

Лабораторная работа №7

Дисциплина: Архитектура компьютера

Малюга Валерия Васильевна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	7
4.1	Реализация переходов в NASM	7
4.2	Изучение структуры файлы листинга	13
4.3	Задание для самостоятельной работы	14
5	Выводы	20
	Список литературы	21

Список иллюстраций

4.1	Создание каталога и файла lab7-1.ams	7
4.2	Ввод текста программы из листинга 7.1	8
4.3	Создание исполняемого файла и его запуск	8
4.4	Изменение текста программы	9
4.5	Создание исполняемого файла и проверка его работы	9
4.6	Изменение текста программы и вывод	11
4.7	Создание файла lab7-2.asm	12
4.8	Создание исполняемого файла и проверка его работы	12
4.9	Файл листинга lab7-2.asm	13
4.10	Удаление операнда из инструкции с двумя операндами	13
4.11	Ошибка трансляции кода	14
4.12	Программа для нахождения наименьшей из 3 переменных	14
4.13	Программа вычисления функции	16

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

1. Реализация переходов в NASM.
2. Изучение структуры файлы листинга.
3. Задания 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`. 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.

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.

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

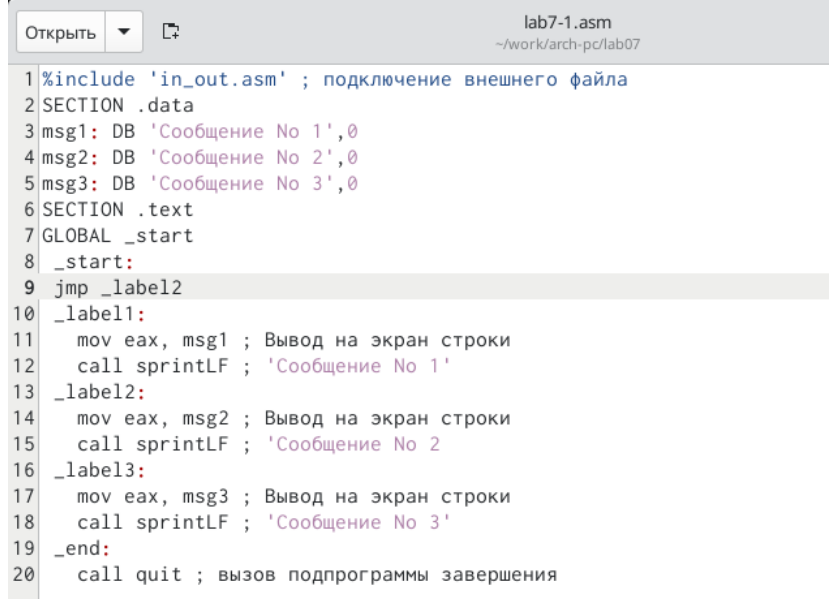
Создала каталог для программ лабораторной работы № 7, перешла в него и создала файл lab7-1.asm (рис. 4.1).

```
vvmalyuga@dk8n74 ~ $ mkdir ~/work/arch-pc/lab07
vvmalyuga@dk8n74 ~ $ cd ~/work/arch-pc/lab07
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ touch lab7-1.asm
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ ls
lab7-1.asm
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $
```

Рис. 4.1: Создание каталога и файла lab7-1.asm

Скопировала файл in_out.asm, так как он будет использоваться в программах. Ввела в файл lab7-1.asm текст программы из листинга 7.1. (рис. 4.2).

```
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ cp ~/work/arch-pc/lab06/in_out.asm ~/work/arch-pc/lab07
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ ls
in_out.asm lab7-1.asm
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ gedit lab7-1.asm
```



```
1 %include 'in_out.asm' ; подключение внешнего файла
2 SECTION .data
3 msg1: DB 'Сообщение No 1',0
4 msg2: DB 'Сообщение No 2',0
5 msg3: DB 'Сообщение No 3',0
6 SECTION .text
7 GLOBAL _start
8 _start:
9 jmp _label2
10 _label1:
11     mov eax, msg1 ; Вывод на экран строки
12     call sprintf ; 'Сообщение No 1'
13 _label2:
14     mov eax, msg2 ; Вывод на экран строки
15     call sprintf ; 'Сообщение No 2'
16 _label3:
17     mov eax, msg3 ; Вывод на экран строки
18     call sprintf ; 'Сообщение No 3'
19 _end:
20     call quit ; вызов подпрограммы завершения
```

Рис. 4.2: Ввод текста программы из листинга 7.1

Создала исполняемый файл и запустила его (рис. 4.3). Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения.

```
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ gedit lab7-1.asm
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 3
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $
```

