

# **Лабораторная работа №7**

**Команды безусловного и условного переходов в Nasm.  
Программирование ветвлений**

Андрюшин Никита Сергеевич

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>3</b>	<b>Выполнение задания для самостоятельной работы</b>	<b>16</b>
<b>4</b>	<b>Выводы</b>	<b>21</b>

# Список иллюстраций

2.1	Создание рабочей директории и файла lab7-1.asm . . . . .	6
2.2	Запуск Midnight commander . . . . .	6
2.3	Вставка кода из файла листинга 7.1 . . . . .	7
2.4	Копирование файла in_out.asm в рабочую директорию . . . . .	7
2.5	Сборка программы из файла lab7-1.asm и её запуск . . . . .	8
2.6	Изменение файла lab7-1.asm согласно листингу 7.2 . . . . .	8
2.7	Повторная сборка программы из файла lab7-1.asm и её запуск . .	9
2.8	Редактирование файла lab7-1.asm . . . . .	9
2.9	Повторная сборка программы из файла lab7-1.asm и её запуск . .	10
2.10	Создание второго файла: lab7-2.asm . . . . .	10
2.11	Запись кода из листинга 7.3 в файл lab7-2.asm . . . . .	10
2.12	сборка программы из файла lab7-2.asm и её запуск . . . . .	11
2.13	Создание файла листинга из файла lab6-2.asm . . . . .	11
2.14	Открытие файла листинга в текстовом редакторе . . . . .	11
2.15	Вид файла листинга . . . . .	12
2.16	Нахождение нашей программы в файле листинга . . . . .	13
2.17	Изменение исходного файла . . . . .	14
2.18	Вывод ошибки при сборке объектного файла . . . . .	14
2.19	Отображение ошибки в листинге . . . . .	15
3.1	Создание первого файла самостоятельной работы . . . . .	16
3.2	Код первого файла самостоятельной работы . . . . .	17
3.3	Код первого файла самостоятельной работы (продолжение) . . .	18
3.4	Сборка и запуск программы первого задания самостоятельной ра- боты, а также результат выполнения . . . . .	18
3.5	Создание второго файла самостоятельной работы . . . . .	18
3.6	Код второго файла самостоятельной работы . . . . .	19
3.7	Код второго файла самостоятельной работы (продолжение) . . . .	20
3.8	Сборка и тестирование второго файла самостоятельной работы .	20

## **Список таблиц**

# 1 Цель работы

Понять принцип работы условных и безусловных переходов в Ассемблере и научиться писать программы с командами, отвечающими за переходы. Научиться работать с файлами листинга и уметь их читать.

## 2 Выполнение лабораторной работы

Для начала выполнения лабораторной работы необходимо создать рабочую папку lab07 и файл lab7-1.asm (рис. 2.1):

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

Рис. 2.1: Создание рабочей директории и файла lab7-1.asm

После чего, для удобства, запустить Midnight commander (рис. 2.2):

```
nsandryushin@nsandryushin:~/work/arch-pc/lab07$ mc
```

