

Отчет по лабораторной работе №7

дисциплина: Архитектура компьютера

Михайлова Регина Алексеевна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выполнение заданий для самостоятельной работы	17
4	Выводы	22
	Список литературы	23

Список иллюстраций

2.1	Создание файла	6
2.2	Листинг 7.1.	8
2.3	Результат работы программы	8
2.4	Измененная программа	10
2.5	Запуск файла	10
2.6	Листинг измененной программы	11
2.7	Запуск измененной программы	11
2.8	Листинг 7.3.	14
2.9	Проверка работы программы	15
2.10	Ошибка при запуске	16
2.11	Изменение в листинге	16
3.1	Программа	18
3.2	Работа программы	18
3.3	Задание	19
3.4	Листинг задания	20
3.5	Проверка работы выполненного задания	21

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 Выполнение лабораторной работы

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

```
mkdir ~/work/arch-pc/lab07 cd ~/work/arch-pc/lab07 touch lab7-1.asm
```

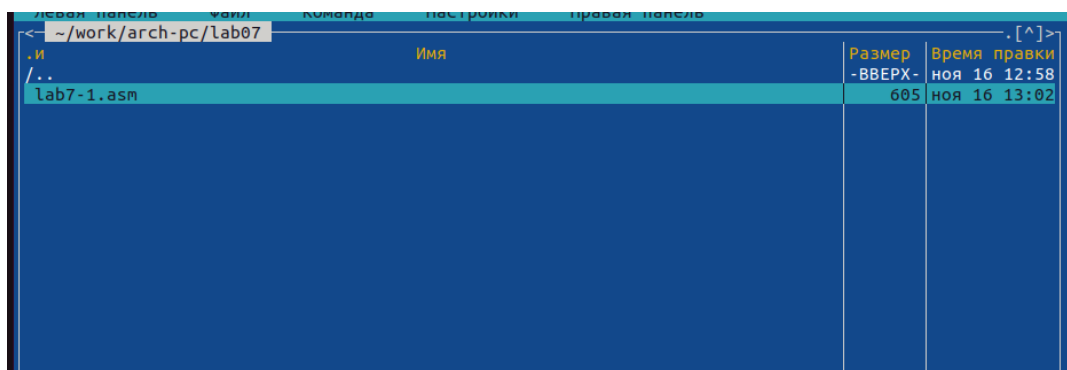


Рис. 2.1: Создание файла

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 (рис. 2.2).

Листинг 7.1. Программа с использованием инструкции jmp

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

```

GNU nano 0.2
%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

```

Рис. 2.2: Листинг 7.1.

Создайте исполняемый файл и запустите его. Результат работы данной программы будет следующим (рис. 2.3):

user@dk4n31:~\$./lab7-1 Сообщение № 2 Сообщение № 3 user@dk4n31:~\$

```

ramikhalova@ramikhalova:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ d -m elf_i386 -o lab7-1 lab7-1.o
d: команда не найдена
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
ramikhalova@ramikhalova:~/work/arch-pc/lab07$

```

Рис. 2.3: Результат работы программы

Таким образом, использование инструкции `jmp _label2` меняет порядок испол-

нения 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. (рис. 2.4)

Листинг 7.2. Программа с использованием инструкции `jmp`

```
%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 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

```

```

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

```

Рис. 2.4: Измененная программа

Создайте исполняемый файл и проверьте его работу (рис. 2.5).

```

ramikhalova@ramikhailova:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
ramikhalova@ramikhailova:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
ramikhalova@ramikhailova:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
ramikhalova@ramikhailova:~/work/arch-pc/lab07$

```

Рис. 2.5: Запуск файла

Измените текст программы добавив или изменив инструкции `jmp`, чтобы вывод программы был следующим (рис. 2.6):

```
user@dk4n31:~$ ./lab7-1 Сообщение № 3 Сообщение № 2 Сообщение № 1
user@dk4n31:~$
```

Запустите программу (рис. 2.7).

```
%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
```

Рис. 2.6: Листинг измененной программы

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

Рис. 2.7: Запуск измененной программы

- 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 С. Значения для А и С задаются в программе, значение В вводится с клавиатуры.

Создайте файл lab7-2.asm в каталоге ~/work/arch-рс/lab07. Внимательно изучите текст программы из листинга 7.3 и введите в lab7-2.asm (рис. 2.8).

