

# **Отчёт по лабораторной работе №7**

**Дисциплина: архитектура компьютеров и операционные системы**

Пестова Ева Константиновна

# 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>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>5</b>	<b>Выводы</b>	<b>19</b>

## Список иллюстраций

4.1	Создание файлов для лабораторной работы . . . . .	8
4.2	Ввод текста программы из листинга 7.1 . . . . .	9
4.3	Запуск программного кода . . . . .	9
4.4	Изменение текста программы . . . . .	10
4.5	Изменение текста программы . . . . .	11
4.6	Вывод программы . . . . .	11
4.7	Создание файла . . . . .	11
4.8	Ввод текста программы из листинга 7.3 . . . . .	12
4.9	Проверка работы файла . . . . .	13
4.10	Создание файла листинга . . . . .	13
4.11	Изучение файла листинга . . . . .	14
4.12	Выбранные строки файла . . . . .	14
4.13	Удаление выделенного операнда из кода . . . . .	15
4.14	Получение файла листинга . . . . .	15
4.15	Написание программы . . . . .	16
4.16	Запуск файла и проверка его работы . . . . .	16
4.17	Написание программы . . . . .	17
4.18	Запуск файла и проверка его работы . . . . .	17
4.19	Запуск файла и проверка его работы . . . . .	18

## Список таблиц

# 1 Цель работы

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

## 2 Задание

1. Реализация переходов в NASM.
2. Изучение структуры файлы листинга.
3. Задания для самостоятельной работы.

### 3 Теоретическое введение

Для реализации ветвлений в ассемблере используются так называемые команды передачи управления или команды перехода. Можно выделить 2 типа переходов:

- условный переход – выполнение или не выполнение перехода в определенную точку программы в зависимости от проверки условия.
- безусловный переход – выполнение передачи управления в определенную точку программы без каких-либо условий.

Безусловный переход выполняется инструкцией `jmp`. Инструкция `cmp` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `cmp` является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания.

Листинг (в рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Он имеет текстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную информацию.

## 4 Выполнение лабораторной работы

### 1. Реализация переходов в NASM

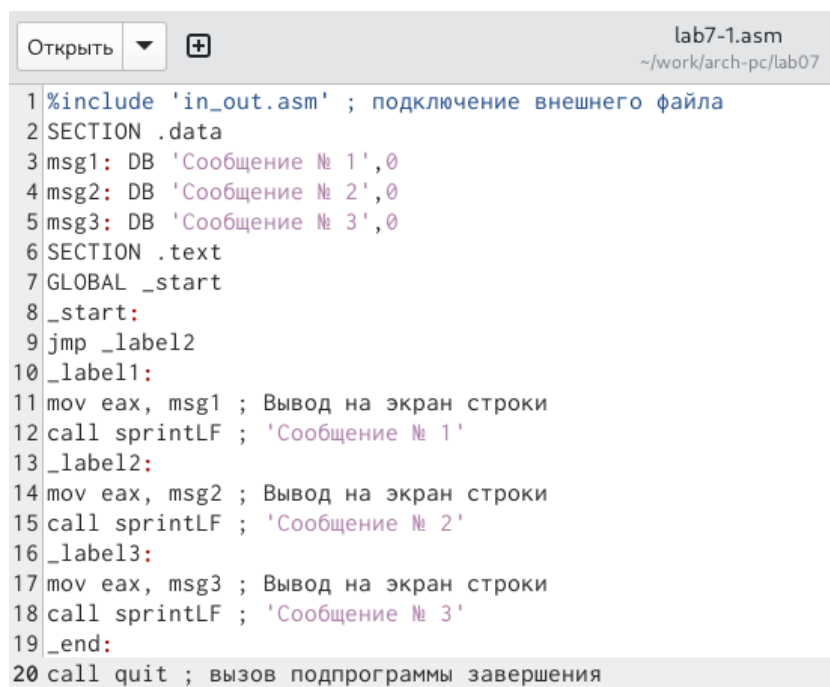
Создаю каталог для программ лабораторной работы № 7, перехожу в него и создаю файл lab7-1.asm (рис. [4.1]).

```
ekpestova@dk3n56 ~ $ mkdir ~/work/study/2023-2024/Архитектура\ компьютеров/arch-pc/lab07
ekpestova@dk3n56 ~ $ mkdir ~/work/arch-pc/lab07
ekpestova@dk3n56 ~ $ cd ~/work/arch-pc/lab07
ekpestova@dk3n56 ~/work/arch-pc/lab07 $ touch lab7-1.asm
```

Рис. 4.1: Создание файлов для лабораторной работы

Ввожу в файл lab7-1.asm текст программы из листинга 7.1 (рис. [4.2]).

