РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук Кафедра прикладной информатики и теории вероятностей

Архипов Олег Константинович

Содержание

Список иллюстраций

1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Безусловный переход

Создаю каталог ~/work/arch-pc/lab07 , перехожу в него и создаю в нем файл lab7-1.asm (рис. 2.1).

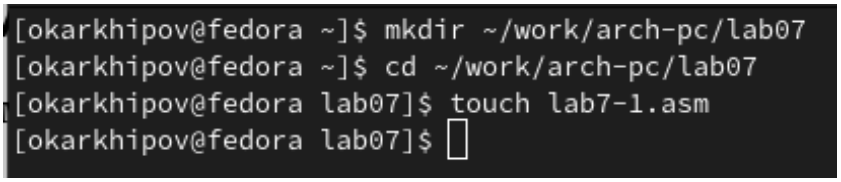

Рис. 2.1: Директория программных файлов новой ЛР

Как и в прошлые разы копирую в созданный каталог внешний файл in_out.asm , а затем при помощи команды gedit lab7-1.asm открываю файл в текстовом редакторе (рис. 2.2).

Рис. 2.2: Открываю файл в редакторе

Вбиваю текст программы с применением инструкции jmp из первого листинга ЛР (рис. 2.3).

```
%include 'in_out.asm' ; подключение внешнего файла
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 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

Рис. 2.3: Текст программы

Создаю исполняемый файл для программы и запускаю его, получаю последовательность сообщений “2”->“3”, т.к. после начала программы реализуется переход к label2 минуя label1 посредством команды `NASM jmp_label2` (рис. 2.3-2.4).

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

Рис. 2.4: Исполнение программы

Изменяю программу так, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого в текст программы после вывода сообщения № 2 добавлю инструкцию jmp с меткой _label1 , а после вывода сообщения № 1 добавлю инструкцию jmp с меткой _end (рис. 2.5).

```
%include 'in_out.asm' ; подключение внешнего файла
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 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

Рис. 2.5: Текст измененной программы

Создаю и запускаю исполняемый файл, выводится верная последовательность сообщений (рис. 2.6).

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

Рис. 2.6: Результат работы измененной программы

В начале программы изменяю инструкцию jmp _label2 на jmp _label3 , затем после вывода сообщения № 3 добавляю jmp _label2 , чтобы вывод програм- мы был следующим

```
[okarkhipov@fedora lab07]$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
[okarkhipov@fedora lab07]$
```

Получаю результат с изображений 2.7 и 2.8.

```
%include 'in_out.asm' ; подключение внешнего файла
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 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 3'
jmp _label2
_end:
call quit ; вызов подпрограммы завершения
```

Рис. 2.7: Второе изменение программы lab7-1.asm

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

Рис. 2.8: Результат работы второй измененной программы

2.2 Условный переход

Создаю файл lab7-2.asm (рис. 2.9).

```
[okarkhipov@fedora lab07]$ touch lab7-2.asm
[okarkhipov@fedora lab07]$
```

Рис. 2.9: Создание файла lab7-2.asm

Ввожу из листинга ЛР программу, которая найдет наибольшее из трех целых чисел A, B, C (рис. 2.10-2.11).

