

Лабораторная работа №7

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

Глобин Никита Анатольевич

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	9
3.3	Задание для самостоятельной работы	12
4	Выводы	16
	Список литературы	17

Список иллюстраций

3.1 photo 1	7
3.2 photo 2	7
3.3 photo 3	8
3.4 photo 4	8
3.5 photo 5	8
3.6 photo 6	9
3.7 photo 7	9
3.8 photo 8	10
3.9 photo 9	10
3.10 photo 10	11
3.11 photo 11	11
3.12 photo 12	11
3.13 photo 13	11
3.14 photo 14	12
3.15 photo 15	12
3.16 photo 16	13
3.17 photo 17	13
3.18 photo 18	14
3.19 photo 19	15

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

Реализация переходов в NASM

Изучение структуры файлы листинга

Задание для самостоятельной работы

3 Выполнение лабораторной работы

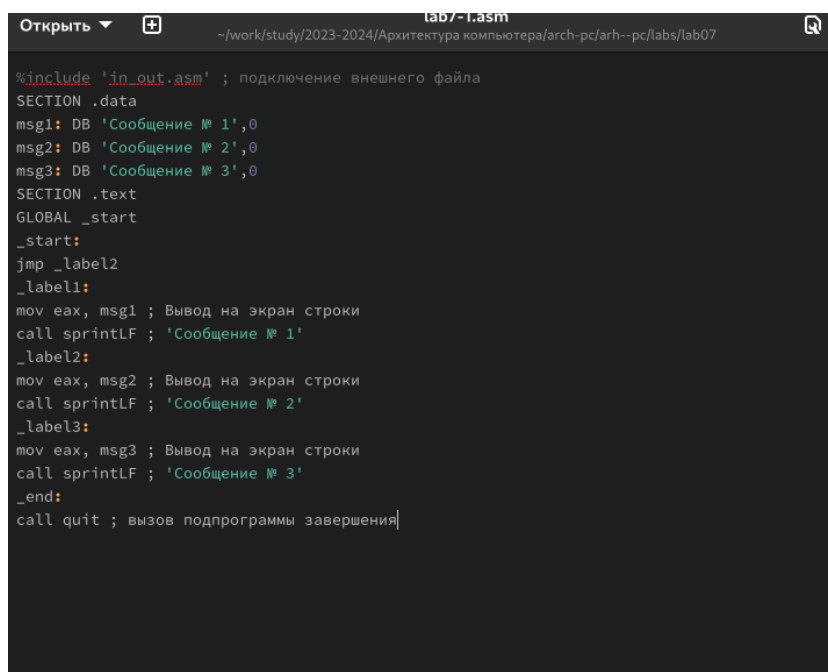
3.1 Реализация переходов в NASM

1. Переходим в каталог lab07 и создаём файл lab7-1.asm (рис. 3.1).

```
dodo@vbox:~$ cd work/study/2023-2024/Архитектура\ компьютера/arch-pc/arh--pc/labs/lab07/
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ ls
presentation  report
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ touch lab7-1.asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$
```

Рис. 3.1: photo 1

2. переписываем программу из ТУИС в наш файл (рис. 3.2).



```
Открыть ▾ + lab7-1.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07

%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: photo 2

3. компилируем и запускаем этот файл (рис. 3.3).

```
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ nasm -f elf lab7-1.asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$
```

Рис. 3.3: photo 3

4. теперь переписываем программу как показано в лабораторной работе (рис. 3.4).

```
%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: photo 4

5. компилируем и запускаем этот файл (рис. 3.5).

```
Сообщение № 1
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ nasm -f elf lab7-1.asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$
```

Рис. 3.5: photo 5

6. создаём файл lab7-2.asm и пишем в нём код новой программы(рис. 3.6).


```

%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]
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [max],ecx
; ----- Вывод результата
fin:

```

Рис. 3.6: photo 6

7. компилируем и запускаем этот файл (рис. 3.7).

```

dodogvbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arth--pc/labs/lab07$ touch lab7-2.asm
dodogvbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arth--pc/labs/lab07$ nasm -f elf lab7-2.asm
dodogvbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arth--pc/labs/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
dodogvbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arth--pc/labs/lab07$ ./lab7-2
Введите B: 10
Наибольшее число: 50
dodogvbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arth--pc/labs/lab07$

```

Рис. 3.7: photo 7

3.2 Изучение структуры файлы листинга

1. создаём файл листинга через команду `nasm -f elf -l lab7-2.lst lab7-2.asm` (рис. 3.8).

```

Наибольшее число: 50
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arch--pc/labs/lab07$ nasm -f elf -l lab7-2.lst lab7-2.
asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arch--pc/labs/lab07$

