

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

по лабораторной работе № 8

дисциплина: Архитектура компьютера

Студент: Ужаков Магомед.

Группа: НПИбд-02-22

МОСКВА

2022 г.

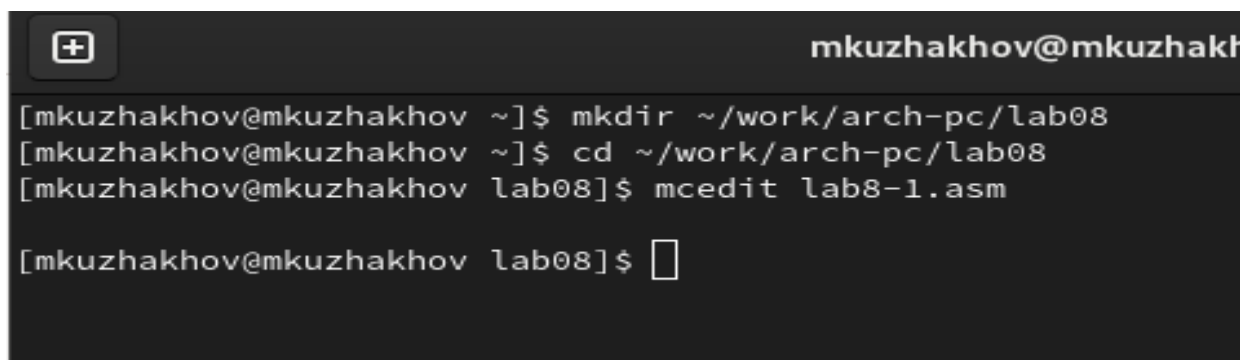
Цель работы:

Изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

Порядок выполнения лабораторной работы:

Реализация переходов в NASM.

Создадим каталог для программ лабораторной работы №8, перейдем в него и создадим файл lab8-1.asm (рис. 1).

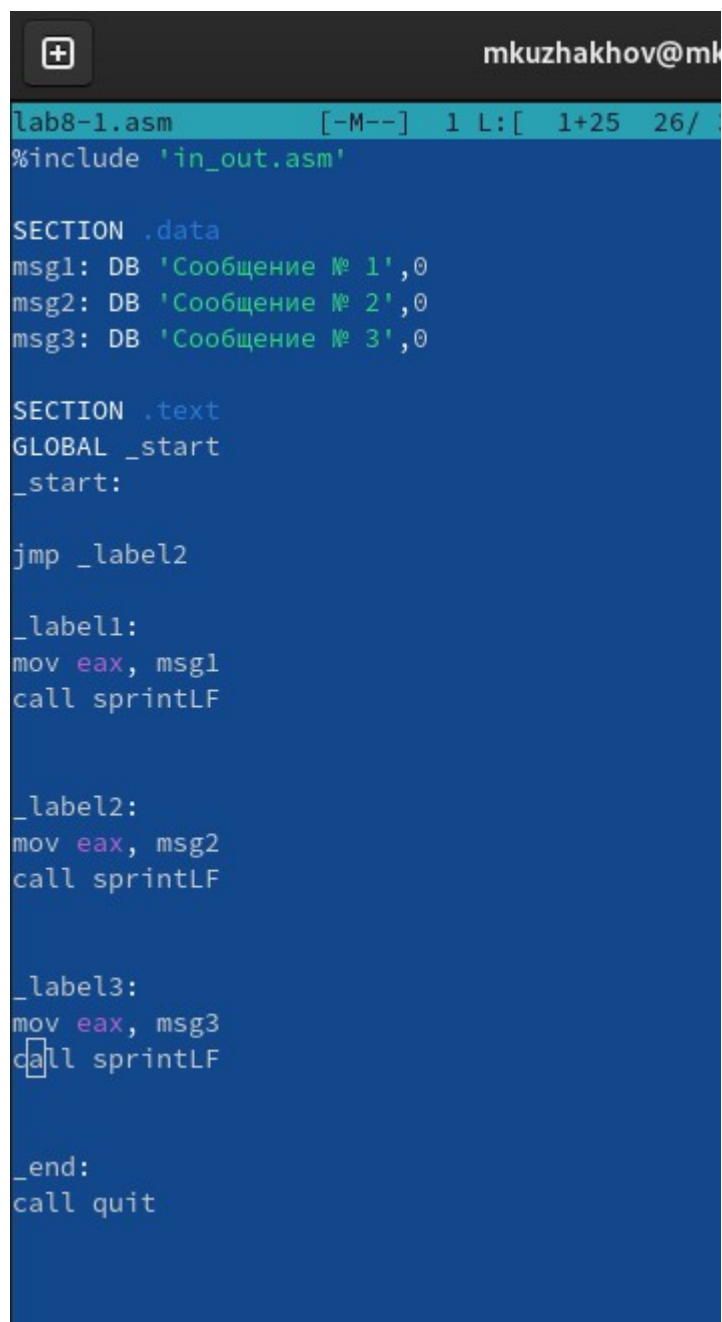


```
mkuzhakhov@mkuzhakhov ~]$ mkdir ~/work/arch-pc/lab08
mkuzhakhov@mkuzhakhov ~]$ cd ~/work/arch-pc/lab08
mkuzhakhov@mkuzhakhov lab08]$ mcedit lab8-1.asm

mkuzhakhov@mkuzhakhov lab08]$
```

