

# **ОТЧЕТ по лабораторной работе №7**

**дисциплина: Архитектура компьютера**

**Студент: Идрисов Д.А.**

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
1.1	Задание . . . . .	5
1.2	Теоретическое введение . . . . .	5
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
2.1	Реализация переходов в NASM . . . . .	7
2.2	Изучение структуры файлы листинга . . . . .	15
2.3	Задание для самостоятельной работы . . . . .	18
<b>3</b>	<b>Выводы</b>	<b>23</b>

## Список иллюстраций

2.1	Изменение кода . . . . .	8
2.2	Запуск программы . . . . .	9
2.3	Изменение кода . . . . .	10
2.4	Запуск программы . . . . .	11
2.5	Изменение кода . . . . .	12
2.6	Запуск программы . . . . .	13
2.7	Изменение кода . . . . .	14
2.8	Запуск программы . . . . .	15
2.9	Файл листинга . . . . .	16
2.10	Ошибка трансляции . . . . .	17
2.11	Файл листинга . . . . .	18
2.12	Изменение кода . . . . .	19
2.13	Запуск программы . . . . .	20
2.14	Изменение кода . . . . .	21
2.15	Запуск программы . . . . .	22

## **Список таблиц**

# 1 Цель работы

Целью работы является изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

## 1.1 Задание

1. Изучение механизма переходов в авссемблере
2. Изучение листинга программы
3. Выполнение заданий для самостоятельной работы

## 1.2 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` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление:

```
jmp <адрес_перехода>
```

Команда условного перехода имеет вид

```
j<мнемоника_перехода> labe
```

Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов.

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:

