

# **ЛаБортаторная работа №7**

**Дисциплина: Архитектура компьютера**

Первий Анастасия Андреевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
4.1	Реализация переходов в NASM . . . . .	9
4.2	Изучение структуры файла листинга . . . . .	14
4.3	Самостоятельная работа . . . . .	17
<b>5</b>	<b>Выводы</b>	<b>22</b>
<b>6</b>	<b>Листинги</b>	<b>23</b>
6.1	Листинг 7.1. Программа с использованием инструкции jmp . . . .	23
6.2	Листинг 7.2. Программа с использованием инструкции jmp . . . .	23
6.3	Листинг 7.3. Программа, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А,В и С. . . . .	24
6.4	Листинг 1. Программа, которая определяет и выводит на экран наименбшую из 3 целочисленных переменных: А,В и С. . . . .	25
6.5	Листинг 2. Программа, которая для введенных с клавиатуры значе- ний $\times$ и $\times$ вычисляет значение заданной функции $\times(\times)$ и выводит результат вычислений . . . . .	26

# Список иллюстраций

4.1	Базовые команды . . . . .	9
-----	---------------------------	---

## **Список таблиц**

# 1 Цель работы

Целью лабораторной работы является изучение команд условного и безусловного переходов, приобретение навыков написания программ с использованием переходов, а также знакомство с назначением и структурой файла листинга.

## 2 Задание

0. Общее ознакомление с командами условного и безусловного переходов.
1. Реализация переходов в NASM.
2. Изучение структуры файла листинга.
3. Выполнение заданий для самостоятельной 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. Безусловный переход выполняется инструкцией `jmp` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление. Адрес перехода может быть либо меткой, либо адресом области памяти, в которую предварительно помещен указатель перехода. Кроме того, в качестве операнда можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре. Как отмечалось выше, для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов. Флаг – это бит, принимающий значение 1 («флаг установлен»), если выполнено некоторое условие, и значение 0 («флаг сброшен») в противном случае. Флаги работают независимо друг от друга, и лишь для удобства они помещены в единый регистр — регистр флагов, отражающий текущее состояние процессора. Инструкция `cmp` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `cmp` является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания. Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов. Листинг (в

рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Он имеет текстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную информацию. Все ошибки и предупреждения, обнаруженные при ассемблировании, транслятор выводит на экран, и файл листинга не создаётся. Итак, структура листинга:

- номер строки — это номер строки файла листинга (нужно помнить, что номер строки в файле листинга может не соответствовать номеру строки в файле с исходным текстом программы);
- адрес — это смещение машинного кода от начала текущего сегмента;
- машинный код представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности. (например, инструкция `int 80h` начинается по смещению `00000020` в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция `int 80h` ассемблируется в `CD80` (в шестнадцатеричном представлении); `CD80` — это инструкция на машинном языке, вызывающая прерывание ядра);

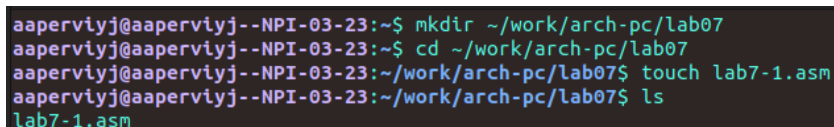
исходный текст программы — это просто строка исходной программы вместе с комментариями (некоторые строки на языке ассемблера, например, строки, содержащие только комментарии, не генерируют никакого машинного кода, и поля «смещение» и «исходный текст программы» в таких строках отсутствуют, однако номер строки им присваивается)



## 4 Выполнение лабораторной работы

Перед тем как начать выполнять лабораторную работу, необходимо создать директорию и файл, в которых я и буду работать, для этого прописываю следующие команды (Рис.1):

```
mkdir ~/work/arch-pc/lab07 cd ~/work/arch-pc/lab07 touch lab7-1.asm
```