Рис. 2.2: Запуск Midnight commander

Вставим код в файл lab7-1.asm из файла листинга (рис. 2.3):

```

GNU nano 6.2 /home/nsandryushin/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 _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. 2.3: Вставка кода из файла листинга 7.1

Теперь скопируем файл in\_out.asm из рабочей директории прошлой лабораторной работы (рис. 2.4):

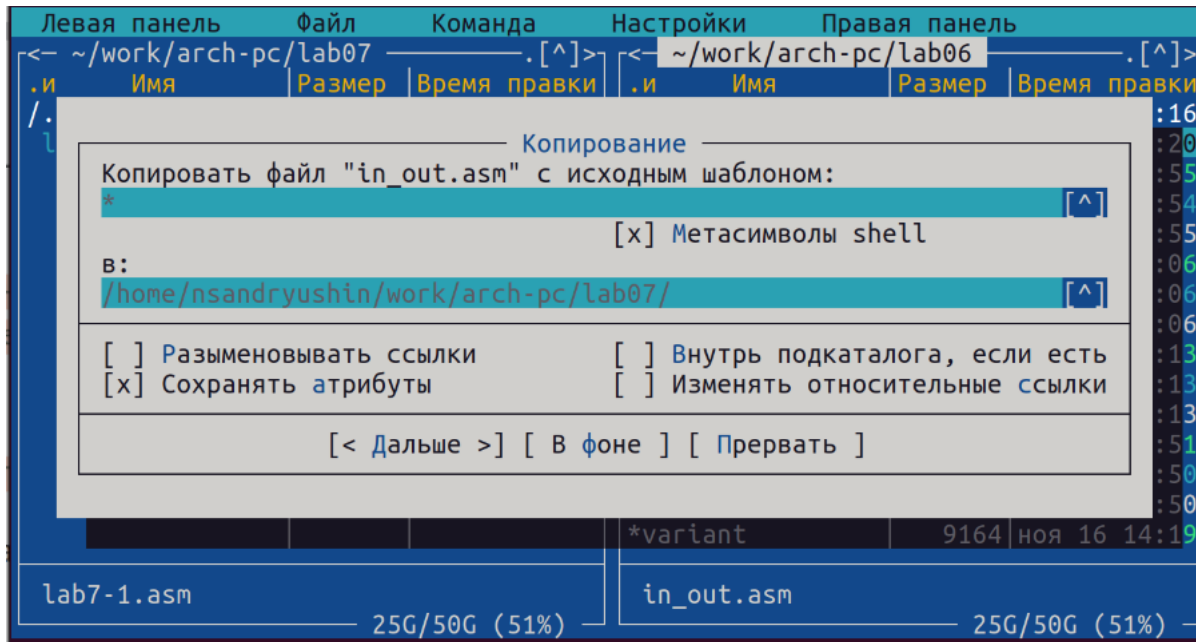


Рис. 2.4: Копирование файла in\_out.asm в рабочую директорию

Теперь соберём программу из файла lab7-1.asm и запустим её (рис. 2.5):

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

Рис. 2.5: Сборка программы из файла lab7-1.asm и её запуск

Изменим файл lab7-1.asm согласно листингу 7.2 (рис. 2.6):

```
GNU nano 6.2 /home/nsandryushin/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 _label2
_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'
_end:
call quit ; вызов подпрограммы завершения
```

Рис. 2.6: Изменение файла lab7-1.asm согласно листингу 7.2

Снова соберём программу и запустим её (рис. 2.7):



```

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

```

Рис. 2.7: Повторная сборка программы из файла lab7-1.asm и её запуск

Теперь сделаем так, чтобы код выводил сообщения в обратном порядке (от 3 сообщения к первому). Для этого внесём в код следующие изменения (рис. 2.8):

```

GNU nano 6.2 /home/nsandryushin/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
_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. 2.8: Редактирование файла lab7-1.asm

И запустим её, предварительно собрав (рис. 2.9):

```

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

```

Рис. 2.9: Повторная сборка программы из файла lab7-1.asm и её запуск

Теперь создадим файл lab7-2.asm (рис. 2.10):

```

nsandryushin@nsandryushin:~/work/arch-pc/lab07$ touch lab7-2.asm
nsandryushin@nsandryushin:~/work/arch-pc/lab07$ █

```

Рис. 2.10: Создание второго файла: lab7-2.asm

Запишем код из листинга 7.3 в файл lab7-2.asm (рис. 2.11):

```

GNU nano 6.2 /home/nsandryushin/work/arch-pc/lab07/lab7-2.asm
#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'

```

Рис. 2.11: Запись кода из листинга 7.3 в файл lab7-2.asm

И запустим его, предварительно собрав (рис. 2.12):

```

nsandryushin@nsandryushin:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
nsandryushin@nsandryushin:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
nsandryushin@nsandryushin:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 15
Наибольшее число: 50
nsandryushin@nsandryushin:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 35
Наибольшее число: 50
nsandryushin@nsandryushin:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 55
Наибольшее число: 55
nsandryushin@nsandryushin:~/work/arch-pc/lab07$

```

Рис. 2.12: сборка программы из файла lab7-2.asm и её запуск

Теперь попробуем создать файл листинга при сборке файла lab7-2.asm (рис. 2.13):

```

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

```

Рис. 2.13: Создание файла листинга из файла lab6-2.asm

Теперь посмотрим, как выглядит файл листинга изнутри. Для этого откроем его в mcedit (рис. 2.14):

```

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

