

# **Лабораторная работа №7**

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

Казначеев Сергей Ильич

# 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>Выполнение заданий для самостоятельной работы</b>	<b>17</b>

# Список иллюстраций

2.1	01	.....	6
2.2	02	.....	7
2.3	03	.....	8
2.4	04	.....	8
2.5	05	.....	9
2.6	06	.....	9
2.7	07	.....	10
2.8	08	.....	10
2.9	09	.....	11
2.10	10	.....	11
2.11	11	.....	12
2.12	12	.....	12
2.13	13	.....	13
2.14	14	.....	14
2.15	15	.....	15
2.16	16	.....	15
2.17	17	.....	16
3.1	18	.....	17
3.2	19	.....	18
3.3	20	.....	19
3.4	21	.....	20
3.5	22	.....	20
3.6	23	.....	21
3.7	24	.....	22

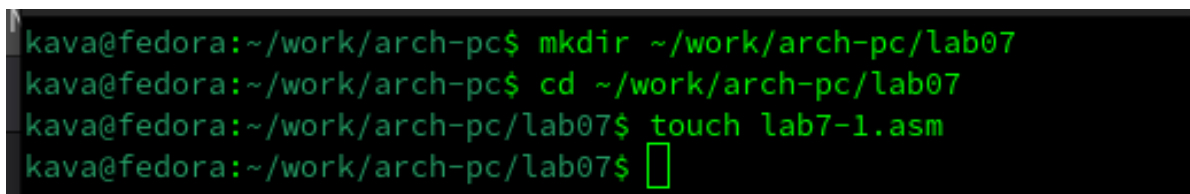
## **Список таблиц**

# 1 Цель работы

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

## 2 Выполнение лабораторной работы

Создаем папку lab07 и файл lab7-1.asm.

A terminal window with a black background and green text. It shows four lines of commands and their execution. The first line creates a directory named 'lab07' in the path '~/work/arch-pc'. The second line changes the current directory to '~/work/arch-pc/lab07'. The third line creates an empty file named 'lab7-1.asm' in the current directory. The fourth line shows the prompt with a cursor, indicating the command has finished.

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

Рис. 2.1: 01

После чего, запускаем Midnight commander для удобства и вставляем код в файл lab7-1.asm из файла листинга.

```

GNU nano 7.2 /home/kava/work/arch-pc/lab07/lab7
#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: 02

Теперь скопируем файл `in_out.asm` из рабочей директории прошлой лабораторной работы.

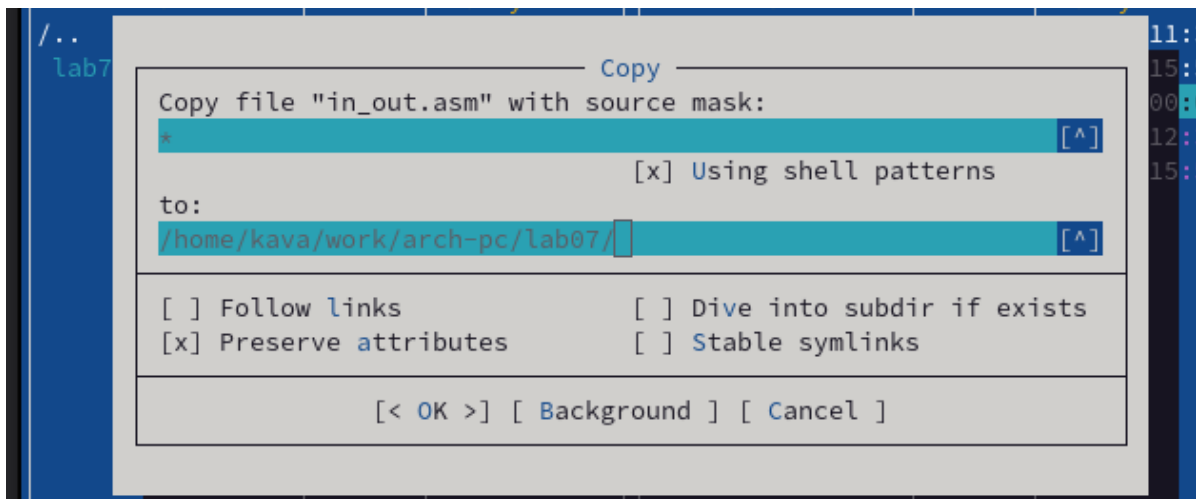


Рис. 2.3: 03

Теперь собираем программу и запускаем ее.

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

Рис. 2.4: 04

После чего, мы меняем файл согласно листингу 7.2



```

GNU nano 7.2 /home/kava/work/arch-pc/lab07/lab7-1.asm
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:

```

Рис. 2.5: 05

Снова соберём программу и запустим её.

