

Отчёт по лабораторной работе №7

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

Лань Цянын

1. Цель работы

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

2. Порядок выполнения лабораторной работы

2.1. Реализация переходов в NASM

1. Сначала был создан каталог `~/work/arch-pc/lab07`, после чего выполнен переход в него командой `cd`. (рис. Рисунок 1).

```
clanj1@clanj1:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/Lab06$  
mkdir ~/work/arch-pc/lab07  
clanj1@clanj1:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab06$  
cd ~/work/arch-pc/lab07
```

Рисунок 1: Создание каталога `lab07` и переход в него

2. В ходе лабораторной работы № 7 был создан каталог `lab07` и файл `lab7-1.asm`, в который введён текст программы из листинга 7.1 с использованием инструкции `jmp`. (рис. Рисунок 2).

The screenshot shows a text editor window titled "lab7-1.asm" with the file path "/work/arch-pc/lab07". The code is written in NASM assembly language. It includes three data sections (.data) containing strings "Сообщение № 1", "Сообщение № 2", and "Сообщение № 3". It has a .text section starting with the _start label, which branches to three labels (_label1, _label2, _label3) that each output a message and then jump to _label2. Finally, it ends with a quit call.

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg1: DB 'Сообщение № 1',0
5 msg2: DB 'Сообщение № 2',0
6 msg3: DB 'Сообщение № 3',0
7
8 SECTION .text
9 GLOBAL _start
10
11 _start:
12     jmp _label2
13
14 _label1:
15     mov eax, msg1
16     call sprintLF
17
18 _label2:
19     mov eax, msg2
20     call sprintLF
21
22 _label3:
23     mov eax, msg3
24     call sprintLF
25
26 _end:
27     call quit
```

Рисунок 2: Редактирование файла lab7-1.asm

Файл `in_out.asm` был скопирован в каталог `lab07`, после чего выполнены компиляция, линковка и запуск программы. (рис. Рисунок 3).

```
clanji@clanji:~/work/arch-pc/lab07$ touch lab7-1.asm
clanji@clanji:~/work/arch-pc/lab07$ cp /media/sf_VB_Share/in_out.asm ~/work/arch
-pc/lab07
clanji@clanji:~/work/arch-pc/lab07$ gedit lab7-1.asm
clanji@clanji:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
clanji@clanji:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
clanji@clanji:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
```

Рисунок 3: Сборка и запуск программы lab7-1

В результате работы программы на экран были выведены строки Сообщение № 2 и Сообщение № 3, так как выполнение начинается с перехода `jmp _label2`.

В ходе лабораторной работы № 7 был изменён текст программы `lab7-1.asm` в соответствии с листингом 7.2 с использованием инструкции `jmp` для изменения порядка выполнения команд. (рис. Рисунок 4).

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg1: DB 'Сообщение № 1',0
5 msg2: DB 'Сообщение № 2',0
6 msg3: DB 'Сообщение № 3',0
7
8 SECTION .text
9 GLOBAL _start
10
11 _start:
12     jmp _label2
13
14 _label1:
15     mov eax, msg1
16     call sprintLF
17     jmp _end
18
19 _label2:
20     mov eax, msg2
21     call sprintLF
22     jmp _label1
23
24 _label3:
25     mov eax, msg3
26     call sprintLF
27
28 _end:
29     call quit|
```

Рисунок 4: Редактирование программы lab7-1.asm по листингу 7.2

После внесения изменений программа была сохранена.

Программа была модифицирована в соответствии с Листингом 7.2, после чего собрана и запущена.(рис. Рисунок 5).

```
clanj1@clanj1:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
clanj1@clanj1:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
clanj1@clanj1:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
```

Рисунок 5: Компиляция и запуск модифицированной программы lab7-1

В результате программа вывела сообщения «Сообщение № 2» и «Сообщение № 1» в указанном порядке, демонстрируя управление порядком выполнения через инструкции jmp. Третье сообщение не было выведено, поскольку управление не возвращалось к соответствующей метке.

Для достижения требуемого порядка вывода (сначала третье сообщение, затем второе, затем первое) программа была модифицирована путем добавления и изменения инструкций jmp.(рис. Рисунок 6).

