

# **Отчет по лабораторной работе № 5**

**дисциплина: Архитектура компьютера**

Клюкин Михаил Александрович

# 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>
3.1	Основные принципы работы компьютера . . . . .	7
3.2	Ассемблер и язык ассемблера . . . . .	9
3.3	Процесс создания и обработки программы на языке ассемблера .	10
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>11</b>
4.1	Программа Hello world! . . . . .	11
4.2	Транслятор NASM . . . . .	11
4.3	Расширенный синтаксис командной строки NASM . . . . .	12
4.4	Компоновщик LD . . . . .	12
4.5	Запуск исполняемого файла . . . . .	13
<b>5</b>	<b>Выполнение заданий для самостоятельной работы</b>	<b>14</b>
<b>6</b>	<b>Контрольные вопросы для самопроверки</b>	<b>16</b>
<b>7</b>	<b>Выводы</b>	<b>18</b>
	<b>Список литературы</b>	<b>19</b>

# Список иллюстраций

4.1	Создание рабочего каталога и файла hello.asm . . . . .	11
4.2	Написание текста программы . . . . .	11
4.3	Компиляция и проверка . . . . .	12
4.4	Компиляция hello.asm в obj.o . . . . .	12
4.5	Создание исполняемого файла hello . . . . .	13
4.6	Создание исполняемого файла main . . . . .	13
4.7	Запуск исполняемого файла . . . . .	13
5.1	Изменение вывода в файле lab5.asm . . . . .	14
5.2	Создание исполняемого файла lab5 . . . . .	15
5.3	Перемещение файлов в репозиторий и загрузка на Github . . . . .	15

## Список таблиц

# 1 Цель работы

Освоить процедуру компиляции и сборки программ, написанных на ассемблере NASM.

## 2 Задание

1. Написать программы, которая печатает на экран “Hello world!”.
2. Провести трансляцию и компоновку этой программы.
3. Написать программу, которая печатает на экран “Клюкин Михаил”.
4. Провести трансляцию и компоновку этой программы.
5. Скопировать полученные файлы в локальный репозиторий, загрузить их на Github.

## 3 Теоретическое введение

### 3.1 Основные принципы работы компьютера

Основными функциональными элементами ЭВМ являются центральный процессор, память и периферийные устройства. Взаимодействие этих устройств осуществляется через общую шину, которая представляет собой большое количество проводников, соединяющих устройства друг с другом.

В состав центрального процессора входят следующие устройства:

- арифметико-логическое устройство;
- устройство управления;
- 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.

Примеры регистров общего назначения:

- 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.

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.

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.

## 3.2 Ассемблер и язык ассемблера

Ассемблер — машинноориентированный язык низкого уровня, который позволяет получить наиболее полный доступ к вычислительным ресурсам компьютера. В современных архитектурах невозможно получить полный доступ к ресурсам компьютера. Самым низким уровнем работы программы является обращение напрямую к ядру ОС. Именно на этом уровне и работают программы, написанные на ассемблере. Трансляция команд с языка ассемблера в исполняемый машинный код осуществляется специальной программой — транслятором — ассемблером.

Открытый проект ассемблера. Его версии доступны под различные операционные системы. Он позволяет получать объектные файлы для этих систем. В NASM используется Intel-синтаксис и поддерживаются инструкции x86-64.

Формат записи команд NASM:

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

Метка перед командой связана с адресом данной команды.

Мнемокод — сама инструкция.

Операндами могут быть числа, данные, адреса регистров, адреса оперативной памяти.

В метках допустимы буквы, цифры, а также символы `,` `$`, `#`, `@`, `~`, `.`, `?`. Метка может начинаться с буквы, `.`, или `?`. Перед идентификаторами, которые пишутся как зарезервированные слова, нужно указывать `$` для экранирования. Максимальная длина идентификатора 4096 символов.

Программа также может содержать директивы — инструкции, управляющие работой транслятора и не переводящиеся непосредственно в машинный код. Директивы, используемые для определения данных (констант и переменных), обычно пишутся большими буквами.

### **3.3 Процесс создания и обработки программы на языке ассемблера**

В процессе создания программы на языке ассемблера можно выделить четыре шага:

- Создание текста программы в текстовом редакторе. Файлы на языке ассемблера имеют расширение `.asm`.
- Трансляция — преобразование с помощью транслятора (например, `nasm`) текста программы в машинный код (объектный). На этом этапе может быть получен листинг программы, содержащий дополнительную информацию, созданную транслятором.
- Компоновка или линковка — объектный код обрабатывается линковщиком, который собирает из объектных файлов исполняемый, который обычно не имеет расширения.
- Запуск программы.

