

Report on Laboratory Work No. 7

Discipline: Computer Architecture

ali hosseiabadi

Table of Contents

1 Purpose of the Work	5
2 Assignment	6
3 Theoretical Introduction	7
4 Laboratory Work Execution	8
4.1 Implementing Jumps in NASM	8
4.2 Study of the Listing File Structure	12
4.3 Independent Work Assignments	13

List of Figures

4.1	Creating a directory and file for the program	8
4.2	Saving the program	8
4.3	Running the program	9
4.4	Modifying the program	9
4.5	Running the modified program	10
4.6	Modifying the program	10
4.7	Checking the changes	11
4.8	Saving the new program	11
4.9	Checking the program from the listing	12
4.10	Checking the listing file	12
4.11	Deleting an operand from the program	13
4.12	Viewing the error in the listing file	13
4.13	First program of independent work	14

List of Tables

1 Purpose of the Work

Study of conditional and unconditional jump instructions. Acquiring skills in writing programs using jumps. Introduction to the purpose and structure of a listing file.

2 Assignment

1. Implement jumps in NASM
2. Study the structure of listing files
3. Independently write programs based on the materials of the laboratory work

3 Theoretical Introduction

To implement branching in assembly language, the so-called control transfer instructions or jump instructions are used. There are two types of jumps: * Conditional jump – the execution or non-execution of a jump to a specific point in the program depending on the condition check. * Unconditional jump – the execution of a control transfer to a specific point in the program without any conditions.

4 Laboratory Work Execution

4.1 Implementing Jumps in NASM

Create a directory for the laboratory work programs No. 7 (Fig. -fig. 4.1).

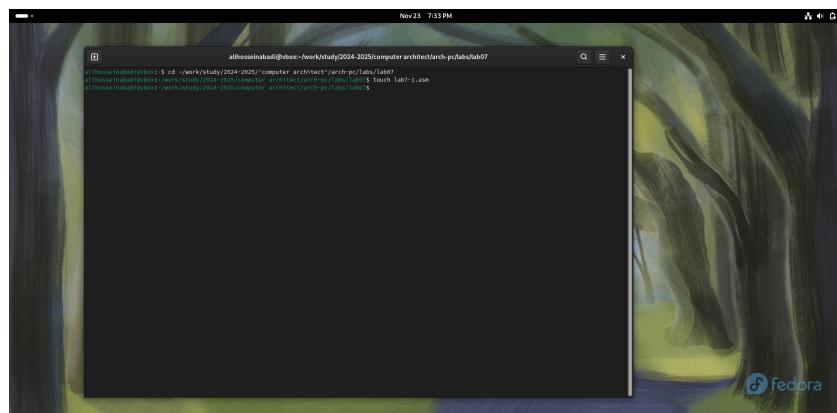


Fig. 4.1: Creating a directory and file for the program

Copy the code from the listing into the file of the future program. (Fig. -fig. 4.2).

A screenshot of a 'mousepad' application window titled 'mousepad * ~/work/study/2024-2025/computer architect/arch-pc/labs/lab07/lab7.1.asm'. The window contains assembly code for a program. The code includes sections for data and code, with labels like .start, .label1, .label2, and .label3, and instructions like mov, call, and jmp. The assembly code is identical to the one shown in Fig. 4.1.

Fig. 4.2: Saving the program

Upon running the program, I confirmed that the unconditional jump indeed changes the order of instruction execution (Fig. -fig. 4.3).

