

Report on Laboratory Work No. 4

Discipline: Computer Architecture

Ali Hosseinabadi

Table of Contents

1	Objective of the Work	5
2	Assignment	6
3	Theoretical Introduction	7
4	Conducting Laboratory Work	10
4.1	Hello World Program	10
4.2	NASM Translator	11
4.3	Extended NASM Command Line Syntax	11
4.4	LD Linker	12
4.5	Running the Executable File	13
4.6	Independent Work Assignments	13
5	Conclusions	16
6	References	17

List of Illustrations

4.1	Creating .asm file	10
4.2	Editing the file	11
4.3	Compiling the program	11
4.4	NASM syntax capabilities	12
4.5	Sending file to the linker	12
4.6	Creating executable file	13
4.7	Running the program	13
4.8	Creating a copy	14
4.9	Checking the functionality of the compiled program	14
4.10	Sending files to local repository	15

List of Tables

1 Objective of the Work

The objective of this laboratory work is to master the procedures for compiling and building programs written in NASM assembly language.

2 Assignment

1. Create a Hello World program!
2. Work with the NASM translator.
3. Work with the extended syntax of the NASM command line.
4. Work with the LD linker.
5. Run the executable file.
6. Complete independent work assignments.

3 Theoretical Introduction

The primary functional elements of any computer are the central processor, memory, and peripheral devices.

The interaction of these devices occurs through a shared bus to which they are connected. Physically, the bus consists of a large number of conductors that connect devices to each other. In modern computers, these conductors are made in the form of conductive tracks on the motherboard.

The main task of the processor is to process information and organize the coordination of all computer nodes. The central processor includes the following units:

- Arithmetic Logic Unit (ALU) — performs logical and arithmetic operations necessary to process information stored in memory;
- Control Unit (CU) — provides control and checks all devices of the computer;
- Registers — ultra-fast temporary storage within the processor for storing intermediate results of instructions; processor registers are divided into two types: general-purpose registers and special registers.

To write programs in assembly language, it is necessary to know what processor registers exist and how they can be used. Most commands in assembly programs use registers as operands. Practically all commands represent transformations of data stored in processor registers, such as moving data between registers or between registers and memory, and performing mathematical or logical operations on data stored in registers.

Access to registers is done by names, rather than by addresses like with main memory. Each register in the x86 architecture has a name consisting of 2 or 3 Latin letters.

For example, here are the names of the main general-purpose registers (these registers are most frequently used in programming):

- RAX, RCX, RDX, RBX, RSI, RDI – 64-bit
- EAX, ECX, EDX, EBX, ESI, EDI – 32-bit
- AX, CX, DX, BX, SI, DI – 16-bit
- AH, AL, CH, CL, DH, DL, BH, BL – 8-bit

Another important node of a computer is Random Access Memory (RAM). RAM is a fast volatile memory that directly interacts with processor nodes, intended for storing programs and data with which the processor works at the current moment. RAM consists of uniform numbered memory cells. The number of a memory cell is the address of the data stored in it.

The peripheral devices of a computer include:

- External memory devices intended for long-term storage of large volumes of data.
- Input-output devices that ensure interaction between the CPU and the external environment.

The computational process of a computer is based on the principle of program control. This means that a computer solves a given task as a sequence of actions recorded in the form of a program.

Instruction codes represent multi-digit binary combinations of 0s and 1s. In the machine instruction code, two parts can be distinguished: operational and address. The operational part stores the code of the command that needs to be executed. The address part contains the data or addresses of the data involved in performing the operation.

When executing each command, the processor carries out a certain standard sequence of actions known as the processor's command cycle. This involves:

1. Forming the address in memory of the next command;

2. Reading the command code from memory and its decoding;
3. Executing the command;
4. Transitioning to the next command.

Assembly language (abbreviated as asm) is a low-level machine-oriented language.

NASM is an open assembler project with versions available for different operating systems, which enables the generation of object files for these systems. NASM uses Intel syntax and supports x86-64 instructions.

4 Conducting Laboratory Work

4.1 Hello World Program

I create a file hello.asm in which I will write a program in assembly language. (Fig. -fig. 4.2)

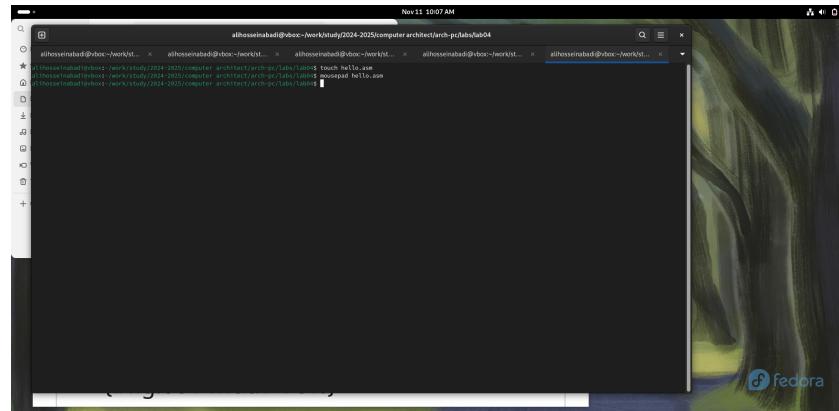


Fig. 4.1: Creating .asm file

Using the editor, I write the program in the created file. (Fig. -fig. 4.2)

