

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютеров и операционные системы

Соколова Александра Олеговна

Содержание

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

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 `str` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `str` 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

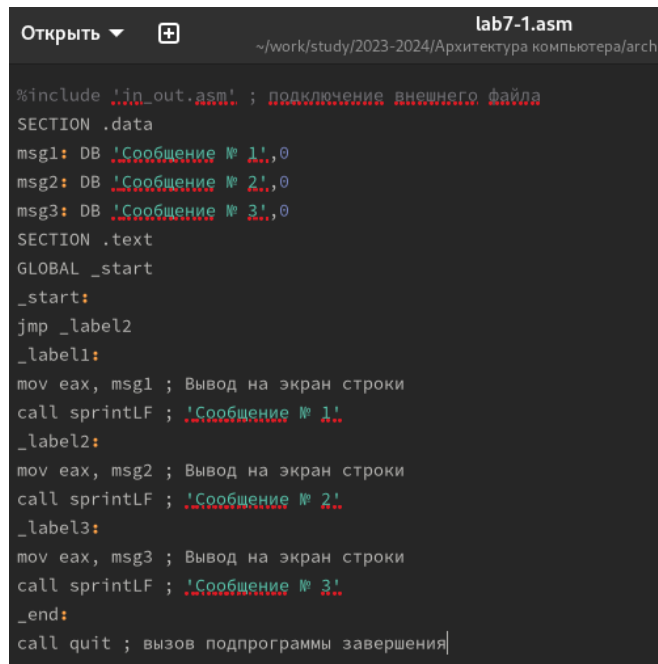
4.1 Реализация переходов в NASM

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

```
[ssokolova@fedora ~]$ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab07/lab7
[ssokolova@fedora ~]$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab07/lab7
[ssokolova@fedora lab7]$ touch lab7-1.asm
[ssokolova@fedora lab7]$
```

рис.1 Создание файлов для лабораторной работы

Ввожу в файл `lab7-1.asm` текст программы из листинга 7.1. (рис. 2).



```

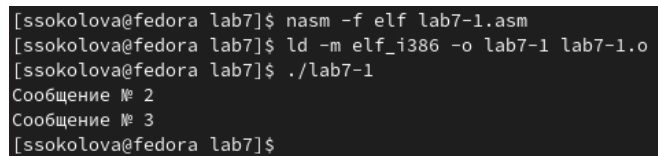
Открыть ▾ + lab7-1.asm
~/work/study/2023-2024/Архитектура компьютера/arch

%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

```

рис.2 Ввод текста программы из листинга 7.1

Создаю исполняемый файл и запускаю его. (рис. 3).



```

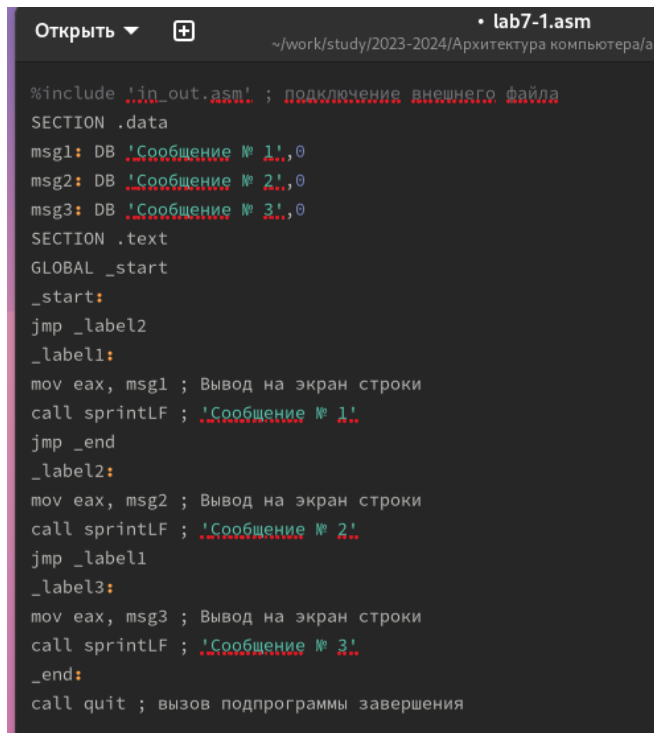
[ssokolova@fedora lab7]$ nasm -f elf lab7-1.asm
[ssokolova@fedora lab7]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[ssokolova@fedora lab7]$ ./lab7-1
Сообщение № 2
Сообщение № 3
[ssokolova@fedora lab7]$

