РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

по лабораторной работе № 7

дисциплина: Архитектура компьютера

Студент: Сачковская София Александровна

Группа: НКАбд-06-25

МОСКВА

2025 г.

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. 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.

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>

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.

В следующем примере рассмотрим использование инструкции jmp:

label:

... ;

... ; команды

... ;

jmp label

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.

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.

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 <операнд_1>, <операнд_2>

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

2.2.3. Описание команд условного перехода.

Команда условного перехода имеет вид

j <мнемоника перехода> label

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

Представлены команды условного перехода, которые обычно ставятся после команды сравнения cmp. 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 jnbe). Программист выбирает, какую из них применить, чтобы получить более простой для понимания текст программы.

3.Выполнение лабораторной работы

1. Создадим каталог для программ лабораторной работы № 7, перейдем в него и создадим файл lab7-1.asm:

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

Рис.3.1

2.Введем в файл lab7-1.asm текст программы из листинга 7.1.

```
/home/sa~b7-1.asm   [----] 41 L:[  1+19  20/ 20] *(649 / 649b) <EOF>      [*][X]
%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.3.2

Создадим исполняемый файл и запустиим его. Результат работы данной программы будет следующим:

```
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ nasm -f elf lab7-1.a
sm
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ ld -m elf_i386 -o la
b7-1 lab7-1.o
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$
```

Рис.3.3

Изменим текст программы в соответствии с листингом 7.2.

Создадим исполняемый файл и проверим его работу.Изменим текст программы добавив или изменив инструкции jmp, чтобы вывод программы был следующим:

user@dk4n31:~$ ./lab7-1

Сообщение № 3

Сообщение № 2

Сообщение № 1

user@dk4n31:~$

```
/home/sachkovskayasofiya/work/arch-pc/lab07/lab7-1.asm   [----]
%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      ; Начинаем с вывода сообщения № 3

_label1:
    mov eax, msg1    ; Вывод на экран строки 'Сообщение № 1'
    call sprintLF
    jmp _end

_label2:
    mov eax, msg2    ; Вывод на экран строки 'Сообщение № 2'
    call sprintLF
    jmp _label1      ; Переход к выводу сообщения № 1

_label3:
    mov eax, msg3    ; Вывод на экран строки 'Сообщение № 3'
    call sprintLF
    jmp _label2      ; Переход к выводу сообщения № 2

_end:
    call quit        ; вызов подпрограммы завершения
```

Рис.3.4

```
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ nasm -f elf lab7-1.a
sm
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ ld -m elf_i386 -o la
b7-1 lab7-1.o
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$
```

Рис.3.5

3.Создадим файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. Внимательно изучим текст программы из листинга 7.3 и введем в lab7-2.asm

```
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ touch ~/work/arch-pc
/lab07/lab7-2.asm
```

```
/home/sachkovskayasofiya/work/arch-pc/lab07/lab7-2.asm   [----] 12 L:[  1+ 6   7/
%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
1Помощь  2Сохранить  3Блок  4Замена  5Копия
```

Рис.3.6,3.7,3.8

Создадим исполняемый файл и проверим его работу для разных значений B.

```
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 1
Наибольшее число: 50
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ nasm -f elf lab7-2.a
sm
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ ld -m elf_i386 -o la
b7-2 lab7-2.o
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 56
Наибольшее число: 56
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ nasm -f elf lab7-2.a
sm
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ ld -m elf_i386 -o la
b7-2 lab7-2.o
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 864
Наибольшее число: 864
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$
```

Рис.3.9

4.Создадим файл листинга для программы из файла lab7-2.asm