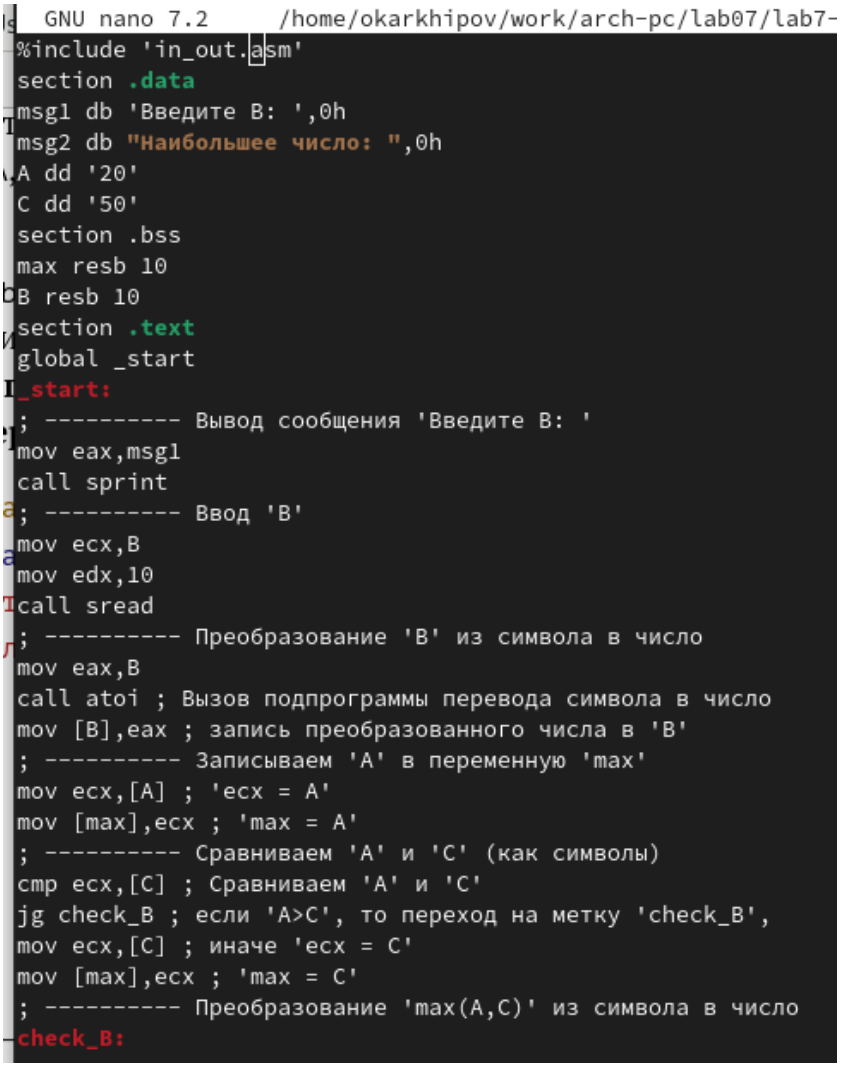

Рис. 2.10: Ввод текста программы в lab7-2.asm

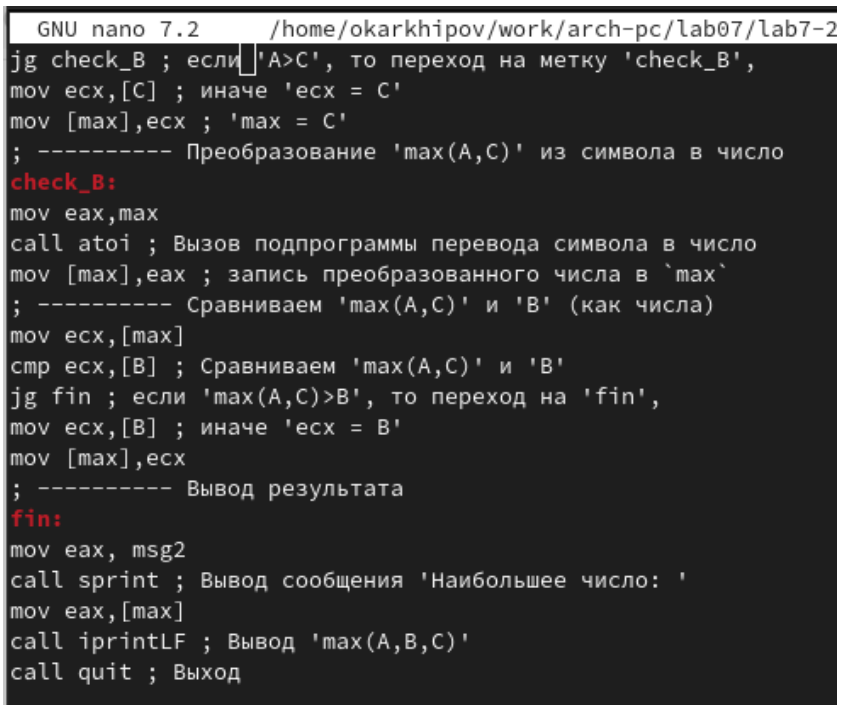

Рис. 2.11: Ввод текста программы в lab7-2.asm прод.

Создаю исполняемый файл (рис. 2.12).