A screenshot of a terminal window showing a series of commands and their outputs. The prompt is 'aaperviyj@aaperviyj--NPI-03-23:~\$'. The first command is 'mkdir ~/work/arch-pc/lab07', followed by 'cd ~/work/arch-pc/lab07', then 'touch lab7-1.asm', and finally 'ls'. The output of 'ls' is 'lab7-1.asm'.

```
aaperviyj@aaperviyj--NPI-03-23:~$ mkdir ~/work/arch-pc/lab07
aaperviyj@aaperviyj--NPI-03-23:~$ cd ~/work/arch-pc/lab07
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ touch lab7-1.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ ls
lab7-1.asm
```

Рис. 4.1: Базовые команды

Теперь, после того, как я создала пространство для дальнейшей работы, я могу переходить к следующему пункту

### 4.1 Реализация переходов в NASM

Открываю созданный файл lab7-1.asm, вставляю в него следующую программу из листинга 7.1: (Рис.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

```

Копирую в текущий каталог файл in\_out.asm с помощью утилиты cp, так как он будет использоваться в дальнейшем. Выполняю компиляцию, компоновку файла и запускаю его. Замечу, что использование инструкции jmp \_label2 меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки \_label2, пропустив вывод первого сообщения (Рис.3).

```

aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ cp ~/in_out.asm in_out.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ ls
in_out.asm  lab7-1.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$

```

Добавляю в текст метки jmp \_label1 jmp \_end. Программа представлена в листинге 7.2(Рис.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

```

Создаю новый исполняемый файл программы и запускаю его. Соответственно, инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. (Рис.5).

```

aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1

```

Изменяю метки `jmp` в программе, чтобы выводились сообщения в порядке от 3 до 1 (Рис.6)

```

%include 'in_out.asm' ; подключение внешнего
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label3 ; начать с метки 3
_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

```

{width=70%}

Выполняю компиляцию и компоновку, и запускаю исполняемый файл. Все работает так, как нужно. (Рис.7).

```

aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1

```

{width=70%}

Создаю файл lab7-2.asm. Редактирую его, вводя предлагаемую программу из листинга 7.3 (Рис.8):

```

%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:
mov eax, msg2
call sprint ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintLF ; Вывод 'max(A,B,C)'
call quit ; Выход

```

{width=70%)

Создаю исполняемый файл и проверяю его работу для разных целочисленных значений В(Рис.9):

```

aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 9
Наибольшее число: 50
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 99
Наибольшее число: 99

```

{width=70%

## 4.2 Изучение структуры файла листинга

Получаю файл листинга для программы lab7-2, указав ключ -l и введя имя листинга в командной строке(Рис.10):

```

aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ mc

```

{width=70%

Открываю полученный файл листинга в МС(Рис.11):

```

1          %include 'in_out.asm'
2          <1> ;----- slen -----
3          <1> ; Функция вычисления длины сообщения
4          <1> slen:
5          <1>     push    ebx
6          <1>     mov     ebx, eax
7          <1>
8          <1> nextchar:
9          <1>     cmp     byte [eax], 0
10         <1>     jz      finished
11         <1>     inc     eax
12         <1>     jmp     nextchar
13         <1>
14         <1> finished:
15         <1>     sub     eax, ebx
16         <1>     pop     ebx
17         <1>     ret
18         <1>
19         <1>
20         <1> ;----- sprint -----
21         <1> ; Функция печати сообщения
22         <1> ; входные данные: mov eax,<message>
23         <1> sprint:
24         <1>     push    edx
25         <1>     push    ecx
26         <1>     push    ebx
27         <1>     push    eax
28         <1>     call    slen
29         <1>
30         <1>     mov     edx, eax
31         <1>     pop     eax
32         <1>

```

<sup>^</sup>G Справка    <sup>^</sup>O Записать    <sup>^</sup>W Поиск    <sup>^</sup>K Вырезать    <sup>^</sup>T Выполнить    <sup>^</sup>C Позиция  
<sup>^</sup>X Выход    <sup>^</sup>R ЧитФайл    <sup>^</sup>\ Замена    <sup>^</sup>U Вставить    <sup>^</sup>J Выводить    <sup>^</sup>/ К строке

{width=70%

Объяснение строк:

Инструкция `mov esx, B` используется, чтобы положить адрес вводимой строки `B` в регистр `esx`. `call sread` - вызов подпрограммы из внешнего файла, обеспечивающей ввод сообщения с клавиатуры. `call atoi` используется для вызова подпрограммы из внешнего файла, которая преобразует `ascii`-код символа в целое число и записывает результат в регистр `eax`.