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

```
%include 'in_out.asm'

section .data
msg1 db 'Введите В: ',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 В: '
mov eax,msg1
call sprint
; ----- Ввод 'В'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'В' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'В'
```

```

; ----- 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 sprintf ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintfLF ; Вывод 'max(A,B,C)'
call quit ; Выход

```

```

%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,0
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

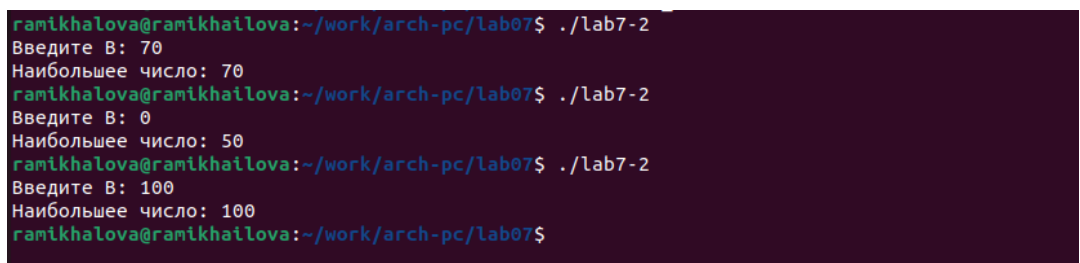
```

^{^G} Справка ^{^O} Записать ^{^W} Поиск ^{^K} Вырезать ^{^T} Выполнить ^{^C} Пози
^{^X} Выход ^{^R} ЧитФайл ^{^L} Замена ^{^U} Вставить ^{^J} Выводить ^{^/} К ст

Рис. 2.8: Листинг 7.3.

Создайте исполняемый файл и проверьте его работу для разных значений B (рис. 2.9). 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.



```
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 70
Наибольшее число: 70
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 0
Наибольшее число: 50
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 100
Наибольшее число: 100
ramikhalova@ramikhalova:~/work/arch-pc/lab07$
```

Рис. 2.9: Проверка работы программы

4. 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`

```
nasm -f elf -l lab7-2.lst lab7-2.asm
```

Откройте файл листинга `lab7-2.lst` с помощью любого текстового редактора, например `mcedit`:

```
mcedit lab7-2.lst
```

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

Откройте файл с программой `lab7-2.asm` и в любой инструкции с двумя операндами удалите один операнд. Выполните трансляцию с получением файла листинга:

```
nasm -f elf -l lab7-2.lst lab7-2.asm
```

На выходе мы получаем ошибку (рис. 2.10), которую так же можем заметить в листинге (рис. 2.11).

```

ramikhalova@ramikhalova:~/work/arch-pc/lab07$ mc
ramikhalova@ramikhalova:~/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
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ mcedit lab7-2.lst
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ touch lab7-y.asm
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ mc
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ nasm -f elf lab7-y.asm
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-y lab7-y.o
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ ./lab7-y
Наименьшее число: 12

```

Рис. 2.10: Ошибка при запуске

```

/home/ramikhalova/work/arch-pc/lab07/lab7-2.lst [B-...] 90 L:[165+25 190/226] *(11682/14544b) 0010 0x00A
165 <1> ;----- quit -----
166 <1> ; Функция завершения программы
167 <1> quit:
168 000000DB BB00000000 <1> mov ebx, 0.....
169 000000E0 B801000000 <1> mov eax, 1.....
170 000000E5 CD80 <1> int 80h
171 000000E7 C3 <1> ret
2 section .data
3 00000000 D092D0B2D0B5D0B4D0- msg1 db 'Введите B: ',0h
3 00000009 B8D182D0B520423A20-
3 00000012 00.....
4 00000013 D09D0B0D0B8D0B1D0- msg2 db "Наибольшее число: ",0h
4 0000001C BED0BBD18CD188D0B5-
4 00000025 D0B520D187D0B8D181-
4 0000002E D0BBD0BE3A2000 ....
5 00000035 32300000 A dd '20'
6 00000039 35300000 C dd '50'
7 section .bss
8 00000000 <res Ah> max resb 10
9 0000000A <res Ah> B resb 10
10 section .text
11 global _start
12 _start:
13 ; ----- Вывод сообщения 'Введите B: '
14 mov eax
14 *****
15 000000E8 E822FFFFFF error: invalid combination of opcode and operands
16 call sprint
17 ; ----- Ввод 'B'
17 000000ED B9[0A000000] mov ecx,B
18 000000F2 BA0A000000 mov edx,10
19 000000F7 E847FFFFFF call sread
20 ; ----- Преобразование 'B' из символа в число
21 000000FC B8[0A000000] mov eax,B
22 00000101 E896FFFFFF call atoi ; Вызов подпрограммы перевода символа в число
23 00000106 A3[0A000000] mov [B],eax ; запись преобразованного числа в 'B'
24 ; ----- Записываем 'A' в переменную 'max'
25 0000010B 8B0D[35000000] mov ecx,[A] ; 'ecx = A'
26 00000111 890D[00000000] mov [max],ecx ; 'max = A'
27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 00000117 3B0D[39000000] cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 0000011D 7F0C jg check_B ; если 'A>C', то переход на метку 'check_B',
30 0000011F 8B0D[39000000] mov ecx,[C] ; иначе 'ecx = C'
31 00000125 890D[00000000] mov [max],ecx ; 'max = C'
32 ; ----- Преобразование 'max(A,C)' из символа в число
33 check_B:
34 0000012B B8[00000000] mov eax,max
35 00000130 E867FFFFFF call atoi ; Вызов подпрограммы перевода символа в число
36 00000135 A3[00000000] mov [max],eax ; запись преобразованного числа в 'max'
37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
38 0000013A 8B0D[00000000] mov ecx,[max]
39 00000140 3B0D[0A000000] cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
40 00000146 7F0C jg fin ; если 'max(A,C)>B', то переход на 'fin',
41 00000148 8B0D[0A000000] mov ecx,[B] ; иначе 'ecx = B'