```

рис.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 завершала работу. Для этого изменяю текст программы в соответствии с листингом 7.2. (рис. 4).



```
Открыть ▾ + lab7-1.asm
~/work/study/2023-2024/Архитектура компьютера/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 _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call printf ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call printf ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call printf ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

рис.4 Изменение текста программы

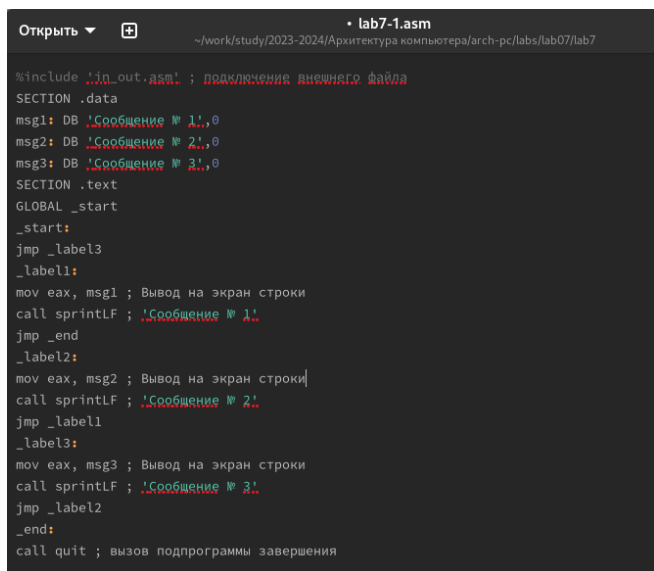
Создаю исполняемый файл и проверяю его работу. (рис. 5).



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

рис.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. 6).



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

%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 printf ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call printf ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call printf ; 'Сообщение № 3'
jmp _label2
_end:
call quit ; вызов подпрограммы завершения
```

рис.6 Изменение текста программы

чтобы вывод программы был следующим: (рис. 7).

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

рис.7 Вывод программы

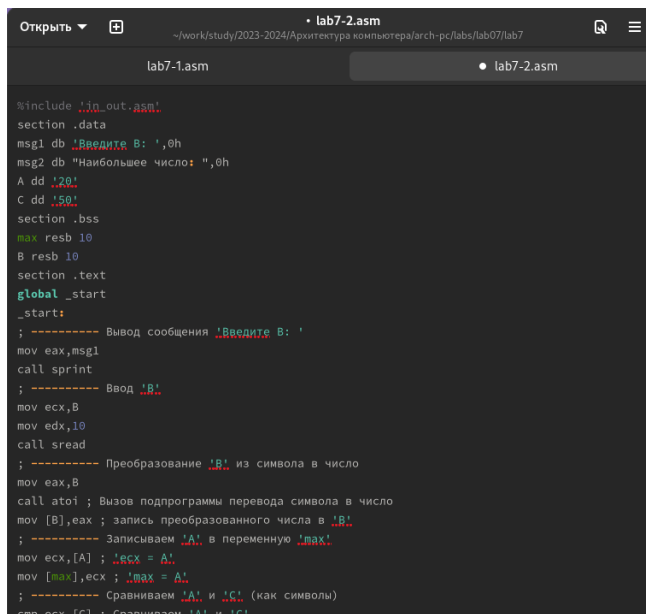
Рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: A, B и C. Значения для A и C задаются в программе, значение B вводится с клавиатуры.

Создаю файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. (рис. 8).

