

Отчёт по лабораторной работе №7

Лобанова Екатерина Евгеньевна

Содержание

1	1. Цель работы	5
2	2. Порядок выполнения лабораторной работы	6
2.1	Реализация переходов в NASM	6
2.2	Изучение структуры файлы листинга	11
2.3	Задание для самостоятельной работы	14
3	Выводы	19

Список иллюстраций

2.1	Создание каталога и файла для лабораторной работы 7	6
2.2	Используя команду <code>nanp</code> вводим текст листинга в файл.	7
2.3	Компелируем файл.	7
2.4	Меняем программу	8
2.5	Скомпелировали файл и сморим на результат изменения программы	8
2.6	Вводим новый текст программы и проверяем корректность вывода .	9
2.7	Создаем новый файл	9
2.8	Заполняем файл	10
2.9	Запускаем файл и проверяем его работу для разных значений	11
2.10	Создаем файл листинга	12
2.11	Читаем файл	12
2.12	Удаляем операнд	13
2.13	Выполняем трансляцию файла	13
2.14	Заполняем файл	15
2.15	Рис 15	16
2.16	Код для вычисления выражения	17
2.17	Смотрим на результат работы кода	18

Список таблиц

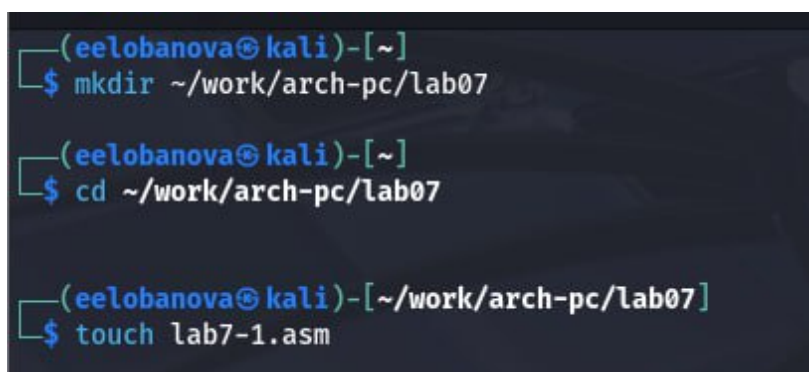
1 1. Цель работы

Изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

2.2. Порядок выполнения лабораторной работы

2.1 Реализация переходов в NASM

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



```
(eelobanova@kali)-[~]  
$ mkdir ~/work/arch-pc/lab07  
  
(eelobanova@kali)-[~]  
$ cd ~/work/arch-pc/lab07  
  
(eelobanova@kali)-[~/work/arch-pc/lab07]  
$ touch lab7-1.asm
```

Рисунок 2.1: Создание каталога и файла для лабораторной работы 7

Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`.

Вводим в файл lab7-1.asm текст программы из листинга 7.1. (рис. 2.2).

```
GNU nano 8.4 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
```

Рисунок 2.2: Используя команду nano вводим текст листинга в файл.

Копируем файл in_out.asm в новый каталог lab07. Создаем исполняемый файл и запускаем его, видим что результатом компиляции являются две строки Сообщение № 2 Сообщение № 3. (рис. 2.3).

```
(eelobanova@kali)-[~/work/arch-pc/lab07]
$ nasm -f elf lab7-1.asm

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ld -m elf_i386 -o lab7-1 lab7-1.o

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ./lab7-1
Сообщение № 2
Сообщение № 3
```

Рисунок 2.3: Компилируем файл.

Использование инструкции jmp _label2 меняет порядок исполнения 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.(рис. 2.4).

```

GNU nano 8.4                                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

```

Рисунок 2.4: Меняем программу

Снова создаем исполняемый файл и запустите его, сморим на изменения. Выводятся строки в другом порядке. (рис. 2.5).

```

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ nano lab7-1.asm

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ nasm -f elf lab7-1.asm

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ld -m elf_i386 -o lab7-1 lab7-1.o

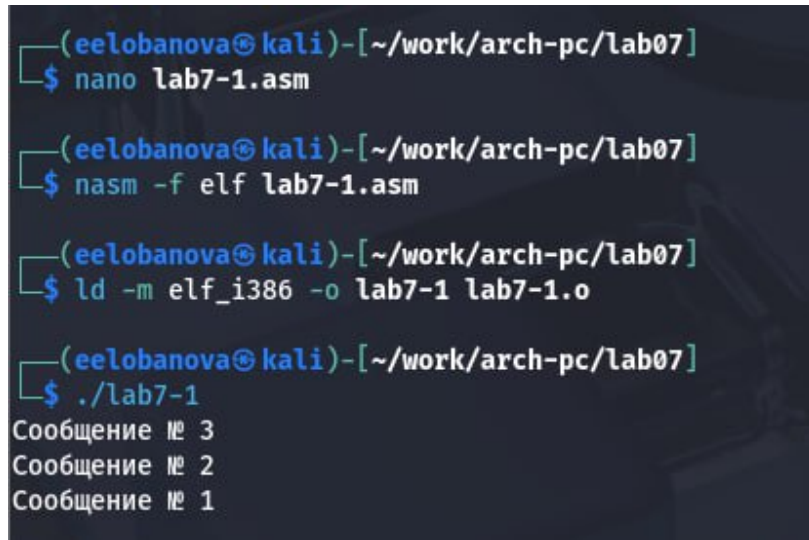
(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ./lab7-1
Сообщение № 2
Сообщение № 1

