

# **Лабораторная работа №7. Команды безусловного и условного переходов в Nasm. Программирование ветвлений**

**Простейший вариант**

Диана Садова Алексеевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
2.1	Порядок выполнения лабораторной работы . . . . .	6
2.1.1	Реализация переходов в NASM . . . . .	6
2.1.2	Изучение структуры файлы листинга . . . . .	12
<b>3</b>	<b>Теоретическое введение</b>	<b>16</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>17</b>
4.0.1	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 № 7. Создайте исполняемый файл и проверьте его работу. . . . .	17
4.0.2	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. . . . .	19
<b>5</b>	<b>Выводы</b>	<b>21</b>
	<b>Список литературы</b>	<b>22</b>

## Список иллюстраций

2.1	Создаем каталог и файл для дальнейшей работы . . . . .	6
2.2	Проверяем наличие файла lab7-1.asm . . . . .	6
2.3	Вводим код . . . . .	7
2.4	Создаем и запускаем исполняемый файл . . . . .	7
2.5	Вводим код программы . . . . .	8
2.6	Создаем и запускаем исполняемый файл . . . . .	8
2.7	Изменяем код программы . . . . .	9
2.8	Выводим ответ . . . . .	9
2.9	Создаем файл lab7-2.asm. Проверяем его наличие . . . . .	10
2.10	Вводим код программы . . . . .	11
2.11	Создаем и запускаем исполняемый файл. Вводим разные значения	12
2.12	Создаем файл листинга . . . . .	12
2.13	Открываем файл с помощью текстового редактора . . . . .	13
2.14	Файл в редакторе mcedit . . . . .	13
2.15	Файл листинга . . . . .	13
2.16	Изменяем файл . . . . .	14
2.17	Выполняем трансляцию листинга . . . . .	14
2.18	Строка с ошибкой . . . . .	15
4.1	Данные для ввода . . . . .	17
4.2	Программа под условие задачи . . . . .	18
4.3	Создаем и запускаем исполняемый файл . . . . .	18
4.4	Данные для решения задачи . . . . .	19
4.5	Программа под условие задачи . . . . .	20
4.6	Создаем и запускаем исполняемый файл . . . . .	20

## **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

### 2.1 Порядок выполнения лабораторной работы

#### 2.1.1 Реализация переходов в NASM

2.1.1.1 Создайте каталог для программ лабораторной работы № 7, перейдите в него и создайте файл lab7-1.asm: (рис. 2.1), (рис. 2.2).

```
dasadova@dk8n64 ~ $ mkdir ~/work/arch-pc/lab07
dasadova@dk8n64 ~ $ cd ~/work/arch-pc/lab07
dasadova@dk8n64 ~/work/arch-pc/lab07 $ touch lab7-1.asm
dasadova@dk8n64 ~/work/arch-pc/lab07 $
```

Рис. 2.1: Создаем каталог и файл для дальнейшей работы

```
dasadova@dk8n64 ~/work/arch-pc/lab07 $ ls
lab7-1.asm
dasadova@dk8n64 ~/work/arch-pc/lab07 $
```

Рис. 2.2: Проверяем наличие файла lab7-1.asm

2.1.1.2 Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Введите в файл lab7-1.asm текст программы из листинга 7.1.

Листинг 7.1. Программа с использованием инструкции `jmp`(рис. 2.3).

```

lab7-1.asm      [-M--] 18 L:[ 1+10 11/ 20] *(277 / 642b) 1074 0x432
#include "in_out.asm" ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение No 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение No 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение No 3'
_end:
call quit ; вызов подпрограммы завершения

```

Рис. 2.3: Вводим код

Создайте исполняемый файл и запустите его. Результат работы данной программы будет следующим:(рис. 2.4).

```

dasadova@dk8n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
dasadova@dk8n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
dasadova@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 3
dasadova@dk8n64 ~/work/arch-pc/lab07 $ █

```

Рис. 2.4: Создаем и запускаем исполняемый файл

Таким образом, выясняем, что программа работает корректно

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`).

Измените текст программы в соответствии с листингом 7.2.