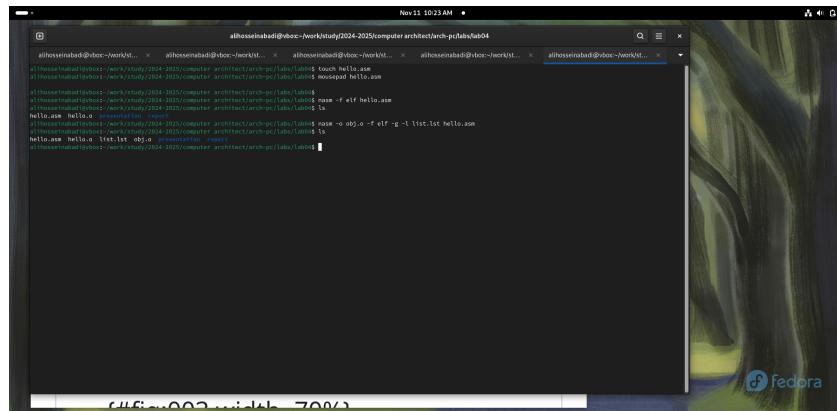


Fig. 4.2: Editing the file

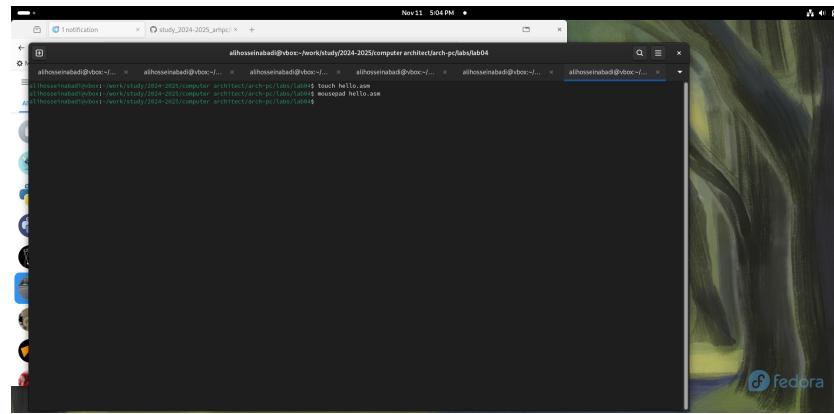
4.2 NASM Translator

I compile my program using NASM. (Fig. -fig. 4.3)

Fig. 4.3: Compiling the program

4.3 Extended NASM Command Line Syntax

I execute the command shown in (Fig. -fig. 4.4), which compiles the source file hello.asm into obj.o; the .o extension indicates that the file is an object file. Additionally, the flags -g and -l prepare the debugging file and listing, respectively.

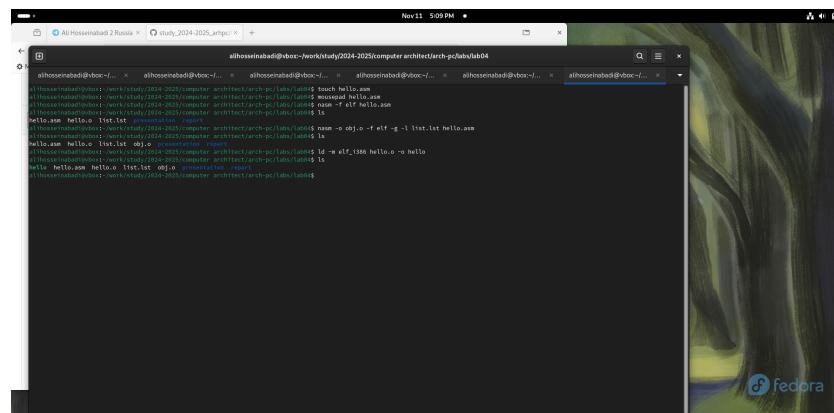


A screenshot of a terminal window titled "Nov 11 5:04 PM". It shows a series of commands being run in a Linux environment. The commands include creating a file ("touch hello.asm"), assembling it ("nasm hello.asm"), linking it ("ld -Telf hello.o -o hello"), and finally running the program ("./hello"). The terminal window has multiple tabs open, and the background shows a Fedora desktop environment.