рис. 1. Создание каталога и файла lab8-1.asm

Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Введем в файл lab8-1.asm следующий текст программы (рис. 2).



```
lab8-1.asm [-M--] 1 L:[ 1+25 26/
#include 'in_out.asm'

SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label2

_label1:
mov eax, msg1
call printf

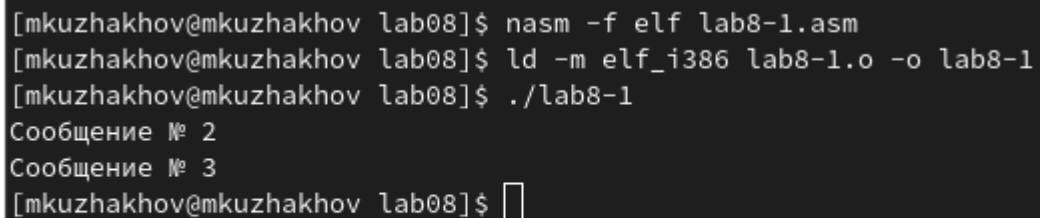
_label2:
mov eax, msg2
call printf

_label3:
mov eax, msg3
call printf

_end:
call quit
```

рис. 2. Текст программы lab8-1

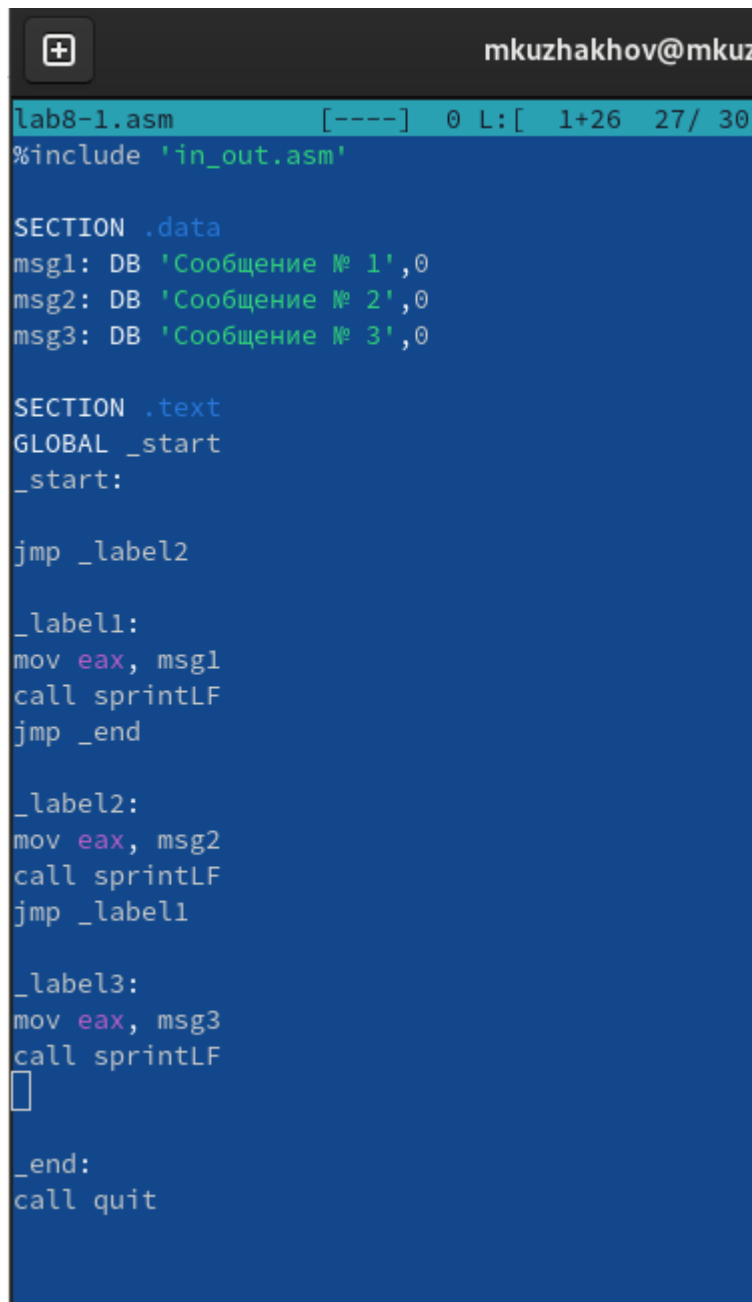
Создадим исполняемый файл и запустим его (рис. 3).



```
[mkuzhakhov@mkuzhakhov lab08]$ nasm -f elf lab8-1.asm
[mkuzhakhov@mkuzhakhov lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1
[mkuzhakhov@mkuzhakhov lab08]$ ./lab8-1
Сообщение № 2
Сообщение № 3
[mkuzhakhov@mkuzhakhov lab08]$
```

рис. 3. Результат работы программы lab8-1

Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения. Инструкция `jmp` позволяет осуществлять переходы не только вперед, но и назад. Изменим программу таким образом, чтобы она выводила сначала 'Сообщение № 2', потом 'Сообщение № 1' и завершала работу. Для этого в текст программы после вывода сообщения № 2 добавим инструкцию `jmp` с меткой `_label1` (т.е. переход к инструкциям вывода сообщения № 1) и после вывода сообщения № 1 добавим инструкцию `jmp` с меткой `_end` (т.е. переход к инструкции `call quit`) (рис. 4).



```
mkuzhakhov@mkuz
lab8-1.asm [----] 0 L: [ 1+26 27/ 30
%include 'in_out.asm'

SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label2

_label1:
mov eax, msg1
call sprintf
jmp _end

_label2:
mov eax, msg2
call sprintf