Листинг 7.2. Программа с использованием инструкции `jmp`(рис. 2.5).

```
lab7-1.asm [-M--] 41 L:[ 1+21 22/ 22] *(664 / 664b) <EOF>
#include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 3'
_end:
call quit ; вызов подпрограммы завершения
```

Рис. 2.5: Вводим код программы

Создайте исполняемый файл и проверьте его работу.(рис. 2.6).

```
dasadova@dk8n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
dasadova@dk8n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
dasadova@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 1
dasadova@dk8n64 ~/work/arch-pc/lab07 $
```

Рис. 2.6: Создаем и запускаем исполняемый файл

Видим, что программа работает верно

Измените текст программы добавив или изменив инструкции `jmp`, чтобы вывод программы был следующим:(рис. 2.7),(рис. 2.8).



```

lab7-1.asm      [-M--] 41 L:[ 1+22 23/ 23] *(676 / 676b) <EOF>
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label3
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение No 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение No 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение No 3'
jmp _label2
_end:
call quit ; вызов подпрограммы завершения

```

Рис. 2.7: Изменяем код программы

```

dasadova@dk8n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
dasadova@dk8n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
dasadova@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 3
Сообщение No 2
Сообщение No 1
dasadova@dk8n64 ~/work/arch-pc/lab07 $

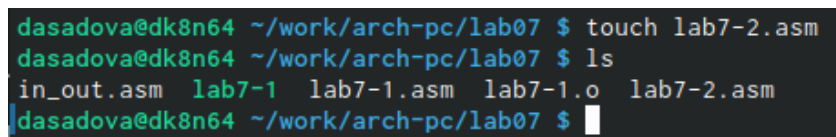
```

Рис. 2.8: Выводим ответ

Таким образом, убеждаемся, что программа работает корректно

**2.1.1.3** 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.

Создайте файл `lab7-2.asm` в каталоге `~/work/arch-pc/lab07`. Внимательно изучите текст программы из листинга 7.3 и введите в `lab7-2.asm`.(рис. 2.9).



```
dasadova@dk8n64 ~/work/arch-pc/lab07 $ touch lab7-2.asm
dasadova@dk8n64 ~/work/arch-pc/lab07 $ ls
in_out.asm  lab7-1  lab7-1.asm  lab7-1.o  lab7-2.asm
dasadova@dk8n64 ~/work/arch-pc/lab07 $
```

Рис. 2.9: Создаем файл `lab7-2.asm`. Проверяем его наличие

Листинг 7.3. Программа, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А,В и С.(рис. 2.10).

```

lab7-2.asm      [-M--] 17 L: [ 1+48 49/ 49] *(1743/1743b) <EOF>
%include "in_out.asm"
section .data
msg1 db "Введите B: ",0h
msg2 db "Наибольшее число: ",0h
A dd '20'
C dd '80'
section .bss
max resb 10
B resb 10
section .text
global _start
_start:
; ----- Вывод сообщения 'Введите B: '
mov eax,msg1
call sprint
; ----- Ввод 'B'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'B' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'B'
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A] ; 'ecx = A'
mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку 'check_B',
mov ecx,[C] ; иначе 'ecx = C'
mov [max],ecx ; 'max = C'
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,max
call atoi ; Вызов подпрограммы перевода символа в число
mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [max],ecx
; ----- Вывод результата
fin:
mov eax, msg2
call sprint ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintLF ; Вывод 'max(A,B,C)'
call quit ; Выход

```

Рис. 2.10: Вводим код программы

Создайте исполняемый файл и проверьте его работу для разных значений В.(рис. 2.11).

```

dasadova@dk8n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
dasadova@dk8n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
dasadova@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 5
Наибольшее число: 50
dasadova@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 50
Наибольшее число: 50
dasadova@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 100
Наибольшее число: 100
dasadova@dk8n64 ~/work/arch-pc/lab07 $

```

Рис. 2.11: Создаем и запускаем исполняемый файл. Вводим разные значения

Видим, что программа работает верно

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

## 2.1.2 Изучение структуры файлы листинга

### 2.1.2.1 Обычно `nasm` создаёт в результате ассемблирования только объектный файл.

Получить файл листинга можно, указав ключ `-l` и задав имя файла листинга в командной строке.