Убираю один из операндов в инструкции двумя операндами(Рис.12):

```

#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
mov [B], eax
; ----- Записываем 'A' в переменную 'max'
mov ecx, [A]
mov [max], ecx
; ----- Сравниваем 'A' и 'C' (как символы)
cmp [A], [C] ; Удален лишний 'ecx'
jg check_B
mov [max], [C] ; Удален лишний 'ecx'
check_B:
; ----- Преобразование 'max(A,C)' из символа в число
mov eax, max
call atoi
mov [max], eax
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
cmp [max], [B] ; Удален лишний 'ecx'
jg fin
mov [max], [B] ; Удален лишний 'ecx'
fin:
; ----- Вывод результата
mov eax, msg2
call sprint
mov eax, [max]
call iprintLF
call quit

```

{width=70%



Заново создаю листинг(Рис.13)

```
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
lab7-2.asm:30: error: invalid combination of opcode and operands
lab7-2.asm:37: error: invalid combination of opcode and operands
lab7-2.asm:39: error: invalid combination of opcode and operands
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab07$
```

Мы видим ошибку, но файл листинга создаётся. Открываю его. Также на месте строки находится сообщение об ошибке(Рис.14)

```
20 ; ----- Преобразование 'B' из символа в число
21 00000101 B8[0A000000] mov eax, B
22 00000106 E891FFFFFF call atoi
23 0000010B A3[0A000000] mov [B], eax
24 ; ----- Записываем 'A' в переменную 'max'
25 00000110 8B0D[35000000] mov ecx, [A]
26 00000116 890D[00000000] mov [max], ecx
27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 cmp [A], [C] ; Удален лишний 'ecx'
28 ***** error: invalid combination of opcode and operands
29 0000011C 7F00 jg check_B
30 mov [max], [C] ; Удален лишний 'ecx'
30 ***** error: invalid combination of opcode and operands
31 check_B:
32 ; ----- Преобразование 'max(A,C)' из символа в чи
33 0000011E B8[00000000] mov eax, max
34 00000123 E874FFFFFF call atoi
35 00000128 A3[00000000] mov [max], eax
36 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
37 cmp [max], [B] ; Удален лишний 'ecx'
37 ***** error: invalid combination of opcode and operands
38 0000012D 7F00 jg fin
39 mov [max], [B] ; Удален лишний 'ecx'
39 ***** error: invalid combination of opcode and operands
40 fin:
41 ; ----- Вывод результата
42 0000012F B8[13000000] mov eax, msg2
43 00000134 E8D6FEFFFF call sprint
44 00000139 A1[00000000] mov eax, [max]
45 0000013E E843FFFFFF call iprintLF
46 00000143 E8D3FEFFFF call quit
```

## 4.3 Самостоятельная работа

Для выполнения самостоятельной работы, создаю файл sr-1.asm, с помощью утилиты touch. Открываю созданный файл, ввожу в него текст программы для определения наименьшего числа из 3-х, предложенных в варианте 16, полученным мной во время выполнении прошлой лабораторной работы(Рис.15):

```

#include 'in_out.asm'
SECTION .data
msg1 DB "Наименьшее число: ", 0h
A DD '44'
B DD '74'
C DD '17'
SECTION .bss
min resb 10
SECTION .text
GLOBAL _start
_start:
    ; ----- Преобразование 'A' из символа в число
    mov eax, A
    call atoi
    mov [A], eax
    ; ----- Преобразование 'B' из символа в число
    mov eax, B
    call atoi
    mov [B], eax
    ; ----- Преобразование 'C' из символа в число
    mov eax, C
    call atoi
    mov [C], eax
    ; ----- Записываем 'A' в переменную 'min'
    mov ecx, [A]
    mov [min], ecx
    ; ----- Сравниваем 'A' и 'B'
    cmp ecx, [B]
    jl check_C
    ; ----- Если 'A >= B', то 'min = B'
    mov ecx, [B]
    mov [min], ecx
    jmp check_C
check_C:
    ; ----- Сравниваем 'min' и 'C'
    cmp ecx, [C]
    jl fin
    ; ----- Если 'min >= C', то 'min = C'
    mov ecx, [C]
    mov [min], ecx
fin:
    ; ----- Вывод результата
    mov eax, msg1
    call sprint
    mov eax, [min]
    call iprintLF
    call quit

