

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютера

Зиани Сид Ахмед

Содержание

1	Цель работы	5
2	Задание	6
2.1	1. Команды условного перехода	6
2.2	2. Реализация переходов в NASM	6
2.3	3. Изучение структуры файлы листинга	6
2.4	4. Самостоятельная работа	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Самостоятельная работа	19
6	Выводы	28

Список иллюстраций

4.1	Создание директории	8
4.2	Создание копии файла для дальнейшей работы, редактирование файла	9
4.3	Запуск исполняемого файла	10
4.4	Редактирование программы	10
4.5	Создание исполняемого файла	11
4.6	Создание файла	11
4.7	Вставляю текст в файл	12
4.8	Вставляю текст в файл	13
4.9	Запуск исполняемого файла	13
4.10	Запуск исполняемого файла	14
4.11	Файл листинга	14
4.12	Файл листинга	16
4.13	asm -f elf -l lab7-2.lst lab7-2.asm	16
4.14	gedit lab7-2.lst	17
4.15	17
4.16	18
5.1	Создание запуск файла	19
5.2	Редактирование файла	20
5.3	Запуск исполняемого файла	20
5.4	создание файла	23
5.5	ввод программы в файл	24
5.6	Создание исполняемого файла	24
5.7	запуск исполняемого файла	25

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 Задание

2.1 1. Команды условного перехода

2.2 2. Реализация переходов в NASM

2.3 3. Изучение структуры файлы листинга

2.4 4. Самостоятельная работа

3 Теоретическое введение

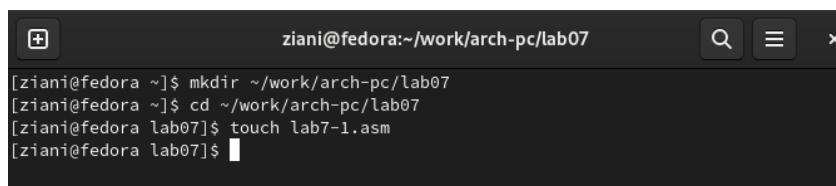
Для реализации ветвлений в ассемблере используются так называемые команды передачи управления или команды перехода. Можно выделить 2 типа переходов:

- 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

1

С помощью утилиты `mkdir` создаю директорию `lab07`, перехожу в нее и создаю файл для работы. (рис. [4.1]).

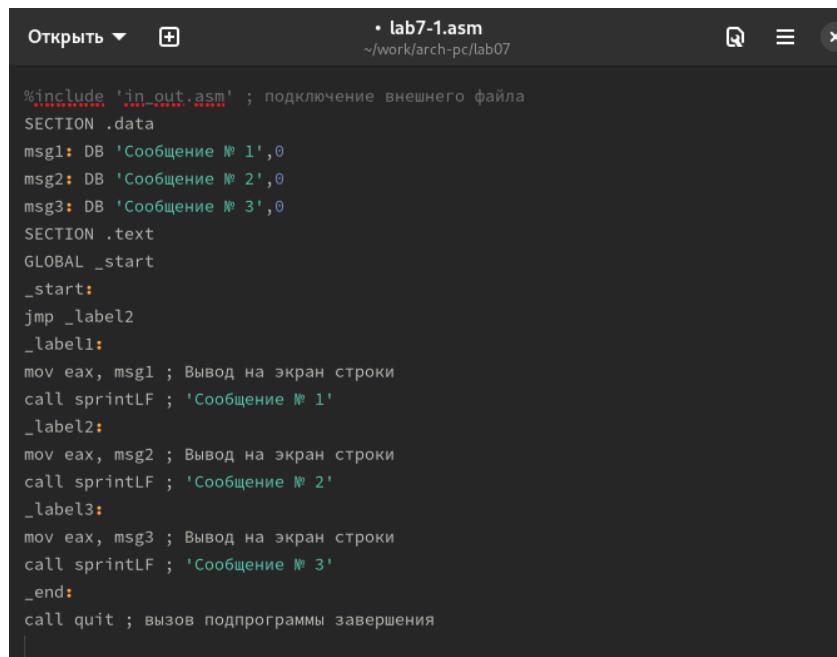


```
ziani@fedora:~/work/arch-pc/lab07
[ziani@fedora ~]$ mkdir ~/work/arch-pc/lab07
[ziani@fedora ~]$ cd ~/work/arch-pc/lab07
[ziani@fedora lab07]$ touch lab7-1.asm
[ziani@fedora lab07]$
```

Рис. 4.1: Создание директории

2

Копирую в текущий каталог файл `in_out.asm` из загрузок, т.к. он будет использоваться в других программах. Открываю созданный файл `lab7-1.asm`, вставляю в него программу реализации безусловных переходов(рис. [fig002?]).