Создайте файл листинга для программы из файла `lab7-2.asm` (рис. 2.12).

```

dasadova@dk8n64 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
dasadova@dk8n64 ~/work/arch-pc/lab07 $ ls
in_out.asm  lab7-1  lab7-1.asm  lab7-1.o  lab7-2  lab7-2.asm  lab7-2.lst  lab7-2.o
dasadova@dk8n64 ~/work/arch-pc/lab07 $

```

Рис. 2.12: Создаем файл листинга

Откройте файл листинга `lab7-2.lst` с помощью любого текстового редактора, например `mcedit`: (рис. 2.13), (рис. 2.14).

```
dasadova@dk8n64 ~/work/arch-pc/lab07 $ mcedit lab7-2.lst
```

Рис. 2.13: Открываем файл с помощью текстового редактора

```
lab7-2.lst [----] 68 L: [ 1+10 11/225] *(780 /14458b) 0010 0x00A
1                                     %include 'in_out.asm'
1                                     <1> ;----- slen -----
2                                     <1> ; Функция вычисления длины сообщения
3                                     <1> slen:
4 00000000 53                         <1>     push     ebx
5 00000001 89C3                       <1>     mov      ebx, eax
6                                     <1>
7                                     <1> nextchar:
8 00000003 803800                     <1>     cmp      byte [eax], 0
9 00000006 7403                       <1>     jz       finished
10 00000008 40                        <1>     inc      eax
11 00000009 EBF8                      <1>     jmp      nextchar
12                                     <1>
13                                     <1> finished:
14 0000000B 29D8                       <1>     sub      eax, ebx
15 0000000D 5B                        <1>     pop      ebx
16 0000000E C3                       <1>     ret
17                                     <1>
18                                     <1>
19                                     <1> ;----- sprint -----
20                                     <1> ; Функция печати сообщения
21                                     <1> ; входные данные: mov eax,<message>
22                                     <1> sprint:
23 0000000F 52                         <1>     push     edx
```

Рис. 2.14: Файл в редакторе mcedit

Внимательно ознакомиться с его форматом и содержимым. Подробно объяснить содержимое трёх строк файла листинга по выбору.

Подробно объясним содержание строк 5,8,9 (рис. 2.15).

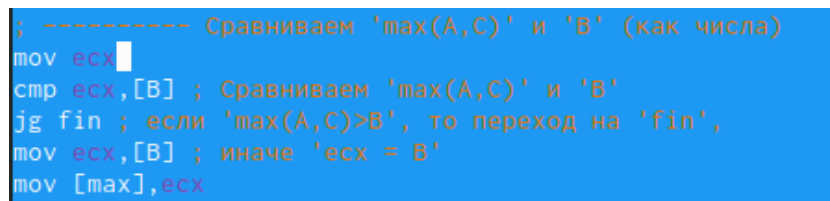
```
5 00000001 89C3                       <1>     mov      ebx, eax
6                                     <1>
7                                     <1> nextchar:
8 00000003 803800                     <1>     cmp      byte [eax], 0
9 00000006 7403                       <1>     jz       finished
```

Рис. 2.15: Файл листинга

- 1) 5 - Номер строки файла листинга 00000001 - Адрес (смещение машинного кода от начала текущего сегмента) 89C3 - Машинный код (ассемблированная исходная строка в виде шестнадцатеричной последовательности) mov ebx,eax - Исходный код программы (в данной строке идет перемещение значения eax в ebx)

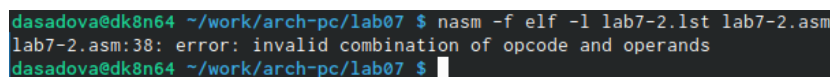
- 2) 8 - Номер строки файла листинга 00000003 - Адрес (смещение машинного кода от начала текущего сегмента) 803800 - Машинный код (ассемблированная исходная строка в виде шестнадцатеричной последовательности) `cmp byte [eax],0` - Исходный код программы (в данной строке идет сравнение значение байта, на который указывает `eax`, и 0)
- 3) 9 - Номер строки файла листинга 00000006 - Адрес (смещение машинного кода от начала текущего сегмента) 7403 - Машинный код (ассемблированная исходная строка в виде шестнадцатеричной последовательности) `jz finished` - Исходный код программы (в данной строке используем `jz` для перехода в `finished`)

Откройте файл с программой `lab7-2.asm` и в любой инструкции с двумя операндами удалить один операнд. Выполните трансляцию с получением файла листинга:(рис. 2.16),(рис. 2.17).



```
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx, [B]
cmp ecx, [B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx, [B] ; иначе 'ecx = B'
mov [max], ecx
```

Рис. 2.16: Изменяем файл



```
dasadova@dk8n64 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:38: error: invalid combination of opcode and operands
dasadova@dk8n64 ~/work/arch-pc/lab07 $
```

Рис. 2.17: Выполняем трансляцию листинга

Какие выходные файлы создаются в этом случае? Что добавляется в листинге?