```
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ nasm -f elf -l lab7-
2.lst lab7-2.asm
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$
```

Рис.3.10

Откроем файл листинга lab7-2.lst с помощью любого текстового редактора, например mcedit

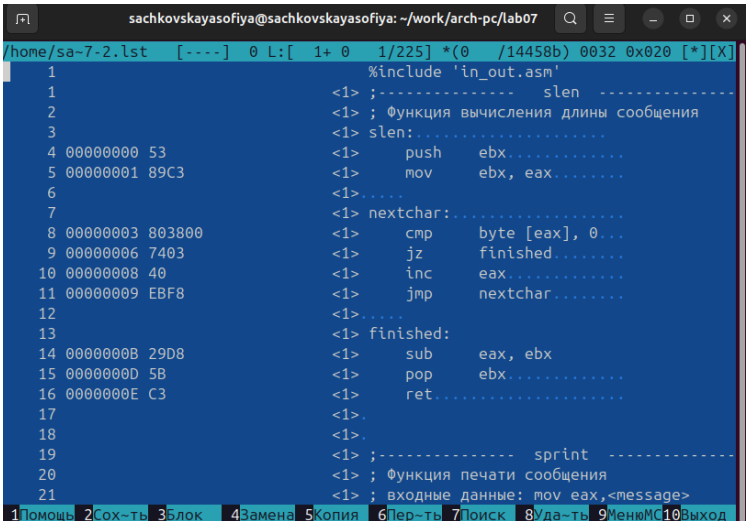


Рис.3.11

5 строка:

00000001 89C3 mov ebx, eax

эта строка выполняет сохранение начального адреса строки перед циклом; инструкция копирует текущее значение из регистра eax (указатель на начало сообщения) в регистр ebx, что необходимо для дальнейшего вычисления длины строки путем вычитания этого сохраненного адреса из конечного адреса после прохождения цикла.

8 строка:

00000003 803800 cmp byte [eax], 0

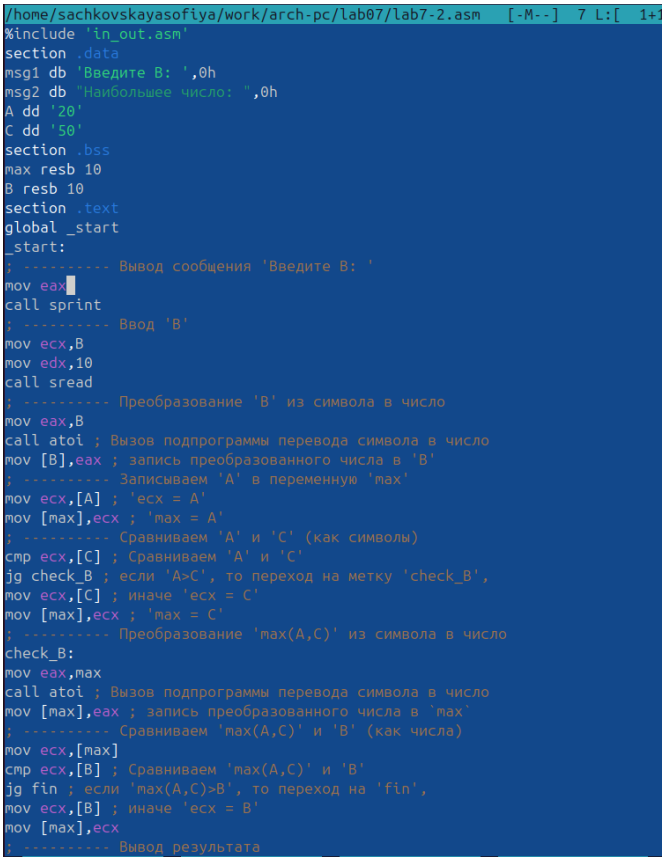
эта строка осуществляет проверку условия выхода из цикла; инструкция сравнивает значение байта, находящегося в памяти по адресу из регистра eax, с нулем (нуль-терминатором), и если они равны, процессор выставляет флаг нуля (ZF) в регистре флагов в единицу.

9 строка:

00000006 7403 jz finished

эта строка реализует условное ветвление программы; инструкция анализирует флаг нуля (ZF), и если он установлен (то есть был найден конец строки), выполнение программы перенаправляется на метку finished (смещение +3 байта), пропуская тело цикла.

Откроем файл с программой lab7-2.asm и в любой инструкции с двумя операндами, удалим один операнд. Выполним трансляцию с получением файла листинга.



