

Содержание

Цель работы

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

Задание

8.3.1. Реализация переходов в NASM

1. Создайте каталог для программ лабораторной работы No 8, перейдите в него и создайте файл lab8-1.asm: `mkdir ~/work/arch-pc/lab08 cd ~/work/arch-pc/lab08 touch lab8-1.asm`
2. Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Введите в файл lab8-1.asm текст программы из предложенного листинга.

Создайте исполняемый файл и запустите его. Результат работы данной программы будет следующим: `user@dk4n31:~$./lab8-1 Сообщение No 2 Сообщение No 3 user@dk4n31:~$` 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 No 2', потом 'Сообщение No 1' и завершала работу. Для этого в текст программы после вывода сообщения No 2 добавим инструкцию `jmp` с меткой `_label1` (т.е. переход к инструкциям вывода сообщения No 1) и после вывода сообщения No 1 добавим инструкцию `jmp` с меткой `_end` (т.е. переход к инструкции `call quit`). Измените текст программы в соответствии с листингом.

Измените текст программы добавив или изменив инструкции `jmp`, чтобы вывод программы был следующим: `user@dk4n31:~$./lab8-1 Сообщение No 3 Сообщение No 2 Сообщение No 1 user@dk4n31:~$` 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: A, B и C. Значения для A и C задаются в программе, значение B вводится с клавиатуры.

Создайте исполняемый файл и проверьте его работу.

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.

8.2.1. Команды безусловного перехода.

Безусловный переход выполняется инструкцией `jmp` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление: `jmp`. Адрес перехода может быть либо меткой, либо адресом области памяти, в которую предварительно помещен указатель перехода. Кроме того, в качестве операнда можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре.

Тип операнда Описание

`jmp label` Переход на метку `label`

`jmp [label]` Переход по адресу в памяти, помеченному меткой `label`

`jmp eax` Переход по адресу из регистра `eax`

В следующем примере рассмотрим использование инструкции `jmp`:

```
label:
...      ;
...      ; команды
...      ;
jmp label
```

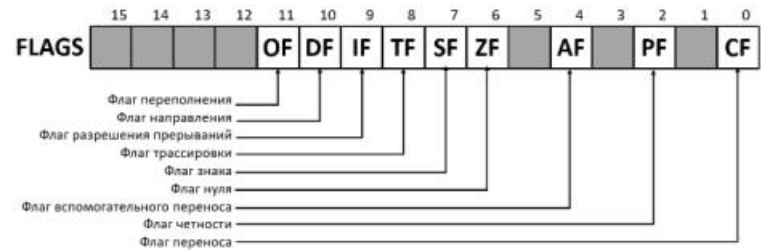
Использование инструкции `jmp`

8.2.2. Команды условного перехода

Как отмечалось выше, для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов.

8.2.2.1. Регистр флагов.

Флаг – это бит, принимающий значение 1 («флаг установлен»), если выполнено некоторое условие, и значение 0 («флаг сброшен») в противном случае. Флаги работают независимо друг от друга, и лишь для удобства они помещены в единый регистр — регистр флагов, отражающий текущее состояние процессора. В следующей таблице указано положение битовых флагов в регистре флагов:



Регистр флагов

Флаги состояния (биты 0, 2, 4, 6, 7 и 11) отражают результат выполнения арифметических инструкций, таких как ADD, SUB, MUL, DIV.

8.2.2.2. Описание инструкции `cmp`.

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 , Команда cmp`, так же как и команда вычитания, выполняет вычитание -, но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов. Примеры:

Типы операндов	Мнемокод	Критерий условного перехода $a \vee b$	Значения флагов	Комментарий
Любые	<code>JE</code>	$a = b$	$ZF = 1$	Переход если равно
Любые	<code>JNE</code>	$a \neq b$	$ZF = 0$	Переход если не равно
Со знаком	<code>JL/JNGE</code>	$a < b$	$SF \neq OF$	Переход если меньше
Со знаком	<code>JLE/JNG</code>	$a \leq b$	$SF \neq OF$ или $ZF = 1$	Переход если меньше или равно
Со знаком	<code>JG/JNLE</code>	$a > b$	$SF = OF$ и $ZF = 0$	Переход если больше
Со знаком	<code>JGE/JNL</code>	$a \geq b$	$SF = OF$	Переход если больше или равно
Без знака	<code>JB/JNAE</code>	$a < b$	$CF = 1$	Переход если ниже

Описание инструкции `cmp`

8.2.2.3. Описание команд условного перехода.

Команда условного перехода имеет вид j label. Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов. В табл. 8.3. представлены команды условного перехода, которые обычно ставятся после команды сравнения `сmp`. В их мнемокодах указывается тот результат сравнения, при котором надо делать переход. Мнемоники, идентичные по своему действию, написаны в таблице через дробь (например, `ja` и `jnb`). Программист выбирает, какую из них применить, чтобы получить более простой для понимания текст программы.

