

# **Команды безусловного и условного переходов в Pascal. Программирование ветвлений.**

**Лабораторная работа №7**

Приходько Иван Иванович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
<b>3</b>	<b>Задания для самостоятельной работы</b>	<b>18</b>
<b>4</b>	<b>Выводы</b>	<b>23</b>

# Список иллюстраций

2.1	Создание lab7-1.asm . . . . .	6
2.2	Вставка кода из файла листинга 7.1 . . . . .	7
2.3	Копирование in_out.asm . . . . .	8
2.4	Сборка и запуск lab7-1.asm . . . . .	8
2.5	Изменение файла lab7-1.asm согласно листингу 7.2 . . . . .	9
2.6	Повторная сборка и запуск lab7-1.asm . . . . .	9
2.7	Редактирование lab7-1.asm . . . . .	10
2.8	Повторная сборка и запуск lab7-1.asm . . . . .	11
2.9	Создание lab7-2.asm . . . . .	11
2.10	Запись кода из листинга 7.3 в файл lab7-2.asm . . . . .	12
2.11	Сборка и запуск lab7-2.asm . . . . .	13
2.12	Создание файла листинга из lab7-2.asm . . . . .	13
2.13	Вид файла lab7-2.lst . . . . .	14
2.14	Нахождение программы в файле листинга . . . . .	15
2.15	Допуск ошибки в lab7-2.asm . . . . .	16
2.16	Сборка файла с ошибкой . . . . .	16
2.17	Ошибка в lab7-2.lst . . . . .	17
3.1	Создание файла task1v6.asm . . . . .	18
3.2	Код task1v6.asm . . . . .	19
3.3	Сборка и запуск task1v6.asm . . . . .	20
3.4	Создание файла task2v6.asm . . . . .	20
3.5	Код task2v6.asm . . . . .	21
3.6	Сборка и запуск task2v6.asm . . . . .	22

## **Список таблиц**

# 1 Цель работы

Понять принцип работы условных и безусловных переходов в Ассемблере и научиться писать программы с командами, отвечающими за переходы. Научиться работать с файлами листинга и уметь их читать.

## 2 Выполнение лабораторной работы

Для начала выполнения лабораторной работы необходимо создать файл lab7-1.asm (рис. 2.1).

```
ivanprithodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc$ cd ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07/  
ivanprithodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ touch lab7-1.asm
```

Рис. 2.1: Создание lab7-1.asm

Вставим в него код из листинга 7.1 (рис. 2.2).

```
GNU nano 7.2 /home/ivanprihc
%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. 2.2: Вставка кода из файла листинга 7.1

Скопируем файл `in_out.asm` из рабочей директории прошлой лабораторной работы (рис. 2.3).

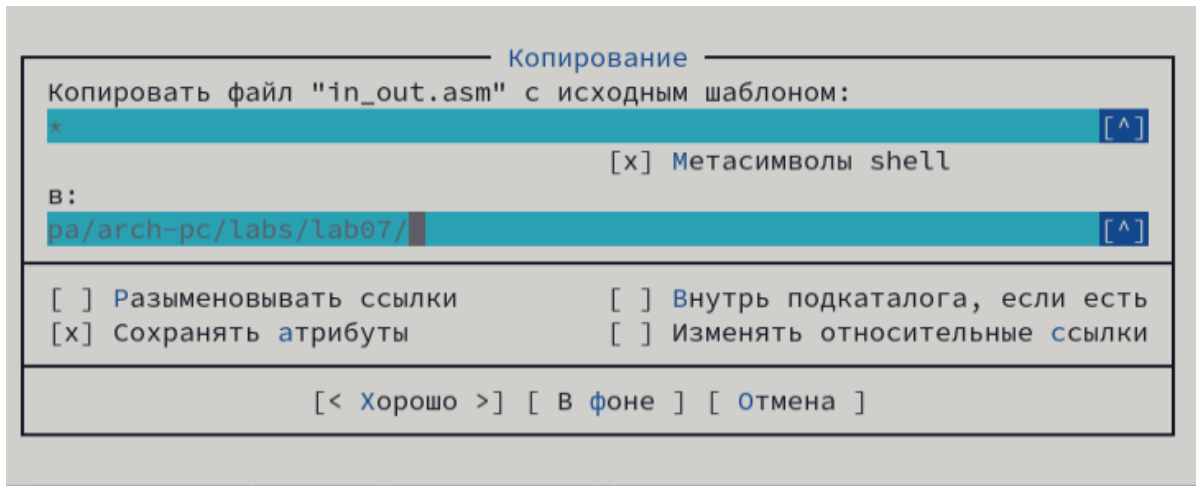


Рис. 2.3: Копирование in\_out.asm

Соберем и запустим (рис. 2.4).