```

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

```

Рис. 2.6: 06

Теперь сделаем так, чтобы код выводил сообщения в обратном порядке (от 3 сообщения к первому). Для этого внесём в код следующие изменения.

```
GNU nano 7.2 /home/kava/work/arch-pc/lab07/lab7-1.asm
%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'
```

Рис. 2.7: 07

Снова соберём программу и запустим её.

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

Рис. 2.8: 08

Теперь создаем файл lab7-2.asm.

```
kava@fedora:~/work/arch-pc/lab07$ touch lab7-2.asm
kava@fedora:~/work/arch-pc/lab07$
```

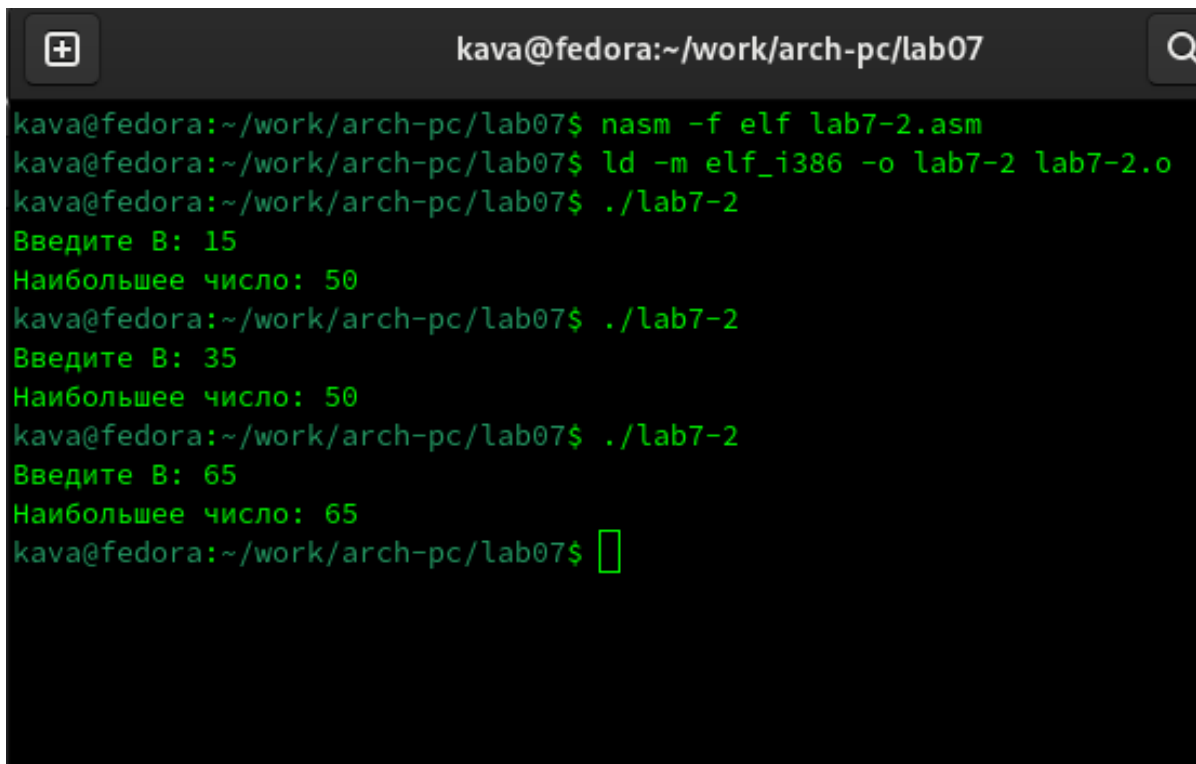
Рис. 2.9: 09

Запишем в него код из листинга 7.3 в файл lab7-2.asm.

```
GNU nano 7.2 /home/kava/work/arch-pc/lab07/lab7-2.asm
%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
```

Рис. 2.10: 10

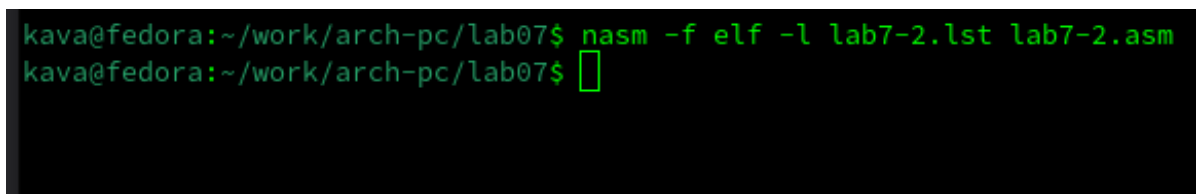
И запустим его, предварительно собрав.

A terminal window with a dark background and green text. The title bar shows 'kava@fedora:~/work/arch-pc/lab07'. The terminal contains the following commands and output:

```
kava@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
kava@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
kava@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 15
Наибольшее число: 50
kava@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 35
Наибольшее число: 50
kava@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 65
Наибольшее число: 65
kava@fedora:~/work/arch-pc/lab07$
```

Рис. 2.11: 11

Теперь попробуем создать файл листинга при сборке файла lab7-2.asm

A terminal window with a dark background and green text. The terminal contains the following commands:

```
kava@fedora:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
kava@fedora:~/work/arch-pc/lab07$
```

Рис. 2.12: 12

Теперь посмотрим, как выглядит файл листинга изнутри. Для этого откроем его в mcedit.

```

ab7-2.lst      [----]  0 L:[ 1+ 0  1/225] *(0  /14458b) 0032 0x020
1              %include 'in_out.asm'
1              <1> ;----- slen -----
2              <1> ; Функция вычисления длины сообще
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>

```

Рис. 2.13: 13

Вот наша программа.

```

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 'Введ
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

```

Рис. 2.14: 14

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 шестнадцатиричной 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 пробуем допустить ошибку убрав у команды move 1 операнд.

```

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
call sread
; ----- Преобразование 'В' из символа в число
mov eax,B

```

Рис. 2.15: 15

Пробуем собрать файл с ошибками.

```

kava@fedora:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:18: error: invalid combination of opcode and operands
kava@fedora:~/work/arch-pc/lab07$ 

```

Рис. 2.16: 16

Теперь зайдём в файл листинг и посмотрим отображается ли в нём ошибка.

```

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
19  000000F7 E847FFFFFF    call sread
20      ; ----- Преобразование 'B' из символа
21  000000FC B8[0A000000]    mov eax,B
22  00000101 E896FFFFFF    call atoi ; Вызов подпрограммы перевода
23  00000106 A3[0A000000]    mov [B],eax ; запись преобразованного ч
24      ; ----- Записываем 'A' в переменную
25  0000010B 8B0D[35000000]    mov ecx,[A] ; 'ecx = A'
26  00000111 890D[00000000]    mov [max],ecx ; 'max = A'
27      ; ----- Сравниваем 'A' и 'C' (как

```

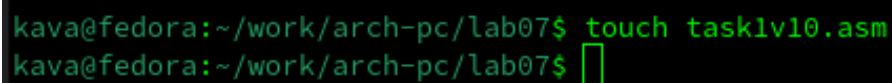
Рис. 2.17: 17

Как видим появилась ошибка



### 3 Выполнение заданий для самостоятельной работы

Создадим файл для выполнения самостоятельной работы. Мой вариант-10



```
kava@fedora:~/work/arch-pc/lab07$ touch tasklv10.asm
kava@fedora:~/work/arch-pc/lab07$
```

Рис. 3.1: 18

Напиши код для выполнения задания.

