

# **Отчёт по лабораторной работе №7**

**Дисциплина: Архитектура компьютера**

Ларина Наталья Денисовна

# 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>
4.1	Реализация переходов в NASM . . . . .	8
4.2	Изучение структуры файлы листинга . . . . .	11
4.3	Задания для самостоятельной работы . . . . .	13
<b>5</b>	<b>Выводы</b>	<b>16</b>
<b>6</b>	<b>Список литературы</b>	<b>17</b>

## Список таблиц

## Список иллюстраций

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

# 1 Цель работы

Целью данной лабораторной работы является 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

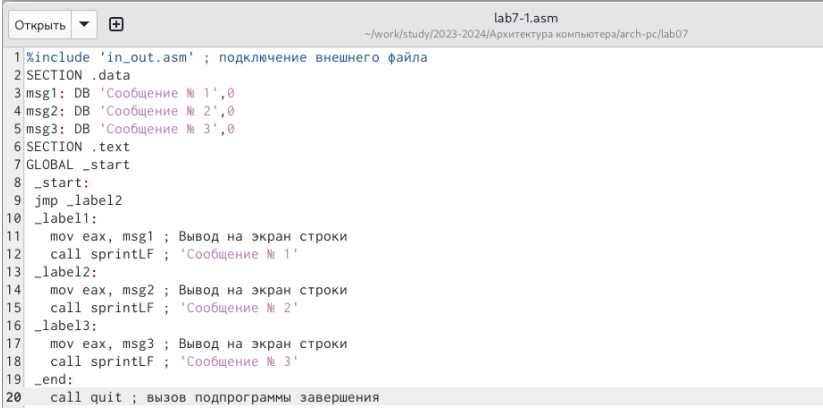
### 4.1 Реализация переходов в NASM

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

```
ndlarina@dk3n31 ~ $ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab07
ndlarina@dk3n31 ~ $ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab07
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ touch lab7-1.asm
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ls
lab7-1.asm
```

Рис. 4.1: Создание файлов для лабораторной работы

Далее 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).

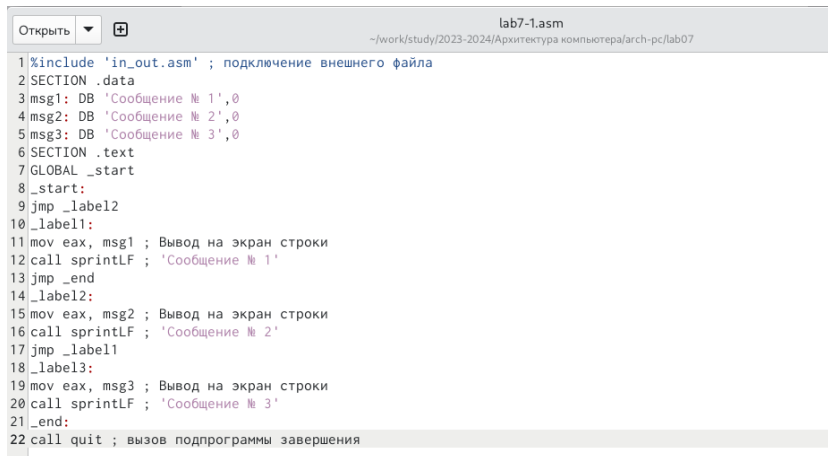


```
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f elf lab7-1.asm
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ./lab7-1
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 затем завершала работу. Для этого изменяю текст программы в соответствии с листингом 7.2. (рис. 4.4).



```
lab7-1.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07
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
```

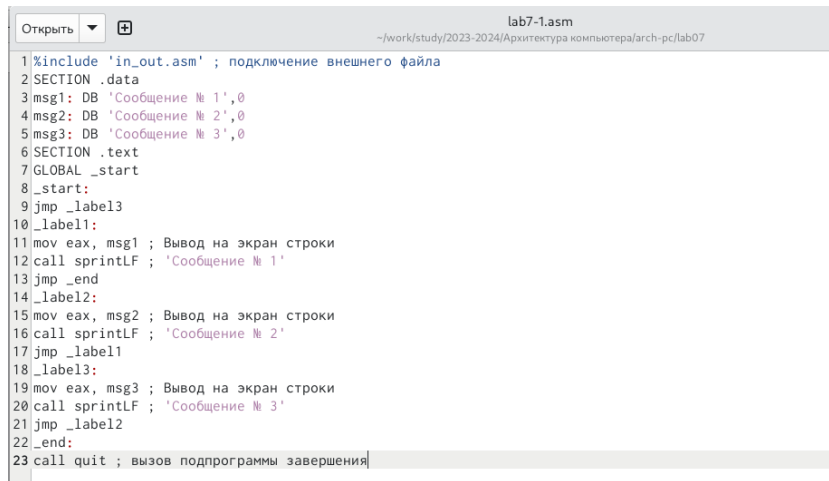
Рис. 4.4: Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. 4.5).