jmp _label1

_label3:
mov eax, msg3
call sprintf
[]

_end:
call quit
```

рис. 4. Измененный текст программы lab8-1

Создадим исполняемый файл и запустим его (рис. 5).

```
mkuzhakhov@mkuzhakhov: lab08]$ nasm -f elf lab8-1.asm
mkuzhakhov@mkuzhakhov: lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1
mkuzhakhov@mkuzhakhov: lab08]$ ./lab8-1
Сообщение № 2
Сообщение № 1
mkuzhakhov@mkuzhakhov: lab08]$
```

рис. 5. Результат работы измененной программы lab8-1

Далее изменим текст программы lab8-1 так, чтобы сообщения выводились в обратном порядке, затем запустим программу (рис. 6-7).

```
lab8-1.asm [-M--] 11 L: [ 1+21 22
%include 'in_out.asm'

SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label3

_label1:
mov eax, msg1
call sprintLF
jmp _end

_label2:
mov eax, msg2
call sprintLF
jmp _label1

_label3:
mov eax, msg3
call sprintLF
jmp _label2

_end:
call quit
```

рис. 6.
Измененный
текст программы
lab8-1

```
[mkuzhakhov@mkuzhakhov lab08]$ nasm -f elf lab8-1.asm
[mkuzhakhov@mkuzhakhov lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1
[mkuzhakhov@mkuzhakhov lab08]$ ./lab8-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
[mkuzhakhov@mkuzhakhov lab08]$ □
```

рис. 7. Результат работы измененной программы lab8-1

Использование инструкции `jmp` приводит к переходу в любом случае. Однако, часто при написании программ необходимо использовать условные переходы, т.е. переход должен происходить если выполнено какое-либо условие. В качестве примера рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А, В и С. Значения для А и С задаются в программе, значение В вводится с клавиатуры.