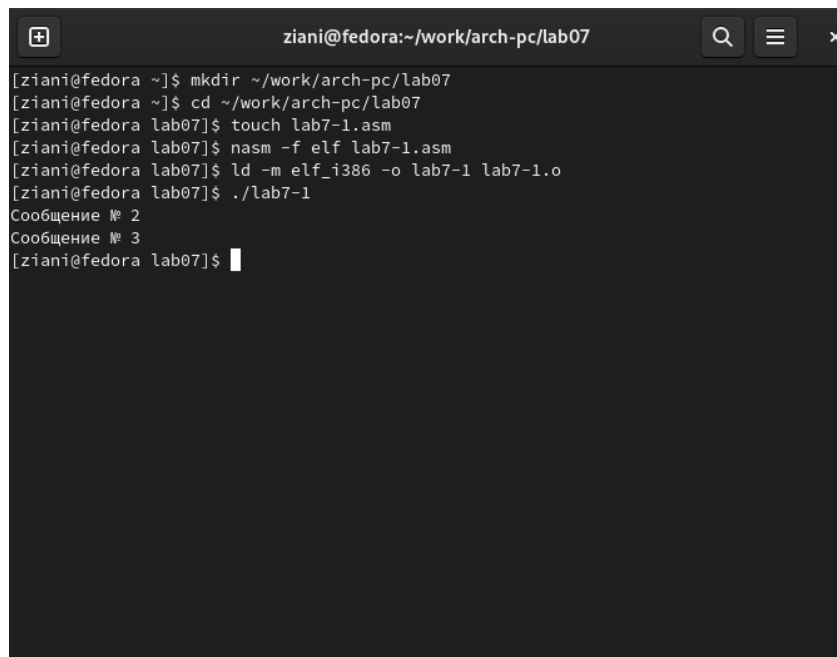
```
Открыть ▾ + lab7-1.asm ~/work/arch-pc/lab07

%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: Создание копии файла для дальнейшей работы, редактирование файла

3

Создаю исполняемый файл программы и запускаю его (рис. [4.3]). Инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`.

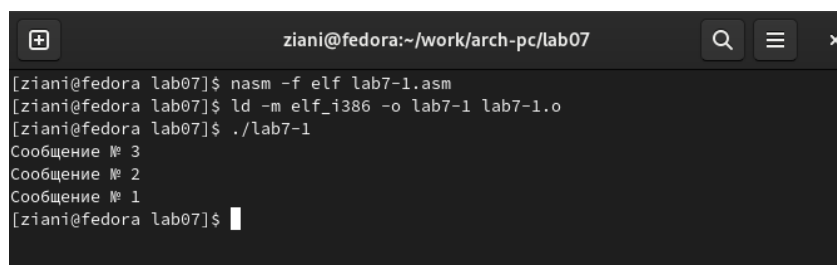


```
ziani@fedora:~/work/arch-pc/lab07
[ziani@fedora ~]$ mkdir ~/work/arch-pc/lab07
[ziani@fedora ~]$ cd ~/work/arch-pc/lab07
[ziani@fedora lab07]$ touch lab7-1.asm
[ziani@fedora lab07]$ nasm -f elf lab7-1.asm
[ziani@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[ziani@fedora lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 3
[ziani@fedora lab07]$
```

Рис. 4.3: Запуск исполняемого файла

4

Изменяю текст программы, так чтобы вывод происходил в обратном порядке (рис. [4.4]).

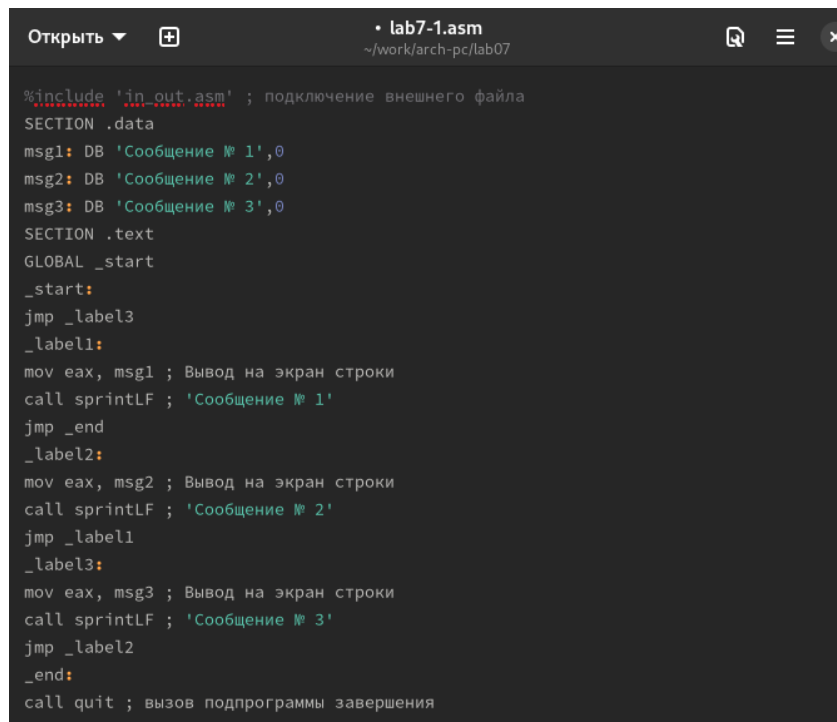


```
ziani@fedora:~/work/arch-pc/lab07
[ziani@fedora lab07]$ nasm -f elf lab7-1.asm
[ziani@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[ziani@fedora lab07]$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
[ziani@fedora lab07]$
```

Рис. 4.4: Редактирование программы

5

Создаю исполняемый файл и проверяю работу программы (рис. [4.5]). Программа отработало верно.



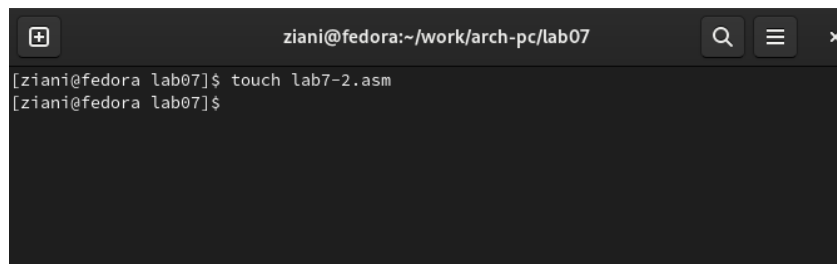
```
Открыть ▾ + lab7-1.asm ~/work/arch-pc/lab07