После выполнение трансляции на экране появляется ошибка. В полученном файле листинга также возникает строчка об ошибке в том месте где мы удалили один из операндов (рис. 2.18).

```
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx
    error: invalid combination of opcode and operands
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
```

Рис. 2.18: Строка с ошибкой

### **3 Теоретическое введение**



## 4 Выполнение лабораторной работы

**4.0.1 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 № 7. Создайте исполняемый файл и проверьте его работу.**

Для дальнейшей работы создадим файл lab7-zadanie1.asm и продолжим работать в нем. Мой вариант в предыдущей лабораторной работе был 19, так что я начала писать программу используя значения из 19 номера (рис. 4.1),(рис. ??),(рис. 4.3)

19	46,32,74
----	----------

Рис. 4.1: Данные для ввода

```

lab7-zadanie1.asm  [----] 49 L:[ 1+ 2 3/ 41] *(102 /14
%include 'in_out.asm'
section .data
msg1 db 'A = 46, B = 32, C = 74. Найти наименьшее.',0h
msg2 db "Наименьшее число: ",0h
A dd '46'
B dd '32'
C dd '74'
section .bss
min resb 10
section .text
global _start
_start:
mov eax,msg1
call sprintf
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A] ; 'ecx = A'
mov [min],ecx ; 'min = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку 'check_B',
mov ecx,[C] ; иначе 'ecx = C'
mov [min],ecx ; 'min = C'
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,min
call atoi ; Вызов подпрограммы перевода символа в число
mov [min],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'min(A,C)' и 'B' (как числа)
mov ecx,[min]
cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
jg fin ; если 'min(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [min],ecx
; ----- Вывод результата
fin:
mov eax, msg2
call sprintf ; Вывод сообщения 'Наименьшее число: '
mov eax,min
call atoi
call iprintLF ; Вывод 'max(A,B,C)'
call quit ; Выход

```

Рис. 4.2: Программа под условие задачи

```

dasadova@dk8n72 ~/work/arch-pc/lab07 $ nasm -f elf lab7-zadanie1.asm
dasadova@dk8n72 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-zadanie1 lab7-zadanie1.o
dasadova@dk8n72 ~/work/arch-pc/lab07 $ ./lab7-zadanie1
A = 46, B = 32, C = 74. Найти наименьшее.
Наименьшее число: 32
dasadova@dk8n72 ~/work/arch-pc/lab07 $

```

Рис. 4.3: Создаем и запускаем исполняемый файл

**4.0.2 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.**

Для дальнейшей работы создадим файл lab7-zadanie2.asm и продолжим работать в нем. Мой вариант в предыдущей лабораторной работе был 19, так что я начала писать программу используя значения из 19 номера. При решении буду использовать `jb (a>b ;Переход если больше)`(рис. 4.4),(рис. 4.6),(рис. ??)

$$19 \quad \begin{cases} a+x, & x > a \\ x, & x \leq a \end{cases} \quad (4;5) \quad (3;2)$$

Рис. 4.4: Данные для решения задачи

```

lab7-zadanie2.asm [----] 2 L: [ 11+28 39/ 54] *(874 /1170b) 0032 0x020
global _start
_start:
; ----- Вывод сообщения'
mov eax,msg1
call sprint
; ----- Ввод x и a
mov ecx,x
mov edx,10
call sread

mov eax,x
call atoi ; Вызов подпрограммы перевода символа в число
mov [x],eax ; запись преобразованного числа в 'x'

mov eax,msg2
call sprint

mov ecx,a
mov edx,10
call sread
; ----- Преобразование x и a из символа в число

mov eax,a
call atoi ; Вызов подпрограммы перевода символа в число
mov [a],eax ; запись преобразованного числа в 'a'

mov ecx,[x]
cmp ecx,[a]
jg[check_B]
mov [max],ecx
jmp fin

check_B:
add ecx,[a]
mov [max],ecx
jmp fin
; ----- Вывод результата
fin:
mov eax, msg3
call sprint ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintLF ; Вывод 'max(A,B,C)'
call quit ; Выход

```

Рис. 4.5: Программа под условие задачи

```

dasadova@dk4n71 ~/work/arch-pc/lab07 $ ./lab7-zadanie2
Введите x: 4
Введите a: 5
Ответ: 4
dasadova@dk4n71 ~/work/arch-pc/lab07 $ ./lab7-zadanie2
Введите x: 3
Введите a: 2
Ответ: 5

```

Рис. 4.6: Создаем и запускаем исполняемый файл

## 5 Выводы

Изучили команды условного и безусловного переходов. Приобрели навыков написания программ с использованием переходов. Познакомились с назначением и структурой файла листинга.

## **Список литературы**