

# **Отчет по лабораторной работе №7**

**Дисциплина: архитектура компьютера**

Пихтовникова Алёна Владимировна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
4.1	Реализация переходов в NASM . . . . .	7
4.2	Изучение структуры файла листинга . . . . .	11
4.3	Задания для самостоятельной работы . . . . .	12
<b>5</b>	<b>Выводы</b>	<b>20</b>
	<b>Список литературы</b>	<b>21</b>

## Список иллюстраций

4.1	Создание каталога и файла для программы . . . . .	7
4.2	Сохранение программы . . . . .	8
4.3	Запуск программы . . . . .	8
4.4	Изменение программы . . . . .	9
4.5	Запуск измененной программы . . . . .	9
4.6	Изменение программы . . . . .	10
4.7	Проверка изменений . . . . .	10
4.8	Сохранение новой программы . . . . .	10
4.9	Проверка программы из листинга . . . . .	11
4.10	Проверка файла листинга . . . . .	11
4.11	Удаление операнда из программы . . . . .	12
4.12	Просмотр ошибки в файле листинга . . . . .	12
4.13	Первая программа самостоятельной работы . . . . .	13
4.14	Проверка работы первой программы . . . . .	16
4.15	Вторая программа самостоятельной работы . . . . .	17
4.16	Проверка работы второй программы . . . . .	19

# 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

1. Реализация переходов в NASM
2. Изучение структуры файлов листинга
3. Самостоятельное написание программ по материалам лабораторной работы

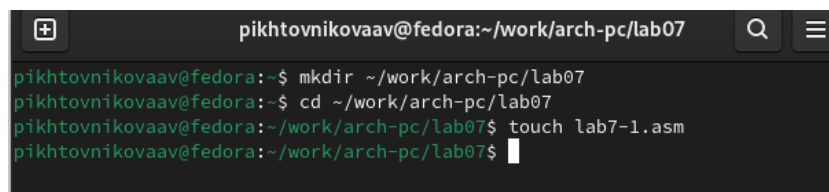
### **3 Теоретическое введение**

Для реализации ветвлений в ассемблере используются так называемые команды передачи управления или команды перехода. Можно выделить 2 типа переходов: • условный переход – выполнение или не выполнение перехода в определенную точку программы в зависимости от проверки условия. • безусловный переход – выполнение передачи управления в определенную точку программы без каких-либо условий.

## 4 Выполнение лабораторной работы

### 4.1 Реализация переходов в NASM

Создаю каталог для программ лабораторной работы №7 (рис. 4.1).

A terminal window with a dark background. The title bar shows the user 'pikhtovnikovaav@fedora' and the current directory '~/work/arch-pc/lab07'. The terminal contains four lines of text: a prompt followed by 'mkdir ~/work/arch-pc/lab07', a prompt followed by 'cd ~/work/arch-pc/lab07', a prompt followed by 'touch lab7-1.asm', and a final prompt with a cursor. The text is in a monospaced font, with the prompt in green and the commands in white.

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

Рис. 4.1: Создание каталога и файла для программы

Копирую код из листинга в файл будущей программы. (рис. 4.2).

```

%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. 4.2: Сохранение программы

При запуске программы я убедилась в том, что безусловный переход действительно изменяет порядок выполнения инструкций (рис. 4.3).

```

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

```

Рис. 4.3: Запуск программы

Изменяю программу таким образом, чтобы поменялся порядок выполнения функций (рис. 4.4).



```

%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
call sprintLF
jmp _end

_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 2'
jmp _label1

_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 3'

end:

```

Рис. 4.4: Изменение программы

Запускаю программу и проверяю, что примененные изменения верны (рис. 4.5).

```

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

```

Рис. 4.5: Запуск измененной программы

Теперь изменяю текст программы так, чтобы все три сообщения вывелись в обратном порядке (рис. 4.6).