```

Рис. 2.14: Открытие файла листинга в текстовом редакторе

Открыв его, мы видим следующую картину (рис. 2.15):

```

/home/ns~7-2.lst  [----]  0 L:[ 1+ 0  1/225] *(0 /14458b) 0032 0x020 [*][X]
1      %include 'in_out.asm'
2      <1> ;----- slen -----
3      <1> ; Функция вычисления длины сообщения
4      <1> slen:.....
5      00000000 53      <1>      push    ebx.....
6      00000001 89C3    <1>      mov     ebx, eax.....
7      <1>.....
8      <1> nextchar:.....
9      00000003 803800  <1>      cmp     byte [eax], 0...
10     00000006 7403    <1>      jz      finished.....
11     00000008 40      <1>      inc     eax.....
12     00000009 EBF8    <1>      jmp     nextchar.....
13     <1>.....
14     <1> finished:
15     0000000B 29D8    <1>      sub     eax, ebx
16     0000000D 5B      <1>      pop     ebx.....
17     0000000E C3      <1>      ret.....
18     <1>.
19     <1>.
20     <1> ;----- sprint -----
21     <1> ; Функция печати сообщения
22     <1> ; входные данные: mov eax,<message>
23     <1> sprint:
24     0000000F 52      <1>      push    edx
25     00000010 51      <1>      push    ecx

```

Рис. 2.15: Вид файла листинга

Наша программа находится чуть ниже (рис. 2.16):

```
/home/nsandryushin/work~ch-pc/lab07/lab7-2.lst      11867/14458      82%
171 000000E7 C3          <1>      ret
2                                section .data
3 00000000 D092D0B2D0B5D0B4D0-    msg1 db 'Введите B: ',0h
3 00000009 B8D182D0B520423A20-
3 00000012 00
4 00000013 D09DD0B0D0B8D0B1D0-    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 000000E8 B8[00000000]      mov eax,msg1
15 000000ED E81DFFFFFF      call sprint
16                               ; ----- Ввод 'B'
17 000000F2 B9[0A000000]      mov ecx,B
18 000000F7 BA0A000000      mov edx,10
19 000000FC E842FFFFFF      call sread
```

Рис. 2.16: Нахождение нашей программы в файле листинга

Разберём несколько строк файла листинга:

1. Строка под номером 14 перемещает содержимое msg1 в регистр eax. Адрес указывается сразу после номера. Следом идёт машинный код, который представляет собой исходную ассемблированную строку в виде шестнадцатиричной системы. Далее идёт исходный код

2. 15-ая строка отвечает за вызов функции sprint. Она также имеет адрес и машинный код

3. Строка 17 отвечает за запись переменной B в регистр ecx. Как видно, все строки имеют номер, адрес, машинный код и исходный код.

Теперь попробуем намеренно допустить ошибку в нашем коде, убрав у команды move 1 операнд (рис. 2.17):

```

GNU nano 6.2 /home/nsandryushin/work/arch-pc/lab07/lab7-2.asm *
#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,
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'

```

Рис. 2.17: Изменение исходного файла

И попробуем собрать файл с ошибкой, генерируя файл листинга (рис. 2.18):

```

nsandryushin@nsandryushin:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2
.asm
lab7-2.asm:18: error: invalid combination of opcode and operands
nsandryushin@nsandryushin:~/work/arch-pc/lab07$

```

Рис. 2.18: Вывод ошибки при сборке объектного файла

Мы видим, что объектный файл не создавался, однако появился файл листинга. Теперь зайдём в файл листинга, и посмотрим, отображается ли в нём ошибка (рис. 2.19):

```

/home/nsandryushin/work~ch-pc/lab07/lab7-2.lst 12377/14546 85%
 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 000000E8 B8[00000000] mov eax,msg1
15 000000ED E81DFFFFFF call sprint
16 ; ----- Ввод 'B'
17 000000F2 B9[0A000000] mov ecx,B
18 mov edx
18 *****
error: invalid combination of opcode and operands
19 000000F7 E847FFFFFF call sread
20 ; ----- Преобразование 'B' из симво
ла 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 чи
сла в 'B'

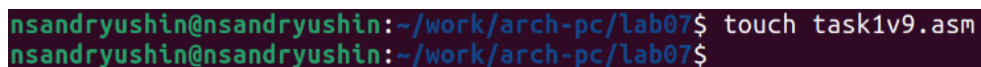
```