```

{width=70%)

Проводим привычные операции и запускаем исполняемый файл, выполняем устную проверку(должно получиться 17) и убеждаемся в правильности работы программы(Рис.16):

```
aaerviyj@aaerviyj--NPI-03-23:~/work/arch-pc/lab07$ nasm -f elf sr-1.asm
aaerviyj@aaerviyj--NPI-03-23:~/work/arch-pc/lab07$ ld -m elf_i386 -o sr-1 sr-1.o
aaerviyj@aaerviyj--NPI-03-23:~/work/arch-pc/lab07$ ./sr-1
Наименьшее число: 17
```

{width=70%

Создаю файл sr-2.asm с помощью утилиты touch. Открываю созданный файл для редактирования, ввожу в него текст программы для своего 16-го варианта:  $f = x + 4$ , если  $x < a$  и  $f = a * x$ , если  $x \geq 4$  (Рис.17)

```

%include 'in_out.asm'
SECTION .data
    msg_x: DB 'Введите значение переменной x: ', 0
    msg_a: DB 'Введите значение переменной a: ', 0
    msg_result: DB 'Результат: ', 0
SECTION .bss
    x: RESB 80
    a: RESB 80
    result: 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 [x], eax
    ; Ввод значения переменной a
    mov eax, msg_a
    call sprint
    mov ecx, a
    mov edx, 80
    call sread
    mov eax, a
    call atoi
    mov [a], eax
    ; Сравнение x и 4
    cmp dword [x], 4
    jl less_than_4
    jge greater_than_or_equal_4
less_than_4:
    ; Вычисление f = x + 4
    mov eax, [x]
    add eax, 4
    mov [result], eax
    jmp end_calculation
greater_than_or_equal_4:
    ; Вычисление f = a * x
    mov eax, [a]
    mov ebx, [x]
    imul eax, ebx
    mov [result], eax
end_calculation:
    ; Вывод результата на экран
    mov eax, msg_result
    call sprint

```

^G Справка  
 ^X Выход

^O Записать  
 ^R ЧитФайл

^W Поиск  
 ^\ Замена

^K В  
 ^U В

{width=70%)

Компилирую, обрабатываю и запускаю исполняемый файл. Ввожу предложенные значения, и, сделав проверку, понимаю, что программа работает верно(Рис.18)

```
аарerviyj@аарerviyj--NPI-03-23:~/work/arch-pc/lab07$ nasm -f elf sr-2.asm
аарerviyj@аарerviyj--NPI-03-23:~/work/arch-pc/lab07$ ld -m elf_i386 -o sr-2 sr-2.o
аарerviyj@аарerviyj--NPI-03-23:~/work/arch-pc/lab07$ ./sr-2
Введите значение переменной x: 1
Введите значение переменной a: 1
Результат: 5
аарerviyj@аарerviyj--NPI-03-23:~/work/arch-pc/lab07$ ./sr-2
Введите значение переменной x: 7
Введите значение переменной a: 1
Результат: 7
аарerviyj@аарerviyj--NPI-03-23:~/work/arch-pc/lab07$ █
```

{width=70%

## 5 Выводы

При выполнении лабораторной работы я изучила команды условного и безусловного переходов, приобрела практический опыт в написании программ с использованием переходов, познакомилась с назначением и структурой файла листинга.