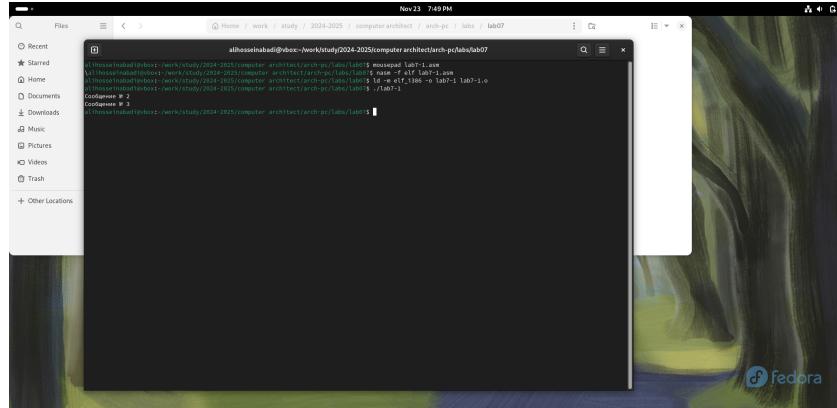


Fig. 4.3: Running the program

I modified the program so that the order of function execution changes (Fig. -fig. 4.4).

```

File Edit Search View Document Help
*/work/study/2024-2025/computer architect/arch-pc-labs/lab07/lab7-1.asm - Mousepad

SECTION .data
msg1 DB 'Confidence # 1', 0
msg2 DB 'Confidence # 2', 0
msg3 DB 'Confidence # 3', 0

SECTION .text
GLOBAL _start
_start:
    jmp _label2
_label1:
    mov eax, msg1
    call sprintf
    jmp _label1

_label2:
    mov eax, msg2
    call sprintf
    jmp _label2

_label3:
    mov eax, msg3
    call sprintf
    jmp _label3

_end:
    call quit

```

Fig. 4.4: Modifying the program

Run the program and check that the applied changes are correct (Fig. -fig. 4.5).

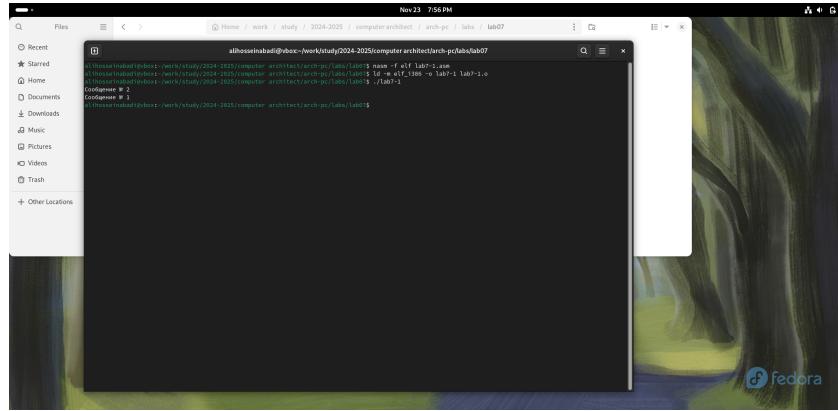


Fig. 4.5: Running the modified program

Now, I modify the program text so that all three messages are displayed in reverse order (Fig. -fig. 4.6).

```

Nov 23 7:57 PM
~/work/study/2024-2025/computer_architect/arch-pc/labs/lab07/lab7-1.asm - Mouspad

File Edit Search View Document Help
#include <io_out.asm>

SECTION .data
msg1 DB "Confidence # 1: ", 0
msg2 DB "Confidence # 2: ", 0
msg3 DB "Confidence # 3: ", 0

SECTION .text
GLOBAL _start
_start:
    jmp _label1
    .label1:
    mov eax, msg1
    call sprintlf
    jmp _label2
    .label2:
    mov eax, msg2
    call sprintlf
    jmp _label3
    .label3:
    mov eax, msg3
    call sprintlf
    jmp _end
    .end:
    call quit

```

Fig. 4.6: Modifying the program

The work was completed correctly, the program displays the messages in the order I wanted (Fig. -fig. 4.7).