%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 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
```

Рис. 4.5: Создание исполняемого файла

6

Создаю новый файл lab7-2.asm для программы с условным оператором. (рис. [4.6]).

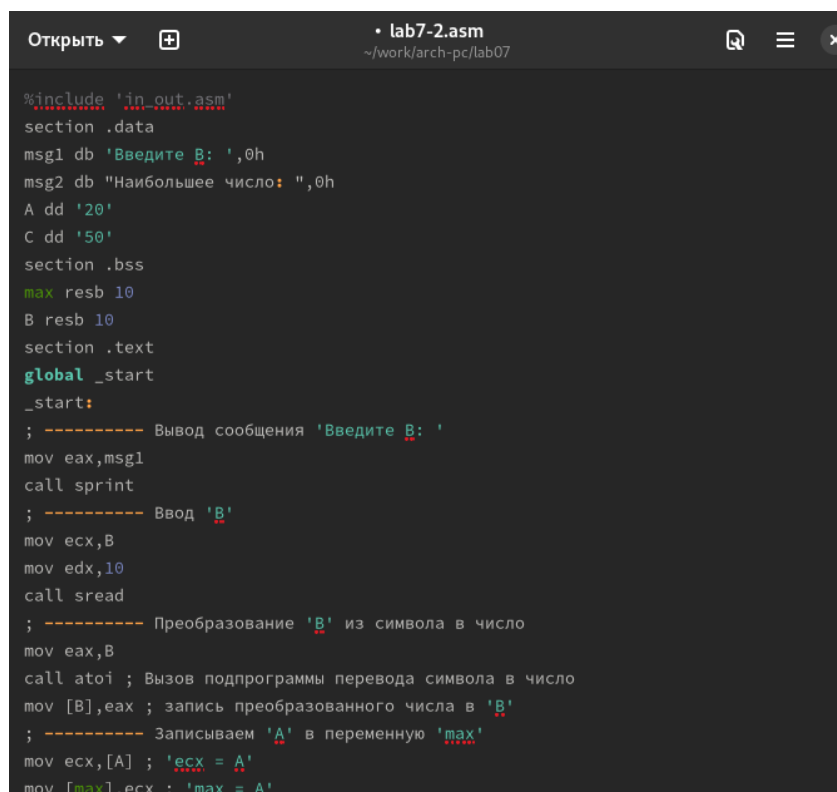


```
ziani@fedora:~/work/arch-pc/lab07
[ziani@fedora lab07]$ touch lab7-2.asm
[ziani@fedora lab07]$
```

Рис. 4.6: Создание файла

7

Вставляю программу, которая определяет и выводит на экран наибольшее число (рис.[4.8]).




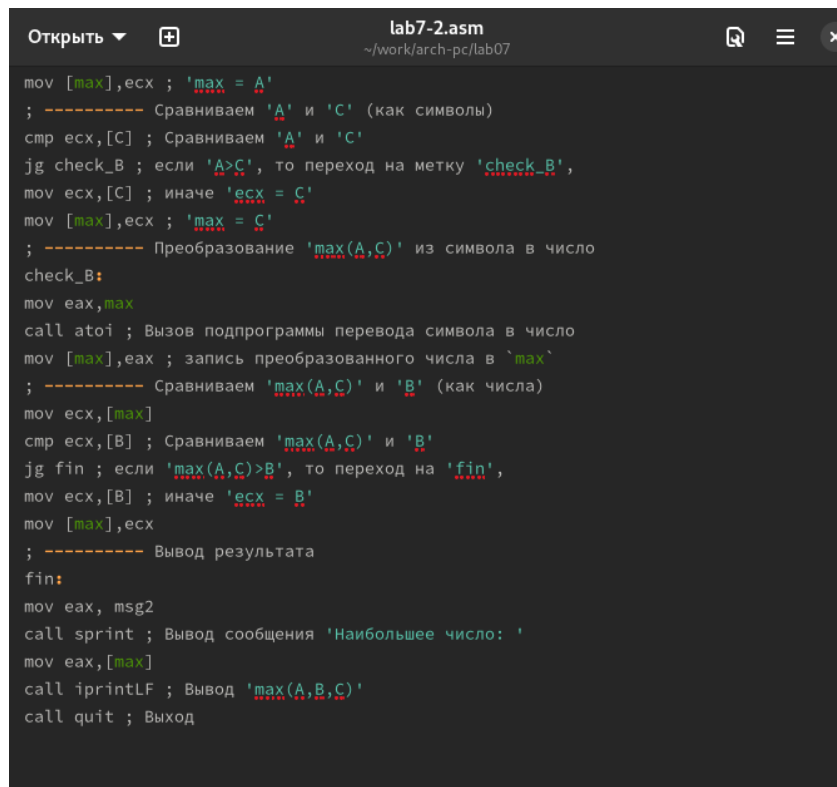
```
Открыть ▾  • lab7-2.asm  
~/work/arch-pc/lab07  
  