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

рис.8 Создание файла

Текст программы из листинга 7.3 ввожу в lab7-2.asm. (рис. 9).



```
lab7-2.asm

%include "io_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'
```

рис.9 Ввод текста программы из листинга 7.3

Создаю исполняемый файл и проверьте его работу. (рис. 10).

```

[ssokolova@fedora lab7]$ nasm -f elf lab7-2.asm
[ssokolova@fedora lab7]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[ssokolova@fedora lab7]$ ./lab7-2
Введите B: 15
Наибольшее число: 50
[ssokolova@fedora lab7]$ nasm -f elf lab7-2.asm
[ssokolova@fedora lab7]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[ssokolova@fedora lab7]$ ./lab7-2
Введите B: 120
Наибольшее число: 120
[ssokolova@fedora lab7]$

```

рис.10 Проверка работы файла

Файл работает корректно.

4.2 Изучение структуры файлы листинга

Создаю файл листинга для программы из файла lab7-2.asm. (рис. 11).

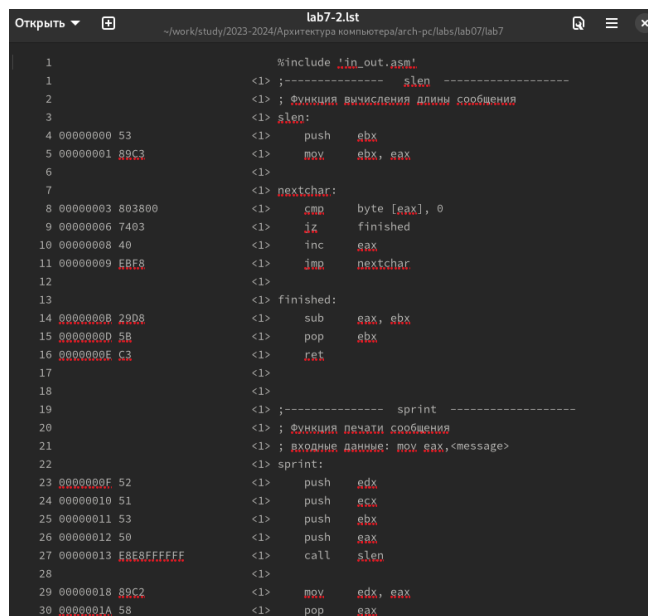
```

[ssokolova@fedora lab7]$ nasm -f elf -l lab7-2.lst lab7-2.asm
[ssokolova@fedora lab7]$

```

рис.11 Создание файла листинга

Открываю файл листинга lab7-2.lst с помощью текстового редактора и внимательно изучаю его формат и содержимое. (рис. 12).



```

1      %include "in_out.asm"
2      <1> ;----- alen -----
3      <1> ; функция вычисления длины сообщения
4      <1> alen:
5      00000000 53      <1> push    ebx
6      00000001 89C3      <1> mov     ebx, eax
7      00000002      <1> nextchar:
8      00000003 803800      <1> cmp     byte [eax], 0
9      00000006 7403      <1> jz      finished
10     00000008 40      <1> inc     eax
11     00000009 FBE8      <1> jmp     nextchar
12     0000000A      <1>
13     0000000B      <1> finished:
14     0000000B 2908      <1> sub     eax, ebx
15     0000000D 5B      <1> pop     ebx
16     0000000E C3      <1> ret
17     0000000F      <1>
18     00000010      <1>
19     00000011      <1> ;----- sprint -----
20     00000012      <1> ; функция печати сообщения
21     00000013      <1> ; входные данные: mov eax, <message>
22     00000014      <1> sprint:
23     00000015 52      <1> push    edx
24     00000016 51      <1> push    ecx
25     00000017 53      <1> push    ebx
26     00000018 50      <1> push    eax
27     00000019 E8E8FFFFFF      <1> call    alen
28     0000001A      <1>
29     0000001B 89C2      <1> mov     edx, eax
30     0000001C 58      <1> pop     eax

```

рис.12 Изучение файла листинга

В представленных трех строчках содержаться следующие данные: (рис. 13).

```

2      <1> ; функция вычисления длины сообщения
3      <1> alen:
4      00000000 53      <1> push    ebx
5      00000001 89C3      <1> mov     ebx, eax

```

рис.13 Выбранные строки файла

“2” - номер строки кода, “;” - функция вычисления длины 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. 14).

```
; ----- Сравниваем '!A!' и '!C!' (как символы)
cmp ecx, ecx; Сравниваем '!A!' и '!C!'
jg check_B; если '!A>C!', то переход на метку '!check_B!';
```

рис.14 Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга. (рис. 15).