```
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf lab7-1.asm
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 2.4: Сборка и запуск lab7-1.asm

Изменим файл lab7-1.asm согласно листингу 7.2 (рис. 2.5).



```
GNU nano 7.2 /home/ivar
%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
```

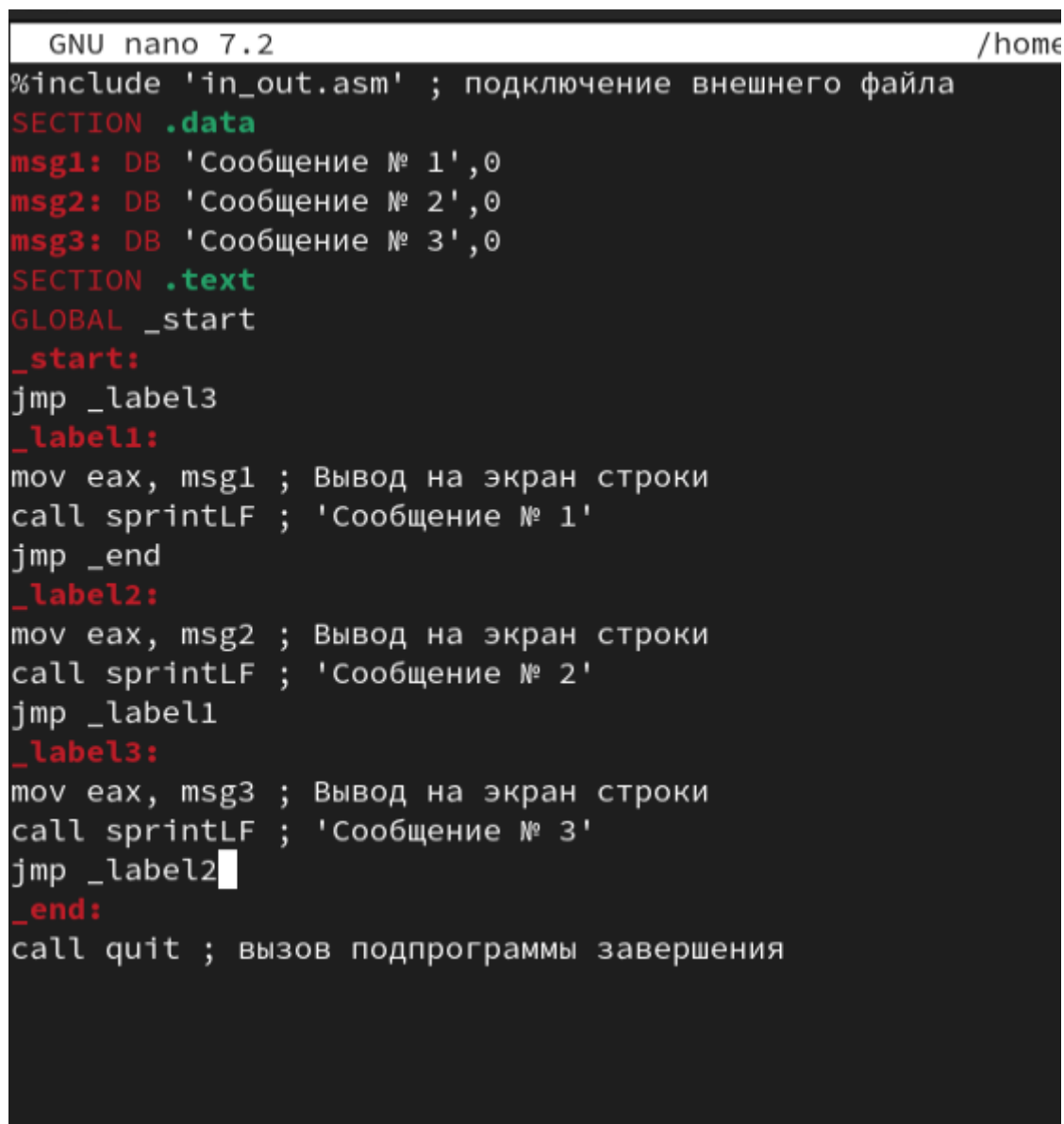
Рис. 2.5: Изменение файла lab7-1.asm согласно листингу 7.2

