answer to labrotary work 6

Discipline: Computer Architecture

ali hosseinabadi

Content

List of illustrations

List of Tables

1 Purpose of the work

The purpose of this lab is to master the arithmetic instructions of the NASM assembly language.

2 Task

1. Symbolic and numerical data in NASM
2. Performing arithmetic operations in NASM
3. Completing assignments for independent work

3 Theoretical Introduction

Most assembly language instructions require operands to be processed. The address of an operand provides the location where the data to be processed is stored. This can be data stored in a register or in a memory cell. - Register addressing - operands are stored in registers and the names of these registers are used in the command, for example: mov ax, bx. - Direct addressing - the operand value is specified directly in the command, for example: mov ax, 2. - Memory addressing - the operand specifies an address in memory. The command specifies a symbolic designation of the memory cell on whose contents the operation must be performed. Information is entered from the keyboard and displayed on the screen in symbolic form. This information is encoded according to the ASCII character code table. ASCII is an abbreviation for American Standard Code for Information Interchange. According to the ASCII standard, each character is encoded by one byte. There is no NASM instruction that outputs numbers (not in symbolic form). Therefore, for example, to output a number, you must first convert its digits into the ASCII codes of these digits and output these codes to the screen, not the number itself. If you output a number to the screen directly, the screen will perceive it not as a number, but as a sequence of ASCII characters - each byte of the number will be perceived as one ASCII character - and will output these characters to the screen. A similar situation occurs when entering data from the keyboard. The entered data will be characters, which will make it impossible to obtain a correct result when performing arithmetic operations on them. To solve this problem, it is necessary to convert ASCII characters to numbers and vice versa.

4 Performing the lab work

4.1 Symbolic and numerical data in NASM

I go into lab work 6, create a file there (рис. -fig. 4.1).

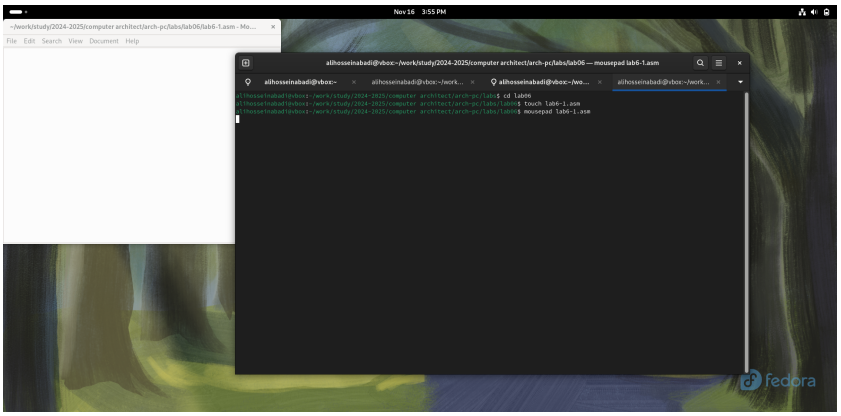

Fig. 4.1: go to lab difrectory

In the created file I enter the program from the listing (рис. -fig. 4.2).

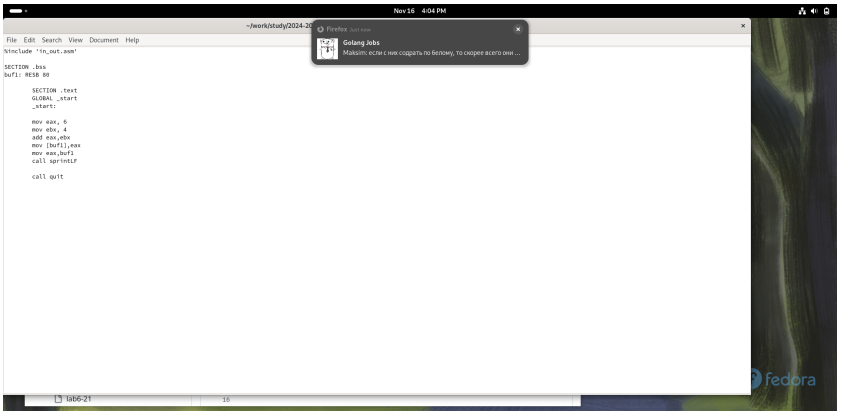

Fig. 4.2: Saving a new program

I create an executable file and run it, the program output differs from what was initially expected, because the character codes together give the character j according to the ASCII table. {#fig:003 width=70%}