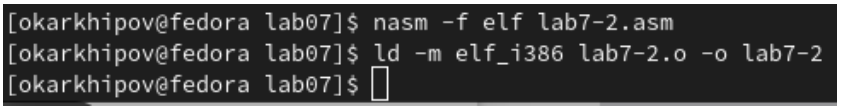

Рис. 2.12: Создание исполняемого файла lab7-2

Ввожу разные значения B: сначала 7 - наименьшее из трех чисел (рис. 2.13), затем 51 - наибольшее (рис. 2.14), 30 (рис. 2.15) и 50, равное наибольшему (рис. 2.16). Во всех случаях получаю верный результат.

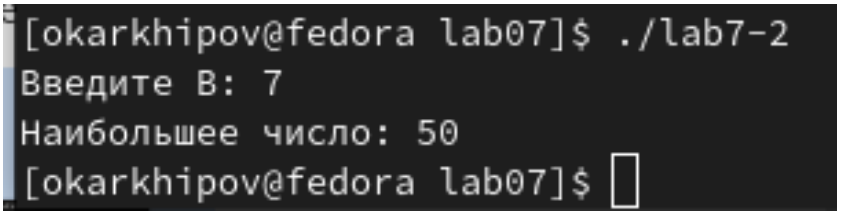

Рис. 2.13: Работа программы lab7-2 с B=7

```
[okarkhipov@fedora lab07]$ ./lab7-2
Введите B: 51
Наибольшее число: 51
[okarkhipov@fedora lab07]$
```

Рис. 2.14: Работа программы lab7-2 с B=51

```
[okarkhipov@fedora lab07]$ ./lab7-2
Введите B: 30
Наибольшее число: 50
[okarkhipov@fedora lab07]$
```

Рис. 2.15: Работа программы lab7-2 с B=30

```
[okarkhipov@fedora lab07]$ ./lab7-2
Введите B: 50
Наибольшее число: 50
[okarkhipov@fedora lab07]$
```

Рис. 2.16: Работа программы lab7-2 с B=50

2.3 Изучение структуры файлы листинга

Ввожу <nasm -f elf -l lab7-2.lst lab7-2.asm> для создания файла листинга lab7-2.lst (рис. 2.17).

```
[okarkhipov@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
[okarkhipov@fedora lab07]$
```

Рис. 2.17: Создание файла листинга

Открываю его в текстовом редакторе (рис. 2.18).

```
[okarkhipov@fedora lab07]$ mcedit lab7-2.lst
```

Рис. 2.18: Команда для открытия файла lab7-2.lst в текстовом редакторе

Рассмотрю, к примеру строки 28-30 файла листинга (отсчет строк без учета внешнего файла) (рис. 2.18). Первая колонка, очевидно, обозначает номер строки, вторая адрес (смещение машинного кода от начала текущего сегмента), например 0000011C означает - адрес команды cmp ecx,[C] (в шестнадцатиричной системе счисления), аналогично для строк 29 и 30, 00000122 и 00000124 - адреса jg check_B и mov ecx,[C] соответственно. Далее во всех трех строках идут машинные коды (3B0D[39000000], 7F0C, 8B0D[39000000]), в которые ассемблируются соответствующие инструкции. Наконец, сами cmp ecx,[C] , jg check_B , mov ecx,[C] являются исходным текстом программы, первая инструкция сравнивает A и C, вторая в зависимости от результата сравнения переводит или не переводит программу на метку ‘check_B’, третья предполагает действия, в случае, если условие jg не выполнено. Каждая из этих инструкций прокомментирована в тексте программы (рис. 2.19).

```
28 0000011C 3B0D[39000000]          cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 00000122 7F0C                    jg check_B ; если 'A>C', то переход на метку 'check_B',
30 00000124 8B0D[39000000]          mov ecx,[C] ; иначе 'ecx = C'
```

Рис. 2.19: Строки 28-30 файла lab7-2.lst

Снова открываю lab7-2.asm . В строке “mov ecx, [C] ; иначе ‘ecx=C’ ” удаляю операнд [C] (рис. 2.20).

```
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку 'check_B',
mov ecx ; иначе 'ecx = C'
```

Рис. 2.20: Удаляю операнд [C]

