

Отчёт по лабораторной работе №7

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

Югай Александр Витальевич

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
3.1	Реализация переходов в NASM	7
3.2	Изучение структуры файлы листинга	14
3.3	Задание для самостоятельной работы	16
4	Выводы	22

Список иллюстраций

3.1	Создаем каталог с помощью команды <code>mkdir</code> и файл с помощью команды <code>touch</code>	7
3.2	Заполняем файл	8
3.3	Запускаем файл и смотрим на его работу	8
3.4	Изменяем файл	9
3.5	Запускаем файл и смотрим на его работу	10
3.6	Редактируем файл	11
3.7	Проверяем, сошелся ли наш вывод с данным в условии выводом .	12
3.8	Создаем файл командой <code>touch</code>	12
3.9	Заполняем файл	13
3.10	Смотрим на работу программ	14
3.11	Создаем файл листинга	14
3.12	Изучаем файл	15
3.13	Удаляем операндум из файла	15
3.14	Транслируем файл	16
3.15	Изучаем файл с ошибкой	16
3.16	Создаем файл командой <code>touch</code>	17
3.17	Пишем программу	18
3.18	Смотрим на работу программы	19
3.19	Создаем файл командой <code>touch</code>	19
3.20	Пишем программу	20
3.21	Проверяем работу программы	21
3.22	Проверяем работу программы	21

Список таблиц

1 Цель работы

Освоить условного и безусловного перехода. Ознакомиться с назначением и структурой файла листинга.

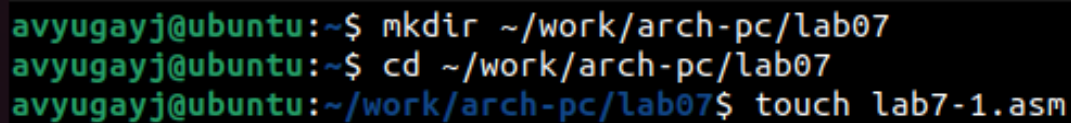
2 Задание

Написать программы для решения системы выражений.

3 Выполнение лабораторной работы

3.1 Реализация переходов в NASM

Создаем каталог для программ Лаб7, и в нем создаем файл



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

Рис. 3.1: Создаем каталог с помощью команды `mkdir` и файл с помощью команды `touch`

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 7.1

```

#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. 3.2: Заполняем файл

Создаем исполняемый файл и запускаем его

```

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

```

Рис. 3.3: Запускаем файл и смотрим на его работу

Снова открываем файл для редактирования и изменяем его в соответствии с


```
%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. 3.4: Изменяем файл

Создаем исполняемый файл и запускаем его

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

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

Рис. 3.5: Запускаем файл и смотрим на его работу

Снова открываем файл для редактирования и изменяем его, чтобы произошел данный вывод

```

#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. 3.6: Редактируем файл

Создаем исполняемый файл и запускаем его

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

Рис. 3.7: Проверяем, сошелся ли наш вывод с данным в условии выводом

Создаем новый файл

```
Сообщение № 1
avyugayj@ubuntu:~/work/arch-pc/lab07$ touch lab7-2.asm
avyugayj@ubuntu:~/work/arch-pc/lab07$
```

Рис. 3.8: Создаем файл командой touch

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 7.3

```

#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 'А' в переменную 'max'

```

Рис. 3.9: Заполняем файл

Создаем исполняемый файл и проверяем его работу, вводя разные значения В

```

avyugayj@ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
avyugayj@ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
avyugayj@ubuntu:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 5
Наибольшее число: 50
avyugayj@ubuntu:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 60
Наибольшее число: 60
avyugayj@ubuntu:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 45
Наибольшее число: 50
avyugayj@ubuntu:~/work/arch-pc/lab07$ █

```

Рис. 3.10: Смотрим на работу программ

3.2 Изучение структуры файлы листинга

Создаем файл листинга для программы lab7-2.asm