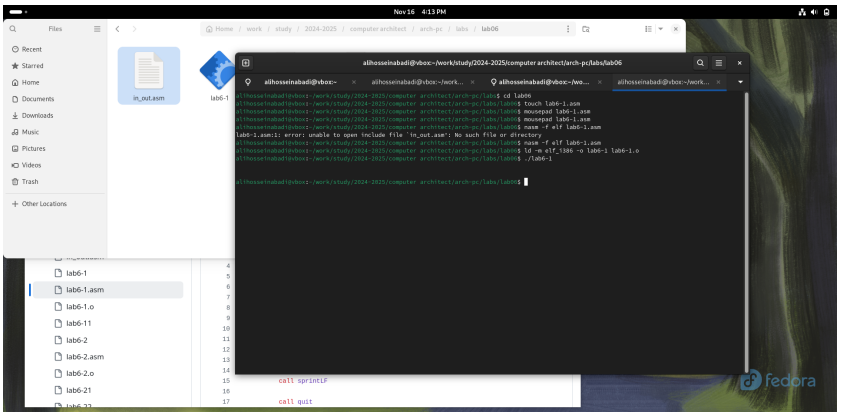

Fig. 4.3: Launching the original program

I change the text of the original program by removing the quotes (рис. -fig. 4.4).

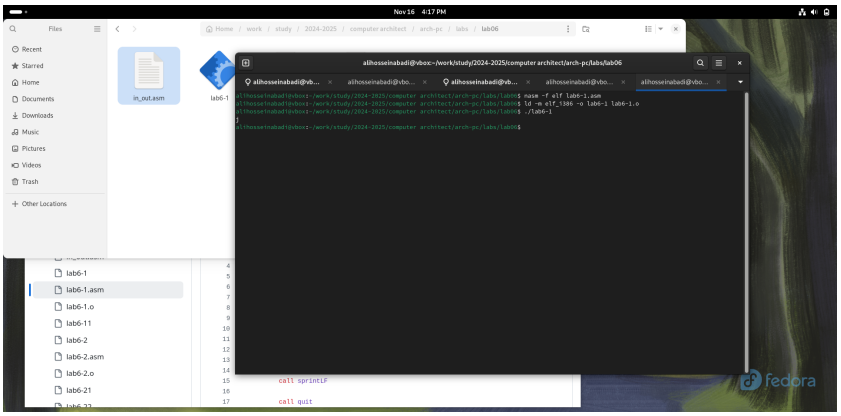

Fig. 4.4: change the program

This time the program returned an empty line, this is because the symbol 10 means a new line. (рис. -fig. 4.5).

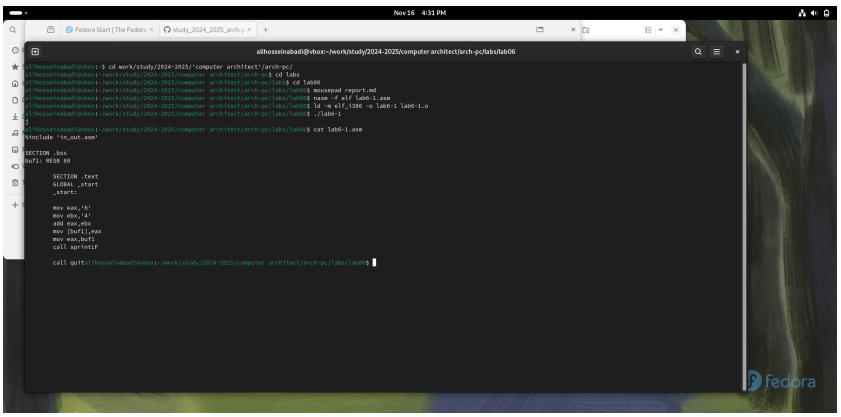

Fig. 4.5: running the new program

I create a new file for the future program and write the code from the listing into it (рис. -fig. 4.6).