Выполняю трансляцию командой nasm -f elf -l lab7-2.lst lab7-2.asm , предварительно удалив старый файл листинга. Файл листинга создается, но без объектного файла (рис. 2.21-2.22), вместо этого терминал выдаёт ошибку, т.к. теперь одна из инструкций записана некорректно.

```
okarkhipov@dk8n60 ~/work/arch-pc/lab07 $ rm lab7-2.lst
okarkhipov@dk8n60 ~/work/arch-pc/lab07 $ ls
in_out.asm  lab7-1.asm  lab7-2.asm
```

Рис. 2.21: Удаляю старый файл листинга

```
okarkhipov@dk8n60 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:30: error: invalid combination of opcode and operands
okarkhipov@dk8n60 ~/work/arch-pc/lab07 $ ls
in_out.asm  lab7-1.asm  lab7-2.asm  lab7-2.lst
```

Рис. 2.22: Выполняю трансляцию

Открываю файл листинга, на месте измененной строки стоят звездочки и описание ошибки (рис. 2.23).

```
    30                              mov ecx ; иначе 'ecx = C'
    30          ******************       error: invalid combination of opcode an
d operands
```

Рис. 2.23: Файл листинга с сообщением об ошибке

После возвращаю операнд [C] на место, чтобы устранить ошибку и вновь выполняю трансляцию с получением файла листинга (рис. 2.24-2.25).

```
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'
```

Рис. 2.24: Исходное состояние файла lab7-2.asm

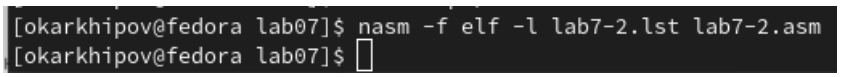

Рис. 2.25: Трансляция файла lab7-2.asm

3 Самостоятельная работа

Напомню, в прошлой ЛР мой номер варианта был 4.

3.1 Задание 1

Создаю файл для программы sol1.asm (рис. 3.1).

```
[okarkhipov@fedora report]$ cd ~/work/arch-pc/lab07
[okarkhipov@fedora lab07]$ touch sol1.asm
[okarkhipov@fedora lab07]$
```

Рис. 3.1: Создание файла sol1.asm

Чтобы решить задачу достаточно предыдущую программу изменить таким образом, чтобы она искала наименьшее значение. Для этого заменяю инструкции jg на jl и операнды max на min (рис. 3.2-3.3).

```
%include 'in_out.asm'
section .data
msg1 db 'Введите B: ',0h
msg2 db "Наименьшее число: ",0h
A dd '8'
C dd '68'
section .bss
min 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 'min'
mov ecx,[A] ; 'ecx = A'
mov [min],ecx ; 'min = A'
; ---------- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jl check_B ; если 'A<C', то переход на метку 'check_B',
mov ecx,[C] ; иначе 'ecx = C'
mov [min],ecx ; 'max = C'
; ---------- Преобразование 'min(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 `min`
```

Рис. 3.2: Программа для нахождения min 1

```
check_B:
mov eax,min
call atoi ; Вызов подпрограммы перевода символа в число
mov [min],eax ; запись преобразованного числа в `min`
; ---------- Сравниваем 'min(A,C)' и 'B' (как числа)
mov ecx,[min]
cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
jl 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 sprint ; Вывод сообщения 'Наименьшее число: '
mov eax,[min]
call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход
```

Рис. 3.3: Программа для нахождения min 2

Создаю исполняемый файл и ввожу неинициализированную переменную B (другие 2 переменные уже есть в тексте программы), получаю ответ 8 (рис. 3.4).

```
[okarkhipov@fedora lab07]$ nasm -f elf sol1.asm
[okarkhipov@fedora lab07]$ ld -m elf_i386 sol1.o -o sol1
[okarkhipov@fedora lab07]$ ./sol1
Введите B: 88
Наименьшее число: 8
[okarkhipov@fedora lab07]$
```

Рис. 3.4: Результат работы программы для нахождения min

3.2 Задание 2

4	$\begin{cases} 2x + a, & a \neq 0 \\ 2x + 1, & a = 0 \end{cases}$	(3;0)	(3;2)