```

Рисунок 2.5: Скомпелировали файл и сморим на результат изменения программы

Меняем текст программы изменив инструкции jmp, чтобы вывод программы был

следующим: user@dk4n31:~\$./lab7-1 Сообщение № 3 Сообщение № 2 Сообщение № 1 user@dk4n31:~\$ (рис. 2.6).



```
(eelobanova@kali)-[~/work/arch-pc/lab07]
$ nano lab7-1.asm

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ nasm -f elf lab7-1.asm

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ld -m elf_i386 -o lab7-1 lab7-1.o

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Рисунок 2.6: Вводим новый текст программы и проверяем корректность вывода

Создаем с помощью команды touch файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. (рис. 2.7).



```
(eelobanova@kali)-[~/work/arch-pc/lab07]
$ touch lab7-2.asm
```

Рисунок 2.7: Создаем новый файл

Вводим текст листинга 7.3 в только что созданный файл.(рис. 2.8).

```
GNU nano 8.4 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'
; ----- 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]
```

Рисунок 2.8: Заполняем файл

Создаем исполняемый файл и проверяем его работу для разных значений B.(рис. 2.9).

```

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ nasm -f elf lab7-2.asm

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ld -m elf_i386 -o lab7-2 lab7-2.o

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ./lab7-2
Введите B: 3
Наибольшее число: 50

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ nasm -f elf lab7-2.asm

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ld -m elf_i386 -o lab7-2 lab7-2.o

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ./lab7-2
Введите B: 100
Наибольшее число: 100

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ nasm -f elf lab7-2.asm

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ld -m elf_i386 -o lab7-2 lab7-2.o

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ./lab7-2
Введите B: 40
Наибольшее число: 50

```

Рисунок 2.9: Запускаем файл и проверяем его работу для разных значений

2.2 Изучение структуры файлы листинга

Обычно `nasm` создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ `-l` и задав имя файла листинга

в командной строке. Создадим файл листинга для программы из файла lab7-2.asm.(рис. 2.10).

```
(eelobanova@kali)-[~]  
$ nasm -f elf -l lab7-2.lst lab7-2.asm
```

Рисунок 2.10: Создаем файл листинга

Открываем файл листинга lab7-2.lst с помощью любого текстового редактора, я использую nano.(рис. 2.11).

```
GNU nano 8.4 /home/eelobanova/work/arch-pc/lab07/lab7-2.lst  
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>  
22                                     <1> sprint:  
23 0000000F 52                         <1>     push    edx  
24 00000010 51                         <1>     push    ecx  
25 00000011 53                         <1>     push    ebx  
26 00000012 50                         <1>     push    eax  
27 00000013 E8E8FFFFFF                <1>     call    slen  
28                                     <1>  
29 00000018 89C2                      <1>     mov     edx, eax  
30 0000001A 58                        <1>     pop     eax  
31                                     <1>  
32 0000001B 89C1                      <1>     mov     ecx, eax  
33 0000001D BB01000000                <1>     mov     ebx, 1  
34 00000022 B804000000                <1>     mov     eax, 4  
35 00000027 CD80                      <1>     int     80h  
36                                     <1>  
37 00000029 5B                        <1>     pop     ebx  
38 0000002A 59                        <1>     pop     ecx  
39 0000002B 5A                        <1>     pop     edx  
40 0000002C C3                        <1>     ret  
41                                     <1>  
42                                     <1>  
^G Help      ^O Write Out ^F Where Is  ^K Cut       ^T Execute   ^C Location  
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line
```

Рисунок 2.11: Читаем файл

Открываем файл с программой lab7-2.asm и в любой инструкции с двумя

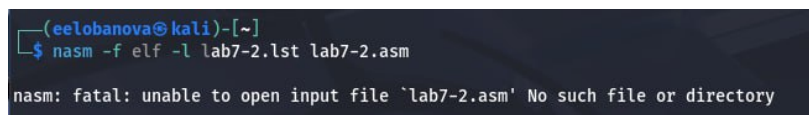
операндами удаляем один операнд. (рис. 2.12)