```
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f elf lab7-1.asm
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 1
```

Рис. 4.5: Создание исполняемого файла

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. 4.6).



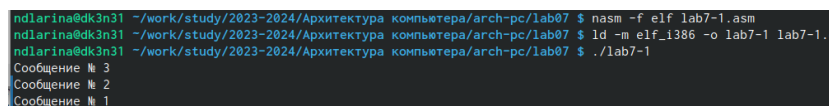
```

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. 4.6: Изменение текста программы

Вывод программы получается следующим: (рис. 4.7).



```

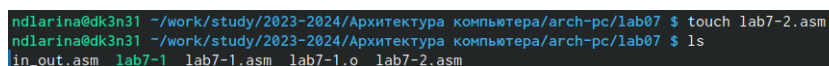
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f elf lab7-1.asm
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1

```

Рис. 4.7: 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/study/2023-2024/Архитектура компьютера/arch-pc/lab07. (рис. 4.8).




```

ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ touch lab7-2.asm
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ls
in_out.asm lab7-1 lab7-1.asm lab7-1.o lab7-2.asm

```

Рис. 4.8: Создание файла

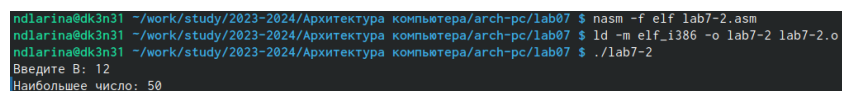
Текст программы из листинга 7.3 ввожу в lab7-2.asm. (рис. 4.9).



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

Рис. 4.9: Ввод текста программы из листинга 7.3

Создаю исполняемый файл и проверяю его работу. (рис. 4.10).



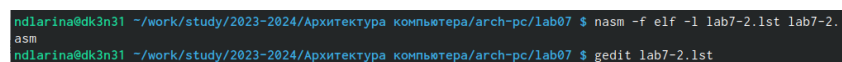
```
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f elf lab7-2.asm
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ./lab7-2
Введите B: 12
Наибольшее число: 50
```

Рис. 4.10: Проверка работы файла

Файл работает корректно.

## 4.2 Изучение структуры файлы листинга

Создаю файл листинга для программы из файла lab7-2.asm. (рис. 4.11).



```
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
ndlarina@dk3n31 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ gedit lab7-2.lst
```

Рис. 4.11: Создание файла листинга

Открываю файл листинга lab7-2.lst с помощью текстового редактора и внимательно изучаю его формат и содержимое. (рис. 4.12).

```

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> mov eax, edx

```

Рис. 4.12: Изучение файла листинга

В представленных трех строчках содержатся следующие данные: (рис. 4.13).

3	2	<1> ; Функция вычисления длины сообщения
4	3	<1> slen:
5	4 00000000 53	<1> push ebx

Рис. 4.13: 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. 4.14).

```

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',

```

Рис. 4.14: Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга. (рис. 4.15).