Рис. 3.5: Мой вариант

Создаю файл для нового задания (рис. 3.6).

```
[okarkhipov@fedora lab07]$ touch sol2.asm
[okarkhipov@fedora lab07]$
```

Рис. 3.6: Результат работы программы для нахождения min

Для начала задам переменные (инициализированные и неинициализированные), затем составлю часть кода вывода сообщений о вводе x и a , ввода x и a и преобразования этих переменных из символов в числа (рис. 3.7).

```
%include 'in_out.asm'
section .data
msg1 db 'Введите переменную x: ',0h
msg2 db 'Введите переменную a: ',0h
msg3 db 'Ответ: ',0h
section .bss
x resb 10
a resb 10
section .text
global _start
_start:
; ------------ Сообщение 'Введите переменную x: '
mov eax,msg1
call sprint
; ------------ Ввод 'x'
mov ecx,x
mov edx,10
call sread
; ------------ Преобразование 'x' из символа в число
mov eax,x
call atoi   ; Вызов подпрограммы перевода символа в число
mov [x],eax ; запись преобразованного числа в 'x'
; ------------ Сообщение 'Введите переменную a: '
mov eax,msg2
call sprint
; ------------ Ввод 'a'
mov ecx,a
mov edx,10
call sread
; ------------ Преобразование 'a' из символа в число
mov eax,a
call atoi   ; Вызов подпрограммы перевода символа в число
mov [a],eax ; запись преобразованного числа в 'a'
```

Рис. 3.7: Первый этап программы

Затем напишу часть кода для решения задачи при a , не равном 0. Завершу программу для этого случая (рис. 3.8).

```
33 mov [a],eax ; запись преобразованного числа в 'a'
34
35 ; ------------ Задание выражения при различных a
36 mov ecx,0   ; 'ecx=0'
37 cmp ecx,[a] ; Сравниваю 0 и 'a'
38 je equal_0  ; Если 'a=0' , то переход в секцию 'equal_0'
39 mov ecx,[a] ; иначе, 'ecx = a'
40 mov eax,[x] ; 'eax=x
41 mov edi, 2  ; 'edi=2'
42 mul edi     ; 'eax=eax*edi=x*2'
43 add eax,ecx ; 'eax=eax+ecx=2x+a'
44 mov edi,eax ; 'edi<-eax=2x+a'
45
46 ; ------------- Завершение программы
47 fin:
48 mov eax, msg3
49 call sprint   ; Вывод сообщения 'Ответ: '
50 mov eax,edi
51 call iprintLF ; Вывод '2x+a или 2x+1'
52 call quit
```

Рис. 3.8: Второй этап программы (a не равно 0 и завершение)

В последней части рассмотрю случай ‘a=0’ и поставлю команду безусловного перехода к секции завершения программы (рис. 3.9).

```
52 call quit
53
54 ; ------------ Если a=0
55 equal_0:
56 cmp ecx,[a] ; Сравнение 'ecx=0' и 'a'
57 jne fin     ; Если a не равно 0, то переход к 'fin'
58 mov ecx,1   ; иначе 'ecx=1'
59 mov eax,[x] ; 'eax=x'
60 mov edi, 2  ; 'edi=2'
61 mul edi     ; 'eax=eax*edi=x*2'
62 add eax,ecx ; 'eax=eax+ecx=2x+1'
63 mov edi,eax ; 'edi<-eax'
64 jmp fin     ; Безусловный переход к 'fin'
```

Рис. 3.9: Третий этап программы

Запускаю то, что получилось и подставляю переменные. Получаю верные ответы в обоих случаях (рис. 3.10).

```
[okarkhipov@fedora lab07]$ nasm -f elf sol2.asm
[okarkhipov@fedora lab07]$ ld -m elf_i386 sol2.o -o sol2
[okarkhipov@fedora lab07]$ ./sol2
Введите переменную x: 3
Введите переменную a: 0
Ответ: 7
[okarkhipov@fedora lab07]$ ./sol2
Введите переменную x: 3
Введите переменную a: 2
Ответ: 8
[okarkhipov@fedora lab07]$
```

Рис. 3.10: Запуск программы

4 Выводы

Усвоены команды условного и безусловного перехода, а также способы их применения. Помимо этого, была изучена структура файла листинга.