## 4 Выполнение лабораторной работы

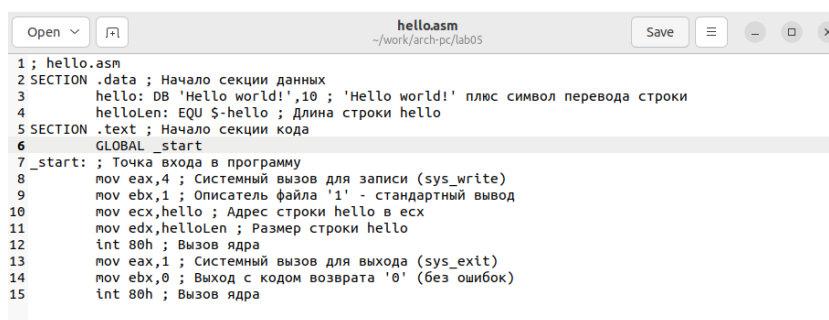
### 4.1 Программа Hello world!

Создали каталог для работы и перешли в него. Создали текстовый файл с именем hello.asm (Рис. 4.1).

```
maklyukin@makyaro-HP-Laptop-15-da0xxx:~$ mkdir ~/work/arch-pc/lab05
maklyukin@makyaro-HP-Laptop-15-da0xxx:~$ cd ~/work/arch-pc/lab05
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ touch hello.asm
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$
```

Рис. 4.1: Создание рабочего каталога и файла hello.asm

Открыли этот файл с помощью текстового редактора gedit и ввели текст программы на языке ассемблера (Рис. 4.2).

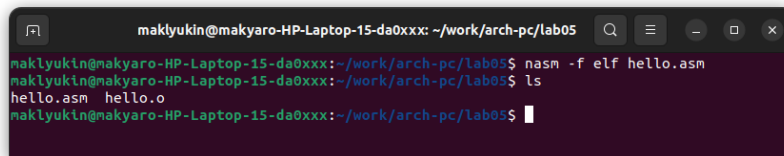


```
1; hello.asm
2 SECTION .data ; Начало секции данных
3     hello: DB 'Hello world!',10 ; 'Hello world!' плюс символ перевода строки
4     helloLen: EQU $-hello ; Длина строки hello
5 SECTION .text ; Начало секции кода
6     GLOBAL _start
7 _start: ; Точка входа в программу
8     mov eax,4 ; Системный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

Рис. 4.2: Написание текста программы

### 4.2 Транслятор NASM

Скомпилировали текст программы с помощью транслятора nasm, проверили, что объектный файл был создан с помощью команды ls (Рис. 4.3).



```
maklyukin@makyaro-HP-Laptop-15-da0xxx: ~/work/arch-pc/lab05
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ nasm -f elf hello.asm
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ ls
hello.asm  hello.o
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$
```

Рис. 4.3: Компиляция и проверка

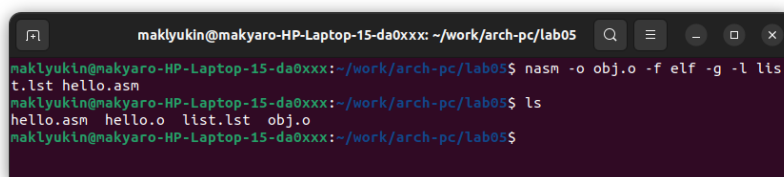
Объектный файл имеет имя hello.o.

## 4.3 Расширенный синтаксис командной строки NASM

Скомпилируем исходный файл hello.asm в obj.o с помощью команды (Рис. 4.4):

```
nasm -o obj.o -f elf -g -l list.lst hello.asm
```

С помощью команды ls проверили, что файлы obj.o и list.lst были созданы (Рис. 4.4).