## 6 Листинги

### 6.1 Листинг 7.1. Программа с использованием инструкции

#### **jmp**

```
'''nasm
(text)
%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 sprintLF ; 'Сообщение № 1' _label2: mov eax, msg2 ; Вывод на
экран строки call sprintLF ; 'Сообщение № 2' _label3: mov eax, msg3 ; Вывод на
экран строки call sprintLF ; 'Сообщение № 3' _end: call quit ; вызов подпрограммы
завершения
'''
```

### 6.2 Листинг 7.2. Программа с использованием инструкции

#### **jmp**

```
'''nasm
(text)
%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 sprintLF ; 'Сообщение № 1' 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: call quit ;
вызов подпрограммы завершения
'''
```

### 6.3 Листинг 7.3. Программа, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А,В и С.

```
'''nasm
(text)
%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' mov ecx,[A] ; 'ecx = A' mov [max],ecx ; 'max = A' ; ——— Сравниваем 'А' и
'С' (как символы) cmp ecx,[C] ; Сравниваем 'А' и 'С' jg check_B ; если 'А>С', 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 'В' (как числа)
mov ecx,[max] cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'В' jg fin ; если 'max(A,C)>В',
то переход на 'fin', mov ecx,[B] ; иначе 'ecx = В' mov [max],ecx ; ——— Вывод
результата fin: mov eax, msg2 call sprint ; Вывод сообщения 'Наибольшее число:'
```



```
mov eax,[max] call iprintLF ; Вывод 'max(A,B,C)' call quit ; Выход
'''
```

## 6.4 Листинг 1. Программа, которая определяет и выводит на экран наименбшую из 3 целочисленных переменных: A,B и C.

```
'''nasm
(text)
%include 'in_out.asm' SECTION .data msg1 DB "Наименьшее число:", 0h A DD '44'
B DD '74' C DD '17' SECTION .bss min resb 10 SECTION .text GLOBAL _start _start:
; ---- Преобразование 'A' из символа в число mov eax, A call atoi mov [A], eax
; ---- Преобразование 'B' из символа в число mov eax, B call atoi mov [B], eax
; ---- Преобразование 'C' из символа в число mov eax, C call atoi mov [C], eax
; ---- Записываем 'A' в переменную 'min' mov ecx, [A] mov [min], ecx ; ----
Сравниваем 'A' и 'B' cmp ecx, [B] jl check_C ; ---- Если 'A >= B', то 'min = B' mov
ecx, [B] mov [min], ecx jmp check_C check_C: ; ---- Сравниваем 'min' и 'C' cmp ecx,
[C] jl fin ; ---- Если 'min >= C', то 'min = C' mov ecx, [C] mov [min], ecx fin: ; ----
Вывод результата mov eax, msg1 call sprint mov eax, [min] call iprintLF call quit
'''
```

## 6.5 Листинг 2. Программа, которая для введенных с клавиатуры значений $x$ и $a$ вычисляет значение заданной функции $f(x)$ и выводит результат вычислений

```
'''nasm
(text) %include 'in_out.asm' SECTION .data msg_x: DB 'Введите значение перемен-
ной x:', 0 msg_a: DB 'Введите значение переменной a:', 0 msg_result: DB 'Результат:',
0 SECTION .bss x: RESB 80 a: RESB 80 result: 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 [x], eax ; Ввод значения переменной a
mov eax, msg_a call sprint mov ecx, a mov edx, 80 call sread mov eax, a call atoi mov
[a], eax ; Сравнение x и 4 cmp dword [x], 4 jl less_than_4 jge greater_than_or_equal_4
less_than_4: ; Вычисление  $f = x + 4$  mov eax, [x] add eax, 4 mov [result], eax jmp
end_calculation greater_than_or_equal_4: ; Вычисление  $f = a * x$  mov eax, [a] mov ebx,
[x] imul eax, ebx mov [result], eax end_calculation: ; Вывод результата на экран mov
eax, msg_result call sprint mov eax, [result] call iprintLF call quit

'''
```