Инструкции условной передачи управления по результатам арифметического сравнения `сmp a,b`

Типы операндов	Мнемокод	Критерий условного перехода $a \vee b$	Значения флагов	Комментарий
Без знака	<code>JBE/JNA</code>	$a \leq b$	$CF = 1$ или $ZF = 1$	Переход если ниже или равно
Без знака	<code>JA/JNBE</code>	$a > b$	$CF = 0$ и $ZF = 0$	Переход если выше
Без знака	<code>JAE/JNB</code>	$a \geq b$	$CF = 0$	Переход если выше или равно

Инструкции условной передачи управления по результатам арифметического сравнения `сmp a,b`

Мне-мокод	Значение флага для осуществления перехода	Мне-мокод	Значение флага для осуществления перехода
<code>JZ</code>	$ZF = 1$	<code>JNZ</code>	$ZF = 0$

Инструкции условной передачи управления по результатам арифметического сравнения `сmp a,b`

Примечание: термины «выше» («a» от англ. «above») и «ниже» («b» от англ. «below») применимы для сравнения беззнаковых величин (адресов), а термины «больше» («g» от англ. «greater») и «меньше» («l» от англ. «lower») используются при учёте знака числа. Таким образом, мнемонику инструкции `JA/JNBE` можно расшифровать как «jump if above (переход если выше) / jump if not below equal (переход если не меньше или равно)». Помимо перечисленных команд условного перехода существуют те, которые можно использовать после любых команд, меняющих значения флагов.

Мне-мокод	Значение флага для осуществления перехода	Мне-мокод	Значение флага для осуществления перехода
<code>JZ</code>	$ZF = 1$	<code>JNZ</code>	$ZF = 0$

Инструкции условной передачи управления

Мне- мокод	Значение флага для осуществления перехода	Мне- мокод	Значение флага для осуществления перехода
JS	SF = 1	JNS	SF = 0
JC	CF = 1	JNC	CF = 0
JO	OF = 1	JNO	OF = 0
JP	PF = 1	JNP	PF = 0

Инструкции условной передачи управления

В качестве примера рассмотрим фрагмент программы, которая выполняет умножение переменных `a` и `b` и если произведение превосходит размер байта, передает управление на метку `Error`. `mov al, a mov bl, b mul bl jc Error`

8.2.3. Файл листинга и его структура.

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. Ниже приведён фрагмент файла листинга.

```

10 00000000 B804000000    mov eax,4
11 00000005 B801000000    mov ebx,1
12 0000000A B9[00000000]    mov ecx,hello
13 0000000F BA00000000    mov edx,helloLen
14
15 00000014 CD80          int 80h

```

Фрагмент файла листинга

Все ошибки и предупреждения, обнаруженные при ассемблировании, транслятор выводит на экран, и файл листинга не создаётся. Итак, структура листинга:

- номер строки — это номер строки файла листинга (нужно помнить, что номер строки в файле листинга может не соответствовать номеру строки в файле с исходным текстом программы);
- адрес — это смещение машинного кода от начала текущего сегмента;
- машинный код представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности. (например, инструкция `int 80h` начинается по смещению `00000020` в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция `int 80h` ассемблируется в `CD80` (в шестнадцатеричном представлении); `CD80` — это инструкция на машинном языке, вызывающая прерывание ядра);
- исходный текст программы — это просто строка исходной программы вместе с комментариями (некоторые строки на языке ассемблера, например, строки, содержащие только комментарии, не генерируют никакого машинного кода, и поля «смещение» и «исходный текст программы» в таких строках отсутствуют, однако номер строки им присваивается).

Выполнение лабораторной работы

8.3.1. Реализация переходов в NASM.

Первым делом я создал каталог для программ лабораторной работы No 8, перешёл в него и создал файл `lab8-1.asm` (рис.9).

```

morissaladonzo@vbox: ~$ mkdir ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07
morissaladonzo@vbox: ~$ cd ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07
morissaladonzo@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ touch lab7-1.asm
morissaladonzo@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$

```

Фрагмент файла листинга

Далее, для корректной работы я скопировал внешний файл в созданный каталог, ввёл текст программы с использованием инструкции `jmp` в текстовый файл `lab8-1.asm`, создал объектный файл и проверил работы программы (рис.10-12).

```

Ouvrir ▾  • lab7-1.asm
~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07

%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 sprintf ; 'Сообщение № 1'

_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'

_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения

```

Текст программы в Midnight Commander

```

morissaladonzo@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07
morissaladonzo@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-1.asm
morissaladonzo@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
morissaladonzo@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-1

```

Процесс создания программы

```

morissaladonzo@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07
Сообщение № 2
Сообщение № 3

```

Работа программы

Затем я изменил текст программы в соответствии с предложенным листингом и проверил её работу (рис.13-14).

```
lab7-1.asm
%include 'in_put.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 sprintfLF ; 'Сообщение № 1'
jmp _end

_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintfLF ; 'Сообщение № 2'
jmp _label1

_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintfLF ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

Текст измененной программы в Midnight Commander

```
marissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-1.asm
marissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
marissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
```

Процесс создания и работа измененной программы

Затем я изменил текст программы таким образом, чтобы сообщения выводились в следующей последовательности: 3, 2, а затем 1 (рис.15).

