

# **Отчёт по выполнению лабораторной работы №7**

**Дисциплина: архитектура компьютера**

Лысенко Маргарита Олеговна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>5</b>	<b>Выполнение самостоятельной работы</b>	<b>11</b>
<b>6</b>	<b>Листинги программ</b>	<b>12</b>
<b>7</b>	<b>Выводы</b>	<b>16</b>
	<b>Список литературы</b>	<b>17</b>

## Список иллюстраций

4.1	Создание каталога и файла . . . . .	8
4.2	Запуск файла . . . . .	8
4.3	Редактирование программы . . . . .	8
4.4	Запуск файла . . . . .	9
4.5	Проверка файла . . . . .	9
4.6	Открытие файла листинга . . . . .	10
4.7	Выявление ошибки . . . . .	10
5.1	Запуск файла . . . . .	11
5.2	Запуск файла . . . . .	11

## **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 Задание

Разобраться в командах условного и безусловного переходов.

### 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.

## 4 Выполнение лабораторной работы

Создала каталог для программ лабораторной работы № 7, перешла в него и создала файл lab7-1.asm (рис. 4.1).

```
molihsenko@dk8n64 ~ $ mkdir ~/work/arch-pc/lab07
molihsenko@dk8n64 ~ $ cd ~/work/arch-pc/lab07
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ touch lab7-1.asm
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ mc
```

Рис. 4.1: Создание каталога и файла

Ввела программу из листинга 7.1. Создала исполняемый файл и запустила его. (рис. 4.2).

```
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 3
```

Рис. 4.2: Запуск файла

Изменила программу так, чтобы сначала выводилась 2, а потом 3 (рис. 4.3).

```
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ mc

molihsenko@dk8n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 1
```

Рис. 4.3: Редактирование программы

Изменила текст программы в соответствии с листингом 7.2. Создала исполняемый файл и проверила его работу. Изменила текст программы добавив или изменив инструкции jmp, чтобы вывод программы был следующим: user@dk4n31:~\$



./lab7-1 Сообщение № 3 Сообщение № 2 Сообщение № 1 user@dk4n31:~\$ (рис. 4.4).

```
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 3
Сообщение No 2
Сообщение No 1
molihsenko@dk8n64 ~/work/arch-pc/lab07 $
```

Рис. 4.4: Запуск файла

Создала файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. Внимательно изучила текст программы из листинга 7.3 и ввела в lab7-2.asm. Создала исполняемый файл и проверила его работу для разных значений В. (рис. 4.5).

```
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-2
Введите В: 1
Наибольшее число: 50
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-2
Введите В: 2
Наибольшее число: 50
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-2
Введите В: 13
Наибольшее число: 50
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-2
Введите В: 51
Наибольшее число: 51
molihsenko@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-2
Введите В: 100
Наибольшее число: 100
```

Рис. 4.5: Проверка файла

Открыла файл листинга lab7-2.lst с помощью текстового редактора. Внимательно ознакомилась с его форматом и содержимым. В строке 9 содержится собственно номер сторки [9], адрес [00000003], машинный код [803800] и содержимое строки кода [cmp byte [eax], 0] в строке 11 содержится номер сторки [11], адрес [00000008], машинный код [40] и содержимое строки кода [inc eax] в строке 24 содержится номер сторки [24], адрес [0000000F], машинный код [52] и содержимое строки кода [push edx] (рис. 4.6).

```

lab7-2.lst      [-----] 0 L: 1+ 0 1/225] *(0 /14458b) 0032 0x020
1      kinclude 'in_out.asm'
2      <|> ;----- slen -----
3      <|> ; Функция вычисления длины сообщения
4      <|> slen:
5      00000000 53      <|> push ebx
6      00000001 89C3    <|> mov ebx, eax
7      <|>
8      <|> nextchar:
9      00000003 803800  <|> cmp byte [eax], 0
10     00000006 7403    <|> jz finished
11     00000008 40      <|> inc eax
12     00000009 EBF8    <|> jmp nextchar
13     <|>
14     <|> finished:
15     0000000B 2908    <|> sub eax, ebx
16     0000000D 5B      <|> pop ebx
17     0000000E C3      <|> ret
18     <|>
19     <|> ;----- sprint -----
20     <|> ; Функция печати сообщения
21     <|> ; входные данные: mov eax,<message>
22     <|> sprint:
23     0000000F 52      <|> push edx
24     00000010 51      <|> push ecx
25     00000011 53      <|> push ebx
26     00000012 50      <|> push eax
27     00000013 E8E8FFFF <|> call slen
28     <|>
29     00000018 89C2    <|> mov edx, eax
30     0000001A 58      <|> pop eax
31     <|>
32     0000001B 89C1    <|> mov ecx, eax
33     0000001D BB010000 <|> mov ebx, 1
34     00000022 B8040000 <|> mov eax, 4
35     00000027 CD80    <|> int 80h
36     <|>
37     00000029 5B      <|> pop ebx
38     0000002A 59      <|> pop ecx
39     0000002B 5A      <|> pop edx
40     0000002C C3      <|> ret
41     <|>