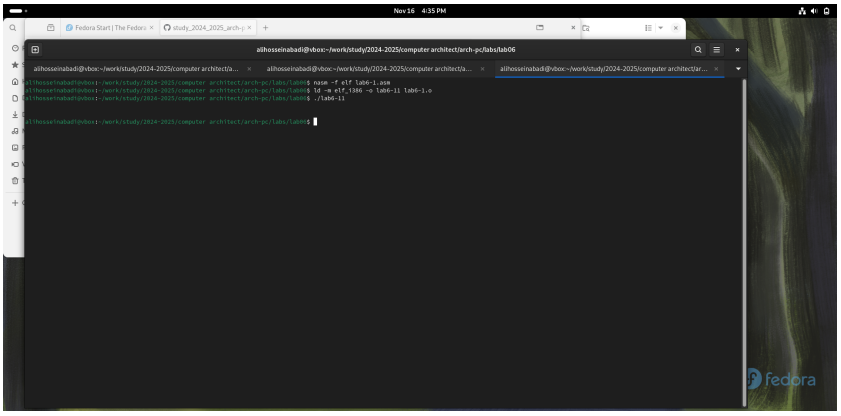

Fig. 4.6: second program

I create an executable file and run it, now the result 106 is displayed, the program, as the first time, added up the character codes, but output the number itself, and not its symbol, thanks to replacing the output function with iprintLF (рис. -fig. 4.7).

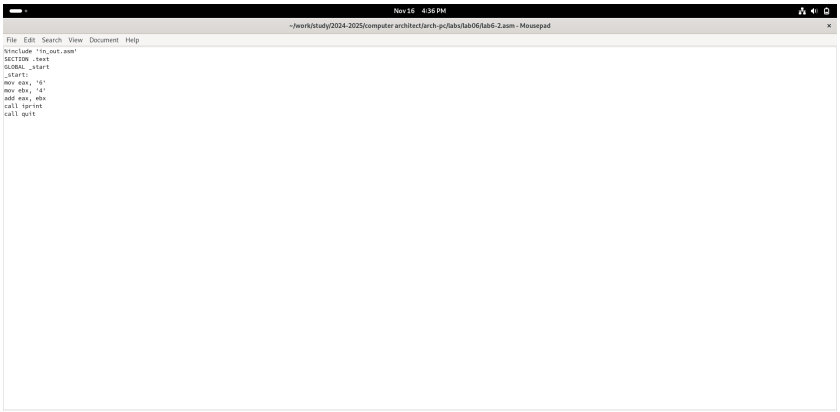

Fig. 4.7: Output of the second program

After removing the quotes in the program, I run it again and get the result I originally intended. (рис. -fig. 4.8).

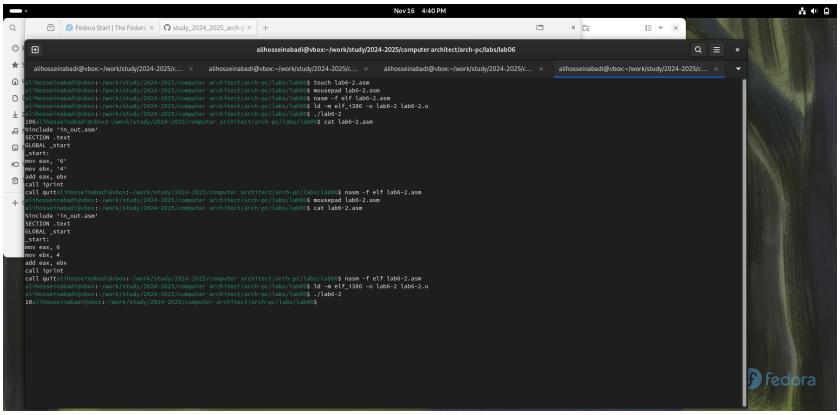

Fig. 4.8: output the changed program

Replacing the output function with iprint gives me the same result but without the line break (рис. -fig. 4.9).