%include 'in-out.asm'  
section .data  
msg1 db 'Введите B: ',0h  
msg2 db "Наибольшее число: ",0h  
A dd '20'  
C dd '50'  
section .bss  
max resb 10  
B resb 10  
section .text  
global _start  
_start:  
; ----- Вывод сообщения 'Введите B: '  
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'
```

Рис. 4.7: Вставляю текст в файл



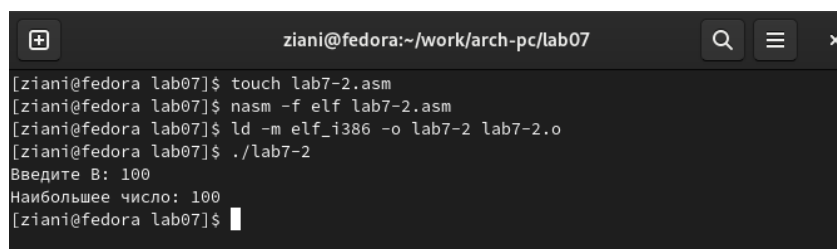
```
Открыть  lab7-2.asm
~/work/arch-pc/lab07

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
```

Рис. 4.8: Вставляю текст в файл

8

Создаю и запускаю новый исполняемый файл, проверяю работу программы (рис. [4.10]).



```
ziani@fedora:~/work/arch-pc/lab07

[ziani@fedora lab07]$ touch lab7-2.asm
[ziani@fedora lab07]$ nasm -f elf lab7-2.asm
[ziani@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[ziani@fedora lab07]$ ./lab7-2
Введите B: 100
Наибольшее число: 100
[ziani@fedora lab07]$
```

Рис. 4.9: Запуск исполняемого файла

```
ziani@fedora:~/work/arch-pc/lab07
[ziani@fedora lab07]$ nasm -f elf lab7-2.asm
[ziani@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[ziani@fedora lab07]$ ./lab7-2
Введите B: 1
Наибольшее число: 50
[ziani@fedora lab07]$
```

Рис. 4.10: Запуск исполняемого файла

9

Открываю файл листинга с помощью редактора mcedit. Рассмотрим 9-11 строки: (рис. [4.11]).

```
GNU nano 7.2 /home/dayanchberdyev/work/arch-pc/lab07/lab7-2.asm
#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'
```

Рис. 4.11: Файл листинга

9 строка:

- Первые цифры [9] - это номер строки файла листинга.

- Следующие цифры [00000006] адрес — это смещение машинного кода от начала текущего сегмента, состоит из 8 чисел.
- следующие числа [7403] - это машинный код, который представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности, поэтому и появляются буквы латинского алфавита.
- следующее [jz finished] - исходный текст программы, которая просто состоит из строк исходной программы вместе с комментариями.

10 строка:

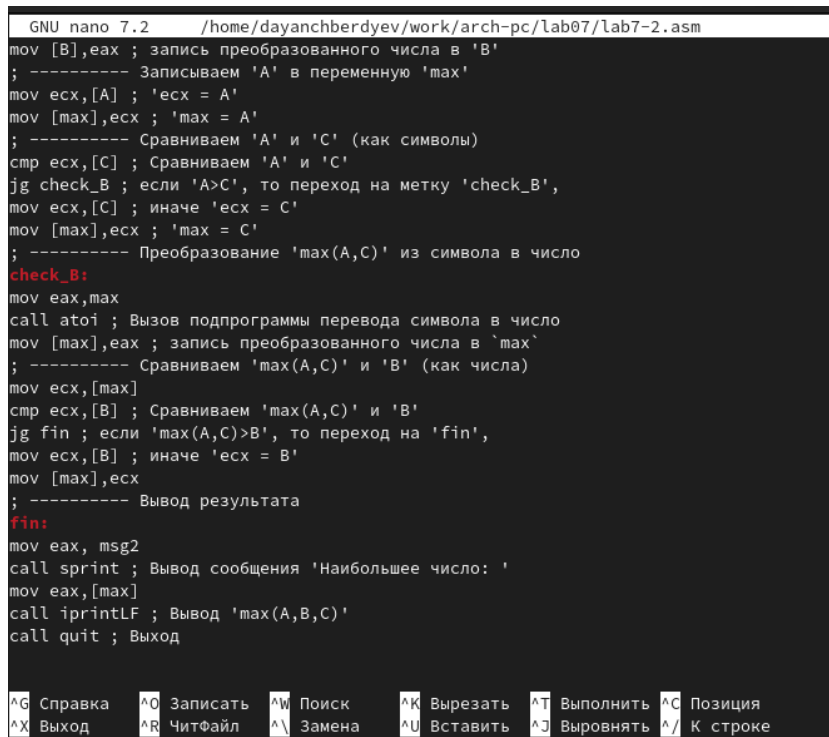
- Первые цифры [10] - это номер строки файла листинга.
- Следующие цифры [00000008] адрес — это смещение машинного кода от начала текущего сегмента, состоит из 8 чисел.
- следующие числа [40] - это машинный код, который представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности, поэтому и появляются буквы латинского алфавита.
- следующее [inc eax] - исходный текст программы, которая просто состоит из строк исходной программы вместе с комментариями

11 строка:

- Первые цифры [11] - это номер строки файла листинга.
- Следующие цифры [00000009] адрес — это смещение машинного кода от начала текущего сегмента, состоит из 8 чисел.
- следующие числа [EBF8] - это машинный код, который представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности, поэтому и появляются буквы латинского алфавита.
- следующее [jmp nextchar] - исходный текст программы, которая просто состоит из строк исходной программы вместе с комментариями

10

Открываю файл листинга с помощью редактора mcedit и замечаю, что в файле листинга появляется ошибка. (рис. [4.12]).