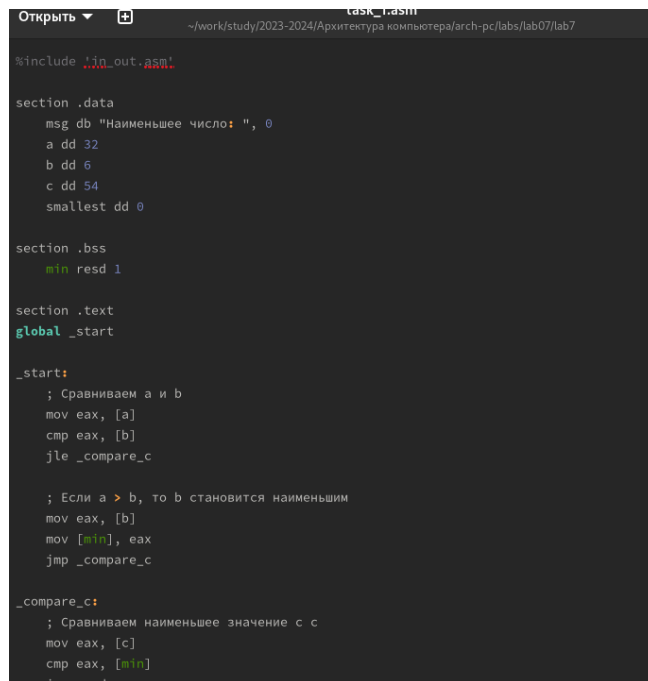
```
[ssokolova@fedora lab7]$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
[ssokolova@fedora lab7]$
```

рис.15 Получение файла листинга

На выходе я не 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 15, поэтому мои значения - 32, 6 и 54. (рис. 16).



```

task_1.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07/lab7

%include 'in_out.asm'

section .data
msg db "Наименьшее число: ", 0
a dd 32
b dd 6
c dd 54
smallest dd 0

section .bss
min resd 1

section .text
global _start

_start:
; Сравниваем а и b
mov eax, [a]
cmp eax, [b]
jle _compare_c

; Если а > b, то b становится наименьшим
mov eax, [b]
mov [min], eax
jmp _compare_c

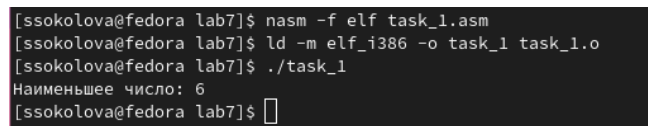
_compare_c:
; Сравниваем наименьшее значение с c
mov eax, [c]
cmp eax, [min]
jge _end

_end:

```

рис.16 Написание программы

Создаю исполняемый файл и проверяю его работу, подставляя необходимые значения. (рис. 17).



```

[ssokolova@fedora lab7]$ nasm -f elf task_1.asm
[ssokolova@fedora lab7]$ ld -m elf_i386 -o task_1 task_1.o
[ssokolova@fedora lab7]$ ./task_1
Наименьшее число: 6
[ssokolova@fedora lab7]$

```

рис.17 Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```
%include 'in_out.asm'
```

```
section .data msg db "Наименьшее число:", 0 a dd 32 b dd 6 c dd 54
smallest dd 0
```