Создадим файл `lab8-2.asm` в каталоге `~/work/arch-pc/lab08` и введем в него следующий текст программы (рис. 8-9).

```

mkuzhakhov@mkuzhak
lab8-2.asm [-----] 0 L: [ 1+32 33/ 53] *
#include 'in_out.asm'

SECTION .data
msg1 db 'Введите B: ',0h
msg2 db "Наибольшее число: ",0h
A dd '20'
C dd '50'

SECTION .bss
max 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 [max],ecx

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

check_B:
mov eax,max
call atoi
mov [max],eax

mov ecx,[max]
cmp ecx,[B]
jg fin
mov ecx,[B]
mov [max],ecx

fin:
mov eax, msg2
call sprint
mov eax,[max]
call iprintLF
call quit

```

рис. 8. Текст программы lab8-2
(1)

рис. 9. Текст программы lab8-2
(2)

Создадим файл и проверим его работу для разных значений В (рис. 10).

```
[mkuzhakhov@mkuzhakhov lab08]$ nasm -f elf lab8-2.asm
[mkuzhakhov@mkuzhakhov lab08]$ ld -m elf_i386 lab8-2.o -o lab8-2
[mkuzhakhov@mkuzhakhov lab08]$ ./lab8-2
Введите В: 10
Наибольшее число: 50
[mkuzhakhov@mkuzhakhov lab08]$ ./lab8-2
Введите В: 33
Наибольшее число: 50
[mkuzhakhov@mkuzhakhov lab08]$ ./lab8-2
Введите В: 54
Наибольшее число: 54
[mkuzhakhov@mkuzhakhov lab08]$
```

рис. 10. Работа программы lab8-2

Обратим внимание, что в данном примере переменные А и С сравниваются как символы, а переменная В и максимум из А и С как числа (для этого используется функция `atoi` преобразования символа в число). Это сделано для демонстрации того, как сравниваются данные. Данную программу можно упростить и сравнить все 3 переменные как символы (т.е. не использовать функцию `atoi`). Однако если переменные преобразовать из символов в числа, над ними можно корректно проводить арифметические операции.

Изучение структуры файлы листинга.

Обычно `nasm` создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ `-l` и задав имя файла листинга в командной строке. Создадим файл листинга для программы из файла `lab8-2.asm` (рис. 11).

```
[mkuzhakhov@mkuzhakhov lab08]$
[mkuzhakhov@mkuzhakhov lab08]$ nasm -f elf lab8-2.asm -l lab8-2.lst
[mkuzhakhov@mkuzhakhov lab08]$
```