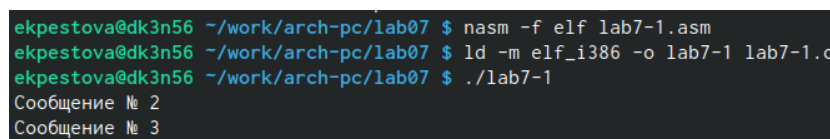




```
1 %include 'in_out.asm' ; подключение внешнего файла
2 SECTION .data
3 msg1: DB 'Сообщение № 1',0
4 msg2: DB 'Сообщение № 2',0
5 msg3: DB 'Сообщение № 3',0
6 SECTION .text
7 GLOBAL _start
8 _start:
9 jmp _label2
10 _label1:
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение № 1'
13 _label2:
14 mov eax, msg2 ; Вывод на экран строки
15 call sprintf ; 'Сообщение № 2'
16 _label3:
17 mov eax, msg3 ; Вывод на экран строки
18 call sprintf ; 'Сообщение № 3'
19 _end:
20 call quit ; вызов подпрограммы завершения
```

Рис. 4.2: Ввод текста программы из листинга 7.1

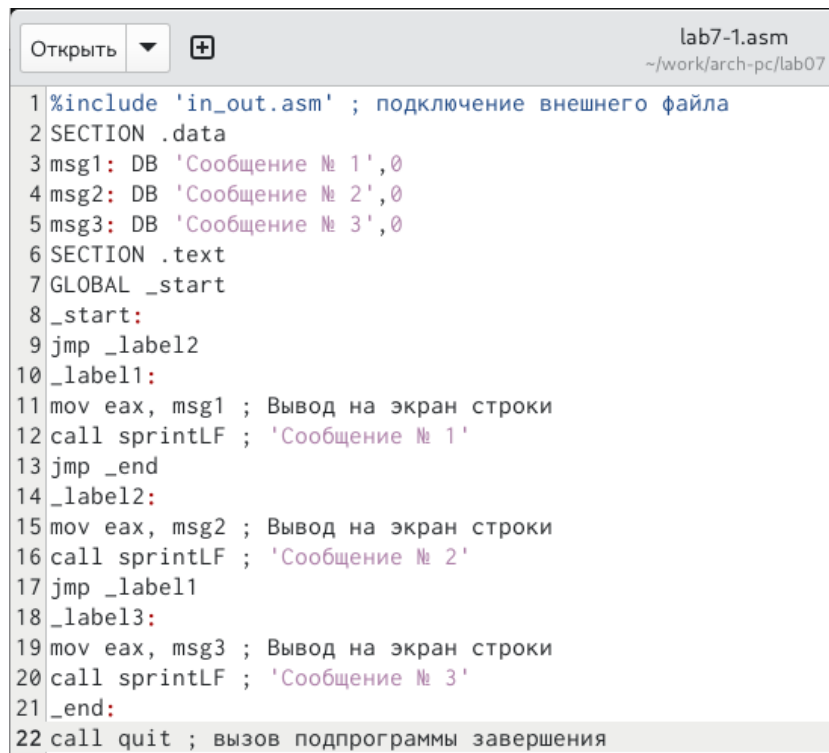
Создаю исполняемый файл и запускаю его (рис. [4.3]).



```
ekpestova@dk3n56 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
ekpestova@dk3n56 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
ekpestova@dk3n56 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 3
```