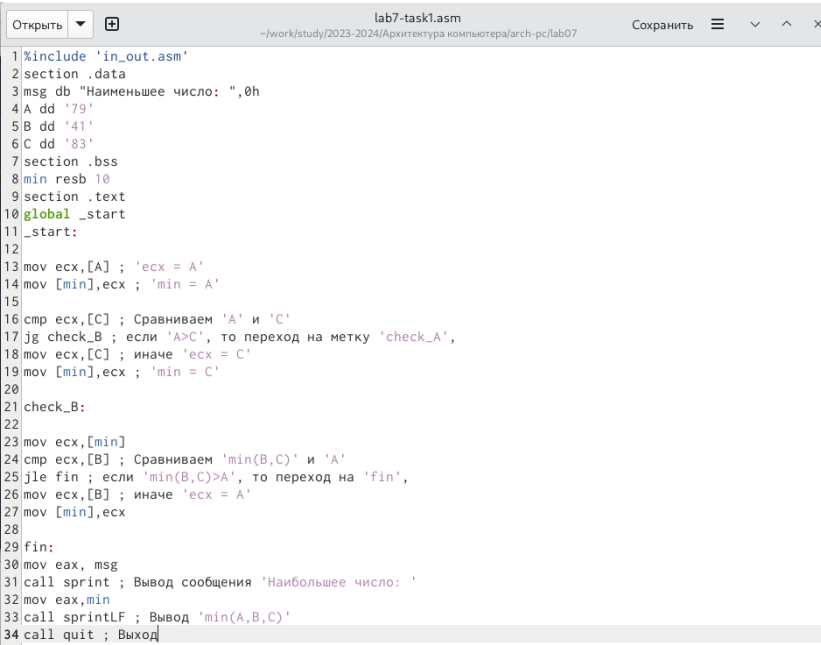
```
ndlarina@dk6n57 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f
elf -l lab7-1.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
```

Рис. 4.15: Получение файла листинга

На выходе я 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.

### 4.3 Задания для самостоятельной работы

1. 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 6, поэтому мои значения - 79, 41 и 83. (рис. 4.16).



```
lab7-task1.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07

1 %include 'in_out.asm'
2 section .data
3 msg db "Наименьшее число: ",0h
4 A dd '79'
5 B dd '41'
6 C dd '83'
7 section .bss
8 min resb 10
9 section .text
10 global _start
11 _start:
12
13 mov ecx,[A] ; 'ecx = A'
14 mov [min],ecx ; 'min = A'
15
16 cmp ecx,[C] ; Сравниваем 'A' и 'C'
17 jg check_B ; если 'A>C', то переход на метку 'check_A',
18 mov ecx,[C] ; иначе 'ecx = C'
19 mov [min],ecx ; 'min = C'
20
21 check_B:
22
23 mov ecx,[min]
24 cmp ecx,[B] ; Сравниваем 'min(B,C)' и 'A'
25 jle fin ; если 'min(B,C)>A', то переход на 'fin',
26 mov ecx,[B] ; иначе 'ecx = A'
27 mov [min],ecx
28
29 fin:
30 mov eax, msg
31 call sprint ; Вывод сообщения 'Наибольшее число: '
32 mov eax,min
33 call sprintLF ; Вывод 'min(A,B,C)'
34 call quit ; Выход
```

Рис. 4.16: Написание программы

Создаю исполняемый файл lab7-task1.asm. (рис. 4.17).

```
ndlarina@dk6n57 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ touch lab7-task1.asm
```

Рис. 4.17: Создание файла

И затем проверяю его работу, подставляя необходимые значения. (рис. 4.18).

```
ndlarina@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ nasm -f elf lab7-task1.asm
ndlarina@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ld -m elf_i386 lab7-task1.o -o lab7-task1
ndlarina@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $ ./lab7-task1
Наименьшее число: 41
ndlarina@dk3n37 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07 $
```

Рис. 4.18: Проверка работы программы

Программа работает корректно.

Код программы:

```
%include 'in_out.asm'
section .data
msg db "Наименьшее число:",0h
A dd '79'
B dd '41'
C dd '83'
section .bss
min resb 10
section .text
global _start
_start:
    mov ecx,[A] ; 'ecx = A'
    mov [min],ecx ; 'min = A'
    cmp ecx,[C] ; Сравниваем 'A' и 'C'
    jg check_B ; если 'A>C', то переход на метку 'check_A',
    mov ecx,[C] ; иначе 'ecx = C'
    mov [min],ecx ; 'min = C'
```

```

check_B:
    mov ecx,[min] cmp ecx,[B] ; Сравниваем 'min(B,C)' и 'A'
    jle fin ; если 'min(B,C)>A', то переход на 'fin',
    mov ecx,[B] ; иначе 'ecx = A'
    mov [min],ecx
    fin:
    mov eax,msg
    call sprint ; Вывод сообщения 'Наибольшее число:'
    mov eax,min
    call sprintLF ; Вывод 'min(A,B,C)'
    call quit ; Выход

```

## 5 Выводы

В ходе работы над лабораторной работой 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.



## 6 Список литературы

1. <https://esystem.rudn.ru/mod/resource/view.php?id=1030555>