рис. 11. Создание файла листинга для программы lab8-2

Затем откроем этот файл (рис. 12-13).

```
mkuzhakhov@mkuzhakhov:~/work/arch-pc/lab08 — mcedit lab8-2.lst
lab8-2.lst [----] 0 L:[ 1+ 0 1/229] *(0 /13433b) 0032 0x020
1      %include 'in_out.asm'
2      <1> ;----- slen -----
3      <1> ; Функция вычисления длины сообщения
4      <1> slen:.....
5 00000000 53      <1>      push     ebx.....
6 00000001 89C3     <1>      mov      ebx, eax.....
7      <1>.....
8      <1> nextchar:.....
9 00000003 803800   <1>      cmp      byte [eax], 0...
10 00000006 7403     <1>      jz       finished.....
11 00000008 40      <1>      inc      eax.....
12 00000009 EBF8     <1>      jmp      nextchar.....
13      <1>.....
14      <1> finished:
15 0000000B 29D8     <1>      sub      eax, ebx
16 0000000D 5B      <1>      pop      ebx.....
17 0000000E C3      <1>      ret.....
18      <1>.
19      <1>.
20      <1> ;----- sprint -----
21      <1> ; Функция печати сообщения
22      <1> ; входные данные: mov eax,<message>
23      <1> sprint:
24 0000000F 52      <1>      push     edx
25 00000010 51      <1>      push     ecx
26 00000011 53      <1>      push     ebx
27 00000012 50      <1>      push     eax
28 00000013 E8E8FFFFFF <1>      call     slen
29      <1>.....
30 00000018 89C2     <1>      mov      edx, eax
31 0000001A 58      <1>      pop      eax
32      <1>.....
33 0000001B 89C1     <1>      mov      ecx, eax
34 0000001D BB01000000 <1>      mov      ebx, 1
35 00000022 B804000000 <1>      mov      eax, 4
36 00000027 CD80     <1>      int      80h
37      <1>.
38 00000029 5B      <1>      pop      ebx
39 0000002A 59      <1>      pop      ecx
40 0000002B 5A      <1>      pop      edx
41 0000002C C3      <1>      ret
42      <1>.
43      <1>.
44      <1> ;----- sprintLF -----
45      <1> ; Функция печати сообщения с переводом строки
46      <1> ; входные данные: mov eax,<message>
47      <1> sprintLF:
```

рис. 12. Файл листинга программы lab8-2 (1)


```

47      <1> sprintf:
48 0000002D E8DDFFFFFF      <1>      call    sprintf
49      <1> .
50 00000032 50      <1>      push    eax
51 00000033 B80A000000      <1>      mov     eax, 0Ah
52 00000038 50      <1>      push    eax
53 00000039 89E0      <1>      mov     eax, esp
54 0000003B E8CFFFFFFF      <1>      call    sprintf
55 00000040 58      <1>      pop     eax
56 00000041 58      <1>      pop     eax
57 00000042 C3      <1>      ret
58      <1> .
59      <1> ;----- sread -----
60      <1> ; Функция считывания сообщения
61      <1> ; входные данные: mov eax,<buffer>, mov ebx,<N>
62      <1> sread:
63 00000043 53      <1>      push    ebx
64 00000044 50      <1>      push    eax
65      <1> . . . .
66 00000045 BB00000000      <1>      mov     ebx, 0
67 0000004A B803000000      <1>      mov     eax, 3
68 0000004F CD80      <1>      int     80h
69      <1> .
70 00000051 5B      <1>      pop     ebx
71 00000052 59      <1>      pop     ecx
72 00000053 C3      <1>      ret
73      <1> . . . .
74      <1> ;----- iprint -----
75      <1> ; Функция вывода на экран чисел в формате ASCII
76      <1> ; входные данные: mov eax,<int>
77      <1> iprint:
78 00000054 50      <1>      push    eax . . . . .
79 00000055 51      <1>      push    ecx . . . . .
80 00000056 52      <1>      push    edx . . . . .
81 00000057 56      <1>      push    esi . . . . .
82 00000058 B900000000      <1>      mov     ecx, 0 . . . . .
83      <1> . . . .

```

рис. 13. Файл листинга программы lab8-2 (2)

Как видим на рис. 12 показаны некоторые функции, прописанные в файле in_out.asm, который мы подключаем, на рис. 13 отображена непосредственно часть текста программы lab8-2, разберем несколько строк из этого текста:

Строка 10: после обозначения строки видим 00000000 это адрес, т.е. смещение машинного кода от начала текущего сегмента, поскольку строка 10 является самым начало сегмента SECTION .bss, ее адрес будет 00000000, затем идет машинный код: <res Ah> показывает, что было зарезервировано А байт (то есть 10 байт) памяти для переменной max, которая уже отображена в самой правой строке: max resb 10 – это код программы, здесь мы выделяем память из 10 однобайтовых ячеек по адресу с меткой max.

Строка 33: ее адрес уже равняется 00000122, 7FOC – ассемблированная инструкция jg, которая используется в этой строке для условной передачи управления по результатам арифметического сравнения в 32 строке ecx и [C].