```

_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
jmp _label2

```

Рис. 4.6: Изменение программы

Работа выполнена корректно, программа в нужном мне порядке выводит сообщения (рис. 4.7).

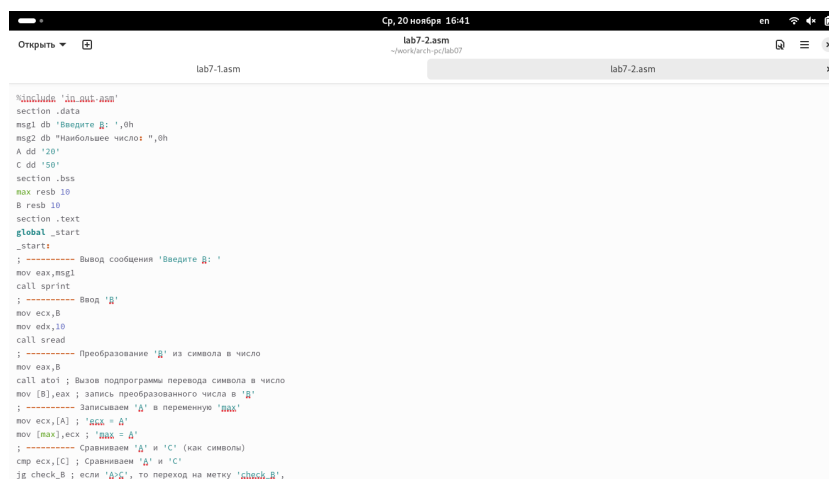
```

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

```

Рис. 4.7: Проверка изменений

Создаю новый рабочий файл и вставляю в него код из следующего листинга (рис. 4.8).



```

lab7-2.asm
~work/arch-pc/lab07

#include 'in_out.asm'
section .data
msg1 db "Введите D: ",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 D: "
mov eax,msg1
call sprintf
; ----- Ввод "D"
mov ecx,B
mov edx,10
call read
; ----- Преобразование "D" из символа в число
mov eax,B
call atoi ; Вывод подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в "B"
; ----- Записываем "D" в переменную "max"
mov ecx,[A] ; "max" = "D"
mov [max],ecx ; "max" = "D"
; ----- Сравниваем "D" и "C" (как символы)
cmp ecx,[C] ; Сравниваем "D" и "C"
jg check_B ; если "D">"C", то переход на метку 'check_B',

```

Рис. 4.8: Сохранение новой программы

Программа выводит значение переменной с максимальным значением, проверяя работу программы с разными входными данными (рис. 4.9).

```

pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ touch lab7-2.asm
pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2 lab7-2.o
lab7-2: fatal: more than one input file specified: lab7-2.o

pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 25
Наибольшее число: 50
pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 10
Наибольшее число: 50
pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 60
Наибольшее число: 60
pikhtovnikovaav@fedora:~/work/arch-pc/lab07$

```

Рис. 4.9: Проверка программы из листинга

## 4.2 Изучение структуры файла листинга

Создаю файл листинга с помощью флага -l команды nasm и открываю его с помощью текстового редактора mousepad (рис. 4.10).

```

~/work/arch-pc/lab07/lab7-2.lst - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь
1      1      <1>  ;include 'in_out.asm'
2      2      <1>  ;----- slen -----
3      3      <1>  ; функция вычисления длины сообщения
4      4      <1>  slen:
5      5      00000000 53      <1>  push    ebx
6      6      00000001 89C3      <1>  mov     ebx, eax
7      7      <1>
8      8      <1> nextchar:
9      9      00000003 803800      <1>  cmp     byte [eax], 0
10     10      00000006 7403      <1>  jz      finished
11     11      00000008 40      <1>  inc     eax
12     12      11 00000009 EBF8      <1>  jmp     nextchar
13     13      <1>
14     14      <1> finished:
15     15      14 0000000B 2008      <1>  sub     eax, ebx
16     16      15 0000000D 58      <1>  pop     ebx
17     17      16 0000000E C3      <1>  ret
18     18      <1>
19     19      <1> ;----- sprint -----
20     20      <1> ; функция печати сообщения
21     21      <1> ; входные данные: mov eax, <message>
22     22      <1> sprint:
23     23      23 0000000F 52      <1>  push    edx
24     24      24 00000010 51      <1>  push    ecx
25     25      25 00000011 53      <1>  push    ebx
26     26      26 00000012 50      <1>  push    eax
27     27      27 00000013 EBECFFFFFF      <1>  call    slen
28     28      <1>
29     29      29 00000018 89C2      <1>  mov     edx, eax
30     30      0000001A 58      <1>  pop     eax

```

Рис. 4.10: Проверка файла листинга

Первое значение в файле листинга - номер строки, и он может вовсе не совпадать с номером строки изначального файла. Второе вхождение - адрес, смещение машинного кода относительно начала текущего сегмента, затем непосредственно идет сам машинный код, а заключает строку исходный текст программы с комментариями.

Удаляю один операнд из случайной инструкции, чтобы проверить поведение файла листинга в дальнейшем (рис. 4.11).