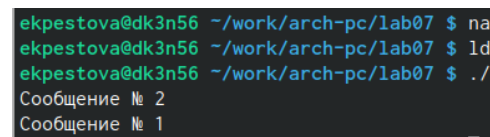
Рис. 4.3: Запуск программного кода

Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения. Изменяю программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого изменяю текст программы в соответствии с листингом 7.2 (рис. [4.4]).



```
1 %include 'in_out.asm' ; подключение внешнего файла
2 SECTION .data
3 msg1: DB 'Сообщение № 1',0
4 msg2: DB 'Сообщение № 2',0
5 msg3: DB 'Сообщение № 3',0
6 SECTION .text
7 GLOBAL _start
8 _start:
9 jmp _label2
10 _label1:
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение № 1'
13 jmp _end
14 _label2:
15 mov eax, msg2 ; Вывод на экран строки
16 call sprintf ; 'Сообщение № 2'
17 jmp _label1
18 _label3:
19 mov eax, msg3 ; Вывод на экран строки
20 call sprintf ; 'Сообщение № 3'
21 _end:
22 call quit ; вызов подпрограммы завершения
```

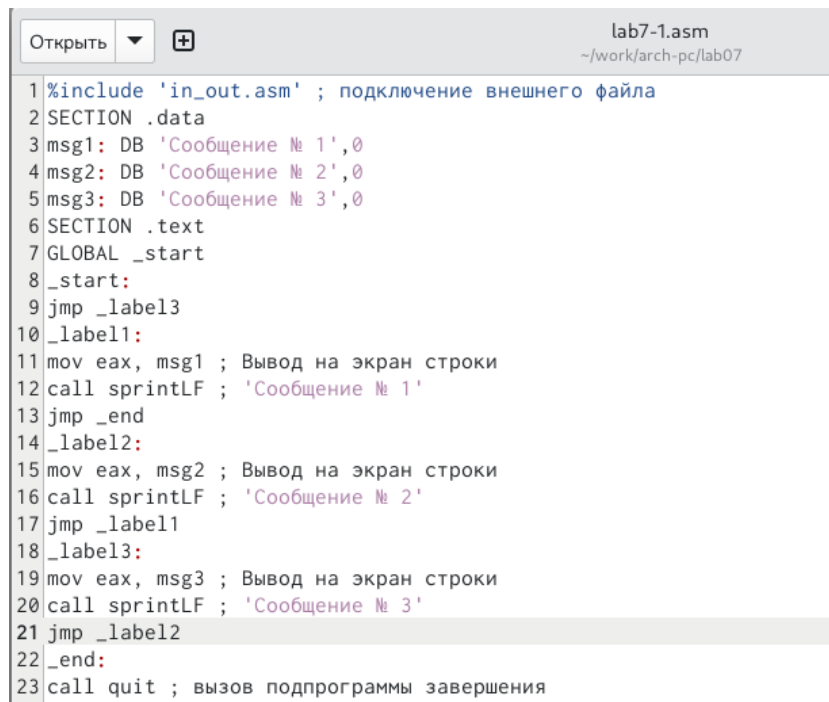
Рис. 4.4: Изменение текста программы



```
ekpestova@dk3n56 ~/work/arch-pc/lab07 $ na
ekpestova@dk3n56 ~/work/arch-pc/lab07 $ ld
ekpestova@dk3n56 ~/work/arch-pc/lab07 $ ./
Сообщение № 2
Сообщение № 1
```

Создаю исполняемый файл и проверяю его работу (рис. [??]).

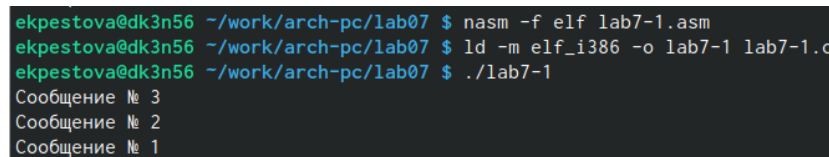
Затем изменяю текст программы, добавив в начале программы `jmp _label3`, `jmp _label2` в конце метки `jmp _label3`, `jmp _label1` добавляю в конце метки `jmp _label2`, и добавляю `jmp _end` в конце метки `jmp _label1` (рис. [4.5]).



```
1 %include 'in_out.asm' ; подключение внешнего файла
2 SECTION .data
3 msg1: DB 'Сообщение № 1',0
4 msg2: DB 'Сообщение № 2',0
5 msg3: DB 'Сообщение № 3',0
6 SECTION .text
7 GLOBAL _start
8 _start:
9 jmp _label3
10 _label1:
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение № 1'
13 jmp _end
14 _label2:
15 mov eax, msg2 ; Вывод на экран строки
16 call sprintf ; 'Сообщение № 2'
17 jmp _label1
18 _label3:
19 mov eax, msg3 ; Вывод на экран строки
20 call sprintf ; 'Сообщение № 3'
21 jmp _label2
22 _end:
23 call quit ; вызов подпрограммы завершения
```