```
maklyukin@makyaro-HP-Laptop-15-da0xxx: ~/work/arch-pc/lab05
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ nasm -o obj.o -f elf -g -l list.lst hello.asm
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ ls
hello.asm  hello.o  list.lst  obj.o
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$
```

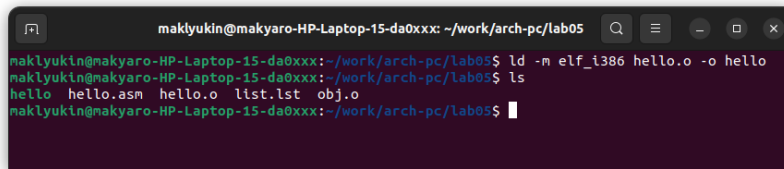
Рис. 4.4: Компиляция hello.asm в obj.o

## 4.4 Компоновщик LD

Передадим объектный файл на обработку компоновщику, чтобы получить исполняемый файл (Рис. 4.5):

```
ld -m elf_i386 hello.o -o hello
```

С помощью команды ls проверим, что исполняемый файл hello был создан (Рис. 4.5).

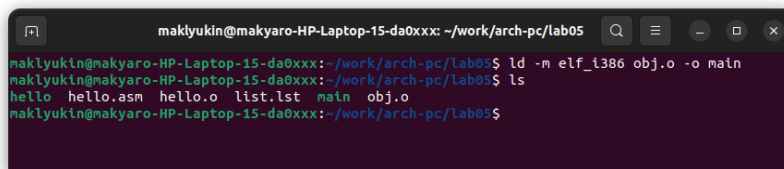


```
maklyukin@makyaro-HP-Laptop-15-da0xxx: ~/work/arch-pc/lab05
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ ld -m elf_i386 hello.o -o hello
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ ls
hello  hello.asm  hello.o  list.lst  obj.o
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$
```

Рис. 4.5: Создание исполняемого файла hello

Также выполним следующую команду, создав при этом исполняемый файл main из объектного файла obj.o (Рис. 4.6):

```
ld -m elf_i386 obj.o -o main
```

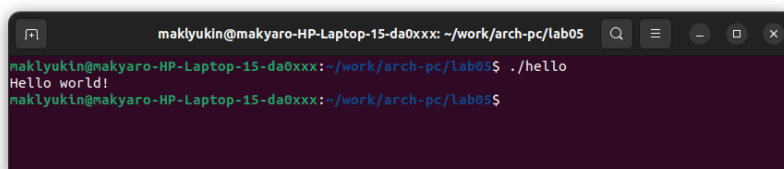


```
maklyukin@makyaro-HP-Laptop-15-da0xxx: ~/work/arch-pc/lab05
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ ld -m elf_i386 obj.o -o main
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ ls
hello  hello.asm  hello.o  list.lst  main  obj.o
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$
```

Рис. 4.6: Создание исполняемого файла main

## 4.5 Запуск исполняемого файла

Запустим исполняемый файл (Рис. 4.7).



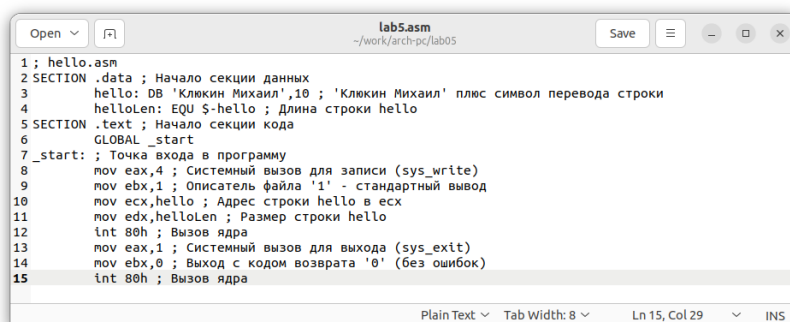
```
maklyukin@makyaro-HP-Laptop-15-da0xxx: ~/work/arch-pc/lab05
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ ./hello
Hello world!
maklyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$
```

Рис. 4.7: Запуск исполняемого файла

## 5 Выполнение заданий для самостоятельной работы

В каталоге `~/work/arch-pc/lab05` с помощью команды `cp` создали копию файла `hello.asm` с именем `lab5.asm`.

С помощью текстового редактора `gedit` внесли изменения в текст программы в файле `lab5.asm` так, чтобы вместо строки “Hello world!” на экран выводилась строка “Клюкин Михаил” (Рис. 5.1).



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

Рис. 5.1: Изменение вывода в файле `lab5.asm`

Оттранслировали полученный текст программы `lab5.asm` в объектный файл, выполнили компоновку объектного файла и запустили полученный исполняемый файл (Рис. 5.2).