Повторно соберем и запустим (рис. 2.6).

```
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура_компьютера/arch-pc/labs/lab07$ nasm -f elf lab7-1.asm
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура_компьютера/arch-pc/labs/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура_компьютера/arch-pc/labs/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура_компьютера/arch-pc/labs/lab07$
```

Рис. 2.6: Повторная сборка и запуск lab7-1.asm

Теперь сделаем так, чтобы код выводил сообщения в обратном порядке, для этого внесём в код следующие изменения (рис. 2.7).



```
GNU nano 7.2 /home
%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 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'
jmp _label2
_end:
call quit ; вызов подпрограммы завершения
```

Рис. 2.7: Редактирование lab7-1.asm

Запустим (рис. 2.8).

```
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf lab7-1.asm
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 2.8: Повторная сборка и запуск lab7-1.asm

Теперь создадим файл lab7-2.asm (рис. 2.9).

```
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ touch lab7-2.asm
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 2.9: Создание lab7-2.asm

Вставим код из листинга 7.3 в файл lab7-2.asm (рис. 2.10).

```

GNU nano 7.2 /home/ivanpr
%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

```

Рис. 2.10: Запись кода из листинга 7.3 в файл lab7-2.asm

Соберем и запустим (рис. 2.11).

```
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf lab7-2.asm
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ ld -m elf_1386 -o lab7-2 lab7-2.o
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ ./lab7-2
Введите B: 15
Наибольшее число: 50
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ ./lab7-2
Введите B: 55
Наибольшее число: 55
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 2.11: Сборка и запуск lab7-2.asm

Теперь попробуем создать файл листинга при сборке файла lab7-2.asm (рис. 2.12).

```
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
ivanprihodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ mcedit lab7-2.lst
```

Рис. 2.12: Создание файла листинга из lab7-2.asm

Открыв его, мы видим следующую картину (рис. 2.13).

```

lab7-2.lst      [----]  0 L:[ 1+ 0  1/225] *(0  /14458b) 0032 0x020
1               %include 'in_out.asm'
1               <1> ;----- slen -----
2               <1> ; Функция вычисления длины сообщения
3               <1> slen:.....
4 00000000 53     <1>     push     ebx.....
5 00000001 89C3   <1>     mov      ebx, eax.....
6               <1>.....
7               <1> nextchar:.....
8 00000003 803800 <1>     cmp      byte [eax], 0...
9 00000006 7403   <1>     jz       finished.....
10 00000008 40    <1>     inc      eax.....
11 00000009 EBF8  <1>     jmp      nextchar.....
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 BB01000000 <1>     mov      ebx, 1
34 00000022 B804000000 <1>     mov      eax, 4
35 00000027 CD80   <1>     int      80h
36             <1>.
37 00000029 5B    <1>     pop      ebx
38 0000002A 59    <1>     pop      ecx

```

Рис. 2.13: Вид файла lab7-2.lst

Наша программа находится ниже (рис. 2.14).

```

lab7-2.lst      [----]  0 L:[171+ 0 171/225] *(10588/14458b) 0032 0x020
170 000000E7 C3          <1>      ret
2                                section .data
3 00000000 D092D0B2D0B5D0B4D0-    msg1 db 'Введите B: ',0h
3 00000009 B8D182D0B520423A20-
3 00000012 00.....
4 00000013 D09DD0B0D0B8D0B1D0-    msg2 db "Наибольшее число: ",0h
4 0000001C BED0BBD18CD188D0B5-
4 00000025 D0B520D187D0B8D181-
4 0000002E D0BBD0BE3A2000....
5 00000035 32300000          A dd '20'
6 00000039 35300000          C dd '50'
7                                section .bss
8 00000000 <res Ah>          max resb 10
9 0000000A <res Ah>          B resb 10
10                               section .text
11                               global _start
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 000000F7 BA0A000000      mov edx,10
19 000000FC E842FFFFFF      call sread
20                               ; ----- Преобразование 'B' из символа в число
21 00000101 B8[0A000000]      mov eax,B
22 00000106 E891FFFFFF      call atoi ; Вызов подпрограммы перевода символа в число
23 0000010B A3[0A000000]      mov [B],eax ; запись преобразованного числа в 'B'
24                               ; ----- Записываем 'A' в переменную 'max'
25 00000110 8B0D[35000000]    mov ecx,[A] ; 'ecx = A'
26 00000116 890D[00000000]    mov [max],ecx ; 'max = A'
27                               ; ----- Сравниваем 'A' и 'C' (как символы)
28 0000011C 3B0D[39000000]    cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 00000122 7F0C             jg check_B ; если 'A>C', то переход на метку 'check_B',
30 00000124 8B0D[39000000]    mov ecx,[C] ; иначе 'ecx = C'
31 0000012A 890D[00000000]    mov [max],ecx ; 'max = C'

```

Рис. 2.14: Нахождение программы в файле листинга

Разберём несколько строк файла листинга:

1. Строка под номером 14 перемещает содержимое msg1 в регистр eax. Адрес указывается сразу после номера. Следом идёт машинный код, который представляет собой исходную ассемблированную строку в виде шестнадцатичной системы. Далее идёт исходный код
2. 15-ая строка отвечает за вызов функции sprint. Она также имеет адрес и машинный код
3. Строка 17 отвечает за запись переменной B в регистр ecx. Как видно, все строки имеют номер, адрес, машинный код и исходный код.

Теперь попробуем намеренно допустить ошибку в нашем коде, убрав у команды `mov 1` операнд (рис. 2.15).