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg1: DB 'Сообщение № 1',0
5 msg2: DB 'Сообщение № 2',0
6 msg3: DB 'Сообщение № 3',0
7
8 SECTION .text
9 GLOBAL _start
10
11 _start:
12     jmp _label3
13
14 _label1:
15     mov eax, msg1
16     call sprintLF
17     jmp _end
18
19 _label2:
20     mov eax, msg2
21     call sprintLF
22     jmp _label1
23
24 _label3:
25     mov eax, msg3
26     call sprintLF
27     jmp _label2
28
29 _end:
30     call quit|
```

Рисунок 6: Код модифицированной программы lab7-1

Модифицированный код использует инструкции `jmp` для организации обратного порядка вывода, начиная с метки `_label3`

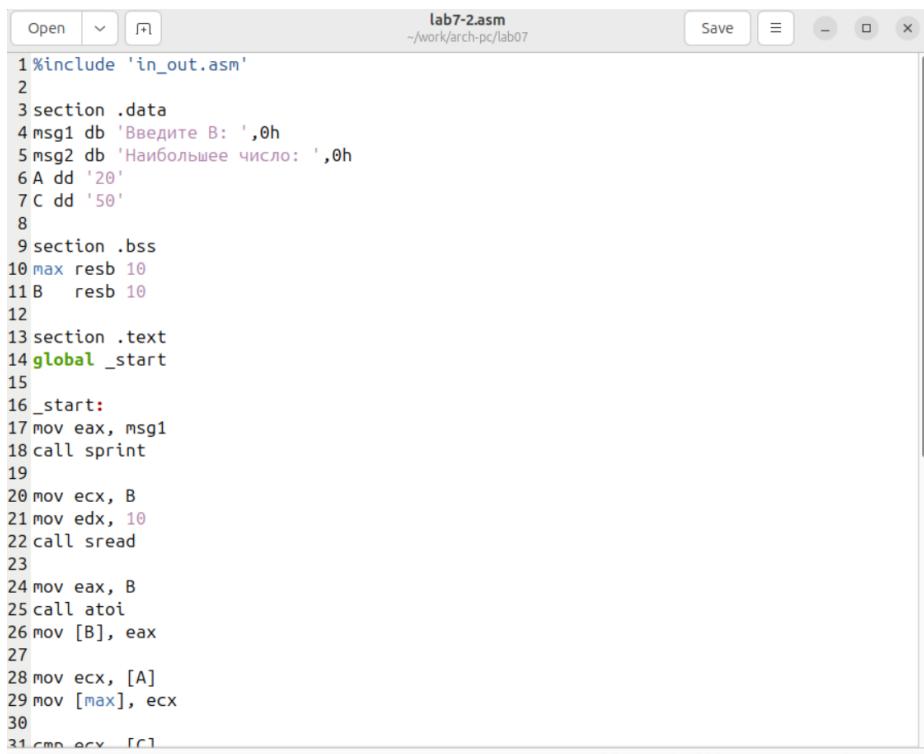
Модифицированная программа была откомпилирована и запущена, выводя три сообщения в требуемом порядке. (рис. Рисунок 7)

```
clanj1@clanj1:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
clanj1@clanj1:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
clanj1@clanj1:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Рисунок 7: Выполнение модифицированной программы lab7-1

Результат выполнения показал, что программа успешно выводит сообщения в порядке «Сообщение № 3», «Сообщение № 2», «Сообщение № 1», что соответствует поставленной задаче по изменению потока управления с помощью инструкций `jmp`.

3. Программа lab7-2.asm реализована в точном соответствии с Листингом 7.3 из методического пособия.(рис. Рисунок 8)

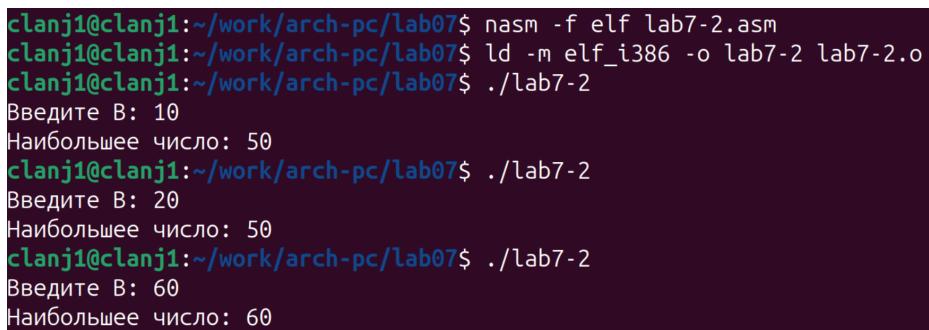


```
1 %include 'in_out.asm'
2
3 section .data
4 msg1 db 'Введите В: ',0h
5 msg2 db 'Наибольшее число: ',0h
6 A dd '20'
7 C dd '50'
8
9 section .bss
10 max resb 10
11 B resb 10
12
13 section .text
14 global _start
15
16 _start:
17 mov eax, msg1
18 call sprint
19
20 mov ecx, B
21 mov edx, 10
22 call sread
23
24 mov eax, B
25 call atoi
26 mov [B], eax
27
28 mov ecx, [A]
29 mov [max], ecx
30
31 cmp ecx, [C]
```