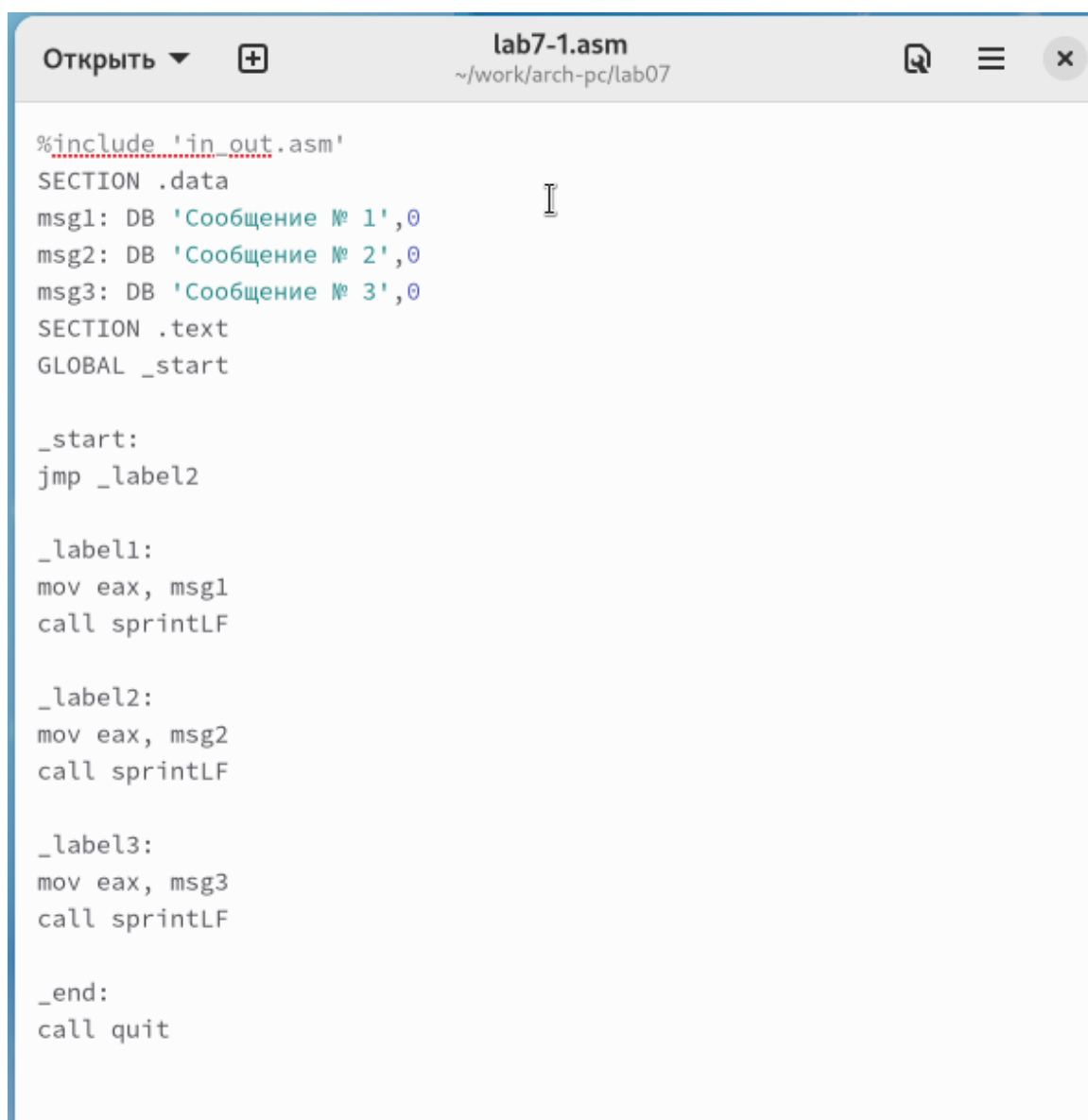
```
cmp <операнд_1>, <операнд_2>
```

## 2 Выполнение лабораторной работы

### 2.1 Реализация переходов в NASM

Я создал каталог для программ лабораторной работы № 7 и файл lab7-1.asm.

Инструкция `jmp` в NASM используется для выполнения безусловных переходов. Рассмотрим пример программы, в которой используется инструкция `jmp`. Написал текст программы из листинга 7.1 в файле lab7-1.asm. (рис. [2.1])



```
Открыть ▾ + lab7-1.asm ~/work/arch-pc/lab07
%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 sprintLF

_label2:
mov eax, msg2
call sprintLF

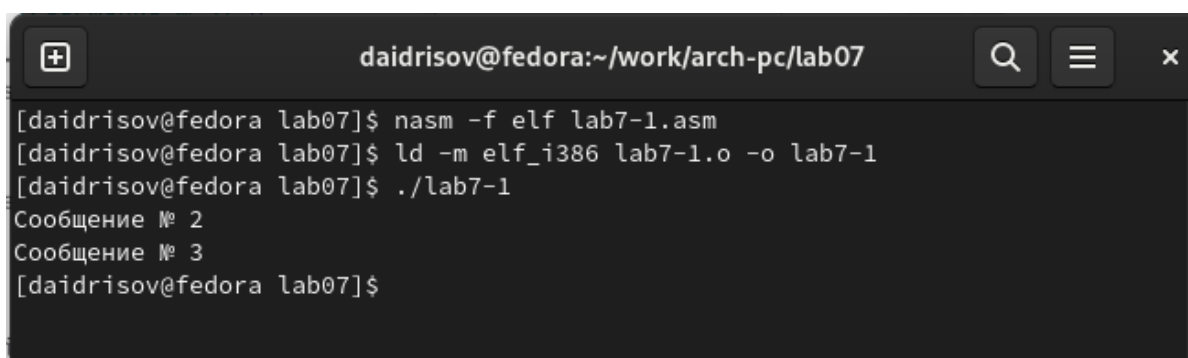
_label3:
mov eax, msg3
call sprintLF

_end:
call quit
```

Рис. 2.1: Изменение кода

Создал исполняемый файл и запустил его. (рис. [2.2])