```

Рис. 4.6: Открытие файла листинга

В инструкции с двумя операндами удалила один операнд. Выполнила трансляцию с получением файла листинга: `nasm -f elf -l lab7-2.lst lab7-2.asm` Видно, что если в коде появляется ошибка, то ее видно в листинге. (рис. 4.7).

```

15      _start:
16      mov eax,
17      *****
17 000000E8 E8AFFFFFFF      error: invalid combination of opcode and operand
                           call atoi ; Вызов подпрограммы перевода символа в

```

Рис. 4.7: Выявление ошибки

## 5 Выполнение самостоятельной работы

Написала программу нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом 10. Создала исполняемый файл и проверила его работу. (рис. 5.1).

```
molihsenko@dk5n53 ~/work/arch-pc/lab07 $ nasm -f elf lab7-3.asm
molihsenko@dk5n53 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-3 lab7-3.o
molihsenko@dk5n53 ~/work/arch-pc/lab07 $ ./lab7-3
Наименьшее число: 35
```

Рис. 5.1: Запуск файла

Написала программу, которая для введенных с клавиатуры значений  $x$  и  $a$  вычисляет значение заданной функции  $f(x, a)$  и выводит результат вычислений. Вид функции  $f(x, a)$  выбрала из таблицы 7.6 вариантов заданий в соответствии с вариантом 10. Создала исполняемый файл и проверила его работу для значений  $x$  и  $a$ . (рис. 5.2).

```
molihsenko@dk8n52 ~/work/arch-pc/lab07 $ ./lab7-4
Введите x: 3
Введите a: 0
1
molihsenko@dk8n52 ~/work/arch-pc/lab07 $ ./lab7-4
Введите x: 1
Введите a: 2
6
```

Рис. 5.2: Запуск файла

## 6 Листинги программ

```
%include 'in_out.asm'

section .data
    msg1 db "Наименьшее число:"
    a dd 41
    b dd 62
    c dd 35

section .bss
    min resb 10

section .text
global _start

_start:
    mov eax, msg1
    call sprint

    mov ecx, [a]
    mov [min], ecx ; 'min = A'
    ; ----- Сравниваем 'A' и 'C' (как числа)
    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 ; 'min = C'
; ----- Преобразование 'min(A,C)' из символа в число

```

check\_B:

```

    ; ----- 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, [min]
    call iprintLF ; Вывод 'min(A,B,C)'
    call quit ; Выход

```

```

#include 'in_out.asm'

```

```

SECTION .data

```

```

input1 db "Введите x: ",0h

```

```

input2 db "Введите a: ",0h

```

```

SECTION .bss

```

```

max resb 10

```

```

x resb 10

```

```

a resb 10

```

```

SECTION .text
GLOBAL _start

_start:
mov eax,input1
call sprint

mov ecx,x
mov edx,10
call sread

mov eax,x
call atoi
mov [x],eax

mov eax,input2
call sprint

mov ecx,a
mov edx,10
call sread

mov eax,a
call atoi
mov [a],eax

mov ebx, 2
cmp [x], ebx

```

```
jle check
```

```
mov eax, [x]  
mov ebx, 2  
sub eax, ebx  
call iprintLF  
call quit
```

```
check:  
mov eax, [a]  
mov ebx, 3  
imul ebx  
call iprintLF  
call quit
```

## 7 Выводы

В ходе лабораторной и самостоятельной работ я изучила команды условного и безусловного переходов. Приобрела навыки написания программ с использованием переходов. Познакомилась с назначением и структурой файла листинга.



## **Список литературы**