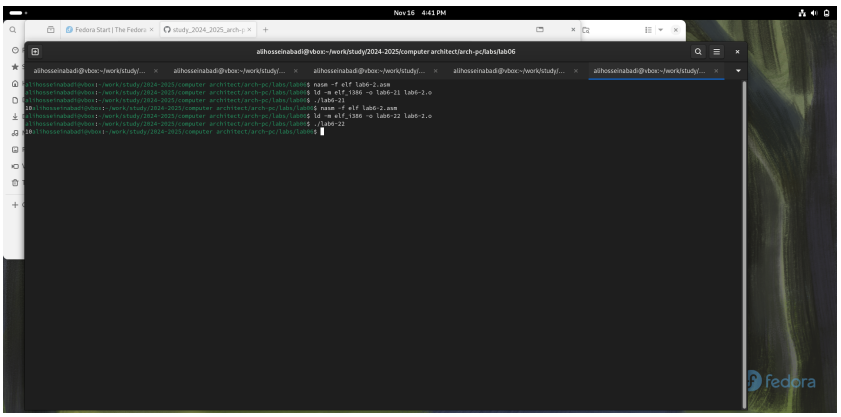

Fig. 4.9: Replacing the output function in the second program

4.2 Performing arithmetic operations in NASM

I create a new file and copy the contents of the listing into it(рис. -fig. 4.10).

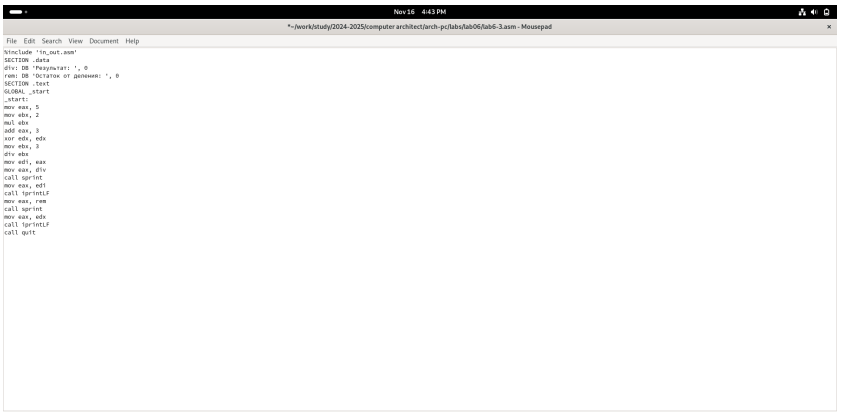

Fig. 4.10: third program

The program performs arithmetic calculations, the resulting expression and its remainder from division are output (рис. -fig. 4.11).

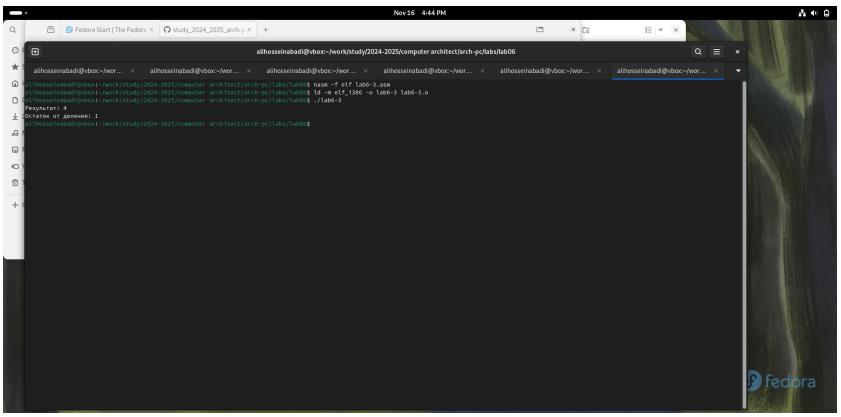

Fig. 4.11: Launching the third program

Replacing variables in the program for the expression f(x) = (4*6+2)/5 (рис. -fig. 4.12).