```
GNU nano 8.4 lab7-2.asm *
#include 'in_out.asm'
section .data
    msg1 db '',0h
    msg2 db '',0h
    A dd '20'
    C dd '50'
section .bss
    max resb 10
    B resb 10
section .text
    global _start
_start:
    mov eax,msg1
    call sprint
    mov ecx,B
    mov edx
    call sread
    mov eax,B
    call atoi
    mov [B],eax
    mov ecx,[A]
    mov [max],ecx
check_B:
    mov eax,max
    call atoi
    mov [max],eax
    mov ecx,[max]
    cmp ecx,[B]
    jg fin
    mov ecx,[B]
    mov [max],ecx
fin:
    mov eax, msg2
    call sprint
    mov eax,[max]
    call iprintLF
    call quit
```

Рисунок 2.12: Удаляем операнд

Запускаем программу(рис. 2.13)



```
(eelobanova@kali)-[~]
└─$ nasm -f elf -l lab7-2.lst lab7-2.asm
nasm: fatal: unable to open input file `lab7-2.asm' No such file or directory
```

Рисунок 2.13: Выполняем трансляцию файла

Программа выдает ошибку.

2.3 Задание для самостоятельной работы

1) Напишите программу нахождения наименьшей из 3 целочисленных переменных x , y и z . Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу

Вводим программу в новый файл (lab7-3.asm). (рис. 2.14).

```

GNU nano 8.4                                lab7-3.asm
#include 'in_out.asm'

SECTION .data
msg db 'Введите В: ',0
res db 'Наименьшее число: ',0
a dd 26
c dd 68
b dd 0

SECTION .text
GLOBAL _start

_start:
mov eax, msg
call sprint

mov ecx, b
mov edx, 10
call sread

mov eax, b
call atoi
mov [b], eax

mov eax, [a]
mov ebx, [b]
mov ecx, [c]

cmp eax, ebx
jle check_c
mov eax, ebx

check_c:
cmp eax, ecx
jle finish
mov eax, ecx

finish:
mov ebx, eax

mov eax, res
call sprint

mov eax, ebx
call iprintLF

call quit

```

Рисунок 2.14: Заполняем файл

Создаем исполняемый файл и запускаем его. (рис. 2.15)


```

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ nano lab7-3.asm

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ nasm -f elf -l lab7-3.lst lab7-3.asm

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ld -m elf_i386 -o lab7-3 lab7-3.o

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ./lab7-3
Введите В: 12
Наименьшее число: 12

```

Рисунок 2.15: Рис 15

Программа работает корректно.

Напишите программу, которая для введенных с клавиатуры значений x и y вычисляет значение заданной функции $f(x, y)$ и выводит результат вычислений. Вид функции $f(x, y)$ выбрать из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу для значений x и y из 7.6.

Вводим программу в новый файл (lab7-4.asm). (рис. 2.15)


```

GNU nano 8.4                                lab7-4.asm
#include 'in_out.asm'

SECTION .data
msg_x db 'Введите x: ',0
msg_a db 'Введите a: ',0
res db 'Результат: ',0
x dd 0
a dd 0

SECTION .text
GLOBAL _start

_start:
    mov eax, msg_x
    call sprint
    mov ecx, x
    mov edx, 10
    call sread
    mov eax, x
    call atoi
    mov [x], eax

    mov eax, msg_a
    call sprint
    mov ecx, a
    mov edx, 10
    call sread
    mov eax, a
    call atoi
    mov [a], eax

    mov eax, [a]
    cmp eax, 8
    jl less_than_8

    mov eax, [a]
    mov ebx, [x]
    mul ebx
    jmp print_result

less_than_8:
    mov eax, [a]
    add eax, 8

print_result:
    mov ebx, eax
    mov eax, res
    call sprint

```

Рисунок 2.16: Код для вычисления выражения

Создаем исполняемый файл и проверяем работу. (рис. 2.17)

```
(eelobanova@kali)-[~/work/arch-pc/lab07]
$ nano lab7-4.asm

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ nasm -f elf -l lab7-4.lst lab7-4.asm

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ld -m elf_i386 -o lab7-4 lab7-4.o

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ./lab7-4
Введите x: 3
Введите a: 4
Результат: 12

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ nasm -f elf -l lab7-4.lst lab7-4.asm

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ld -m elf_i386 -o lab7-4 lab7-4.o

(eelobanova@kali)-[~/work/arch-pc/lab07]
$ ./lab7-4
Введите x: 2
Введите a: 9
Результат: 18
```

Рисунок 2.17: Смотрим на результат работы кода

3 Выводы

В ходе выполнения лабораторной работы были изучены команды условного и безусловного переходов в ассемблере NASM. Приобретены практические навыки написания программ с использованием переходов, что позволяет управлять порядком выполнения инструкций. Также была изучена структура файла листинга и его назначение для отладки программ. В рамках самостоятельной работы успешно написаны программы для нахождения минимального значения из трёх переменных и вычисления значения функции с условными переходами. Полученные знания позволяют эффективно разрабатывать программы с нелинейной структурой управления.