Рис. 4.5: Изменение текста программы

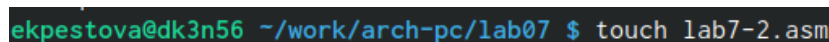
Получаю следующий вывод программы (рис. [4.6]):



```
ekpestova@dk3n56 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
ekpestova@dk3n56 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
ekpestova@dk3n56 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Рис. 4.6: Вывод программы

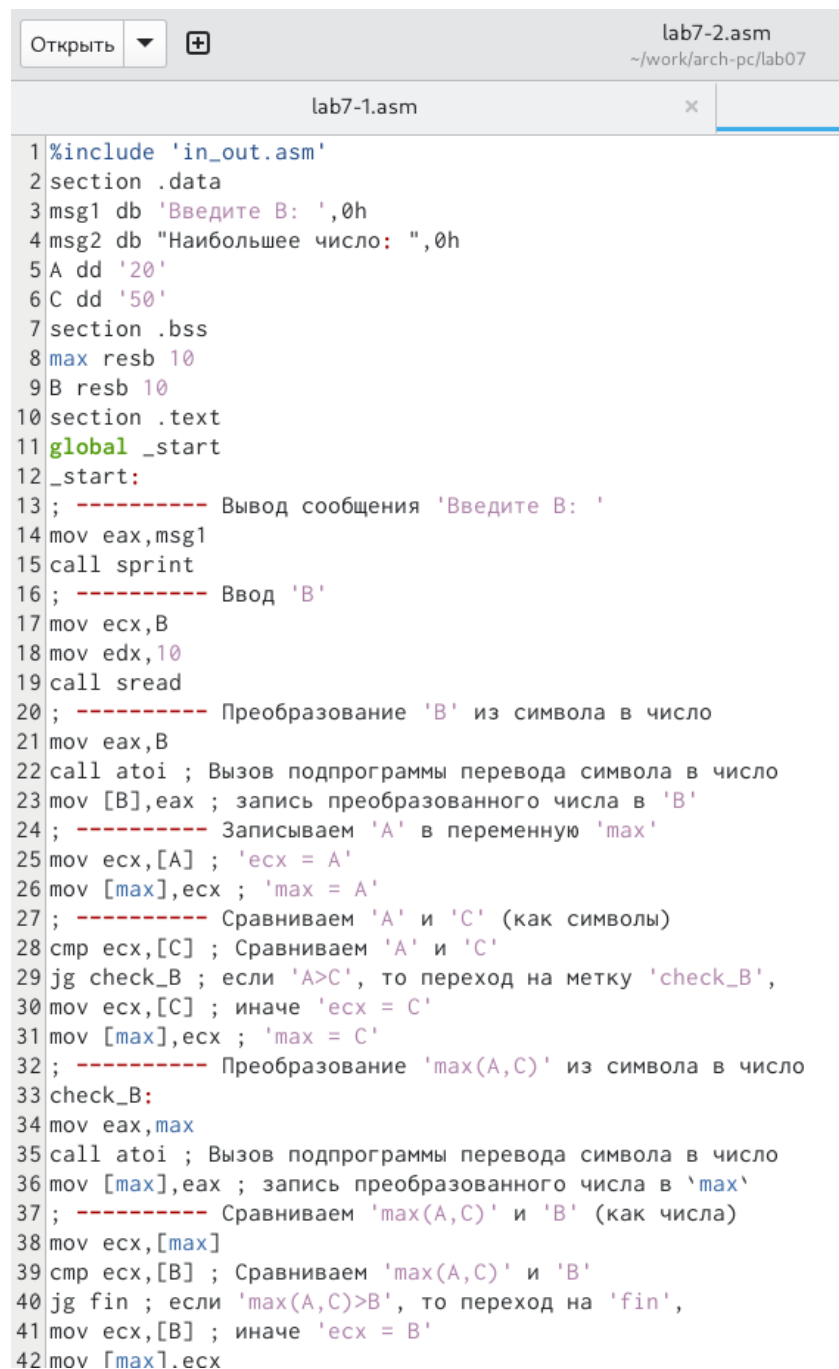
Рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А,В и С. Значения для А и С задаются в программе, значение В вводится с клавиатуры. Создаю файл lab7-2.asm в каталоге ~/work/arch-pc/lab07 (рис. [4.7]).