```
GNU nano 7.2 /home/dayanchberdyev/work/arch-pc/lab07/lab7-2.asm
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 sprintf ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintLF ; Вывод 'max(A,B,C)'
call quit ; Выход

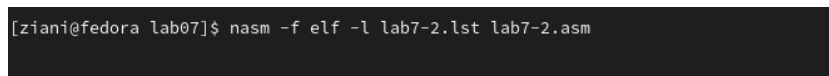
^G Справка  ^O Записать  ^W Поиск    ^K Вырезать ^T Выполнить ^C Позиция
^X Выход    ^R Читфайл  ^\ Замена   ^U Вставить ^J Выровнять ^/ К строке
```

Рис. 4.12: Файл листинга

Отсюда можно сделать вывод, что, если в коде появляется ошибка, то ее описание появится в файле листинга

11

Создал файл листинга для программы из файла lab7-2.asm (рис. [4.13]).



```
[ziani@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
```

Рис. 4.13: asm -f elf -l lab7-2.lst lab7-2.asm

12

Открыл файл листинга lab7-2.lst с помощью любого текстового редактора, например gedit: {#fig:012 width=70%}


```
[ziani@fedora lab07]$ gedit lab7-2.lst
```

Рис. 4.14: gedit lab7-2.lst

13

Открыл файл с программой lab7-2.asm и в любой инструкции с двумя операндами удалить один операнд. Выполните трансляцию с получением файла листинга: {#fig:013 width=70%}

```
1 1 %include 'in_out.asm'
2 1 <1> ;----- slen
3 2 <1> ; Функция вычисления длины сообщения
4 3 <1> slen:
5 4 00000000 53 <1> push ebx
6 5 00000001 89C3 <1> mov ebx, eax
7 6 <1>
8 7 <1> nextchar:
9 8 00000003 803800 <1> cmp byte [eax], 0
10 9 00000006 7403 <1> jz finished
11 10 00000008 40 <1> inc eax
12 11 00000009 EBF8 <1> jmp nextchar
13 12 <1>
14 13 <1> finished:
15 14 0000000B 29D8 <1> sub eax, ebx
16 15 0000000D 5B <1> pop ebx
17 16 0000000E C3 <1> ret
18 17 <1>
19 18 <1>
20 19 <1> ;----- sprint
21 20 <1> ; Функция печати сообщения
22 21 <1> ; входные данные: mov eax, <message>
23 22 <1> sprint:
24 23 0000000F 52 <1> push edx
25 24 00000010 51 <1> push ecx
26 25 00000011 53 <1> push ebx
27 26 00000012 50 <1> push eax
28 27 00000013 E8E8FFFFFF <1> call slen
29 28 <1>
```

Рис. 4.15:

Открыть

+

lab7-2.lst

Сохранить

☰

✕

~/work/arch-pc/lab07