```
/home/sachkovskayasofiya/work/arch-pc/lab07/lab7-2.asm   [-M--]  7 L:[  1+1
%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
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
```

```
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ nasm -f elf -l lab7-
2.lst lab7-2.asm
lab7-2.asm:14: error: invalid combination of opcode and operands
```

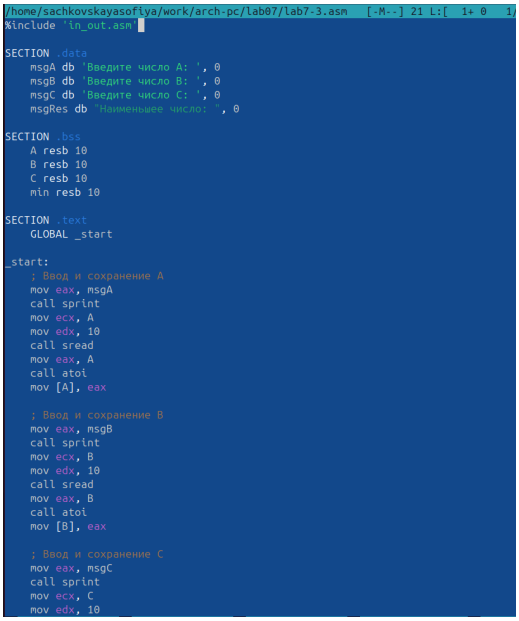
Рис.3.12,3.13

Появляется ошибка на экране,т.к. инструкция mov не может работать с одним аргументом,и,как следствие, транслятор не создаст новый файл листинга,поскольку листинг генерируется только при успешной компиляции без ошибок

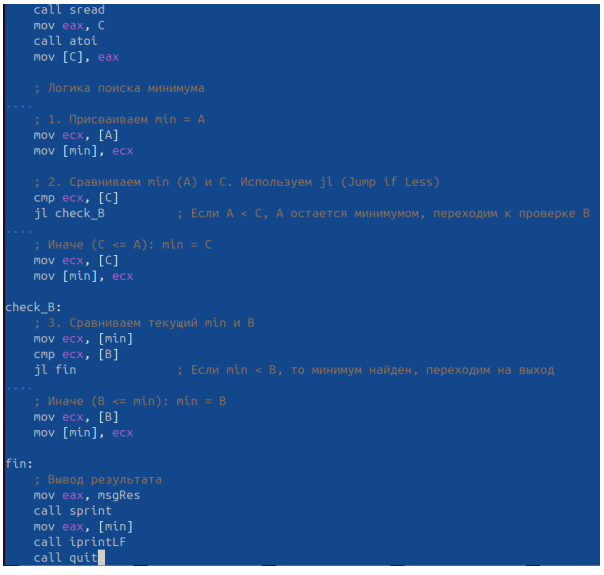
4.Выполнение самостоятельной работы

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

Ответ:Напишем программу



```
/home/sachkovskayasofiya/work/arch-pc/lab07/lab7-3.asm   [-M--] 21 L:[  1+ 0   1/
%include 'in_out.asm'

SECTION .data
    msgA db 'Введите число A: ', 0
    msgB db 'Введите число B: ', 0
    msgC db 'Введите число C: ', 0
    msgRes db 'Наименьшее число: ', 0

SECTION .bss
    A resb 10
    B resb 10
    C resb 10
    min resb 10

SECTION .text
    GLOBAL _start

_start:
    ; Ввод и сохранение A
    mov eax, msgA
    call sprint
    mov ecx, A
    mov edx, 10
    call sread
    mov eax, A
    call atoi
    mov [A], eax

    ; Ввод и сохранение B
    mov eax, msgB
    call sprint
    mov ecx, B
    mov edx, 10
    call sread
    mov eax, B
    call atoi
    mov [B], eax