```
ekpestova@dk3n56 ~/work/arch-pc/lab07 $ touch lab7-2.asm
```

Рис. 4.7: Создание файла

Текст программы из листинга 7.3 ввожу в lab7-2.asm (рис. [4.8]).



```
1 %include 'in_out.asm'
2 section .data
3 msg1 db 'Введите B: ',0h
4 msg2 db "Наибольшее число: ",0h
5 A dd '20'
6 C dd '50'
7 section .bss
8 max resb 10
9 B resb 10
10 section .text
11 global _start
12 _start:
13 ; ----- Вывод сообщения 'Введите B: '
14 mov eax,msg1
15 call sprint
16 ; ----- Ввод 'B'
17 mov ecx,B
18 mov edx,10
19 call sread
20 ; ----- Преобразование 'B' из символа в число
21 mov eax,B
22 call atoi ; Вызов подпрограммы перевода символа в число
23 mov [B],eax ; запись преобразованного числа в 'B'
24 ; ----- Записываем 'A' в переменную 'max'
25 mov ecx,[A] ; 'ecx = A'
26 mov [max],ecx ; 'max = A'
27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 jg check_B ; если 'A>C', то переход на метку 'check_B',
30 mov ecx,[C] ; иначе 'ecx = C'
31 mov [max],ecx ; 'max = C'
32 ; ----- Преобразование 'max(A,C)' из символа в число
33 check_B:
34 mov eax,max
35 call atoi ; Вызов подпрограммы перевода символа в число
36 mov [max],eax ; запись преобразованного числа в 'max'
37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
38 mov ecx,[max]
39 cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
40 jg fin ; если 'max(A,C)>B', то переход на 'fin',
41 mov ecx,[B] ; иначе 'ecx = B'
42 mov [max],ecx
```

Рис. 4.8: Ввод текста программы из листинга 7.3

Создаю исполняемый файл и проверяю его работу (рис. [4.9]).

```

ekpestova@dk3n56 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
ekpestova@dk3n56 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
ekpestova@dk3n56 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 50
Наибольшее число: 50

```

Рис. 4.9: Проверка работы файла

Файл работает корректно.

## 2. Изучение структуры файлы листинга

Создаю файл листинга для программы из файла lab7-2.asm (рис. [4.10]).