Рисунок 8: Исходный код программы lab7-2 (Листинг 7.3)

Программа демонстрирует сравнение переменных A, C как символов и B – как числа.

Программа была протестирована с тремя различными значениями B в соответствии с требованием задания. (рис. Рисунок 9)



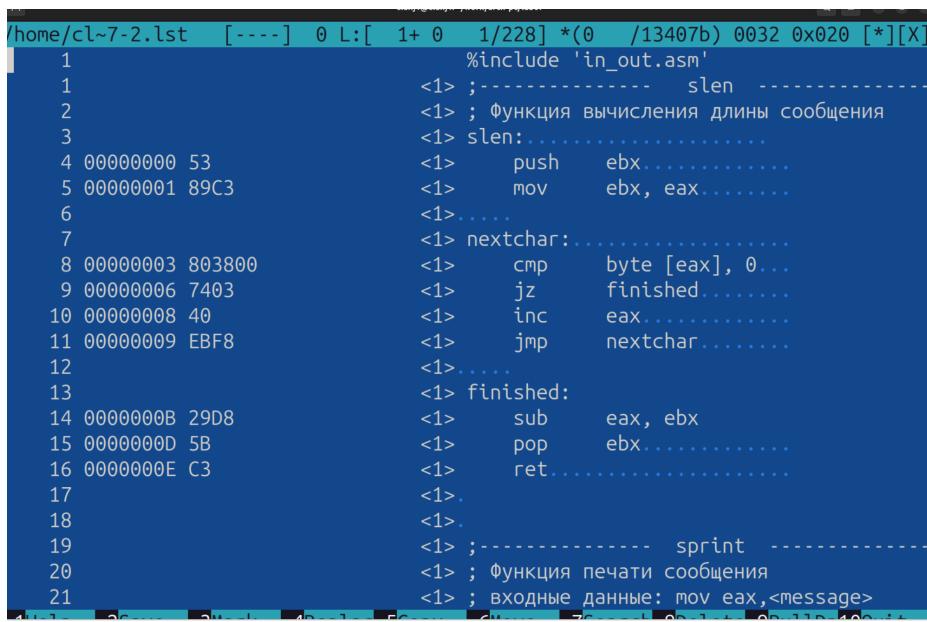
```
clanj1@clanj1:~/work/arch-pc/Lab07$ nasm -f elf lab7-2.asm
clanj1@clanj1:~/work/arch-pc/Lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
clanj1@clanj1:~/work/arch-pc/Lab07$ ./lab7-2
Введите В: 10
Наибольшее число: 50
clanj1@clanj1:~/work/arch-pc/Lab07$ ./lab7-2
Введите В: 20
Наибольшее число: 50
clanj1@clanj1:~/work/arch-pc/Lab07$ ./lab7-2
Введите В: 60
Наибольшее число: 60
```

Рисунок 9: Тестирование программы lab7-2 (Листинг 7.3)

Результаты тестирования подтверждают корректную работу программы:
при B=10 и B=20 выводится С (50), при B=60 – значение В (60).

2.2. Изучение структуры файлы листинга

Для программы lab7-2.asm был создан файл листинга командой ‘nasm -f elf -l lab7-2.lst lab7-2.asm(рис. Рисунок 10)