```

30 29 00000018 89C2      <1>    mov     edx, eax
31 30 0000001A 58      <1>    pop      eax
32 31      <1>
33 32 0000001B 89C1      <1>    mov     ecx, eax
34 33 0000001D BB01000000    <1>    mov     ebx, 1
35 34 00000022 B804000000    <1>    mov     eax, 4
36 35 00000027 CD80      <1>    int      80h
37 36      <1>
38 37 00000029 5B      <1>    pop      ebx
39 38 0000002A 59      <1>    pop      ecx
40 39 0000002B 5A      <1>    pop      edx
41 40 0000002C C3      <1>    ret
42 41      <1>
43 42      <1>
44 43      <1> ;----- sprintLF
-----
45 44      <1> ; Функция печати сообщения с переводом
строки
46 45      <1> ; входные данные: mov eax,<message>
47 46      <1> sprintLF:
48 47 0000002D E8DDFFFFFF    <1>    call    sprint
49 48      <1>
50 49 00000032 50      <1>    push    eax
51 50 00000033 B80A000000    <1>    mov     eax, 0AH
52 51 00000038 50      <1>    push    eax
53 52 00000039 89E0      <1>    mov     eax, esp
54 53 0000003B E8CFFFFFFF    <1>    call    sprint
55 54 00000040 58      <1>    pop     eax
56 55 00000041 58      <1>    pop     eax
57 56 00000042 C3      <1>    ret
58 57      <1>

```

Текст

Ширина табуляции: 8

Ln 1, Col 1

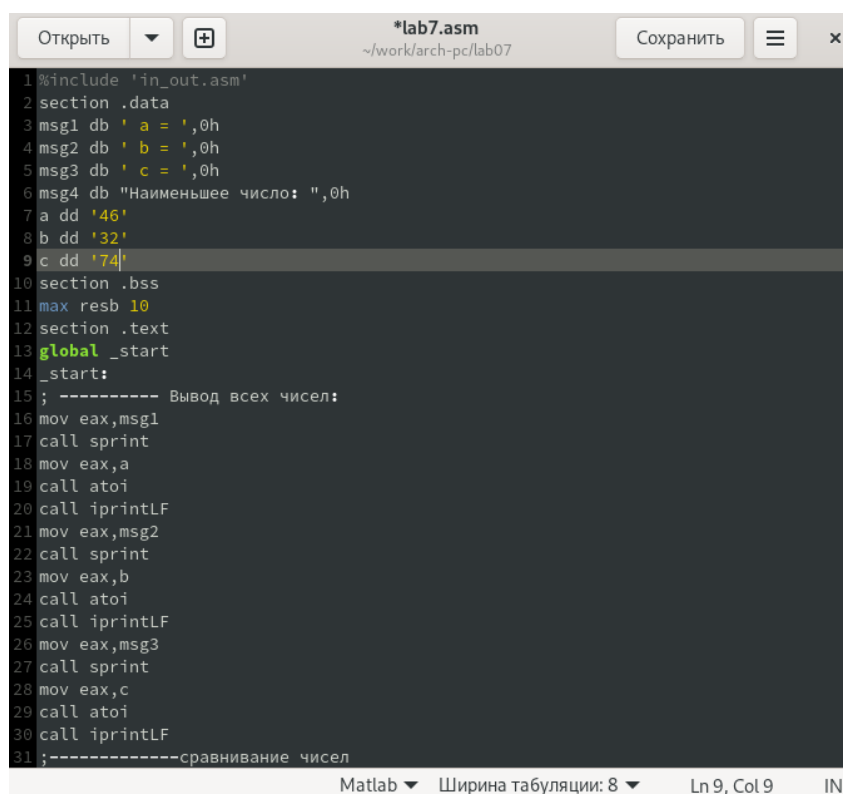
IN

Рис. 4.16:

5 Самостоятельная работа

1

Создаю файл lab7.asm с помощью утилиты touch и запускаю редактора gedit (рис. [5.1]).



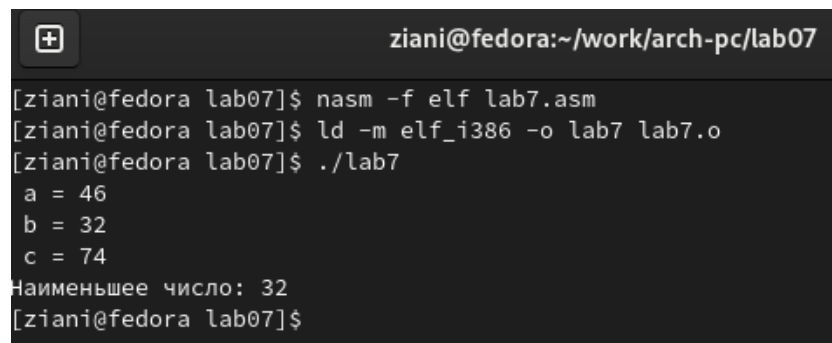
```
1 %include 'in_out.asm'
2 section .data
3 msg1 db ' a = ',0h
4 msg2 db ' b = ',0h
5 msg3 db ' c = ',0h
6 msg4 db "Наименьшее число: ",0h
7 a dd '46'
8 b dd '32'
9 c dd '74'
10 section .bss
11 max resb 10
12 section .text
13 global _start
14 _start:
15 ; ----- Вывод всех чисел:
16 mov eax,msg1
17 call sprint
18 mov eax,a
19 call atoi
20 call iprintlnLF
21 mov eax,msg2
22 call sprint
23 mov eax,b
24 call atoi
25 call iprintlnLF
26 mov eax,msg3
27 call sprint
28 mov eax,c
29 call atoi
30 call iprintlnLF
31 ;-----сравнение чисел
```

Рис. 5.1: Создание запуск файла

2

Ввожу в созданный файл текст программы для вычисления наименьшего из 3 чисел. Числа беру, учитывая свой вариант из прошлой лабораторной работы. 19

вариант (рис. [5.2]).

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

```
[ziani@fedora lab07]$ nasm -f elf lab7.asm
[ziani@fedora lab07]$ ld -m elf_i386 -o lab7 lab7.o
[ziani@fedora lab07]$ ./lab7
a = 46
b = 32
c = 74
Наименьшее число: 32
[ziani@fedora lab07]$
```

Рис. 5.2: Редактирование файла

3

Создаю исполняемый файл и запускаю его (рис. [5.3]).

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

```
[ziani@fedora lab07]$ touch lab7-3.asm
[ziani@fedora lab07]$
```

Рис. 5.3: Запуск исполняемого файла

Текст программы

```
%include 'in_out.asm'

section .data
msg1 db ' a = ',0h
msg2 db ' b = ',0h
msg3 db ' c = ',0h
msg4 db "Наименьшее число: ",0h
a dd '92'
```