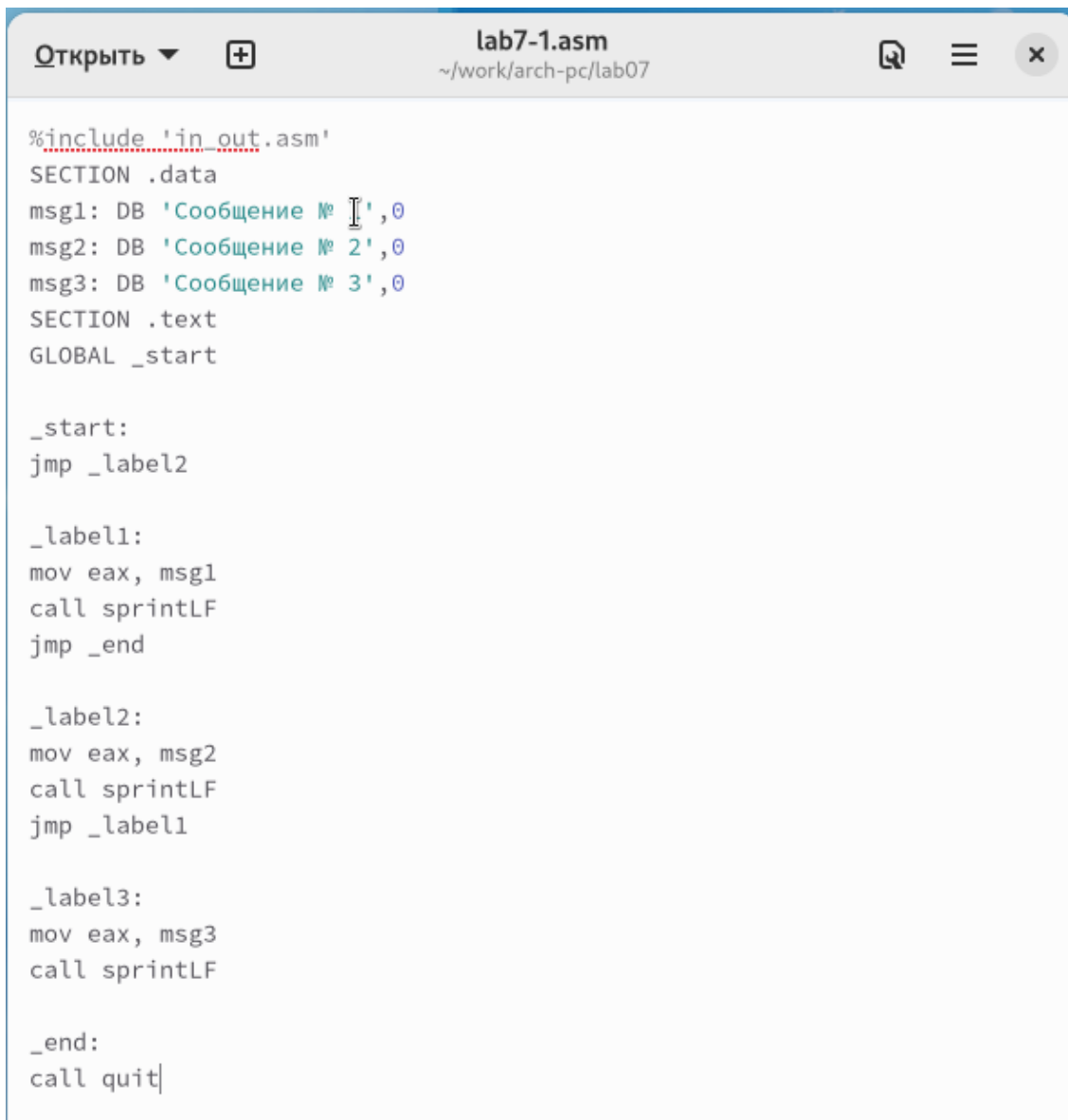


```
daidrisov@fedora:~/work/arch-pc/lab07
[daidrisov@fedora lab07]$ nasm -f elf lab7-1.asm
[daidrisov@fedora lab07]$ ld -m elf_i386 lab7-1.o -o lab7-1
[daidrisov@fedora lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 3
[daidrisov@fedora lab07]$
```

Рис. 2.2: Запуск программы

Инструкция `jmp` позволяет осуществлять переходы не только вперед, но и назад. Мы изменим программу так, чтобы она сначала выводила “Сообщение № 2”, затем “Сообщение № 1” и завершала работу. Для этого мы добавим в текст программы после вывода “Сообщения № 2” инструкцию `jmp` с меткой `_label1` (чтобы перейти к инструкциям вывода “Сообщения № 1”) и после вывода “Сообщения № 1” добавим инструкцию `jmp` с меткой `_end` (чтобы перейти к инструкции `call quit`).