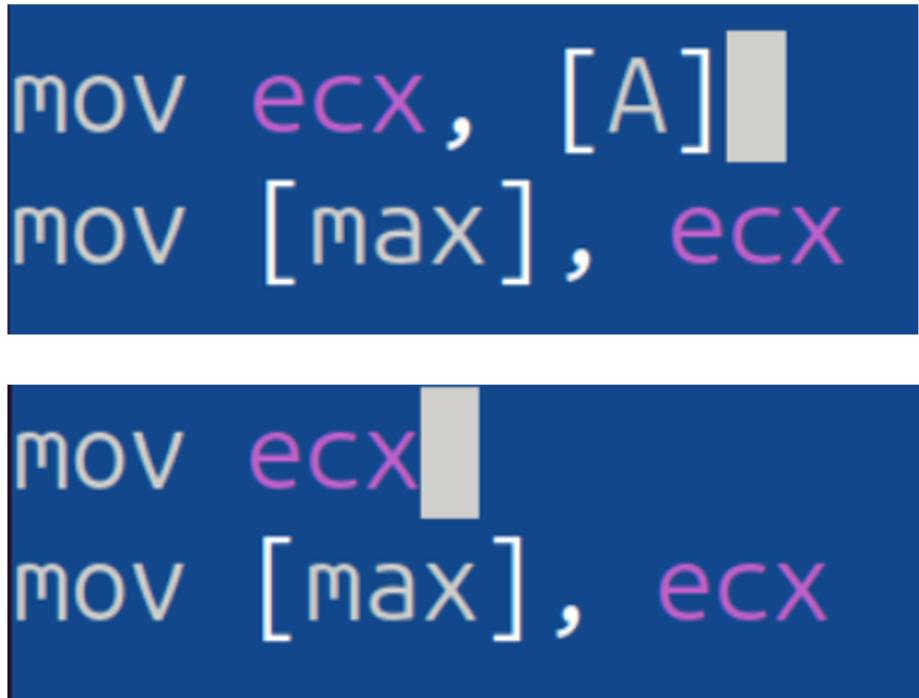
```
/home/cl~7-2.lst [---] 0 L:[ 1+ 0 1/228] *(0 /13407b) 0032 0x020 [*][X]
1           %include 'in_out.asm'
1           ;-----slen-----
2           ; Функция вычисления длины сообщения
3           slen:
4 00000000 53      push    ebx.....
5 00000001 89C3    mov     ebx, eax.....
6           .....
7           nextchar:
8 00000003 803800  cmp     byte [eax], 0...
9 00000006 7403    jz      finished.....
10 00000008 40     inc    eax.....
11 00000009 EBF8   jmp    nextchar.....
12           .....
13           finished:
14 0000000B 29D8   sub    eax, ebx
15 0000000D 5B     pop    ebx.....
16 0000000E C3     ret.....
17           .....
18           .....
19           ;-----sprint-----
20           ; Функция печати сообщения
21           ; входные данные: mov eax,<message>
```

Рисунок 10: Структура файла листинга lab7-2.lst

Объяснение трёх строк листинга :

1. **00000000 53 push ebx:** Инструкция push ebx сохраняет текущее значение регистра ebx в стеке перед использованием его в подпрограмме.
2. **00000003 803800 cmp byte [eax], 0:** Инструкция cmp byte [eax], 0 сравнивает текущий символ строки с нулевым байтом („\0“), определяя конец строки.
3. **00000006 7403 jz finished:** Инструкция jz finished выполняет переход на метку finished, если результат предыдущего сравнения равен нулю.

В файле lab7-2.asm в инструкции `mov ecx, [A]` был удалён один операнд, в результате чего она превратилась в неполную инструкцию `mov ecx`(рис. Рисунок 11)



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

Рисунок 11: Изменённый исходный код с ошибкой

При попытке ассемблирования такой программы, транслятор NASM обнаруживает ошибку `invalid combination of opcode and operands` и прекращает процесс(рис. Рисунок 12)

```
clanj1@clanj1:~/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
```

```
28          *****  
28          mov ecx  
29 00000110 890D[00000000]      error: invalid combination of opcode and  
29                                     operands  
30
```

Рисунок 12: Ошибка ассемблирования

Какие выходные файлы создаются в этом случае?