```

ekpestova@dk3n56 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm

```

Рис. 4.10: Создание файла листинга

Открываю файл листинга lab7-2.lst с помощью текстового редактора и внимательно изучаю его формат и содержимое (рис. [4.11]).

lab7-1.asm	lab7-2.asm	lab7-2.lst
1	1	%include 'in_out.asm'
2	1	<1> ;----- slen -----
3	2	<1> ; Функция вычисления длины сообщения
4	3	<1> slen:
5	4 00000000 53	<1> push ebx
6	5 00000001 89C3	<1> mov ebx, eax
7	6	<1>
8	7	<1> nextchar:
9	8 00000003 803800	<1> cmp byte [eax], 0
10	9 00000006 7403	<1> jz finished
11	10 00000008 40	<1> inc eax
12	11 00000009 EBF8	<1> jmp nextchar
13	12	<1>
14	13	<1> finished:
15	14 0000000B 29D8	<1> sub eax, ebx
16	15 0000000D 5B	<1> pop ebx
17	16 0000000E C3	<1> ret
18	17	<1>
19	18	<1>
20	19	<1> ;----- sprint -----
21	20	<1> ; Функция печати сообщения
22	21	<1> ; входные данные: mov eax,<message>
23	22	<1> sprint:
24	23 0000000F 52	<1> push edx
25	24 00000010 51	<1> push ecx
26	25 00000011 53	<1> push ebx
27	26 00000012 50	<1> push eax
28	27 00000013 E8E8FFFFFF	<1> call slen
29	28	<1>
30	29 00000018 89C2	<1> mov edx, eax
31	30 0000001A 58	<1> pop eax
32	31	<1>
33	32 0000001B 89C1	<1> mov ecx, eax
34	33 0000001D BB01000000	<1> mov ebx, 1
35	34 00000022 B804000000	<1> mov eax, 4
36	35 00000027 CD80	<1> int 80h
37	36	<1>
38	37 00000029 5B	<1> pop ebx
39	38 0000002A 59	<1> pop ecx
40	39 0000002B 5A	<1> pop edx
41	40 0000002C C3	<1> ret
42	41	<1>
43	42	<1>
44	43	<1> ;----- sprintLF -----
45	44	<1> ; Функция печати сообщения с переводом строки

Рис. 4.11: Изучение файла листинга

В представленных трех строчках содержатся следующие данные (рис. [4.12]):

2	<1> ; Функция вычисления длины сообщения
3	<1> slen:
4 00000000 53	<1> push ebx

Рис. 4.12: Выбранные строки файла

“2” - номер строки кода, “; Функция вычисления длинны сообщения” - комментарий к коду, не имеет адреса и машинного кода. “3” - номер строки кода, “slen” - название функции, не имеет адреса и машинного кода. “4” - номер строки кода, “00000000” - адрес строки, “53” - машинный код, “push ebx” - исходный текст программы, инструкция “push” помещает операнд “ebx” в стек.

Открываю файл с программой lab7-2.asm и в выбранной мной инструкции с двумя операндами удаляю выделенный операнд (рис. [4.13]).

```

27 ; ----- Сравниваем 'A' и 'C' (как символ
28 cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 jg check B : если 'A>C'. то переход на метку

```

Рис. 4.13: Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга (рис. [4.14]).

```

ekpestova@dk3n56 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands

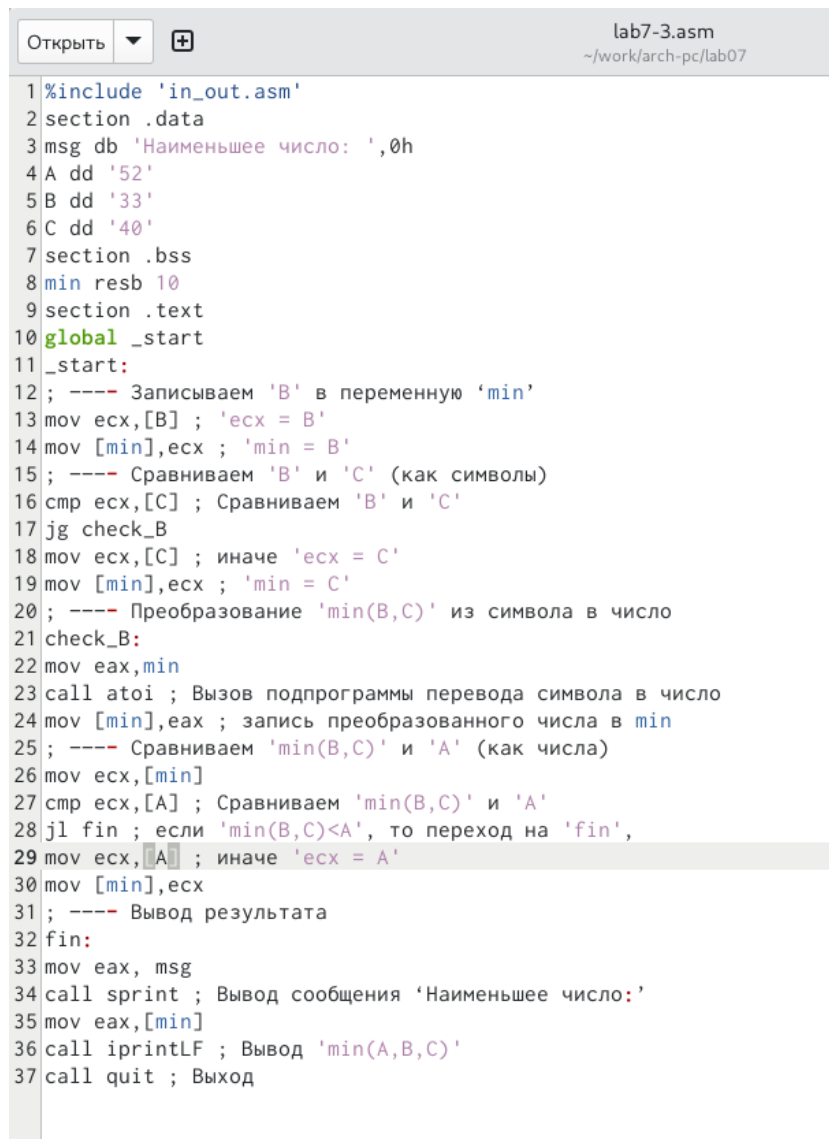
```

Рис. 4.14: Получение файла листинга

На выходе я не получаю ни одного файла из-за ошибки:инструкция mov (единственная в коде содержит два операнда) не может работать, имея только один операнд, из-за чего нарушается работа кода.

### 3. Задания для самостоятельной работы

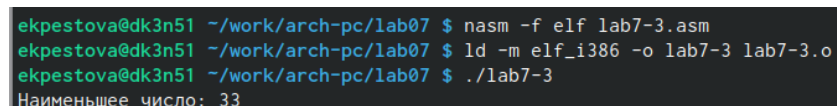
Пишу программу нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения переменных выбираю из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Мой вариант под номером 8, поэтому мои значения - 52, 33 и 40 (рис. [4.15]).