Изменил текст программы в соответствии с листингом 7.2. (рис. [2.3] [2.4])



The screenshot shows a code editor window with the title bar 'lab7-1.asm' and a path '~/.work/arch-pc/lab07'. The code is written in assembly language and includes comments in Russian. It defines three data messages and three labels to print them sequentially.

```
%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 sprintLF
jmp _end

_label2:
mov eax, msg2
call sprintLF
jmp _label1

_label3:
mov eax, msg3
call sprintLF

_end:
call quit
```

Рис. 2.3: Изменение кода

```
[daidrisov@fedora lab07]$  
[daidrisov@fedora lab07]$ nasm -f elf lab7-1.asm  
[daidrisov@fedora lab07]$ ld -m elf_i386 lab7-1.o -o lab7-1  
[daidrisov@fedora lab07]$ ./lab7-1  
Сообщение № 2  
Сообщение № 1  
[daidrisov@fedora lab07]$
```

I

Рис. 2.4: Запуск программы

Изменил текст программы (рис. [2.5] [2.6]), изменив инструкции `jmp`, чтобы вывод программы был следующим:

Сообщение № 3

Сообщение № 2

Сообщение № 1



```
Открыть ▾  lab7-1.asm     
~/work/arch-pc/lab07  
  
%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. 2.5: Изменение кода

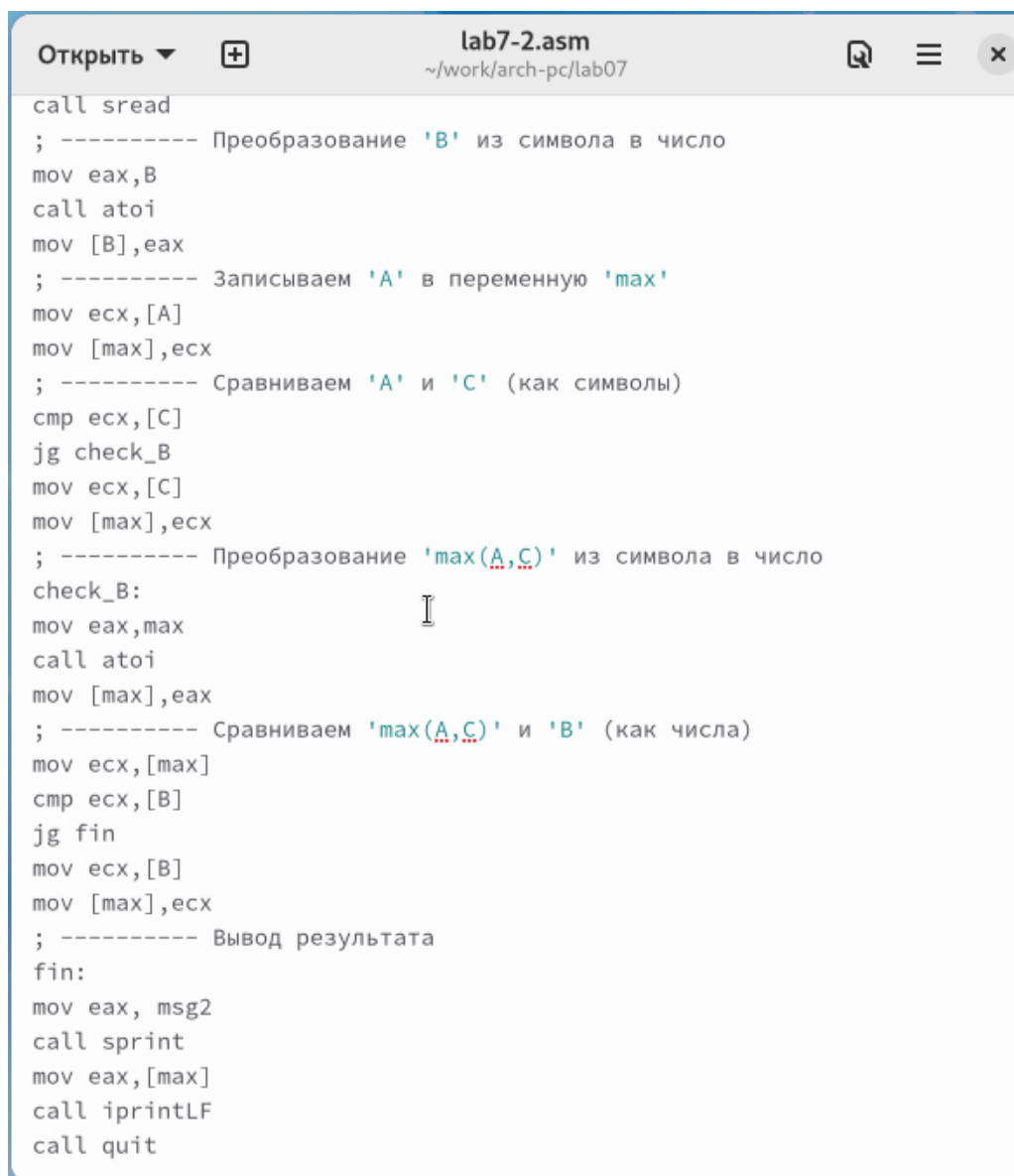
```
[daidrisov@fedora lab07]$  
[daidrisov@fedora lab07]$ nasm -f elf lab7-1.asm  
[daidrisov@fedora lab07]$ ld -m elf_i386 lab7-1.o -o lab7-1  
[daidrisov@fedora lab07]$ ./lab7-1  
Сообщение № 3  
Сообщение № 2  
Сообщение № 1  
[daidrisov@fedora lab07]$
```

Рис. 2.6: Запуск программы

Использование инструкции `jmp` приводит к переходу в любом случае. Однако, часто при написании программ необходимо использовать условные переходы, то есть переход должен происходить, если выполнено какое-либо условие.

Давайте рассмотрим программу, которая определяет и выводит на экран наибольшую из трех целочисленных переменных: А, В и С. Значения для А и С задаются в программе, а значение В вводится с клавиатуры.

Создал исполняемый файл и проверил его работу для разных значений В. (рис. [2.7] [2.8])