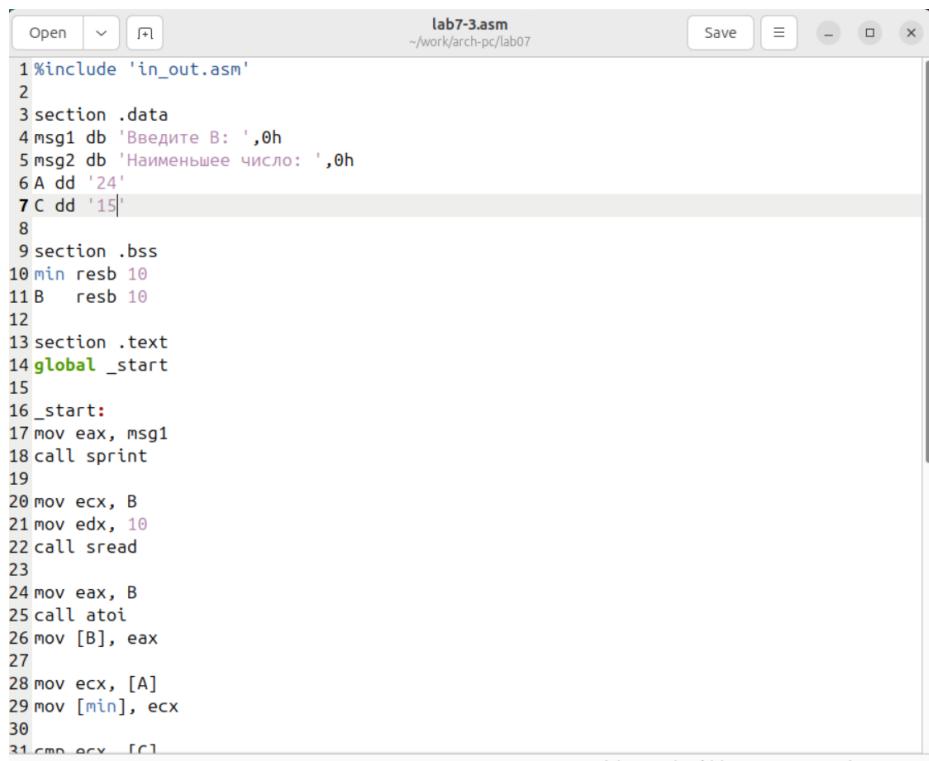
- **Объектный файл (lab7-2.o): НЕ создаётся**, так как трансляция не прошла успешно.
- **Файл листинга (lab7-2.lst):** Создаётся, но его содержимое может быть неполным (до момента ошибки).

Что добавляется в листинге?

В файл листинга **не добавляются** специальные отметки об ошибках Сообщения об ошибках выводятся **только на экран (в стандартный поток ошибок)** Листинг содержит только информацию об успешно ассемблированных строках до точки возникновения фатальной ошибки. Таким образом, файл листинга в данном случае является лишь частичным и не содержит указания на синтаксическую ошибку в самом файле.

3 .Задание для самостоятельной работы

1.Для выполнения задания №1 (вариант 9) на нахождение наименьшего из трёх целых чисел, программа была модифицирована на основе Листинга 7.3. Вместо поиска максимума теперь выполняется поиск минимума, а также изменены значения переменных А и С на соответствующие варианту (24 и 15)(рис. Рисунок 1)



```
1 %include 'in_out.asm'
2
3 section .data
4 msg1 db 'Введите В: ',0h
5 msg2 db 'Наименьшее число: ',0h
6 A dd '24'
7 C dd '15'
8
9 section .bss
10 min resb 10
11 B resb 10
12
13 section .text
14 global _start
15
16 _start:
17 mov eax, msg1
18 call sprint
19
20 mov ecx, B
21 mov edx, 10
22 call sread
23
24 mov eax, B
25 call atoi
26 mov [B], eax
27
28 mov ecx, [A]
29 mov [min], ecx
30
31 cmp ecx, [C]
```

Рисунок 1: Исходный код программы lab7-3.asm для нахождения минимума (вариант 9)

В программе произведены изменения для поиска минимума: переименованы переменные и сообщения, заменены инструкции условного перехода, установлены значения a=24 и c=15 (значение b вводится с клавиатуры)

Модифицированная программа lab7-3.asm была откомпилирована и запущена с входным значением B=98, соответствующим варианту 9(рис. Рисунок 2)