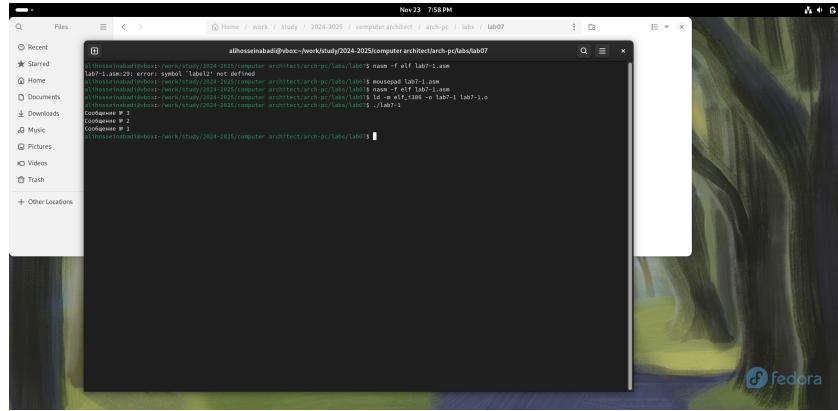


Fig. 4.7: Checking the changes

Create a new working file and insert the code from the next listing (Fig. -fig. 4.8).

```

Nov 23 8:04 PM
~/work/study/2024-2025/computer architect/arch-pc/labs/lab07/lab7-2.asm - Mousepad

File Edit Search View Document Help
#include <in,out.asm>

SECTION .data
msg db "Hello World!", 0h
msg_db equ %offset msg, 0h
len equ $ - msg_db
C dd 24h

SECTION .bss
msg resb 10
B resb 10

SECTION .text
global _start

_start:
    mov ax, msg
    call sstr
    mov ebx, B
    mov edx, 10
    call write
    mov ebx, B
    call sstr
    mov edx, 10
    call write
    mov ebx, A
    mov [bx], ecx
    mov ebx, C
    jg check
    mov ebx, C
    mov [bx], ecx
    check:
    mov eax, ebx
    call atoi
    mov ebx, eax
    mov ebx, [bx]
    cmp ebx, [B]
    jg no
    mov ebx, [bx]
    mov [bx], ecx
no:
    mov ebx, msg
    call sstr
    mov ebx, [bx]
    call printf
    call quit

```

Fig. 4.8: Saving the new program

The program displays the value of the variable with the maximum value. I check the program with different input data (Fig. -fig. 4.9).

The terminal window shows the assembly code being run:

```

Nov 23 8:41 PM
alibesetiahd@box:~/work/study/2024-2025/computer architect/arch-pc/labs/lab07$ ./lab07-2
largest number: 50

```

The mousepad window shows the assembly listing file:

```

; Set input size
80: mov edx, 10 ; Read input
81: call read ; Read input
82: mov eax, B ; Load A
83: cmp eax, C ; Compare A and C
84: mov [B], eax ; Store integer value
85: mov eax, B ; Load B
86: cmp eax, C ; Compare B and C
87: mov [C], eax ; Store integer value
88: mov eax, B ; Load B
89: cmp eax, C ; Compare B and C
90: mov ecx, [A] ; Load A
91: cmp eax, C ; Compare A and C
92: jg check_B ; Jump if A > C
93: mov eax, C ; Otherwise load C
94: check_B:
95: mov eax, B ; Load B
96: cmp eax, C ; Compare with B
97: jg fin ; Jump if max found
98: mov eax, [B] ; Load B as max
99: check_B:
100: mov ecx, [A] ; Load A
101: cmp eax, C ; Compare A and C
102: jg check_B ; Jump if A > C
103: mov eax, C ; Otherwise load C
104: check_B:
105: mov eax, [B] ; Load B
106: cmp eax, C ; Compare with B
107: jg fin ; Jump if max found
108: mov eax, [B] ; Load B as max
109: fin:
110: mov eax, msg2 ; Load result message
111: call printf ; Print message
112: mov eax, [max] ; Load max value
113: call printf ; Print max value
114: call exit ; Exit program
115: 
```

Fig. 4.9: Checking the program from the listing

4.2 Study of the Listing File Structure

Create a listing file using the -l flag of the nasm command and open it using the text editor mousepad (Fig. -fig. 4.10).