Fig. 4.4: NASM syntax capabilities

4.4 LD Linker

Then I need to transfer the object file to the linker, which I do using the ld command. (Fig. -fig. 4.5)



A screenshot of a terminal window titled "Nov 11 5:09 PM". It shows the user running the ld command to link an object file named "hello.o" into an executable file named "Hello". The command is "ld -Telf hello.o -o Hello". The terminal window has multiple tabs open, and the background shows a Fedora desktop environment.

Fig. 4.5: Sending file to the linker

Next, I execute the following command ..., the result of this command will be a created file main, compiled from the object file obj.o. (Fig. -fig. 4.6)

Fig. 4.6: Creating executable file

4.5 Running the Executable File

I run the executable file from the current directory. (Fig. -fig. 4.7)

Fig. 4.7: Running the program

4.6 Independent Work Assignments

I create a copy of the file for subsequent work. (Fig. -fig. 4.8)

```

aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ cp hello.asm lab4.asm
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ ls
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ touch hello.asm
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ moosepad hello.asm
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ name -f elf hello.o
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ ld -m elf_i386 hello.o -o hello
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ objdump -p hello.o
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ presentation report
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ ls
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ hello.o
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ cp hello.o lab4.o
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ ls
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ lab4.o
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ moosepad lab4.asm
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ name -f elf lab4.o
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ ld -m elf_i386 lab4.o -o lab4
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ objdump -p lab4.o
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ presentation report
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ ls
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ lab4
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ ./lab4

```

Fig. 4.8: Creating a copy

I edit the copy of the file, replacing the text with my first and last name. (Fig. -fig. 4.9)
I translate the copy of the file into an object file, link it, and run it. (Fig. -fig. 4.9)

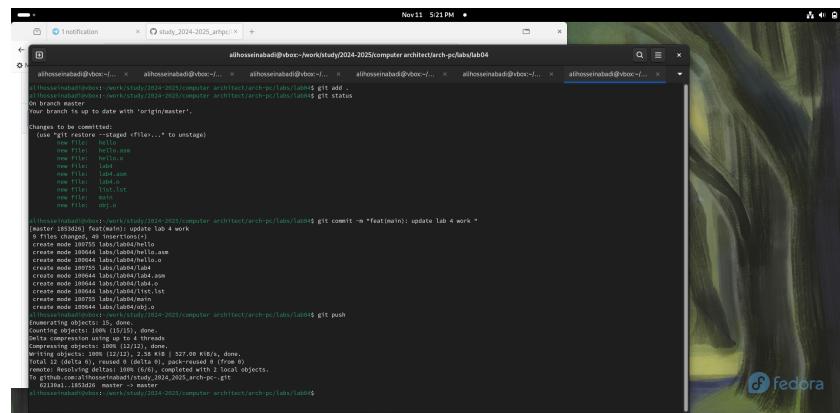
```

aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4
aihossseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab4$ ./lab4
Hello, world!

```

Fig. 4.9: Checking the functionality of the compiled program

Having ensured the program works correctly, I copy the working files into my local repository. (Fig. -fig. 4.10)



The screenshot shows a terminal window with four tabs open, all displaying the same command-line session. The session starts with a 'git add' command, followed by a 'git commit' with a message, and finally a 'git push' command. The terminal is running on a Fedora system, as indicated by the logo in the bottom right corner.

```
alhosseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab04$ git add -A
alhosseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab04$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)

    new file:  hello.asm
    new file:  list.asm
    new file:  list.c
    new file:  list.h
    new file:  obj.o

alhosseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab04$ git commit -m "feat(main): update lab 4 work"
[master 16d3a2e] feat(main): update lab 4 work
 0 files changed, 49 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/list.asm
create mode 100644 labs/lab04/list.c
create mode 100644 labs/lab04/list.h
create mode 100644 labs/lab04/obj.o
alhosseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab04$ git push
Enumerating objects: 10, done.
Counting objects: 1000 (15/15), done.
Delta compression using up to 8 threads.
Compressing objects: 1000 (12/12), done.
Writing objects: 1000 (1000/1000), done.
Total 1000 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0).
remote: Resolving deltas: 100% (1000/1000) done.
To github.com:alhosseiniabadi/study_2024_2025_computer_pc.git
   62130aa..16d3a2e master -> master
alhosseiniabadi@box:~/work/study2024-2025/computer architect/arch-pc-labs/lab04$
```

Fig. 4.10: Sending files to local repository

Uploading changes to my remote repository on GitHub.

5 Conclusions

In carrying out this laboratory work, I learned the procedures for compiling and building programs written in the NASM assembly language.

6 References

1. Example of laboratory work execution
2. Course at RUDN University
3. Laboratory Work No. 4
4. Programming in NASM Assembly Language by A.V. Stolyarov