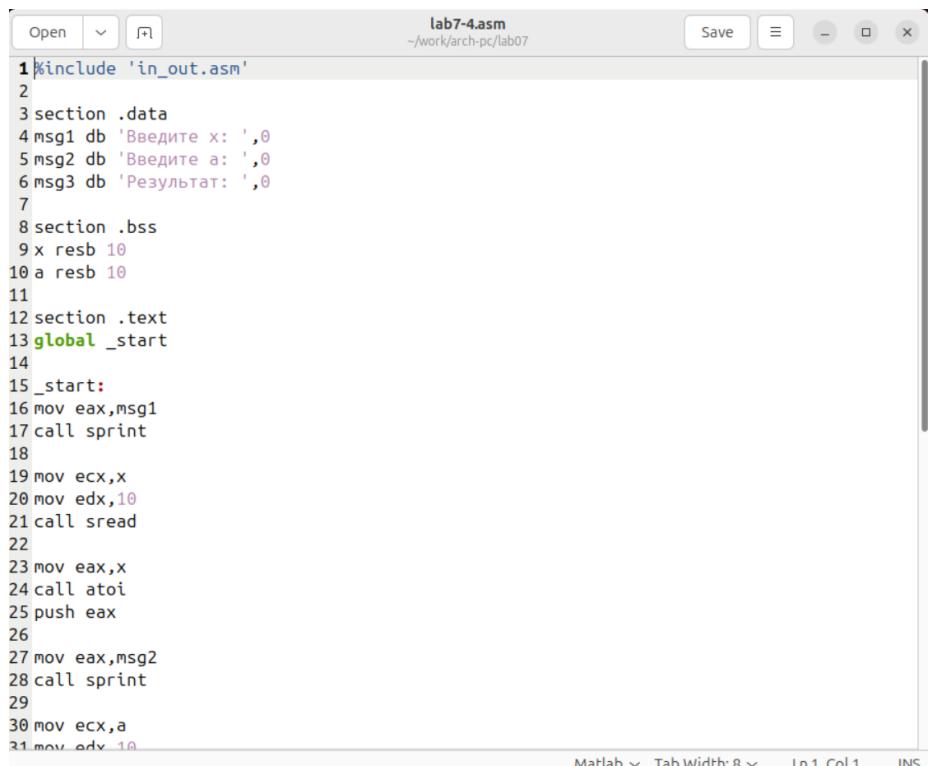
```
clanj1@clanj1:~/work/arch-pc/lab07$ touch lab7-3.asm
clanj1@clanj1:~/work/arch-pc/lab07$ gedit lab7-3.asm
clanj1@clanj1:~/work/arch-pc/lab07$ nasm -f elf lab7-3.asm
clanj1@clanj1:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
clanj1@clanj1:~/work/arch-pc/lab07$ ./lab7-3
Введите В: 98
Наименьшее число: 24
```

Рисунок 2: Компиляция и выполнение программы lab7-3 (вариант 9)

Программа вывела результат «Наименьшее число: 24», что соответствует

ожидающему минимальному значению среди $a=24$, $b=98$ и $c=15$

2. Для выполнения задания №2 (вариант 9) была написана программа lab7-4.asm, вычисляющая значение функции $f(x)$ в зависимости от введённых x и a .(рис. Рисунок 3)



```
Open Save lab7-4.asm
~/work/arch-pc/lab07
1 %include 'in_out.asm'
2
3 section .data
4 msg1 db 'Введите x: ',0
5 msg2 db 'Введите a: ',0
6 msg3 db 'Результат: ',0
7
8 section .bss
9 x resb 10
10 a resb 10
11
12 section .text
13 global _start
14
15 _start:
16 mov eax, msg1
17 call sprint
18
19 mov ecx, x
20 mov edx, 10
21 call sread
22
23 mov eax, x
24 call atoi
25 push eax
26
27 mov eax, msg2
28 call sprint
29
30 mov ecx, a
31 mov edx, 10
```

Matlab Tab Width: 8 Ln 1, Col 1 INS

Рисунок 3: Исходный код программы lab7-4.asm

Программа запрашивает у пользователя ввод двух целых чисел: x и a . Для этого используются подпрограммы ввода-вывода из файла in_out.asm.

Программа lab7-4.asm была успешно откомпилирована и запущена для тестирования функции, соответствующей варианту 9.(рис. Рисунок 4)

```
clanj1@clanj1:~/work/arch-pc/lab07$ nasm -f elf lab7-4.asm
clanj1@clanj1:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-4 lab7-4.o
clanj1@clanj1:~/work/arch-pc/lab07$ ./lab7-4
Введите x: 5
Введите a: 7
Результат: 12
clanj1@clanj1:~/work/arch-pc/lab07$ ./lab7-4
Введите x: 6
Введите a: 4
Результат: 4
```

Рисунок 4: Выполнение программы lab7-4.asm с тестовыми данными

Программа была протестирована на двух парах значений (x , a) из таблицы 7.6. Результаты выполнения соответствуют заданной функции: при (5, 7) (где $x \leq a$) выведено 12 ($7+5$), а при (6, 4) (где $x > a$) – 4 (значение a).

Выводы

В лабораторной работе были изучены команды условного и безусловного перехода в NASM. Реализованы программы с ветвлениеми, выполнено сравнение числовых данных и проанализирована структура файла листинга. Цель лабораторной работы достигнута.