```

GNU nano 7.2
%include 'in_out.asm'
section .data
msg2 db "Наименьшее число: ",0h
A dd '41'
B dd '62'
C dd '35'
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

```

Рис. 3.2: 19

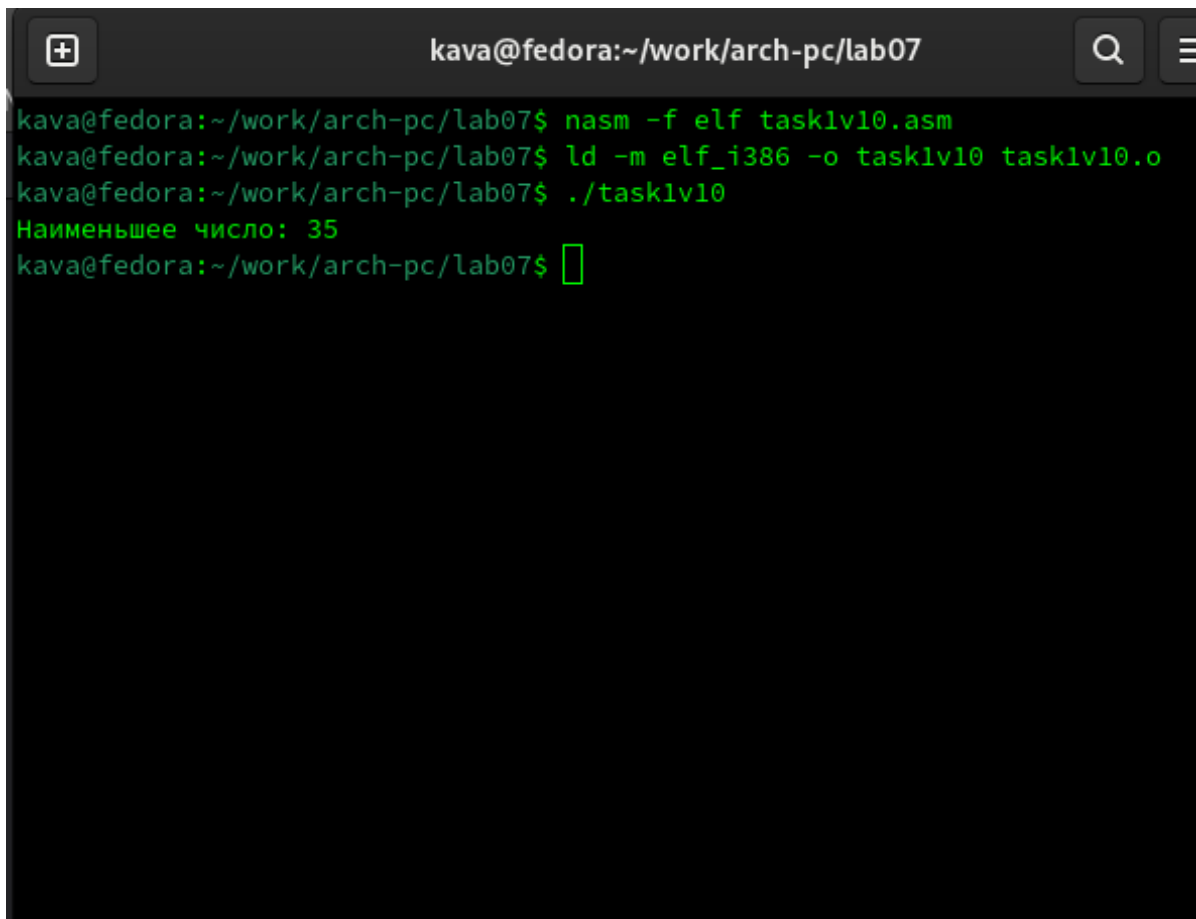
```

check_B:
; ----- Сравниваем 'min(A,C)' и 'B' (как числа)
mov ecx,[min]
cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
jle 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. 3.3: 20

Соберем и запустим его.

A terminal window with a dark background. The title bar shows 'kava@fedora:~/work/arch-pc/lab07'. The terminal contains the following text:

```
kava@fedora:~/work/arch-pc/lab07$ nasm -f elf task1v10.asm
kava@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o task1v10 task1v10.o
kava@fedora:~/work/arch-pc/lab07$ ./task1v10
Наименьшее число: 35
kava@fedora:~/work/arch-pc/lab07$
```

Рис. 3.4: 21

Теперь создаем второй файл для второго задания.

A terminal window with a dark background. The title bar shows 'kava@fedora:~/work/arch-pc/lab07'. The terminal contains the following text:

```
kava@fedora:~/work/arch-pc/lab07$ touch task2v10.asm
kava@fedora:~/work/arch-pc/lab07$
```

Рис. 3.5: 22

И напишем следующий код.

```

/home/kava/work/arch-pc/lab07/task2v10.asm
ans: RESB 80
section .text
global _start
_start:
; Ввод X
mov eax,msg1
call sprint
mov ecx,x
mov edx,80
call sread
mov eax,x
call atoi
mov [x],eax

; Ввод A
mov eax,msg2
call sprint
mov ecx,a
mov edx,80
call sread
mov eax,a
call atoi
mov [a],eax

; Проверка условия X > 2
mov eax, [x]
cmp eax, 2
jg x_greater_2 ; Переход, если X > 2

; Если X <= 2, вычисляем X - 2
mov eax, [x]
sub eax, 2
jmp ansv

x_greater_2:
; Если X > 2, вычисляем 3 * A
mov eax, [a]
mov ebx, 3
mul ebx ; Умножаем eax на ebx, результат в eax

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

```

Рис. 3.6: 23

Соберём и запустим этот файл.

```
kava@fedora:~/work/arch-pc/lab07$ nasm -f elf task2v10.asm
kava@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o task2v10 task2v10.o
kava@fedora:~/work/arch-pc/lab07$ ./task1v10
Наименьшее число: 35
kava@fedora:~/work/arch-pc/lab07$ ./task2v10
Введите X: 10
Введите A: 5
Ответ=15
```

Рис. 3.7: 24

Как видим программа всё посчитала правильно. # Выводы После выполнения лабораторной работы. Я изучил команды условного и безусловного перехода. Приобрёл навыки написания программ с использованием переходов, познакомился с назначением и структурой файла листинга # Список литературы{unnumbered}