```
Открыть ▾ [иконка] lab7-2.asm ~/work/arch-pc/lab07 [иконка] [меню] [крестик]

call sread
; ----- Преобразование 'B' из символа в число
mov eax,B
call atoi
mov [B],eax
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A]
mov [max],ecx
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C]
jg check_B
mov ecx,[C]
mov [max],ecx
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,max
call atoi
mov [max],eax
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
cmp ecx,[B]
jg fin
mov ecx,[B]
mov [max],ecx
; ----- Вывод результата
fin:
mov eax,msg2
call sprint
mov eax,[max]
call iprintLF
call quit
```

Рис. 2.7: Изменение кода

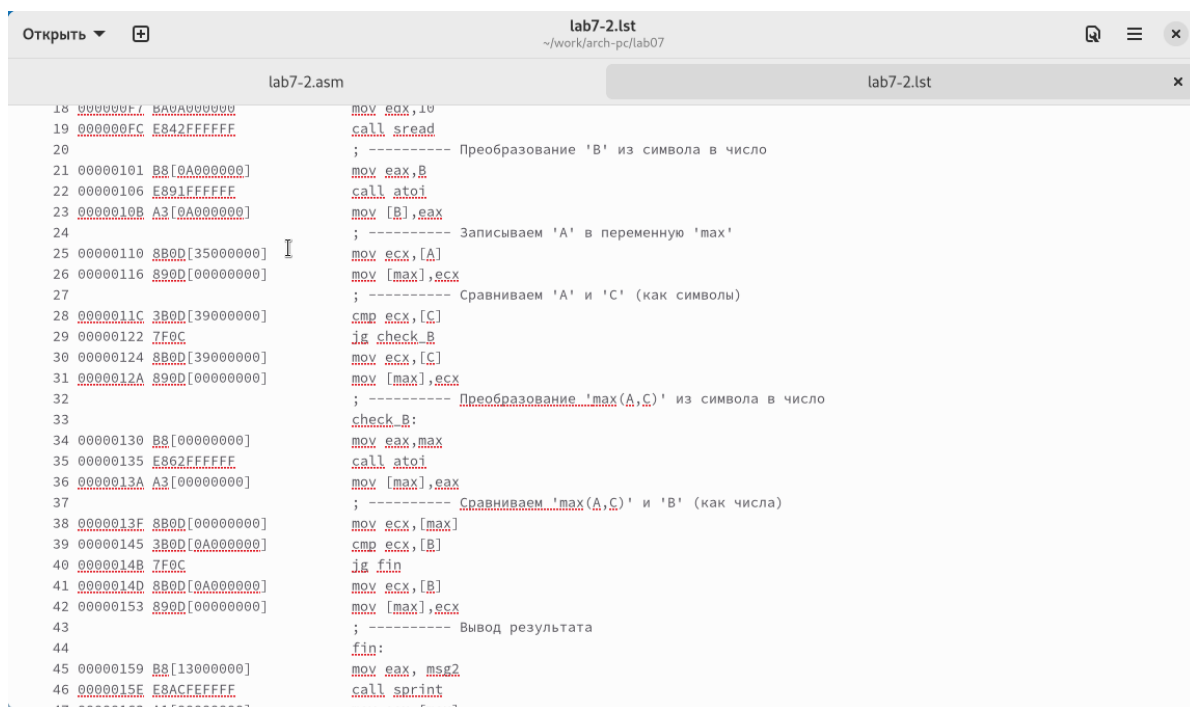
```
[daidrisov@fedora lab07]$ nasm -f elf lab7-2.asm
[daidrisov@fedora lab07]$ ld -m elf_i386 lab7-2.o -o lab7-2
[daidrisov@fedora lab07]$ ./lab7-2
Введите B: 10
Наибольшее число: 50
[daidrisov@fedora lab07]$ ./lab7-2
Введите B: 40
Наибольшее число: 50
[daidrisov@fedora lab07]$ ./lab7-2
Введите B: 60
Наибольшее число: 60
[daidrisov@fedora lab07]$
```

Рис. 2.8: Запуск программы

## 2.2 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.

Создал файл листинга для программы из файла `lab7-2.asm` (рис. [2.9])