```
section .bss min resd 1
```

```
section .text global _start
```

```
_start: ; Сравниваем а и b mov eax, [a] cmp eax, [b] jle _compare_c
```

```
; Если а > b, то b становится наименьшим
```

```
mov eax, [b]
```

```
mov [min], eax
```

```
jmp _compare_c
```


_compare_c: ; Сравниваем наименьшее значение с c mov eax, [c] str
eax, [min] jge _end

; Если c < наименьшего значения, то c становится новым наименьшим
mov [min], eax

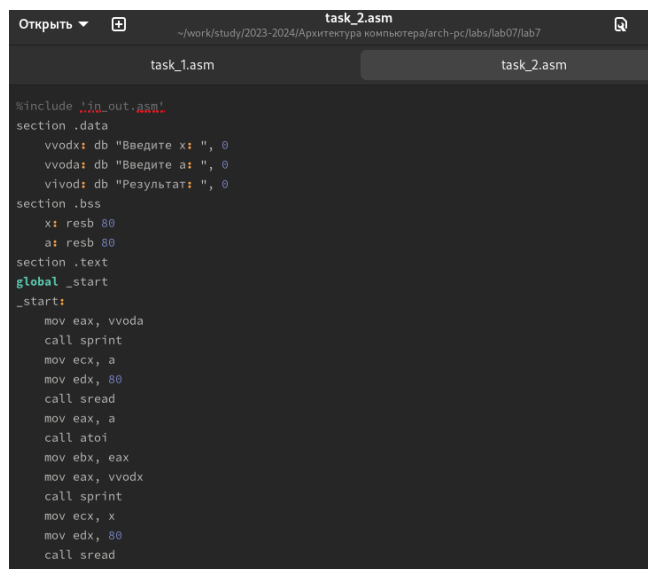
_end: ; Выводим наименьшее значение mov eax, msg call sprint ; Вывод сообщения 'Наименьшее число:' mov eax, [min] call iprintLF ; Вывод 'min(A,B,C)' call quit

1. Пишу программу, которая для введенных с клавиатуры значений x и a вычисляет значение и выводит результат вычислений заданной для моего варианта функции $f(x)$:

$a+10$, при $x < a$

$x + 10$, при $x \geq a$

(рис. 18).



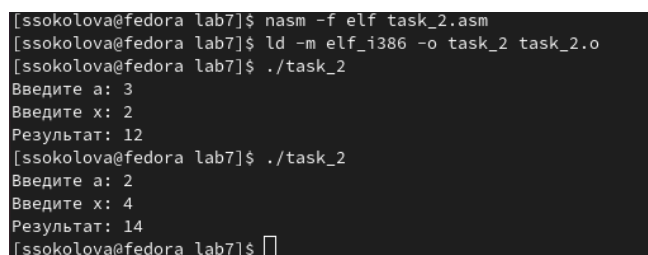
```
task_2.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07/lab7

task_1.asm task_2.asm

%include "in_out.asm"
section .data
    vvodx: db "Введите x: ", 0
    vvoda: db "Введите a: ", 0
    vivod: db "Результат: ", 0
section .bss
    x: resb 80
    a: resb 80
section .text
global _start
_start:
    mov eax, vvoda
    call sprint
    mov ecx, a
    mov edx, 80
    call sread
    mov eax, a
    call atoi
    mov ebx, eax
    mov eax, vvodx
    call sprint
    mov ecx, x
    mov edx, 80
    call sread
    mov eax, x
```

рис.18 Написание программы

Создаю исполняемый файл и проверяю его работу для значений x и a соответственно: (2, 3), (4, 2). (рис. 19).



```
[ssokolova@fedora lab7]$ nasm -f elf task_2.asm
[ssokolova@fedora lab7]$ ld -m elf_i386 -o task_2 task_2.o
[ssokolova@fedora lab7]$ ./task_2
Введите a: 3
Введите x: 2
Результат: 12
[ssokolova@fedora lab7]$ ./task_2
Введите a: 2
Введите x: 4
Результат: 14
[ssokolova@fedora lab7]$
```

рис.19 Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```
%include 'in_out.asm' section .data vvodx: db "Введите x:", 0 vvoda: db  
"Введите a:", 0 vivod: db "Результат:", 0 section .bss x: resb 80 a: resb 80  
section .text global _start _start: mov eax, vvoda call sprint mov ecx, a  
mov edx, 80 call sread mov eax, a call atoi mov ebx, eax mov eax, vvodx  
call sprint mov ecx, x mov edx, 80 call sread mov eax, x call atoi cmp eax,  
ebx jl _functiona jmp _functionx
```

```
_functiona: add eax, 10 jmp _end
```

```
_functionx: add eax, 10 jmp _end
```

```
_end: mov ecx, eax mov eax, vivod call sprint mov eax, ecx call iprintLF  
call quit
```

5 Выводы

По итогам данной лабораторной работы 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. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Lupin C. A. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).