

Лабораторная работа №7

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

Павличенко Родион Андреевич

Содержание

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

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	8
2.5	Сборка программы из файла lab7-1.asm и её запуск	8
2.6	Изменение файла lab7-1.asm согласно листингу 7.2	9
2.7	Повторная сборка программы из файла lab7-1.asm и её запуск . .	9
2.8	Редактирование файла lab7-1.asm	10
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	11
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	Изменение исходного файла	14
2.17	Вывод ошибки при сборке объектного файла	14
2.18	Отображение ошибки в листинге	15
3.1	Создание первого файла самостоятельной работы	16
3.2	Код первого файла самостоятельной работы	17
3.3	Сборка и запуск программы первого задания самостоятельной ра- боты, а также результат выполнения	18
3.4	Создание второго файла самостоятельной работы	18
3.5	Код второго файла самостоятельной работы	19
3.6	Код второго файла самостоятельной работы (продолжение)	20
3.7	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 :

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

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

После чего, для удобства, запустить Midnight commander :

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

Рис. 2.2: Запуск Midnight commander

Вставим код в файл lab7-1.asm из файла листинга :

```

GNU nano 7.2 /home/rapavlichenko/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 с помощью команды F5:

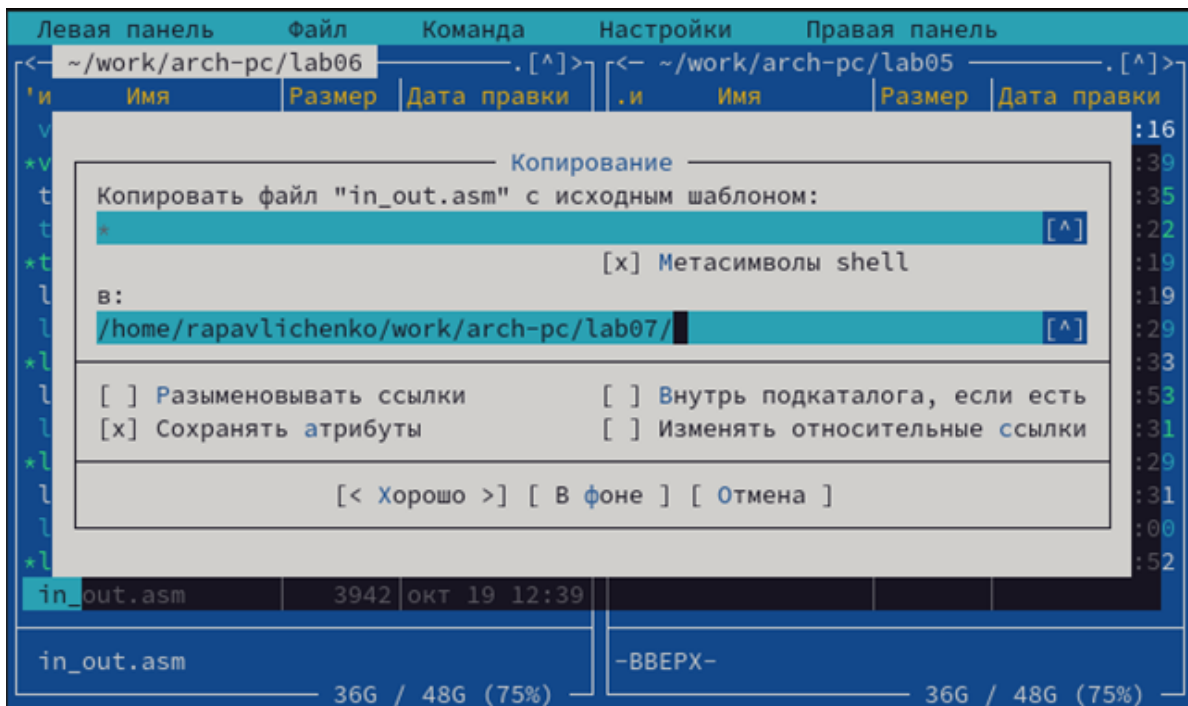


Рис. 2.4: Копирование файла in_out.asm в рабочую директорию

Теперь соберём программу из файла lab7-1.asm и запустим её :

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