```

Рис. 2.11: Изменение в листинге

3 Выполнение заданий для самостоятельной работы

В прошлой лабораторной работе мой вариант был 17.

1. Напишите программу (рис. 3.1) нахождения наименьшей из 3 целочисленных переменных x, y и z . Значения переменных выбрать из табл. в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу (рис. 3.2).

```

%include 'in_out.asm'
section .data
msg db 'Наименьшее число: ',0h
a dd 26
b dd 12
c dd 68
section .bss
min resb 10
section .text
global _start
_start:

;Записываем а в переменную min
mov ecx,[a]
mov [min],ecx
;Сравниваем А и С
cmp ecx,[c]
jl check_B ;Если A<C, то переход на метку 'check_B'
mov ecx,[c] ;Иначе 'ecx=C'
mov [min],ecx

check_B:
mov ecx,[min]
cmp ecx,[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,msg ;Вывод "Наименьшее число"
call sprint
mov eax,[min] ;Вывод min(A,B,C)
call iprintLF
call quit

```

Рис. 3.1: Программа

```

ramikhalova@ramikhalova:~/work/arch-pc/lab07$ mc
ramikhalova@ramikhalova:~/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
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ mcedit lab7-2.lst

ramikhalova@ramikhalova:~/work/arch-pc/lab07$ touch lab7-y.asm
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ mc

ramikhalova@ramikhalova:~/work/arch-pc/lab07$ nasm -f elf lab7-y.asm
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-y lab7-y.o
ramikhalova@ramikhalova:~/work/arch-pc/lab07$ ./lab7-y
Наименьшее число: 12

```

Рис. 3.2: Работа программы

2. Напишите программу, которая для введенных с клавиатуры значений x и y вычисляет значение заданной функции $f(x,y)$ и выводит результат вы-

числений. Вид функции $\mathbb{R}(\mathbb{R})$ выбрать из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7 (рис. 3.3). Создайте исполняемый файл (рис. 3.4) и проверьте его работу для значений \mathbb{R} и \mathbb{R} из 7.6 (рис. 3.5).

17	$\begin{cases} a + 8, & a < 8 \\ ax, & a \geq 8 \end{cases}$	(3;4)	(2;9)	
18	$\begin{cases} a^2, & a \neq 1 \end{cases}$	(1;2)	(2;1)	

Рис. 3.3: Задание

```

#include 'in_out.asm'
SECTION .data
msg1: DB 'Введите x: ',0
msg2: DB 'Введите a: ',0
msg3: DB 'f(x) = ',0
section .bss
x resb 10
a resb 10
f resb 10
SECTION .text
GLOBAL _start
_start:
;Вывод сообщения и ввод x
mov eax,msg1
call sprint
mov ecx,x
mov edx,10
call sread
;Вывод сообщения и ввод a
mov eax,msg2
call sprint
mov ecx,a
mov edx,10
call sread
;преобразование x из сивола в число
mov eax,x
call atoi
mov [x],eax
;преобразование a из симвла в число
mov eax,a
call atoi
mov [a],eax
;Сравниваем a и 8
mov ecx,[a]
mov ebx,8
cmp ecx,ebx
jl _label ;если a меньше 8
mov eax,[x] ;иначе
mov ebx,[a]
mul ebx
mov [f],eax
jmp _end

_label:
add ecx,ebx
mov [f],ecx
jmp _end

_end:
;Вывод сообщения и результата
mov eax,msg3

```

Рис. 3.4: Листинг задания

```
ramikhalova@ramikhailova:~/work/arch-pc/lab07$ ./lab7-c
Введите x: 3
Введите a: 4
f(x) = 12
ramikhalova@ramikhailova:~/work/arch-pc/lab07$ ./lab7-c
Введите x: 2
Введите a: 9
f(x) = 18
```

Рис. 3.5: Проверка работы выполненного задания

4 Выводы

Во время выполнения лабораторной работы я изучила команды условного и безусловного переходов. Приобрела навыки написания программ с использованием переходов. Познакомилась с назначением и структурой файла листинга.

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/Learning-bash-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., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс,
- 11.
12. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
13. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
14. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
15. 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.
16. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
 17. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
 18. — 1120 с. — (Классика Computer Science).