```
18 00000000 8A0A00000000 mov ecx,10
19 000000FC E842FFFFFF call sread
20 ; ----- Преобразование 'B' из символа в число
21 00000101 B8[0A000000] mov eax,B
22 00000106 E891FFFFFF call atoi
23 0000010B A3[0A000000] mov [B],eax
24 ; ----- Записываем 'A' в переменную 'max'
25 00000110 8B0D[35000000] mov ecx,[A]
26 00000116 890D[00000000] mov [max],ecx
27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 0000011C 3B0D[39000000] cmp ecx,[C]
29 00000122 7F0C jg check_B
30 00000124 8B0D[39000000] mov ecx,[C]
31 0000012A 890D[00000000] mov [max],ecx
32 ; ----- Преобразование 'max(A,C)' из символа в число
33 check_B:
34 00000130 B8[00000000] mov eax,max
35 00000135 E862FFFFFF call atoi
36 0000013A A3[00000000] mov [max],eax
37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
38 0000013F 8B0D[00000000] mov ecx,[max]
39 00000145 3B0D[0A000000] cmp ecx,[B]
40 0000014B 7F0C jg fin
41 0000014D 8B0D[0A000000] mov ecx,[B]
42 00000153 890D[00000000] mov [max],ecx
43 ; ----- Вывод результата
44 fin:
45 00000159 B8[13000000] mov eax,msg2
46 0000015F E8ACFFFFFF call sprint
47 00000163 A100000000 mov ecx,[eax]
```

Рис. 2.9: Файл листинга

Внимательно ознакомился с его форматом и содержимым. Подробно объясню содержимое трёх строк файла листинга.

строка 203

- 28 - номер строки в подпрограмме \*
- 0000011C - адрес \*
- 3B0D[39000000] - машинный код \*
- str ecx,[C] - код программы - сравнивает ecx и переменную \*

строка 204

- 29 - номер строки в подпрограмме \*
- 00000122 - адрес \*
- 7F0C - машинный код \*



- `je check_B` - код программы - если сравнение покажет что одно число больше то переход к метке `check_B*`

строка 205

- 30 - номер строки в подпрограмме \*
- 00000124 - адрес \*
- `8B0D[39000000]` - машинный код \*
- `mov ecx,[C]` - код программы - копирует переменную C в `ecx` \*

Открыл файл с программой `lab7-2.asm` и в инструкции с двумя операндами удалил один операнд. Выполнил трансляцию с получением файла листинга. (рис. [2.10]) (рис. [2.11])

```
[daidrisov@fedora lab07]$
[daidrisov@fedora lab07]$ nasm -f elf lab7-2.asm -l lab7-2.lst
[daidrisov@fedora lab07]$
[daidrisov@fedora lab07]$ nasm -f elf lab7-2.asm -l lab7-2.lst
lab7-2.asm:34: error: invalid combination of opcode and operands
[daidrisov@fedora lab07]$
[daidrisov@fedora lab07]$
[daidrisov@fedora lab07]$
```

Рис. 2.10: Ошибка трансляции