```
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,
call sread
; ----- Преобразование 'В' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в чи
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)
```

Рис. 2.15: Допуск ошибки в lab7-2.asm

И попробуем собрать файл с ошибкой, генерируя файл листинга (рис. 2.16).

```
ivanprikhodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:18: error: invalid combination of opcode and operands
ivanprikhodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 2.16: Сборка файла с ошибкой

Посмотрим как выглядит ошибка в файле листинга (рис. 2.17).



```

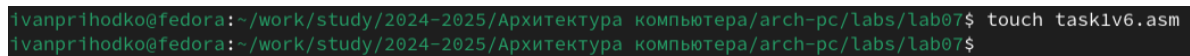
8 00000000 <res Ah>          max resb 10
9 0000000A <res Ah>          B resb 10
10                               section .text
11                               global _start
12                               _start:
13                               ; ----- Вывод сообщения 'Введите B: '
14 000000E8 B8[00000000]      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 8B0D[35000000]    mov ecx,[A] ; 'ecx = A'
26 00000111 890D[00000000]    mov [max],ecx ; 'max = A'
27                               ; ----- Сравниваем 'A' и 'C' (как символы)

```

Рис. 2.17: Ошибка в lab7-2.lst

## 3 Задания для самостоятельной работы

Создадим файл для выполнения самостоятельной работы (рис. 3.1).



```
ivanpr1hodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ touch task1v6.asm
ivanpr1hodko@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 3.1: Создание файла task1v6.asm

Напишем код для первого задания (рис. 3.2).

```

GNU nano 7.2 /home/i
%include 'in_out.asm'
section .data
msg2 db "Наименьшее число: ",0h
A dd '79'
B dd '83'
C dd '41'
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,C
call atoi ; Вызов подпрограммы перевода символа в число
mov [C],eax ; запись преобразованного числа в 'C'

mov eax,A
call atoi ; Вызов подпрограммы перевода символа в число
mov [A],eax ; запись преобразованного числа в 'A'

; ----- 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'
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. 3.2: Код task1v6.asm

Соберем и запустим его (рис. 3.3).

```
ivanprihodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf task1v6.asm
ivanprihodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ ld -m elf_i386 -o task1v6 task1v6.o
ivanprihodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ ./task1v6
Наименьшее число: 41
ivanprihodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 3.3: Сборка и запуск task1v6.asm

Теперь создадим второй файл (рис. 3.4).

```
ivanprihodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ touch task2v6.asm
ivanprihodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 3.4: Создание файла task2v6.asm

Мой код получился таким (рис. 3.5).

```

GNU nano 7.2
#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]
je _equal

imul eax, 5
jmp _print_result

_equal:
add eax, [a]

_print_result:
mov [ans],eax
mov eax, msg3
call sprint
mov eax, [ans]
call iprintLF
call quit

```

Рис. 3.5: Код task2v6.asm

Соберем и проведем тесты (рис. 3.6).

```
ivanprihodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf task2v6.asm
ivanprihodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ ld -m elf_i386 -o task2v6 task2v6.o
ivanprihodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ ./task2v6
Введите X: 2
Введите A: 2
Ответ= 4
ivanprihodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$ ./task2v6
Введите X: 2
Введите A: 1
Ответ= 10
ivanprihodko@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 3.6: Сборка и запуск task2v6.asm

## **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.