Откроем файл с программой lab8-2.asm и в любой инструкции с двумя операндами удалим один операнд. Выполним трансляцию с получением файла листинга (14-15).

```
SECTION .text

GLOBAL _start
_start:

mov eax, 
call sprint

mov ecx, B
mov edx, 10
call sread

mov eax, B
```

рис. 14. Удаление операнда msg1 в строке mov eax, msg1

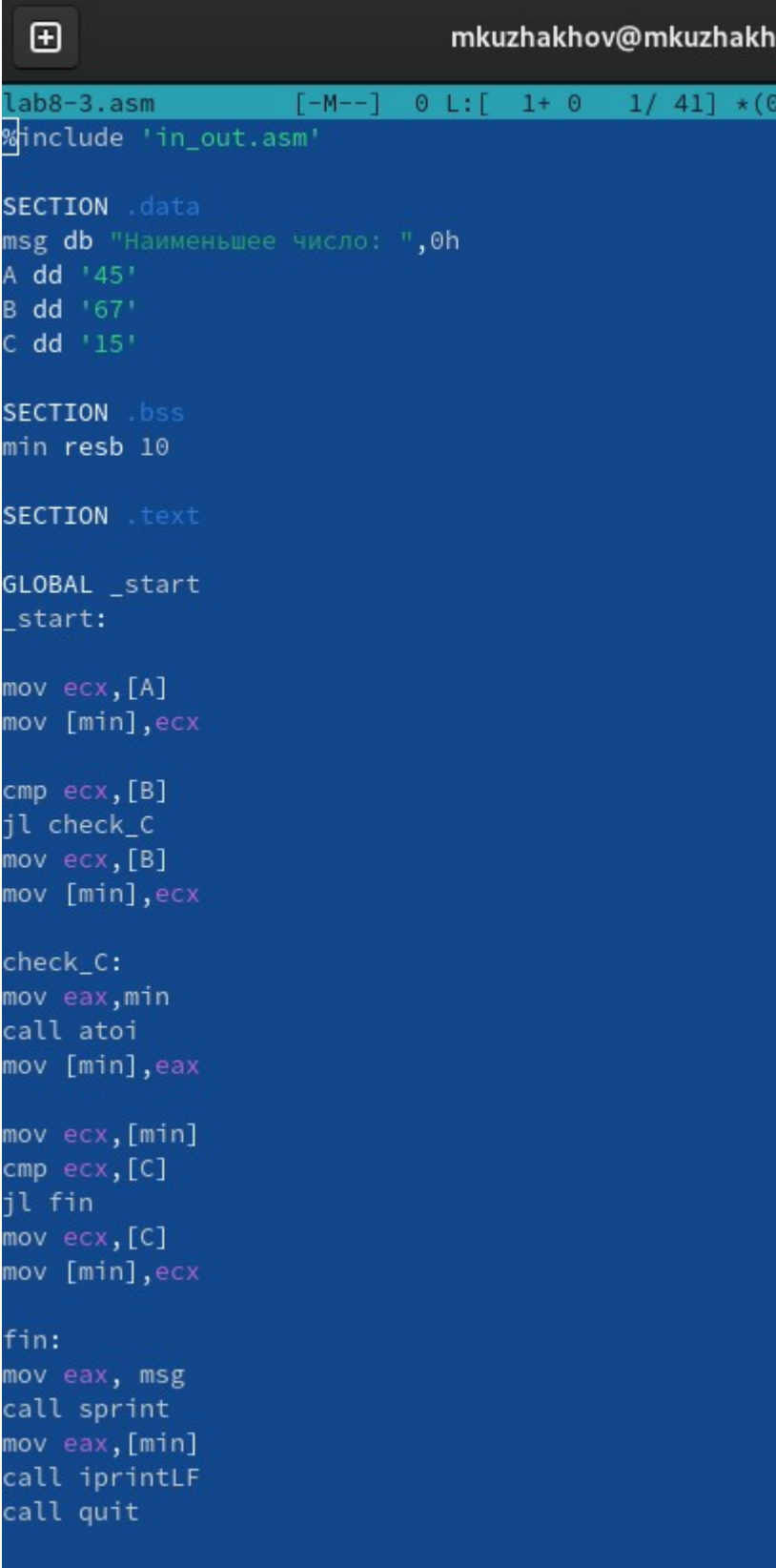
```
[mkuzhakhov@mkuzhakhov lab08]$ nasm -f elf lab8-2.asm -l lab8-2.lst
lab8-2.asm:18: error: invalid combination of opcode and operands
[mkuzhakhov@mkuzhakhov lab08]$
```

рис. 15. Листинг программы с удаленным операндом

В листинге отображается, что указана неверная комбинация операндов как раз в той строке, в которой мы убрали один операнд.