```
1 %include 'in_out.asm'
2 section .data
3 msg db 'Наименьшее число: ',0h
4 A dd '52'
5 B dd '33'
6 C dd '40'
7 section .bss
8 min resb 10
9 section .text
10 global _start
11 _start:
12 ; ---- Записываем 'B' в переменную 'min'
13 mov ecx,[B] ; 'ecx = B'
14 mov [min],ecx ; 'min = B'
15 ; ---- Сравниваем 'B' и 'C' (как символы)
16 cmp ecx,[C] ; Сравниваем 'B' и 'C'
17 jg check_B
18 mov ecx,[C] ; иначе 'ecx = C'
19 mov [min],ecx ; 'min = C'
20 ; ---- Преобразование 'min(B,C)' из символа в число
21 check_B:
22 mov eax,min
23 call atoi ; Вызов подпрограммы перевода символа в число
24 mov [min],eax ; запись преобразованного числа в min
25 ; ---- Сравниваем 'min(B,C)' и 'A' (как числа)
26 mov ecx,[min]
27 cmp ecx,[A] ; Сравниваем 'min(B,C)' и 'A'
28 jl fin ; если 'min(B,C)<A', то переход на 'fin',
29 mov ecx,[A] ; иначе 'ecx = A'
30 mov [min],ecx
31 ; ---- Вывод результата
32 fin:
33 mov eax, msg
34 call sprint ; Вывод сообщения 'Наименьшее число:'
35 mov eax,[min]
36 call iprintLF ; Вывод 'min(A,B,C)'
37 call quit ; Выход
```

Рис. 4.15: Написание программы

Создаю исполняемый файл и проверяю его работу, подставляя необходимые значение (рис. [4.16]).



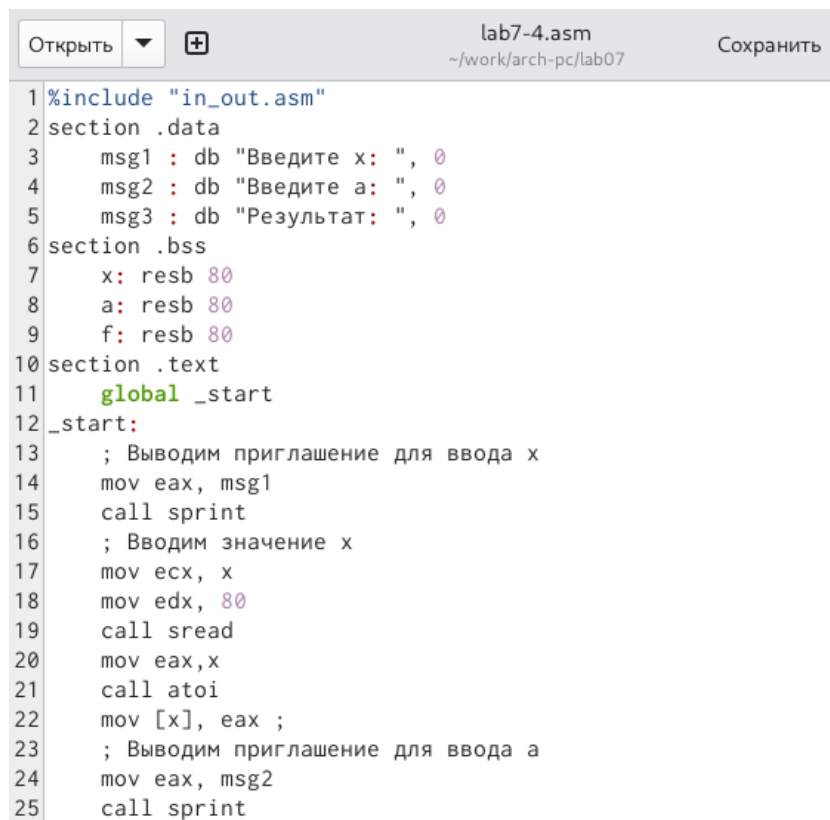
```
ekpestova@dk3n51 ~/work/arch-pc/lab07 $ nasm -f elf lab7-3.asm
ekpestova@dk3n51 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-3 lab7-3.o
ekpestova@dk3n51 ~/work/arch-pc/lab07 $ ./lab7-3
Наименьшее число: 33
```

Рис. 4.16: Запуск файла и проверка его работы



Программа работает корректно.

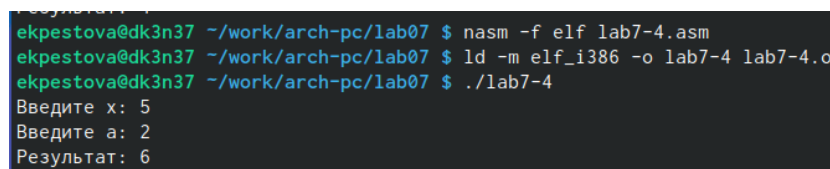
- Пишу программу, которая для введенных с клавиатуры значений  $x$  и  $a$  вычисляет значение и выводит результат вычислений заданной для моего варианта функции  $f(x)$ :  $3 * a$ , если  $a < 3x + 1$ , если  $a \geq 3$  (рис. [4.17]).