```

GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab07/lab7-1.asm Изменён
#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
Архитектура ЭВМ
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки

```

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

Снова соберём программу и запустим её :

```

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

```

GNU nano 7.2 /home/rapavlichenko/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'

```

Рис. 2.8: Редактирование файла lab7-1.asm

И запустим её, предварительно собрав :

```

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

```

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

Теперь создадим файл lab7-2.asm :

```

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

```

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

Запишем код из листинга 7.3 в файл lab7-2.asm :

```
GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab07/lab7-2.asm Изменён
%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
```

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

И запустим его, предварительно собрав :

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

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

Теперь попробуем создать файл листинга при сборке файла lab7-2.asm :

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

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

Рис. 2.14: Открытие файла листинга в текстовом редакторе

Открыв его, мы видим следующую картину :

```

lab7-2.lst      [----]  0 L:[ 1+ 0  1/225] *(0  /14458b) 0032 0x020      [*][X]
1               %include 'in_out.asm'
1               <1> ;----- slen -----
2               <1> ; Функция вычисления длины сообщения
3               <1> slen:.....
4 00000000 53      <1>      push    ebx.....
5 00000001 89C3    <1>      mov     ebx, eax.....
6               <1>.....
7               <1> nextchar:.....
8 00000003 803800  <1>      cmp     byte [eax], 0...
9 00000006 7403    <1>      jz      finished.....
10 00000008 40     <1>      inc     eax.....
11 00000009 EBF8   <1>      jmp     nextchar.....
12               <1>.....
13               <1> finished:
14 0000000B 29D8   <1>      sub     eax, ebx
15 0000000D 5B     <1>      pop     ebx.....
16 0000000E C3     <1>      ret.....
17               <1>.
18               <1>.
19               <1> ;----- sprint -----
20               <1> ; Функция печати сообщения
21               <1> ; входные данные: mov eax, <message>
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ить 7Поиск 8Удалить 9МенюМС 10Выход

```

Рис. 2.15: Вид файла листинга

Наша программа находится чуть ниже :

```

lab7-2.lst      [----]  0 L:[170+21 191/225] *(11649/14458b) 0032 0x020      [*][X]
169 000000E5 CD80 <1>      int     80h
170 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'
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ить 7Поиск 8Удалить 9МенюМС 10Выход

```

Разберём несколько строк файла листинга:

1. Строка `msg1 db 'Введите B:', 0h` Это объявление строки `msg1` в разделе `.data`,

которая содержит текст 'Введите В:' и заканчивается нулевым байтом (0h). Этот нулевой байт используется для обозначения конца строки в ассемблере. Строка msg1 предназначена для вывода на экран, чтобы запросить у пользователя ввод значения переменной В.

2. Строка `A dd '20'` В этой строке создается переменная А в разделе .data, и ей присваивается значение '20', представленное как dd, что означает "double word" (двойное слово или 4 байта). Значение '20' хранится в 16-ричном формате (т.е. это не число 20 в десятичной системе, а символы '20'). Это значение может использоваться в программе для выполнения различных операций.

3. Строка `mov eax, msg1` В этой строке в разделе .text перемещается адрес строки msg1 в регистр eax. Это делается для того, чтобы передать указатель на строку msg1 функции `sprint`, которая будет отвечать за вывод сообщения на экран. Эта строка помогает организовать вывод текста, запрашивающего ввод пользователя. Теперь попробуем намеренно допустить ошибку в нашем коде, убрав у команды `move` 1 операнд

Теперь попробуем намеренно допустить ошибку в нашем коде, убрав у команды `move` 1 операнд :

```
GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab07/lab7-2.asm Изменён
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
```