```

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

```

Рис. 3.11: Создаем файл листинга

Открываем файл листинга с помощью команды `mcedit` и изучаем его

```

1      <1> include "in_out.asm"
2      <1> ;----- strlen -----
3      <1> ; Функция вычисления длины сообщения
4      <1> strlen:
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> jnp     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> ;----- sprintf -----
21     <1> ; Функция печати сообщения
22     <1> ; входные данные: mov eax,<message>
23     <1> sprintf:
24     0000000F 52      <1> push    edx
25     00000010 51      <1> push    ecx
26     00000011 53      <1> push    ebx
27     00000012 50      <1> push    eax
28     00000013 E8E8FFFFFF <1> call    strlen
29     <1>
30     00000018 89C2     <1> mov     edx, eax
31     0000001A 58      <1> pop     eax
32     <1>
33     0000001B 89C1     <1> mov     ecx, eax
34     0000001D B801000000 <1> mov     ebx, 1
35     00000022 B804000000 <1> mov     eax, 4
36     00000027 CD80     <1> int     80h
37     <1>
38     00000029 5B      <1> pop     ebx
39     0000002A 59      <1> pop     ecx
40     0000002B 5A      <1> pop     edx
41     0000002C C3      <1> ret
42     <1>
43     <1>

```

Рис. 3.12: Изучаем файл

Строка 33: 0000001D-адрес в сегменте кода, B801000000-машинный код, mov ebx,1-присвоение переменной ecx значения 1.

Строка 34: 00000022-адрес в сегменте кода, B804000000-машинный код, mov eax,4-присвоение переменной eax значения 4.

Строка 35 00000027-адрес в сегменте кода, CD80-машинный код, int 80h-вызов ядра.

Открываем файл и удаляем один операндум

```

; ----- Ввод 'В'
mov ecx,B
mov edx
call sread

```

Рис. 3.13: Удаляем операндум из файла

Транслируем с получением файла листинга

```

avyugayj@ubuntu:~/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
avyugayj@ubuntu:~/work/arch-pc/lab07$ ls
in_out.asm  lab7-1  lab7-1.asm  lab7-1.o  lab7-2  lab7-2.asm  lab7-2.lst

```

Рис. 3.14: Транслируем файл

При трансляции файла, выдается ошибка, но создаются исполнительный файл lab7-2 и lab7-2.lst

Снова открываем файл листинга и изучаем его

```

 99 0000007C 83F900      <1>    cmp     ecx, 0...
100 0000007F 75F2       <1>    jnz     printLoop..
101                                     <1>
102 00000081 5E         <1>    pop     esi...
103 00000082 5A         <1>    pop     edx....
104 00000083 59         <1>    pop     ecx...
105 00000084 58         <1>    pop     eax.....
106 00000085 C3         <1>    ret
107                                     <1>
108                                     <1>
109                                     <1> ;----- iprintf -----
110 <1> ; функция вывода на экран чисел в формате ASCII
111 <1> ; входные данные: mov eax,<int>
112 <1> iprintf:
113 00000086 E8C9FFFFFF <1>    call    iprintf.....
114                                     <1>
115 0000008B 50         <1>    push    eax.....
116 0000008C B80A000000 <1>    mov     eax, 0Ah.....
117 00000091 50         <1>    push    eax.....
118 00000092 89E0      <1>    mov     eax, esp.....
119 00000094 E876FFFFFF <1>    call    sprintf.....
120 00000099 58         <1>    pop     eax.....
121 0000009A 58         <1>    pop     eax.....
122 0000009B C3         <1>    ret
123                                     <1>
124                                     <1> ;----- atoi -----
125 <1> ; функция преобразования ascii-код символа в целое число
126 <1> ; входные данные: mov eax,<int>
127 <1> atoi:
128 0000009C 53         <1>    push    ebx.....
129 0000009D 51         <1>    push    ecx.....
130 0000009E 52         <1>    push    edx.....
131 0000009F 56         <1>    push    esi.....
132 000000A0 89C6      <1>    mov     esi, eax.....
133 000000A2 B800000000 <1>    mov     eax, 0.....
134 000000A7 B900000000 <1>    mov     ecx, 0.....
135                                     <1>
136                                     <1> .multiplyLoop:
137 000000AC 31DB      <1>    xor     ebx, ebx.....
138 000000AE 8A1C0E   <1>    mov     bl, [esi+ecx]
139 000000B1 80FB30   <1>    cmp     bl, 48
140 000000B4 7C14     <1>    jl      .finished.
141 000000B6 80FB39   <1>    cmp     bl, 57

```

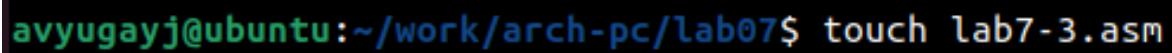
Рис. 3.15: Изучаем файл с ошибкой

3.3 Задание для самостоятельной работы

Вариант №3

Напишите программу нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом,

полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу. Создаем новый файл