```
1 %include "in_out.asm"
2 section .data
3     msg1 : db "Введите x: ", 0
4     msg2 : db "Введите a: ", 0
5     msg3 : db "Результат: ", 0
6 section .bss
7     x: resb 80
8     a: resb 80
9     f: resb 80
10 section .text
11     global _start
12 _start:
13     ; Выводим приглашение для ввода x
14     mov eax, msg1
15     call sprint
16     ; Вводим значение x
17     mov ecx, x
18     mov edx, 80
19     call sread
20     mov eax, x
21     call atoi
22     mov [x], eax ;
23     ; Выводим приглашение для ввода a
24     mov eax, msg2
25     call sprint
```

Рис. 4.17: Написание программы

Создаю исполняемый файл и проверяю его работу для значений  $x$  и  $a$  соответственно (рис. [4.18]), (рис. [4.19]).



```
ekpestova@dk3n37 ~/work/arch-pc/lab07 $ nasm -f elf lab7-4.asm
ekpestova@dk3n37 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-4 lab7-4.o
ekpestova@dk3n37 ~/work/arch-pc/lab07 $ ./lab7-4
Введите x: 5
Введите a: 2
Результат: 6
```

Рис. 4.18: Запуск файла и проверка его работы

```
ekpestova@dk3n37 ~/work/arch-pc/lab07 $ ./lab7-4  
Введите x: 5  
Введите a: 10  
Результат: 6
```

Рис. 4.19: Запуск файла и проверка его работы

Программа работает корректно.

## 5 Выводы

По итогам данной лабораторной работы я изучила команды условного и безусловного переходов, приобрела навыки написания программ с использованием переходов и ознакомилась с назначением и структурой файла листинга, что поможет мне при выполнении последующих лабораторных работ.