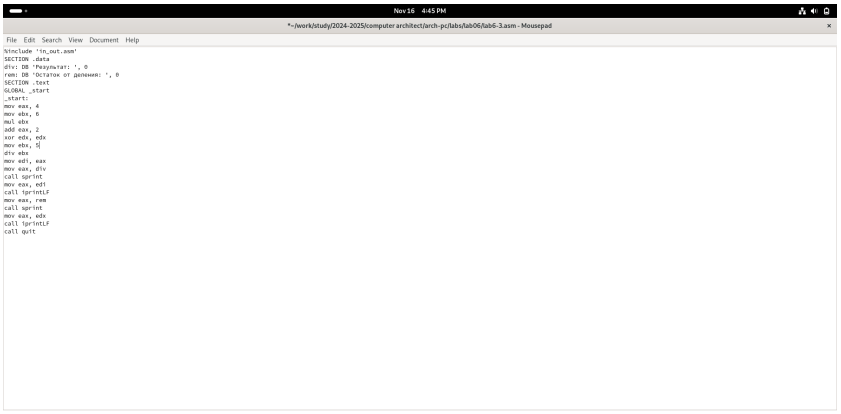

Fig. 4.12: change the third program

Running the program gives the correct result (рис. -fig. 4.13).

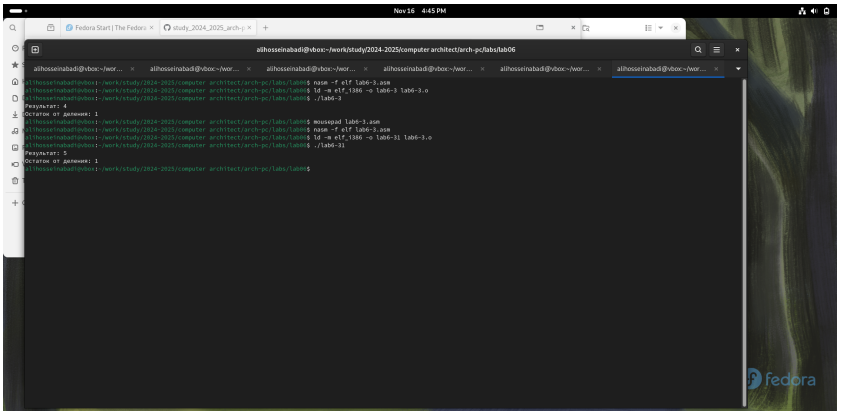

Fig. 4.13: run the modified program

I create a new file and place the text from the listing (рис. -fig. 4.14).

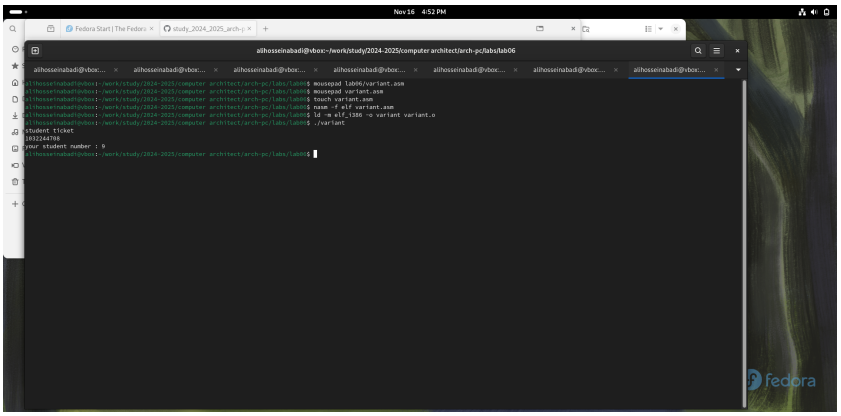

Fig. 4.14: Program for calculating the variant

After running the program and entering my student ID number, I received my version for further work. (рис. -fig. 4.15).