Рис. 2.16: Изменение исходного файла

И попробуем собрать файл с ошибкой, генерируя файл листинга :

```
rapavlichenko@rapavlichenko:~/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
rapavlichenko@rapavlichenko:~/work/arch-pc/lab07$
```

Рис. 2.17: Вывод ошибки при сборке объектного файла

Мы видим, что объектный файл не создался, однако появился файл листинга. Теперь зайдём в файл листинга, и посмотрим, отображается ли в нём ошибка :


```
lab7-2.lst      [----]  0 L:[179+10 189/226] *(11544/14546b) 0032 0x020      [*][X]
 4 0000002E D08BD0BE3A2000.....
 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
19                                     *****
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
23 00000106 A3[0A000000]            mov [B],eax ; запись преобразованного числа в 'B'
24                                     ; ----- Записываем 'A' в переменную 'max'
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ить 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 2.18: Отображение ошибки в листинге

Как видим, в листинге прописана ошибка

3 Выполнение задания для самостоятельной работы

Создадим файл для выполнения самостоятельной работы. Мой вариант - 1 :

A terminal window with a dark background. The prompt is 'rapavlichenko@rapavlichenko:~/work/arch-pc/lab07\$' and the command entered is 'touch task1.asm'.

```
rapavlichenko@rapavlichenko:~/work/arch-pc/lab07$ touch task1.asm
```

Рис. 3.1: Создание первого файла самостоятельной работы

Напишем код для выполнения задания. Код выглядит так:


```

GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab07/task1.asm
#include 'in_out.asm'
section .data
msg2 db "Наименьшее число: ",0h
A dd '17'
B dd '23'
C dd '45'
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'
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,msg2
call sprintf ; Вывод сообщения 'Наименьшее число: '
mov eax,[min]
call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход

```

Рис. 3.2: Код первого файла самостоятельной работы

Соберём, запустим его и посмотрим на результат:

```
rapavlichenko@rapavlichenko:~/work/arch-pc/lab07$ nasm -f elf task1.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab07$ ld -m elf_i386 -o task1 task1.o
rapavlichenko@rapavlichenko:~/work/arch-pc/lab07$ ./task1
Наименьшее число: 17
```

Рис. 3.3: Сборка и запуск программы первого задания самостоятельной работы, а также результат выполнения

Теперь создадим второй файл самостоятельной работы для второго задания :

```
rapavlichenko@rapavlichenko:~/work/arch-pc/lab07$ touch task2.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab07$
```

Рис. 3.4: Создание второго файла самостоятельной работы

Код будет выглядеть так :

```

GNU nano 7.2 /home/rapavlichenko/work/arch-pc/lab07/task2.asm Изменён
#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:
; Ввод x
mov eax, msg1
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
mov [x], eax
; Ввод a
mov eax, msg2
call sprint
mov ecx, a
mov edx, 80
call sread
mov eax, a
call atoi
mov [a], eax
; Проверка условий
mov eax, [x]
cmp eax, [a]
jnl less_than_a ; Переход, если x < a
jmp greater_equal_a ; Переход, если x >= a
less_than_a:
; Если x < a, то ans = a - 1
mov eax, [a]
sub eax, 1
jmp save_result
greater_equal_a:
; Если x >= a, то ans = 8
mov eax, 8
save_result:

```

Рис. 3.5: Код второго файла самостоятельной работы

```
; Сохранение результата в ans
mov [ans], eax
; Вывод результата
mov eax, msg3
call sprint
mov eax, [ans]
call iprintLF
call quit
```

Рис. 3.6: Код второго файла самостоятельной работы (продолжение)

Соберём исполняемый файл и запустим его :

```
rapavlichenko@rapavlichenko:~/work/arch-pc/lab07$ nasm -f elf task2.asm
rapavlichenko@rapavlichenko:~/work/arch-pc/lab07$ ld -m elf_i386 -o task2 task2.o
rapavlichenko@rapavlichenko:~/work/arch-pc/lab07$ ./task2
Введите X: 1
Введите A: 2
Ответ=1
```

Рис. 3.7: 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.