```
makyukin@makyaro-HP-Laptop-15-da0xxx: ~/work/arch-pc/lab05
makyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ nasm -f elf lab5.asm
makyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ ls
hello  hello.asm  hello.o  lab5.asm  lab5.o  list.lst  main  obj.o
makyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ ld -m elf_i386 lab5.o -o lab5
makyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ ls
hello  hello.asm  hello.o  lab5  lab5.asm  lab5.o  list.lst  main  obj.o
makyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$ ./lab5
Ключин Михаил
makyukin@makyaro-HP-Laptop-15-da0xxx:~/work/arch-pc/lab05$
```

Рис. 5.2: Создание исполняемого файла lab5

Скопировали файлы hello.asm и lab5.asm в локальный репозиторий в каталог ~/work/study/2022-2023/“Архитектура компьютера”/arch-pc/labs/lab05/, загрузили файлы на Github (Рис. 5.3).

```
makyukin@makyaro-HP-Laptop-15-da0xxx: ~/work/study/2022-2023/Архитектура компьютера/arch-pc
makyukin@makyaro-HP-Laptop-15-da0xxx:~/work/study/2022-2023/Архитектура компьютера/arch-pc$ cp ./hello.asm ~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab05/hello.asm
makyukin@makyaro-HP-Laptop-15-da0xxx:~/work/study/2022-2023/Архитектура компьютера/arch-pc$ cp ./lab5.asm ~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/lab05/lab5.asm
makyukin@makyaro-HP-Laptop-15-da0xxx:~/work/study/2022-2023/Архитектура компьютера/arch-pc$ cd ~/work/study/2022-2023/Архитектура компьютера/arch-pc
makyukin@makyaro-HP-Laptop-15-da0xxx:~/work/study/2022-2023/Архитектура компьютера/arch-pc$ git add
makyukin@makyaro-HP-Laptop-15-da0xxx:~/work/study/2022-2023/Архитектура компьютера/arch-pc$ git commit -am 'Создали файлы hello.asm и lab5.asm'
[master deafc3] Создали файлы hello.asm и lab5.asm
2 files changed, 36 insertions(+)
create mode 100644 labs/lab05/hello.asm
create mode 100644 labs/lab05/lab5.asm
makyukin@makyaro-HP-Laptop-15-da0xxx:~/work/study/2022-2023/Архитектура компьютера/arch-pc$ git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 102 bytes | 501.00 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:makyaro/study-2022-2023_arch-pc.git
makyukin@makyaro-HP-Laptop-15-da0xxx:~/work/study/2022-2023/Архитектура компьютера/arch-pc$
```

Рис. 5.3: Перемещение файлов в репозиторий и загрузка на Github

## 6 Контрольные вопросы для самопроверки

1. Какие основные отличия ассемблерных программ от программ на языках высокого уровня?

Программы, написанные на языке ассемблера, работают на уровне ядра операционной системы. Также они содержат только тот код, который ввел программист.

2. В чем состоит отличие инструкции от директивы на языке ассемблера?

Директивы не переводятся непосредственно в машинный код, они лишь управляют работой транслятора.

3. Перечислите основные правила оформления программ на языке ассемблера.

Каждая команда в ассемблерной программе располагается на отдельной строке. Размещение нескольких команд на одной строке недопустимо. Синтаксис ассемблера чувствителен к регистру.

4. Каковы этапы получения исполняемого файла?

Набор текста программы, трансляция, компоновка или линковка.

5. Каково назначение этапа трансляции?



Преобразование с помощью транслятора текста программы в машинный код, называемый объектным.

6. Каково назначение этапа компоновки?

Обработка объектного кода компоновщиком, который принимает на вход объектные файлы и собирает по ним исполняемый файл.

7. Какие файлы могут создаваться при трансляции программы, какие из них создаются по умолчанию?

При трансляции по умолчанию создаются объектные файлы, а также может быть получен листинг программы.

8. Каковы форматы файлов для `nasm` и `ld`?

Форматы файлов для `nasm` и `ld`: `.o`, `.elf`, `.lst`.

## **7 Выводы**

Освоили процедуру компиляции и сборки программ, написанных на ассемблере NASM.

# Список литературы

1. Демидова А. В. Лабораторная работа №5. Создание и процесс обработки программ на языке ассемблера NASM – Методическое пособие