The listing file contains the assembly code with line numbers and comments:

```

File Edit Search View Document Help
E: /home/alibesetiahd/work/study/2024-2025/computer architect/arch-pc/labs/lab07/lab07-2.sasm - Mousepad
1: ; Set input size
2: 80: mov edx, 10 ; Read input
3: 81: call read ; Read input
4: 82: mov eax, B ; Load A
5: 83: cmp eax, C ; Compare A and C
6: 84: mov [B], eax ; Store integer value
7: 85: mov eax, B ; Load B
8: 86: cmp eax, C ; Compare B and C
9: 87: mov [C], eax ; Store integer value
10: 88: mov eax, B ; Load B
11: 89: cmp eax, C ; Compare B and C
12: 90: mov ecx, [A] ; Load A
13: 91: cmp eax, C ; Compare A and C
14: 92: jg check_B ; Jump if A > C
15: 93: mov eax, C ; Otherwise load C
16: 94: check_B:
17: 95: mov eax, B ; Load B
18: 96: cmp eax, C ; Compare with B
19: 97: jg fin ; Jump if max found
20: 98: mov eax, [B] ; Load B as max
21: 99: check_B:
22: 100: mov ecx, [A] ; Load A
23: 101: cmp eax, C ; Compare A and C
24: 102: jg check_B ; Jump if A > C
25: 103: mov eax, C ; Otherwise load C
26: 104: check_B:
27: 105: mov eax, [B] ; Load B
28: 106: cmp eax, C ; Compare with B
29: 107: jg fin ; Jump if max found
30: 108: mov eax, [B] ; Load B as max
31: 109: fin:
32: 110: mov eax, msg2 ; Load result message
33: 111: call printf ; Print message
34: 112: mov eax, [max] ; Load max value
35: 113: call printf ; Print max value
36: 114: call exit ; Exit program
37: 
```

Fig. 4.10: Checking the listing file

The first value in the listing file is the line number, which may not match the line number of the original file. The second entry is the address, the offset of the machine code relative to the start of the current segment, followed by the machine code itself, and the line ends with the original program text with comments.

Delete one operand from a random instruction to check the behavior of the listing file in the future (Fig. -fig. 4.11).

```

Nov 23 8:45 PM
/home/lab7-2.asm - Mesched
File Edit Search View Document Help
file:///home/lab7-2.asm

SECTION .data
msg db 'enter B: ', dh
msg2 db 'largest number: ', dh
A dd ?B?
B dd ?C?
C dd ?D?

SECTION .text
GLOBAL _start_
_start:
    mov eax, msg
    call sprint
    mov ecx, B
    mov edx, 10
    call sread
    mov eax, B
    add eax, B
    mov [B], eax

    mov ecx, [A]
    mov [max], ecx
    cmp ecx, [C]
    jg check_B
    mov ecx, [C]
    mov [max], ecx

    check_A:
    mov eax, [max]
    cmp eax, [B]
    jg check_B
    mov eax, [B]
    mov [max], eax

    mov ecx, [max]
    cmp ecx, [D]
    jg check_D
    mov eax, [D]
    mov [max], eax

    mov eax, max
    call sprint
    mov eax, max
    call printF
    mov eax, exit
    call exit

```

Fig. 4.11: Deleting an operand from the program

In the new listing file, an error that occurred during the translation of the file is displayed. No output files are created other than the listing file. (Fig. -fig. 4.12).

```

Nov 23 8:50 PM
alhosseiniabadi@box:~/work/study/2024-2025/computer_architect/arch-pr-labs/lab7-2
alhosseiniabadi@box:~/work/study/2024-2025/computer_architect/arch-pr-labs/lab7-2$ nasm -f elf lab7-2.asm
alhosseiniabadi@box:~/work/study/2024-2025/computer_architect/arch-pr-labs/lab7-2$ ld -m elf_i386 -o lab7-2 lab7-2.o
lab7-2.o: In function `main':
/home/alhosseiniabadi/box/work/study/2024-2025/computer_architect/arch-pr-labs/lab7-2.c:27:1: error: syntax error near unexpected token <eof>
}
^
/home/alhosseiniabadi/box/work/study/2024-2025/computer_architect/arch-pr-labs/lab7-2.o:1: error: /tmp/ccHkWdZt.o: reloc R_386_JUMP_SIB_EBP32 against symbol _start_
/home/alhosseiniabadi/box/work/study/2024-2025/computer_architect/arch-pr-labs/lab7-2.o:1: error: /tmp/ccHkWdZt.o: relocation R_X86_64_JUMP_SIB_EBP32 against symbol _start_ overwriting _start_
ld: 1 error
alhosseiniabadi@box:~/work/study/2024-2025/computer_architect/arch-pr-labs/lab7-2$ ./lab7-2