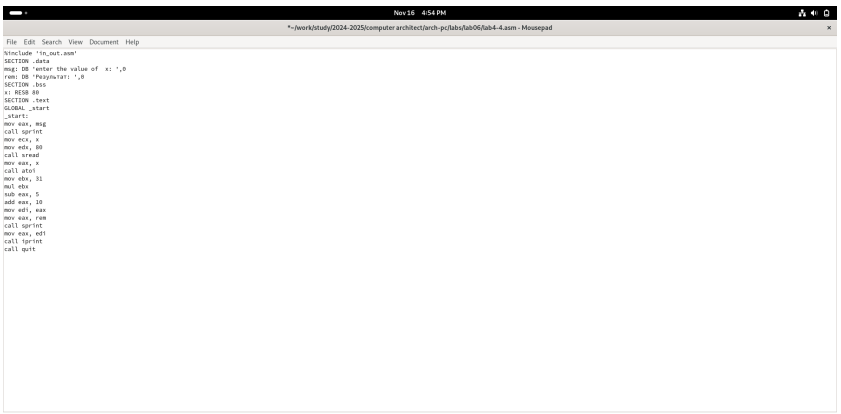

Fig. 4.15: Running the program to calculate the variant

4.3 Answers to security questions

1. The following lines of code are responsible for displaying the message “Your option”:

```
mov eax,rem
call sprint
```

2. The instruction mov ecx, x is used to put the address of the input string x into the ecx register mov edx, 80 - writes the length of the input string into the edx register call sread - calls a subroutine from an external file that provides input of a message from the keyboard.

3. call atoi is used to call a subroutine from an external file that converts the ascii code of a character into an integer and writes the result into the eax register.

4. The lines responsible for calculating the variant are:

```
xor edx,edx ; reset edx for correct work div
mov ebx,20 ; ebx = 20
div ebx ; eax = eax/20, edx - remainder from division
inc edx ; edx = edx + 1
```

5. When the div ebx instruction is executed, the remainder of the division is written to the edx register.

6. The inc edx instruction increases the value of the edx register by 1.

7. The following lines are responsible for displaying the results of calculations on the screen:

```
mov eax,edx
call iprintLF
```

4.4 Assignment for independent work

In accordance with the selected option, I will implement a program for calculating the function f(x) = 10+(31x-5), checking on several variables shows the correct execution of the program (рис. -fig. 4.16).

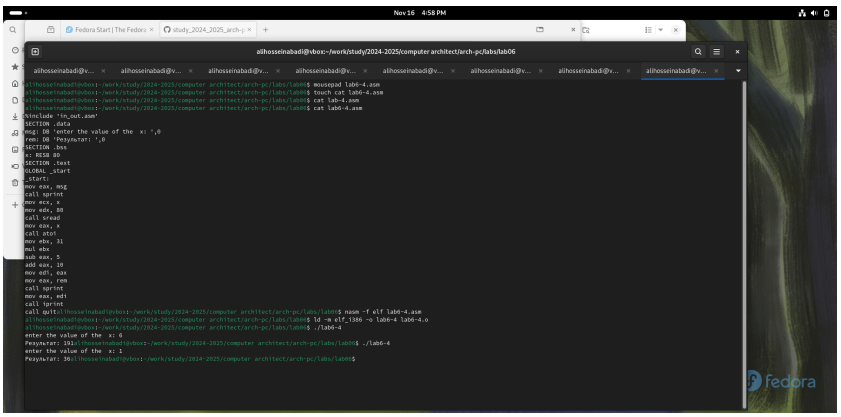

Fig. 4.16: run the program

I am attaching the code of my program:

```
%include 'in_out.asm'
SECTION .data
msg: DB 'enter a number for x: ',0
rem: DB 'result: ',0
SECTION .bss
x: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
mov ebx, 31
mul ebx
sub eax, 5
```

```
add eax, 10
mov edi, eax
mov eax, rem
call sprint
mov eax, edi
call iprint
```

5 Conclusions

During this lab I mastered the arithmetic instructions of the NASM assembly language.

6 References

1. Пример выполнения лабораторной работы
2. Курс на ТУИС
3. Лабораторная работа №6
4. Программирование на языке ассемблера NASM Столяров А. В.