```

b dd '2'
c dd '61'

section .bss
max resb 10

section .text
global _start
_start:
; ----- Вывод всех чисел:
mov eax,msg1
call sprint
mov eax,a
call atoi
call iprintLF

mov eax,msg2
call sprint
mov eax,b
call atoi
call iprintLF

mov eax,msg3
call sprint
mov eax,c
call atoi
call iprintLF

;-----сравнивание чисел

```

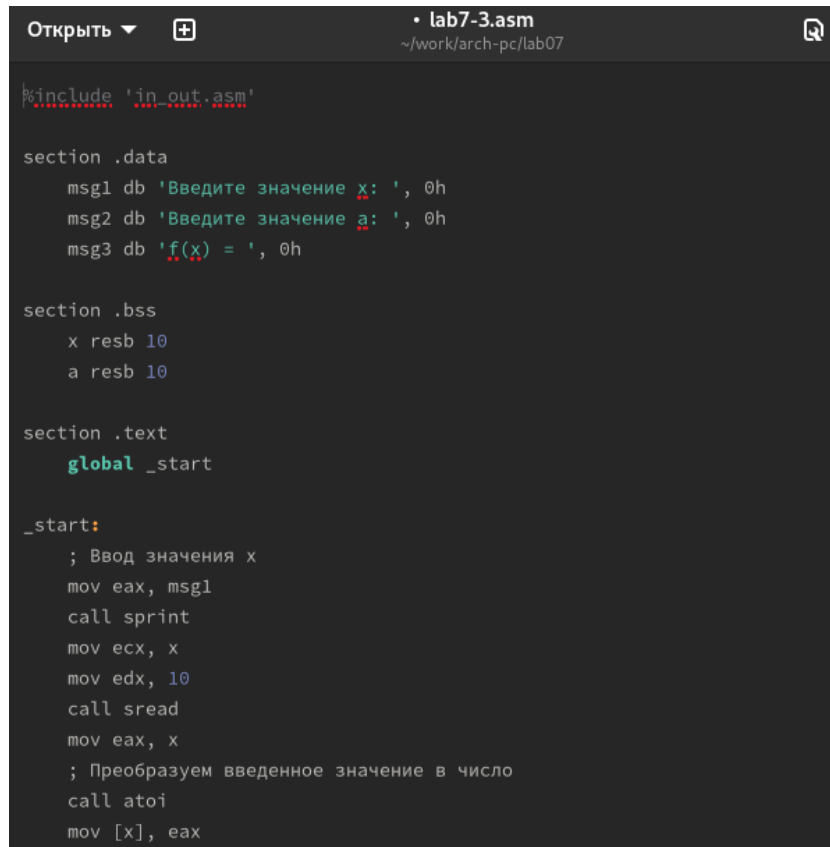
```

mov eax,b
call atoi ;перевод символа в число
mov [b],eax ; запись преобразованного числа в b
;----- запись b в переменную max
mov ecx,[a] ;
mov [max],ecx ;
;-----сравнивание чисел a с
cmp ecx,[c]; if a>c
jl check_b ; то перход на метку
mov ecx,[c] ;
mov [max],ecx ;
;-----метка check_b
check_b:
mov eax,max ;
call atoi
mov [max],eax ;
;-----
mov ecx,[max] ;
cmp ecx,[b] ;
jl check_c ;
mov ecx,[b] ;
mov [max],ecx ;
;-----
check_c:
mov eax,msg4 ;
call sprint ;
mov eax,[max];
call iprintLF ;
call quit

```

4

Создаю новый файл lab7-3 для написания программы второго задания. (рис. [5.4]).



```
Открыть ▾ + lab7-3.asm
~/work/arch-pc/lab07

%include 'in_out.asm'

section .data
    msg1 db 'Введите значение x: ', 0h
    msg2 db 'Введите значение a: ', 0h
    msg3 db 'f(x) = ', 0h

section .bss
    x resb 10
    a resb 10

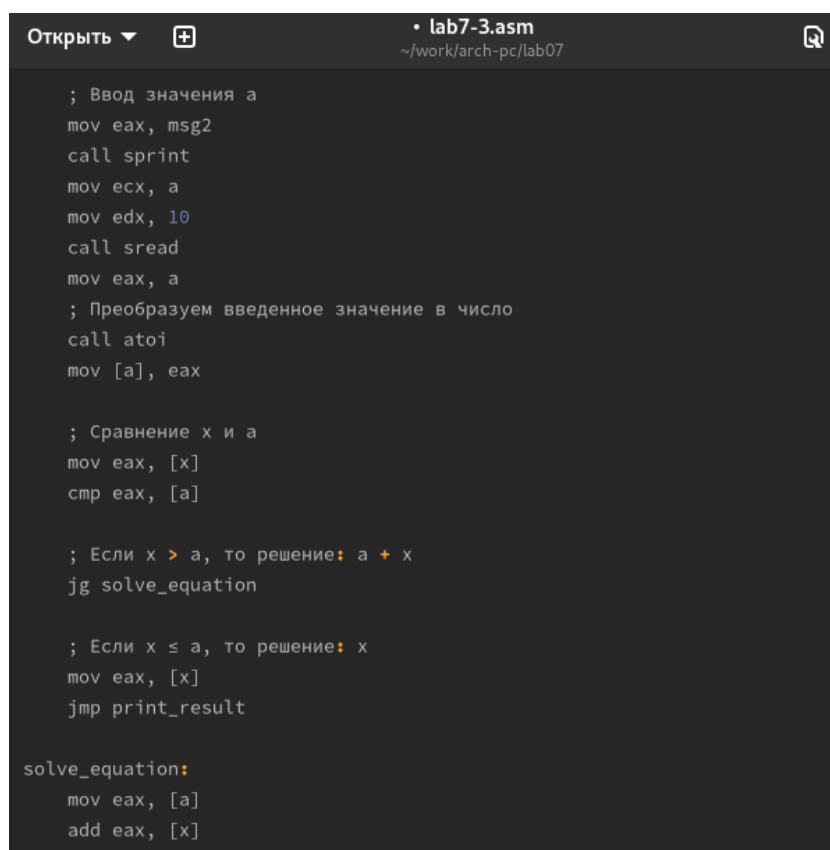
section .text
    global _start

_start:
    ; Ввод значения x
    mov eax, msg1
    call sprint
    mov ecx, x
    mov edx, 10
    call sread
    mov eax, x
    ; Преобразуем введенное значение в число
    call atoi
    mov [x], eax
```

Рис. 5.4: создание файла

5

Ввожу в него программу, в которую ввожу значения 19 x и a, и которая выводит значения функции. Функцию беру из таблицы в соответствии со своим вариантом (рис. [5.5]).