```
avyugayj@ubuntu:~/work/arch-pc/lab07$ touch lab7-3.asm
```

Рис. 3.16: Создаем файл командой touch

Открываем его и пишем программу, которая выберет наименьшее число из трех (2 числа уже в программе, 3е вводится из консоли)

```

#include 'in_out.asm'
section .data
msg1 db 'Введите B: ',0h
msg2 db "Наименьшее число: ",0h
A dd '94'
C dd '58'
section .bss
min resb 10
B resb 10
section .text
global _start
_start:
mov eax,msg1
call sprint
mov ecx,B
mov edx,10
call sread
mov eax,B
call atoi
mov [B],eax
mov ecx,[A]
mov [min],ecx
cmp ecx,[C]
jl check_B
mov ecx,[C]
mov [min],ecx
check_B:
mov eax,min
call atoi
mov [min],eax
mov ecx,[min]
cmp ecx,[B]
jl fin
mov ecx,[B]
mov [min],ecx
fin:
mov eax, msg2
call sprint
mov eax,[min]
call iprintfLF
call quit

```

Рис. 3.17: Пишем программу

Транслируем файл и смотрим на работу программы

```
avyugayj@ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-3.asm
avyugayj@ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
avyugayj@ubuntu:~/work/arch-pc/lab07$ ./lab7-3
Введите В: 5
Наименьшее число: 5
```

Рис. 3.18: Смотрим на работу программы

Напишите программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений. Вид функции $f(x)$ 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 a из 7.6.

Создаем новый файл

```
avyugayj@ubuntu:~/work/arch-pc/lab07$ touch lab7-4.asm
```

Рис. 3.19: Создаем файл командой touch

Открываем его и пишем программу, которая решит систему уравнений, при данных, введенных в консоль

```

%include 'in_out.asm'
section .data
msg1 DB 'Введите x: ',0h
msg2 DB "Введите a: ",0h
otv: DB 'F(x)=',0h
section .bss
x: RESB 80
a: RESB 80
res: 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, 3
je x_is_3
mov eax, [a]
add eax, 1
jmp calc_res
x_is_3:
mov eax, [x]
imul eax,3
calc_res:
mov [res],eax
fin:
mov eax,otv

```

Рис. 3.20: Пишем программу

Транслируем файл и проверяем его работу при $x=3$ и $a=4$

```
avyugayj@ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-4.asm
avyugayj@ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-4 lab7-4.o
avyugayj@ubuntu:~/work/arch-pc/lab07$ ./lab7-4
Введите x: 3
Введите a: 4
F(x)=9
```

Рис. 3.21: Проверяем работу программы

Теперь проверяем его работу при $x=1$ и $a=4$

```
avyugayj@ubuntu:~/work/arch-pc/lab07$ ./lab7-4
Введите x: 1
Введите a: 4
F(x)=5
```

Рис. 3.22: Проверяем работу программы

4 Выводы

Мы познакомились с структурой файла листинга, изучили команды условного и безусловного перехода.