```

Рис. 3.8: photo 8

2. открываем этот файл командой `mcedit lab7-2.lst` (рис. 3.9).

```

lab7-2.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          00000003 803800     <1> nextchar:-----
9          00000006 7403       <1> cmp byte [eax], 0...
10         00000008 40        <1> jz finished-----
11         00000009 EBF8      <1> inc eax-----
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 E8E8FFFF   <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 8B01000000 <1> mov ebx, 1-----
34         00000022 8B04000000 <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>-----
43         <1> ;----- sprintf -----
44         <1> ; Функция печати сообщения с переводом строки
45         <1> ; входные данные: mov eax, <message>
46         <1> sprintf:-----

```

Рис. 3.9: photo 9

тут в строке: (рис. ??). 1. Строка 51: `push eax` Содержимое регистра `eax` помещается в стек. Это значение необходимо для последующей операции вывода. 2. Строка 53: `call sprint` Вызывается функция `sprint`. эта функция, отвечает за вывод строки на экран. Аргументы для этой функции были предварительно подготовлены и помещены в стек. 3. Строка 55: `pop eax` Значение, которое было ранее помещено в стек на строке 51, извлекается и помещается обратно в регистр `eax`. Это делается для того, чтобы восстановить исходное состояние регистра после выполнения функции `sprint`.

```

47 00000020 E8DFFFFFFF <1> call sprint
48                                     <1>
49 00000032 50                                     <1> push eax
50 00000033 B80A000000 <1> mov eax, 0AH
51 00000038 50                                     <1> push eax
52 00000039 89E0 <1> mov eax, esp
53 0000003B E8CFFFFFFF <1> call sprint
54 00000040 58                                     <1> pop eax
55 00000041 58                                     <1> pop eax
56 00000042 C3                                     <1> ret
57

```

Рис. 3.10: photo 10

3. открываем lab7-2.asm и попробуем намеренно допустить ошибку в нашем коде, убрав у команды move 1 операнд (рис. ??).

```

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. 3.11: photo 11

4. компилируем и запускаем этот файл (рис. 3.12).

```

dodo@vbox: ~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ nasm -f elf lab7-2.asm
lab7-2.asm:18: error: invalid combination of opcode and operands
dodo@vbox: ~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$

```

Рис. 3.12: photo 12

5. открываем листинговый файл и видим что там тоже сообщается об ошибке (рис. 3.13).

```

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 символа 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 числа в 'B'
24                                     ; ----- Записываем 'A' в переменную 'max'
25 0000010B 8901[00000000] mov max,A

```

Рис. 3.13: photo 13

3.3 Задание для самостоятельной работы

1. Создадим файл для выполнения самостоятельной работы. Мой вариант - 9 (рис. 3.14).

```
dodogvbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ touch task1v9.asm
dodogvbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$
```

Рис. 3.14: photo 14

Напишем код для выполнения задания. Код выглядит так (рис. 3.15).

```
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'
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 sprint ; Вывод сообщения 'Наименьшее число: '
mov eax,[min]
call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход
```

Рис. 3.15: photo 15

2. Соберём, запустим его и посмотрим на результат (рис. 3.16).

```

dodo@vbox: ~/work/study/2023-2024/Архитектура компьютера/arch-pc/arch--pc/labs/lab07$ touch task1v9.asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arch--pc/labs/lab07$ nasm -f elf task1v9.asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arch--pc/labs/lab07$ ld -m elf_i386 -o task1v9 task1v9.o
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arch--pc/labs/lab07$ ./task1v9
Наименьшее число: 15
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arch--pc/labs/lab07$ 

```

Рис. 3.16: photo 16

3. Теперь создадим второй файл самостоятельной работы для второго задания (рис. 3.17).

```

dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arch--pc/labs/lab07$ touch task2v9.asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arch--pc/labs/lab07$ 

```

Рис. 3.17: photo 17

4. Код будет выглядеть так (рис. 3.18).

```

#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:
mov eax, [a]
add eax, [x]

ansv:
mov [ans],eax
mov eax,msg3
call sprint

```

Рис. 3.18: photo 18

5. Соберём исполняемый файл и запустим его (рис. 3.19).

```

dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ touch task2v9.asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ nasm -f elf task2v9.asm
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ ld -m elf_i386 -o task2v9 task2v9.o
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$ ./task2v9
Введите X: 24
Введите A: 98
Ответ=122
dodo@vbox:~/work/study/2023-2024/Архитектура компьютера/arch-pc/arh--pc/labs/lab07$
```

Рис. 3.19: photo 19

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.

Список литературы