```
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, |
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:
mov eax, msg2
call sprint
mov eax,[max]
call iprintLF
call quit
```

Рис. 4.11: Удаление операнда из программы

В новом файле листинга показывает ошибку, которая возникла при попытке трансляции файла. Никакие выходные файлы при этом помимо файла листинга не создаются. (рис. 4.12).

```
33                                     check_B:
34                                     max: eax,
34                                     error: invalid combination of opcode and operands
35 00000000 00000000
```

Рис. 4.12: Просмотр ошибки в файле листинга

## 4.3 Задания для самостоятельной работы

Во время 6 лабораторной работы я получил 2 варианта. Возвращаю операнд к функции в программе и изменяю ее так, чтобы она выводила переменную с

наименьшим значением (рис. 4.13).

```
%include 'in_out.asm'
SECTION .data
msg1 db 'Введите B: ', 0h
msg2 db 'Наименьшее число: ', 0h
A dd '24'
C dd '15'
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]
jg check_B
mov ecx, [C]
```

Рис. 4.13: Первая программа самостоятельной работы

Код первой программы:

```
%include 'in_out.asm'
```

## SECTION .data

```
A1 DB 'Введите число A: ',0h
B1 DB 'Введите число B: ',0h
C1 DB 'Введите число C: ',0h
otv DB 'Наименьшее число: ',0h
```

## SECTION .bss

```
min RESB 20
A RESB 20
B RESB 20
C RESB 20
```

## SECTION .text

```
GLOBAL _start
```

```
_start:
```

```
mov eax,A1
call sprint
```

```
mov ecx,A
mov edx,20
call sread
```

```
mov eax, A
call atoi
mov [A],eax
```

```
xor eax,eax
```

```
mov eax,B1
```

```
call sprint
```

```
mov ecx,B
```

```
mov edx,20
```

```
call sread
```

```
mov eax,B
```

```
call atoi
```

```
mov [B],eax
```

```
xor eax,eax
```

```
mov ecx, [A]
```

```
mov [min],ecx
```

```
mov ecx,[min]
```

```
cmp ecx,[B]
```

```
jnl check_C
```

```
mov ecx, [B]
```

```
mov [min],ecx
```

```
check_C:
```

```
mov eax,C1
```

```
call sprint
```

```
mov ecx,C
```

```
mov edx,10
```

```
call sread
```

```

mov eax,C
call atoi
mov [C],eax

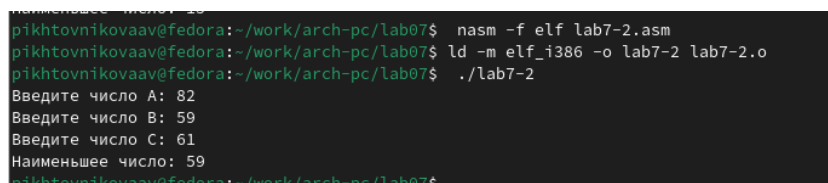
xor eax,eax

mov ecx,[min]
cmp ecx,[C]
jl fin
mov ecx,[C]
mov [min],ecx

fin:
mov eax, otv
call sprint
mov eax, [min]
call iprintLF
call quit

```

Проверяю корректность написания первой программы (рис. 4.14).



```

pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите число A: 82
Введите число B: 59
Введите число C: 61
Наименьшее число: 59
pikhtovnikovaav@fedora:~/work/arch-pc/lab07$