    ; Ввод и сохранение C
    mov eax, msgC
    call sprint
    mov ecx, C
    mov edx, 10
```



```
    call sread
    mov eax, C
    call atoi
    mov [C], eax

    ; Логика поиска минимума
....
    ; 1. Присваиваем min = A
    mov ecx, [A]
    mov [min], ecx

    ; 2. Сравниваем min (A) и C. Используем jl (Jump if Less)
    cmp ecx, [C]
    jl check_B          ; Если A < C, A остается минимумом, переходим к проверке B
....
    ; Иначе (C <= A): min = C
    mov ecx, [C]
    mov [min], ecx

check_B:
    ; 3. Сравниваем текущий min и B
    mov ecx, [min]
    cmp ecx, [B]
    jl fin              ; Если min < B, то минимум найден, переходим на выход
....
    ; Иначе (B <= min): min = B
    mov ecx, [B]
    mov [min], ecx

fin:
    ; Вывод результата
    mov eax, msgRes
    call sprint
    mov eax, [min]
    call iprintLF
    call quit
```

Рис.4.1,4.2

Создадим исполняемый файл и проверим его работу

```
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ nasm -f elf -l lab7-
3.lst lab7-3.asm
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ ld -m elf_i386 -o la
b7-3 lab7-3.o
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ ./lab7-3
Введите число A: 56
Введите число B: 8
Введите число C: 6
Наименьшее число: 6
```

Рис.4.3

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 № 6. Создайте исполняемый файл и проверьте его работу для значений x и a из 7.6.

Ответ:Вариант 11

Напишем программу

```
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$ touch lab7-4.asm
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab07$
```

```
/home/sachkovskayasofiya/work/arch-pc/lab07/lab7-4.asm   [-M--] 13 L:[  1+20  21/
%include 'in_out.asm'

SECTION .data
    msg_x: DB 'Введите значение x: ', 0
    msg_a: DB 'Введите значение a: ', 0
    msg_res: DB 'Результат: ', 0

SECTION .bss
    x: RESB 80
    a: RESB 80
    res: RESB 80

SECTION .text
    GLOBAL _start

_start:
    ; Ввод и сохранение x
    mov eax, msg_x
    call sprint
    mov ecx, x
    mov edx, 80
    call sread
    mov eax, x
    call atol
    mov [x], eax

    ; Ввод и сохранение a
    mov eax, msg_a
    call sprint
    mov ecx, a
    mov edx, 80
    call sread
    mov eax, a
    call atol
    mov [a], eax

    ; Сравнение x и a
    mov ecx, [x]
    cmp ecx, [a]
    jge greater_equal   ; Если x >= a, переходим на метку greater_equal

    ; --- Ветвь: x < a (Формула: 5x + a) ---
    mov eax, [x]        ; eax = x
```

```
    mov ebx, 4
    mul ebx             ; eax = a * 4
    dec eax             ; eax = 4a - 1
    mov [res], eax

print_res:
    ; Вывод результата
    mov eax, msg_res
    call sprint
    mov eax, [res]
    call iprintLF
    call quit
```

Рис.4.4,4.5

Создадим исполняемый файл и проверим его работу для значений x и a

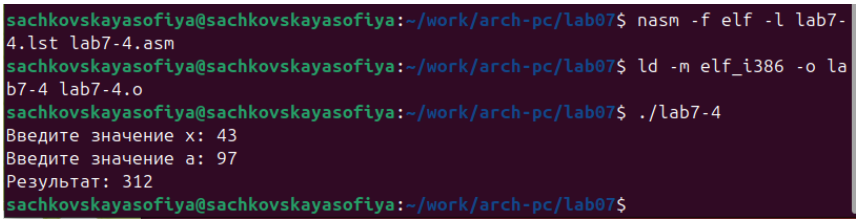


Рис.4.6

Вывод:Выполнили самостоятельную работу

5.Выводы

Я изучила команды условного и безусловного переходов. Приобрела навыки написания программ с использованием переходов. Познакомилась с назначением и структурой файла листинга.