```
Открыть ▾ + lab7-3.asm
~/work/arch-pc/lab07

; Ввод значения a
mov eax, msg2
call sprint
mov ecx, a
mov edx, 10
call sread
mov eax, a
; Преобразуем введенное значение в число
call atoi
mov [a], eax

; Сравнение x и a
mov eax, [x]
cmp eax, [a]

; Если x > a, то решение: a + x
jg solve_equation

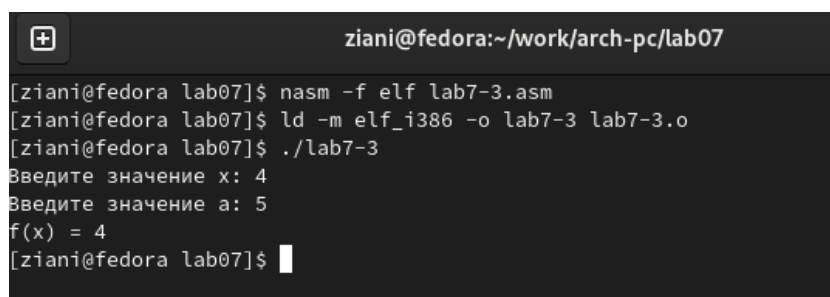
; Если x ≤ a, то решение: x
mov eax, [x]
jmp print_result

solve_equation:
mov eax, [a]
add eax, [x]
```

Рис. 5.5: ввод программы в файл

6

Создаю исполняемый файл и проверяю её выполнение при $x=4$, $a=5$



```
ziani@fedora:~/work/arch-pc/lab07

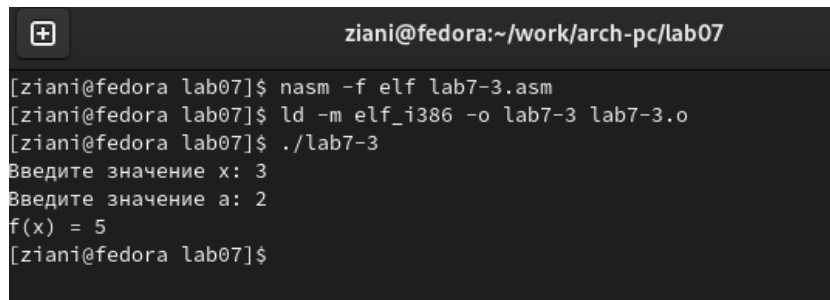
[ziani@fedora lab07]$ nasm -f elf lab7-3.asm
[ziani@fedora lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[ziani@fedora lab07]$ ./lab7-3
Введите значение x: 4
Введите значение a: 5
f(x) = 4
[ziani@fedora lab07]$
```

Рис. 5.6: Создание исполняемого файла

7

Повторный раз запускаю программу и проверяю ее выполнение при $x=3$ и $a=2$

Программа отработала верно!



```
ziani@fedora:~/work/arch-pc/lab07
[ziani@fedora lab07]$ nasm -f elf lab7-3.asm
[ziani@fedora lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[ziani@fedora lab07]$ ./lab7-3
Введите значение x: 3
Введите значение a: 2
f(x) = 5
[ziani@fedora lab07]$
```

Рис. 5.7: запуск исполняемого файла

Текст программы

```
%include 'in_out.asm'

section .data
    msg1 db 'Введите значение x: ', 0h
    msg2 db 'Введите значение a: ', 0h
    msg3 db 'f(x) = ', 0h

section .bss
    x resb 10
    a resb 10

section .text
    global _start

_start:
    ; Ввод значения x
    mov eax, msg1
    call sprint
    mov ecx, x
```

```

mov edx, 10
call sread
mov eax, x
; Преобразуем введенное значение в число
call atoi
mov [x], eax

; Ввод значения a
mov eax, msg2
call sprint
mov ecx, a
mov edx, 10
call sread
mov eax, a
; Преобразуем введенное значение в число
call atoi
mov [a], eax

; Сравнение x и a
mov eax, [x]
cmp eax, [a]

; Если  $x > a$ , то решение:  $a + x$ 
jg solve_equation

; Если  $x \leq a$ , то решение:  $x$ 
mov eax, [x]
jmp print_result

```

solve_equation:

mov **eax**, [a]

add **eax**, [x]

print_result:

mov **ebx**, **eax**

mov **eax**, msg3

call sprint

mov **eax**, **ebx**

call iprintLF

; Завершаем программу

call quit

6 Выводы

При выполнении данной лабораторной работы я освоил инструкции условного и безусловного вывода и ознакомился с структурой файла листинга.

:: {#refs} ::