```
marissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-1.asm
marissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
marissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Процесс создания и работа новой программы


```

Ouvrir ▾  lab7-1.asm
~/work/study/2024-2025/Архитектура_компьютера/arch-pc/lab07

%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 sprintf ; 'Сообщение № 1'
jmp _end

_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label1

_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
jmp _label2

_end:
call quit ; вызов подпрограммы завершения

```

Текст новой программы

Описываются проведённые действия, в качестве иллюстрации даётся ссылка на иллюстрацию (рис. [-@fig:001])

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.

```

morissaladenco@vbox: ~/work/study/2024-2025/Архитектура_компьютера/arch-pc/lab07
morissaladenco@vbox: ~/work/study/2024-2025/Архитектура_компьютера/arch-pc/lab07$ touch lab7-2.asm

```

Текст новой программы

Программа, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: A,B и C


```

Ouvrir ▾ ⓘ • lab7-2.asm
~/work/study/2024-2025/Архитектура н...

%include 'in_out.asm'

section .data
msg1 db 'Введите B: ',0h
msg2 db "Наибольшее число: ",0h
A dd '20'
C dd '50'

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

```

Создайте исполняемый файл и проверьте его работу для разных значений B.

```
mortissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-2.asm
mortissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
mortissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-2
Введите B: 7
Наибольшее число: 50
mortissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-2
Введите B: 30
Наибольшее число: 50
mortissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-2
Введите B: 60
Наибольшее число: 60
mortissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-2
Введите B: 20
Наибольшее число: 50
mortissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-2
Введите B: 75
Наибольшее число: 75
```

Текст 7.3.2. Изучение структуры файлы листингат новой программы

Изучение структуры файлы листинга

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

```
mortissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
mortissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ mcedit lab7-2.lst
mortissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

Текст новой программы

Откройте файл листинга lab7-2.lst с помощью любого текстового редактора,


```
lab7-2.lst  [----]  0 1:1  0 1/23:1  *0 /149500 0032 0x020  [x] [x]
1      <1> ;----- include 'in_out.asm' ----->
2      <1> ;----- slen ----->
3      <1> ; Функция вычисления длины сообщения
4 00000000 53      <1> slen:
5 00000001 09C3     <1>     push    ebx
6      <1>     mov     ebx, eax
7      <1> nextchar:
8 00000003 003000     <1>     cmp     byte [eax], 0
9 00000004 7403     <1>     jz      finished
10 00000006 40      <1>     inc     eax
11 00000009 EBFB     <1>     jmp     nextchar
12      <1>
13      <1> finished:
14 0000000B 290B     <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
24 00000010 51      <1>     push    ecx
25 00000011 53      <1>     push    ebx
26 00000012 50      <1>     push    eax
27 00000013 EBEBFFFFFF     <1>     call    slen
28      <1>
29 00000015 09C2     <1>     mov     edx, eax
30 0000001A 5B      <1>     pop     eax
31      <1>
32 0000001B 09C1     <1>     mov     ecx, eax
33 0000001D 0B01000000     <1>     mov     ebx, 1
```

Текст новой программы

Выполнение заданий 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 № 7. Создайте исполняемый файл и проверьте его работу

```
nerissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ touch lab7-3.asm
nerissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-3.asm
nerissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
nerissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-3
Введите B: 5
Наименьшее число: 5
nerissaladonzo@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

Ouvrir ▾  • lab7-4.asm
~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07

```
#include 'in_out.asm'


section .data
    msg_x db 'Введите x: ', 0
    msg_a db 'Введите a: ', 0
    msg_result db 'Результат f(x): ', 0
    result resb 10

section .bss
    x resd 1
    a resd 1

section .text
global _start

_start:
    ; Ввод x
    mov eax, msg_x
    call sprint
    mov eax, x
    mov edx, 10
    call sread
    call atoi
    mov [x], eax

    ; Ввод a
    mov eax, msg_a
    call sprint
    mov eax, a
    mov edx, 10
    call sread
    call atoi
    mov [a], eax
```

```
Ouvrir ▾  ~/work

; Вычисление f(x)
cmp dword [x], 3
je case1 ; Если x == 3, перейти к case1

case2:
; f(x) = a + 1
mov eax, [a]
add eax, 1
jmp print_result

case1:
; f(x) = 3 * x
mov eax, [x]
imul eax, 3

print_result:
; Вывод результата
mov [result], eax
mov eax, msg_result
call sprint
mov eax, [result]
call iprintLF

call quit
```

Напишите программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений

```
morissaladonzo@ubuntu:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-4.asm
morissaladonzo@ubuntu:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-4.asm
morissaladonzo@ubuntu:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-4
Введите x: 3
```

Текст новой программы

Выводы

В результате выполнения данной лабораторной работы, я изучил команды условного и безусловного переходов, приобрел навыки написания программ с использованием переходов и познакомился с назначением и структурой файла листинга.

Список литературы

Лабораторная работа №7 (Архитектура ЭВМ).