Порядок выполнения самостоятельной работы:

Напишем программу (lab8-3) нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения для моего варианта (7 вариант) будут следующими: a = 45, b = 67, c = 15. Создадим исполняемый файл и проверим его работу.



```
lab8-3.asm [-M--] 0 L: [ 1+ 0 1/ 41] *(C
%include 'in_out.asm'

SECTION .data
msg db "Наименьшее число: ",0h
A dd '45'
B dd '67'
C dd '15'

SECTION .bss
min resb 10

SECTION .text

GLOBAL _start
_start:

mov ecx,[A]
mov [min],ecx

cmp ecx,[B]
jl check_C
mov ecx,[B]
mov [min],ecx

check_C:
mov eax,min
call atoi
mov [min],eax

mov ecx,[min]
cmp ecx,[C]
jl fin
mov ecx,[C]
mov [min],ecx

fin:
mov eax,msg
call sprint
mov eax,[min]
call iprintLF
call quit
```

рис. 16. Текст программы lab8-3

```

[mkuzhakhov@mkuzhakhov lab08]$ nasm -f elf lab8-3.asm
[mkuzhakhov@mkuzhakhov lab08]$ ld -m elf_i386 lab8-3.o -o lab8-3
[mkuzhakhov@mkuzhakhov lab08]$ ./lab8-3
Наименьшее число: 45
[mkuzhakhov@mkuzhakhov lab08]$

```

рис. 17. Результат работы программы

Напишем программу lab8-4

```

lab8-4.asm  [----]  0 L:[ 1+ 0  1/ 67] *(0  / 68
#include 'in_out.asm'

section .data
msg_input_x db "Введите x: ",0h
msg_input_a db "Введите a: ",0h
msg_out_f db "f(x) = "

section .bss
x resb 10
a resb 10

section .text

global _start
_start:

mov eax, msg_input_x
call sprint
mov ecx, x
mov edx, 10
call sread
mov eax, x
call atoi
mov [x], eax

mov eax, msg_input_a
call sprint
mov ecx, a
mov edx, 10
call sread
mov eax, a
call atoi
mov [a], eax

```

рис. 19. Текст программы (1)

```
mov eax, [x]
mov ebx, [a]
cmp eax, ebx
jne _sum
mov eax, [a]
mov ebx, 6
mul ebx
mov edi, eax
mov eax, msg_out_f
call sprint
mov eax, edi
call iprintLF
call quit

_sum:
mov edi, [a]
add edi, [x]
mov eax, msg_out_f
call sprint
mov eax, edi
call iprintLF
call quit
```

рис. 20. Текст программы (2)

```
[mkuzhakhov@mkuzhakhov lab08]$ mcedit lab8-4.asm

[mkuzhakhov@mkuzhakhov lab08]$ nasm -f elf lab8-4.asm
[mkuzhakhov@mkuzhakhov lab08]$ ld -m elf_i386 lab8-4.o -o lab8-4
[mkuzhakhov@mkuzhakhov lab08]$ ./lab8-4
Введите x: 1
Введите a: 1
f(x) = 6
[mkuzhakhov@mkuzhakhov lab08]$ ./lab8-4
Введите x: 1
Введите a: 2
f(x) = 3
[mkuzhakhov@mkuzhakhov lab08]$ ./lab8-4
Введите x: 2
Введите a: 1
f(x) = 3
[mkuzhakhov@mkuzhakhov lab08]$
```

рис. 21. Результат работы программы

Вывод:

Во время выполнения лабораторной работы были изучены команды условного и безусловного переходов, приобретены навыки написания программ с использованием переходов, изучено назначение и структура файла листинга.