Рис. 2.19: Отображение ошибки в листинге

Как видим, в листинге прописана ошибка

### 3 Выполнение задания для самостоятельной работы

Создадим файл для выполнения самостоятельной работы. Мой вариант - 9 (рис. 3.1):



```
nsandryushin@nsandryushin:~/work/arch-pc/lab07$ touch task1v9.asm
nsandryushin@nsandryushin:~/work/arch-pc/lab07$
```

Рис. 3.1: Создание первого файла самостоятельной работы

Напишем код для выполнения задания. Код выглядит так (рис. 3.2 и рис. 3.3):



```

GNU nano 6.2 /home/nsandryushin/work/arch-pc/lab07/task1v9.asm
A dd '24'
B dd '98'
C dd '15'
section .bss
min resb 10
section .text
global _start
_start:
; ----- 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'

mov eax,A
call atoi ; Вызов подпрограммы перевода символа в число
mov [A],eax ; запись преобразованного числа в 'A'

mov eax,C
call atoi ; Вызов подпрограммы перевода символа в число
mov [C],eax ; запись преобразованного числа в 'C'

; ----- Записываем 'A' в переменную 'min'
mov ecx,[A] ; 'ecx = A'
mov [min],ecx ; 'min = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jle 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'
jle fin ; если 'min(A,C)<B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'

```

Рис. 3.2: Код первого файла самостоятельной работы

```

check_B:
; ----- Сравниваем 'min(A,C)' и 'B' (как числа)
mov ecx,[min]
cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
jle fin ; если 'min(A,C)<=B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [min],ecx
; ----- Вывод результата
fin:
mov eax, msg2
call sprint ; Вывод сообщения 'Наименьшее число: '
mov eax,[min]
call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход

```

Рис. 3.3: Код первого файла самостоятельной работы (продолжение)

Соберём, запустим его и посмотрим на результат (рис. 3.4):

```

nsandryushin@nsandryushin:~/work/arch-pc/lab07$ nasm -f elf task1v9.asm
nsandryushin@nsandryushin:~/work/arch-pc/lab07$ ld -m elf_i386 -o task1v9 task1v9.o
nsandryushin@nsandryushin:~/work/arch-pc/lab07$ ./task1v9
Наименьшее число: 15
nsandryushin@nsandryushin:~/work/arch-pc/lab07$

```

Рис. 3.4: Сборка и запуск программы первого задания самостоятельной работы, а также результат выполнения

Теперь создадим второй файл самостоятельной работы для второго задания (рис. 3.5):

```

nsandryushin@nsandryushin:~/work/arch-pc/lab07$ touch task2v9.asm
nsandryushin@nsandryushin:~/work/arch-pc/lab07$

```

Рис. 3.5: Создание второго файла самостоятельной работы

Код будет выглядеть так (рис. 3.6 и рис. 3.7):

```

GNU nano 6.2 /home/nsandryushin/work/arch-pc/lab07/task2v9.asm
#include 'in_out.asm'
section .data
msg1 DB "Введите X: ",0h
msg2 DB "Введите A: ",0h
msg3 DB "Ответ=",0h
section .bss
x: RESB 80
a: RESB 80
ans: RESB 80
section .text
global _start
_start:
mov eax,msg1
call sprint
mov ecx,x
mov edx,80
call sread
mov eax,x
call atoi
mov [x],eax
mov eax,msg2
call sprint
mov ecx,a
mov edx,80
call sread
mov eax,a
call atoi
mov [a],eax

mov eax, [x]
cmp eax, [a]
jle xsa

mov eax, [a]
jmp ansv

xsa:

```

Рис. 3.6: Код второго файла самостоятельной работы

```
xsa:
mov eax, [a]
add eax, [x]

ansv:
mov [ans],eax
mov eax,msg3
call sprint
mov eax,[ans]
call iprintLF
call quit
```

Рис. 3.7: Код второго файла самостоятельной работы (продолжение)

Соберём исполняемый файл и запустим его (рис. 3.8):

```
xsa:
mov eax, [a]
add eax, [x]

ansv:
mov [ans],eax
mov eax,msg3
call sprint
mov eax,[ans]
call iprintLF
call quit
```

Рис. 3.8: Сборка и тестирование второго файла самостоятельной работы

Как видим, программа всё посчитала правильно

## 4 Выводы

В результате работы над лабораторной работой были написаны программы, которые используют команды условных и безусловных переходов, были получены навыки работы с этими командами, а также были созданы и успешно прочитаны листинги для некоторых из программ.