```

Fig. 4.12: Viewing the error in the listing file

4.3 Independent Work Assignments

I sincerely do not understand what variant I should have obtained during laboratory work No. 7, so I will use my variant – the ninth one – from the previous laboratory work. I return the operand to the function in the program and modify it so that it outputs the variable with the smallest value (Fig. -fig. 4.13).

```

Nov 23 8:54 PM
~/work/study/2024-2025/computer architect/arch-pc/labs/lab7_3.asm - Mesged

File Edit Search View Document Help
%include 'in_out.asm'
SECTION .data
msg1 db 'Введіть значення переменної A: ', 0
msg2 db 'Виведіть значення переменної A: ', 0
SECTION .text
_start:
    ; Виведіть значення переменної A:
    mov eax, msg1
    call sprint
    mov edx, 80
    SECTION .start
    _start:
        mov eax, msg2
        call sprint
        mov ecx, 2
        mov edx, 80
        call read
        mov eax, x
        call atoi
        mov ebx, 1
        mov eax, msg_x
        call sprint
        mov ecx, 2
        mov edx, 80
        call read
        mov eax, x
        call atoi
        mov edx, 80
        mov eax, msg_a
        call sprint
        mov ecx, 2
        mov edx, 80
        call read
        mov eax, a
        call atoi
        mov ebx, 1
        mov eax, msg_x
        call sprint
        mov ecx, 2
        mov edx, 80
        call read
        mov eax, x
        call atoi
        mov edx, 80
        cmp edx, ebx
        jle add_values
        mov eax, ebx
        mov edx, 80
        call print_result
        add_values:
        mov eax, edx
        mov edx, 80
        call print_result
        print_result:
        mov edx, eax
        mov ecx, 2
        mov edx, 80
        call sprint
        mov eax, edx
        call quit
    call quit

```

Fig. 4.13: First program of independent work

```

Nov 23 8:55 PM
alhosseiniabadi@ebos:~/work/study/2024-2025/computer architect/arch-pc/labs$ touch lab7_3.asm
alhosseiniabadi@ebos:~/work/study/2024-2025/computer architect/arch-pc/labs$ nasm -f elf32 lab7_3.asm
alhosseiniabadi@ebos:~/work/study/2024-2025/computer architect/arch-pc/labs$ ld -m elf32_i386 -o lab7_3 lab7_3.o
alhosseiniabadi@ebos:~/work/study/2024-2025/computer architect/arch-pc/labs$ ./lab7_3
Введіть значення переменної A: 5
Результат: 5
alhosseiniabadi@ebos:~/work/study/2024-2025/computer architect/arch-pc/labs$ ./lab7_3
Введіть значення переменної A: 4
Введіть значення переменної A: 5
Результат: 5
alhosseiniabadi@ebos:~/work/study/2024-2025/computer architect/arch-pc/labs$ ./lab7_3

```

Code of the first

program:

```

“NASM %include ‘in_out.asm’
SECTION .data msg1 db ‘Enter B:’, 0h msg2 db ‘The smallest number:’, 0h A dd ‘24’ C
dd ‘15’

SECTION .bss min resb 10 B resb 10
SECTION .text GLOBAL _start _start:
    mov eax, msg1 call sprint
    mov ecx, B mov edx, 10 call sread
    mov eax, B call atoi mov [B], eax
    mov ecx, [A] mov [min], ecx
    cmp ecx, [C] jg check_B mov ecx, [C] mov [min], ecx
    check_B: mov eax, min call atoi mov [min], eax
    mov ecx, [min] cmp ecx, [B] jb fin mov ecx, [B] mov [min], ecx

```

fin: mov eax, msg2 call sprint mov eax, [min] call iprintLF call qui