```

lab7-2.asm                                lab7-2.lst
200      25 00000110 8B0D[35000000]          mov ecx,[A]
201      26 00000116 890D[00000000]          mov [max],ecx
202      27                                     ; ----- Сравниваем 'A' и 'C' (как символы)
203      28 0000011C 3B0D[39000000]          cmp ecx,[C]
204      29 00000122 7F0C                      jg check_B
205      30 00000124 8B0D[39000000]          mov ecx,[C]
206      31 0000012A 890D[00000000]          mov [max],ecx
207      32                                     ; ----- Преобразование 'max(A,C)' из
        СИМВОЛА В ЧИСЛО
208      33                                     check_B:
209      34                                     mov eax,
210      34 *****
        operands
211      35 00000130 E867FFFFFF          call atoi
212      36 00000135 A3[00000000]          mov [max],eax
213      37                                     ; ----- Сравниваем 'max(A,C)' и 'B' (как
        числа)
214      38 0000013A 8B0D[00000000]          mov ecx,[max]
215      39 00000140 3B0D[0A000000]          cmp ecx,[B]
216      40 00000146 7F0C                      jg fin
217      41 00000148 8B0D[0A000000]          mov ecx,[B]
218      42 0000014E 890D[00000000]          mov [max],ecx
219      43                                     ; ----- Вывод результата
220      44                                     fin:
221      45 00000154 B8[13000000]          mov eax, msg2
222      46 00000159 E8B1FFFFFF          call sprint
223      47 0000015E A1[00000000]          mov eax,[max]
224      48 00000163 E81EFFFFFF          call iprintLF
225      49 00000168 E86EFFFFFF          call quit

```

Рис. 2.11: Файл листинга

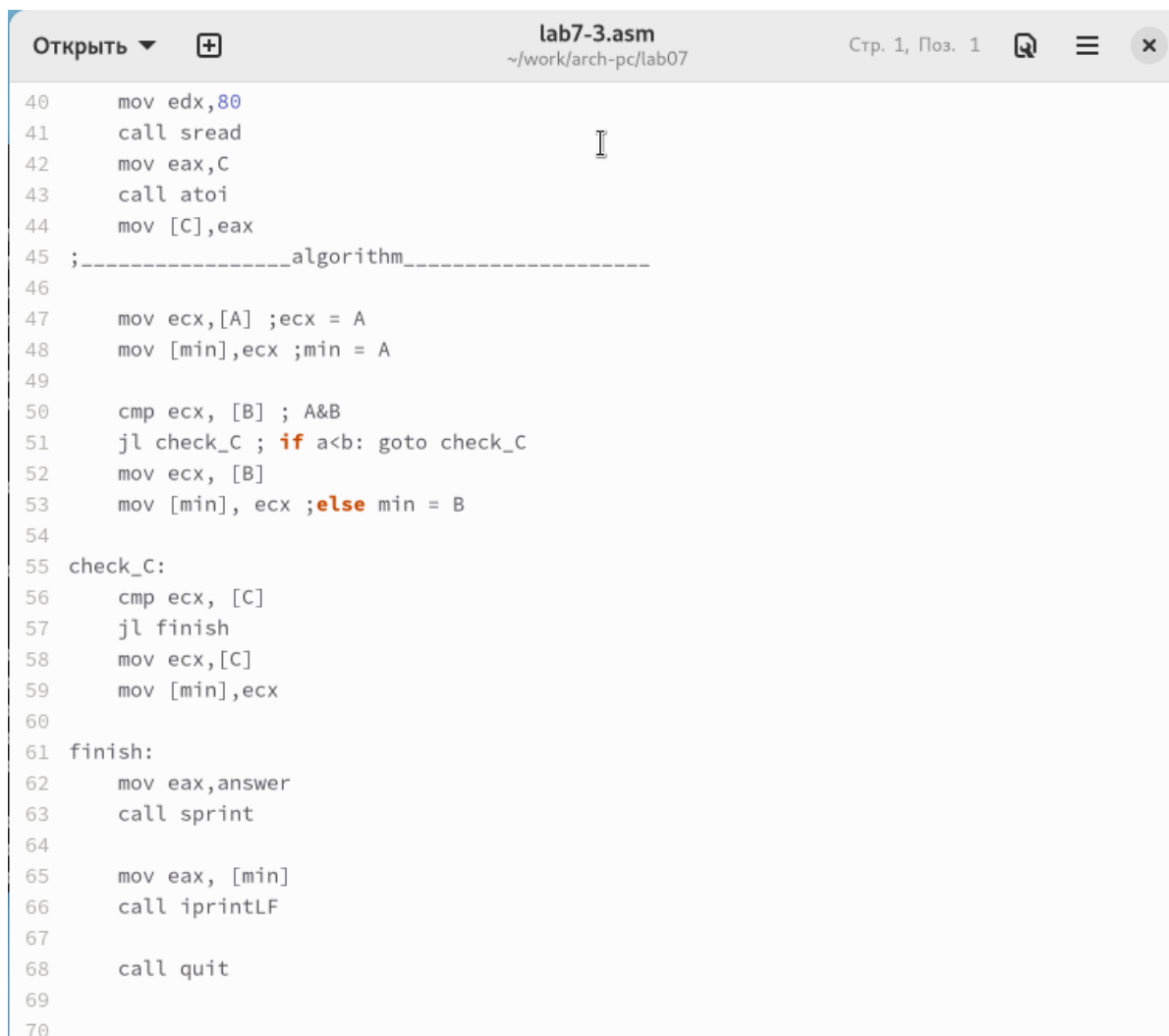
Объектный файл не смог создаться из-за ошибки. Но получился листинг, где выделено место ошибки.

## 2.3 Задание для самостоятельной работы

Напишите программу нахождения наименьшей из 3 целочисленных переменных a,b и c. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Создайте

исполняемый файл и проверьте его работу (рис. [2.12]) (рис. [2.13])

Для варианта 17 - числа: 26,12,68



The screenshot shows a text editor window titled "lab7-3.asm" with the path "~/work/arch-pc/lab07". The editor contains assembly code with line numbers 40 through 70. The code includes instructions for reading input, comparing values, and printing the result. A comment on line 45 indicates a change to the algorithm. The code is as follows:

```
40     mov edx,80
41     call sread
42     mov eax,C
43     call atoi
44     mov [C],eax
45 ;-----algorithm-----
46
47     mov ecx,[A] ;ecx = A
48     mov [min],ecx ;min = A
49
50     cmp ecx, [B] ; A&B
51     j<= check_C ; if a<b: goto check_C
52     mov ecx, [B]
53     mov [min], ecx ;else min = B
54
55 check_C:
56     cmp ecx, [C]
57     j<= finish
58     mov ecx,[C]
59     mov [min],ecx
60
61 finish:
62     mov eax,answer
63     call sprint
64
65     mov eax, [min]
66     call iprintLF
67
68     call quit
69
70
```

Рис. 2.12: Изменение кода

```
[daidrisov@fedora lab07]$ nasm -f elf lab7-3.asm
[daidrisov@fedora lab07]$ ld -m elf_i386 lab7-3.o -o lab7-3
[daidrisov@fedora lab07]$ ./lab7-3
Input A: 26
Input B: 12
Input C: 68
Smallest: 12
[daidrisov@fedora lab07]$
```

Рис. 2.13: Запуск программы

Напишите программу, которая для введенных с клавиатуры значений  $x$  и  $a$  вычисляет значение заданной функции  $f(x)$  и выводит результат вычислений. Вид функции  $f(x)$  выбрать из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу для значений  $X$  и  $a$  из 7.6. (рис. [2.14]) (рис. [2.15])

Мой вариант 17

$$\begin{cases} a + 8, a < 8 \\ ax, a \geq 8 \end{cases}$$

Если подставить  $x = 3, a = 4$ , тогда  $f(x) = 12$

Если подставить  $x = 2, a = 9$ , тогда  $f(x) = 18$

```
lab7-4.asm
~/work/arch-pc/lab07

Открыть ▾ ⊕

22     mov [A],eax
23
24     mov eax,msgX
25     call sprint
26     mov ecx,X
27     mov edx,80
28     call sread
29     mov eax,X
30     call atoi
31     mov [X],eax
32 ;-----algorithm-----
33
34     mov ebx, [A]
35     mov edx, 8
36     cmp ebx, edx
37     jb first
38     jmp second
39
40 first:
41     mov eax,[A]
42     add eax,8
43     call iprintLF
44     call quit
45 second:
46     mov eax,[X]
47     mov ebx,[A]
48     mul ebx
49     call iprintLF
50     call quit
51
52
```

Рис. 2.14: Изменение кода

```
[daidrisov@fedora lab07]$ nasm -f elf lab7-4.asm
[daidrisov@fedora lab07]$ ld -m elf_i386 lab7-4.o -o lab7-4
[daidrisov@fedora lab07]$ ./lab7-4
Input A: 4
Input X: 6
12
[daidrisov@fedora lab07]$ ./lab7-4
Input A: 9
Input X: 2
18
[daidrisov@fedora lab07]$
```

Рис. 2.15: Запуск программы

## 3 Выводы

Изучили команды условного и безусловного переходов, познакомились с фалом листинга.