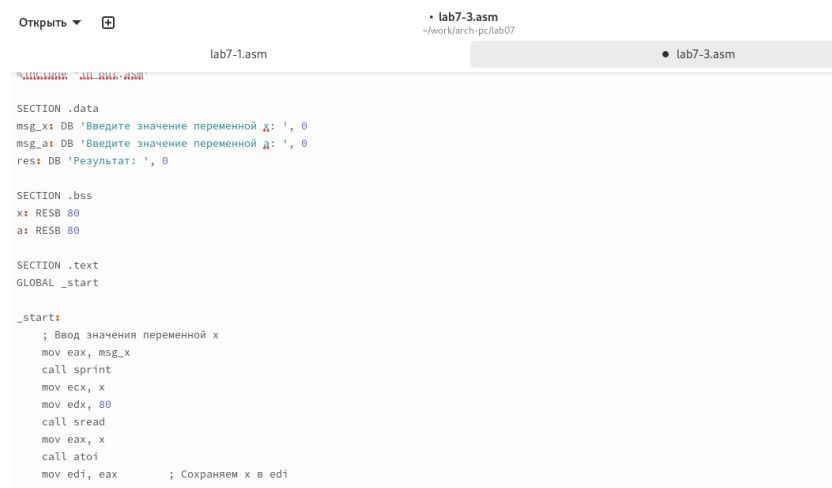
```

Рис. 4.14: Проверка работы первой программы

Пишу программу, которая будет вычислять значение заданной функции согласно моему варианту для введенных с клавиатуры переменных а и х (рис.



4.15).




```
Открыть ▾  lab7-3.asm  
~/work/arch-projects/lab07 lab7-1.asm lab7-3.asm  
%include 'in_out.asm'  
  
SECTION .data  
msg_x: DB 'Введите значение переменной x: ', 0  
msg_a: DB 'Введите значение переменной a: ', 0  
res: DB 'Результат: ', 0  
  
SECTION .bss  
x: RESB 80  
a: RESB 80  
  
SECTION .text  
GLOBAL _start  
  
_start:  
    ; Ввод значения переменной x  
    mov eax, msg_x  
    call sprint  
    mov ecx, x  
    mov edx, 80  
    call sread  
    mov eax, x  
    call atoi  
    mov edi, eax    ; Сохраняем x в edi
```

Рис. 4.15: Вторая программа самостоятельной работы

Код второй программы:

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg_x: DB 'Введите значение переменной x: ', 0
```

```
msg_a: DB 'Введите значение переменной a: ', 0
```

```
res: DB 'Результат: ', 0
```

```
SECTION .bss
```

```
x: RESB 80
```

```
a: RESB 80
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
    ; Ввод значения переменной x
```

```

mov eax, msg_x
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
mov edi, eax          ; Сохраняем x в edi

; Ввод значения переменной a
mov eax, msg_a
call sprint
mov ecx, a
mov edx, 80
call sread
mov eax, a
call atoi
mov esi, eax          ; Сохраняем a в esi

; Сравнение x и a
cmp edi, esi          ; Сравниваем x (edi) и a (esi)
jl calculate_a        ; Если x < a, переходим к calculate_a

; Если x >= a, выполняем x - 1
dec edi              ; x - 1
jmp print_result

calculate_a:
dec esi              ; a - 1

```

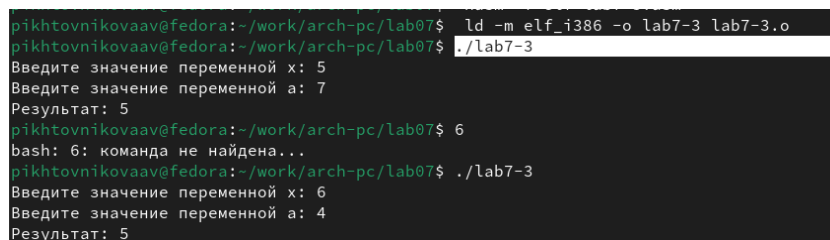
print\_result:

```
mov eax, res
call sprint          ; Вывод сообщения 'Результат: '

mov eax, edi         ; Загружаем результат в eax (либо a-1, либо x-1)
call iprintLF        ; Выводим результат

call quit            ; Выход из программы
```

Транслирую и компоную файл, запускаю и проверяю работу программы для различных значений а и х (рис. 4.16).



```
pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ ./lab7-3
Введите значение переменной x: 5
Введите значение переменной a: 7
Результат: 5
pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ 6
bash: 6: команда не найдена...
pikhtovnikovaav@fedora:~/work/arch-pc/lab07$ ./lab7-3
Введите значение переменной x: 6
Введите значение переменной a: 4
Результат: 5
```

Рис. 4.16: Проверка работы второй программы

## **5 Выводы**

При выполнении лабораторной работы я изучила команды условных и безусловных переходов, а также приобрела навыки написания программ с использованием переходов, познакомилась с назначением и структурой файлов листинга.

# Список литературы

1. Курс на ТУИС
2. Лабораторная работа №7
3. Программирование на языке ассемблера NASM Столяров А. В.