Рис. 4.3: Создание исполняемого файла и его запуск

Изменила программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого изменила текст программы в соответствии с листингом 7.2. (рис. 4.4).


```
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ gedit lab7-1.asm
Открыть lab7-1.asm
~/work/arch-pc/lab07
1 %include 'in_out.asm' ; подключение внешнего файла
2 SECTION .data
3 msg1: DB 'Сообщение № 1',0
4 msg2: DB 'Сообщение № 2',0
5 msg3: DB 'Сообщение № 3',0
6
7 SECTION .text
8 GLOBAL _start
9 _start:
10
11 jmp _label2
12
13 _label1:
14 mov eax, msg1 ; Вывод на экран строки
15 call sprintf ; 'Сообщение No 1'
16 jmp _end
17 _label2:
18 mov eax, msg2 ; Вывод на экран строки
19 call sprintf ; 'Сообщение No 2'
20 jmp _label1
21 _label3:
22 mov eax, msg3 ; Вывод на экран строки
23 call sprintf ; 'Сообщение No 3'
24 _end:
25 call quit ; вызов подпрограммы завершения
```

Рис. 4.4: Изменение текста программы

Создала исполняемый файл и проверила его работу (рис. 4.5).

```
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 1
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $
```

Рис. 4.5: Создание исполняемого файла и проверка его работы

Изменила текст программы, чтобы она выводила сначала 'Сообщение № 3', потом 'Сообщение № 2', 'Сообщение № 1' и завершала работу. Создала исполняемый файл и проверила работу (рис. 4.6). Прилагаю измененный код:

```
%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 No 1'
```

```
    jmp _end
```

```
_label2:
```

```
    mov eax, msg2 ; Вывод на экран строки
```

```
    call sprintf ; 'Сообщение No 2'
```

```
    jmp _label1
```

```
_label3:
```

```
    mov eax, msg3 ; Вывод на экран строки
```

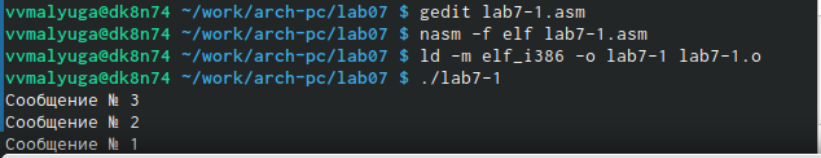
```
    call sprintf ; 'Сообщение No 3'
```

```
    jmp _label2    ;
```

```
_end:
```

```
    call quit ; вызов подпрограммы завершения
```

```
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ gedit lab7-1.asm
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```



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

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

Рис. 4.6: Изменение текста программы и вывод

Создала файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. Изучила текст программы из листинга 7.3 и ввела в lab7-2.asm (рис. 4.7).

```
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ gedit lab7-1.asm
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ touch lab7-2.asm
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ gedit lab7-2.asm

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

1 %include 'in_out.asm'
2 section .data
3     msg1 db 'Введите B: ',0h
4     msg2 db "Наибольшее число: ",0h
5     A dd '20'
6     C dd '50'
7 section .bss
8     max resb 10
9     B resb 10
10 section .text
11 global _start
12 _start:
13     mov eax,msg1
14     call sprint      ; Вывод сообщения 'Введите B: '
15     mov ecx,B
16     mov edx,10
17     call sread      ; Ввод 'B'
18     mov eax,B        ; Преобразование 'B' из символа в число
19     call atoi        ; Вызов подпрограммы перевода символа в число
20     mov [B],eax      ; запись преобразованного числа в 'B'
21     mov ecx,[A] ; 'ecx = A'
22     mov [max],ecx ; 'max = A' ; Записываем 'A' в переменную 'max'
23 ; ----- Сравниваем 'A' и 'C' (как символы)
24     cmp ecx,[C] ; Сравниваем 'A' и 'C'
25     jg check_B ; если 'A>C', то переход на метку 'check_B',
26     mov ecx,[C] ; иначе 'ecx = C'
27     mov [max],ecx ; 'max = C'
28 ; ----- Преобразование 'max(A,C)' из символа в число
29 check_B:
30     mov eax,max
```

Рис. 4.7: Создание файла lab7-2.asm

Создала исполняемый файл и проверила его работу для разных значений B (рис. 4.8).

```
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 34
Наибольшее число: 50
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 999
Наибольшее число: 999
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $
```

Рис. 4.8: Создание исполняемого файла и проверка его работы

4.2 Изучение структуры файлы листинга

Создала файл листинга для программы из файла lab7-2.asm. Открыла файл листинга lab7-2.lst с помощью gedit. Объясню содержимое трёх строк файла листинга (рис. 4.9).

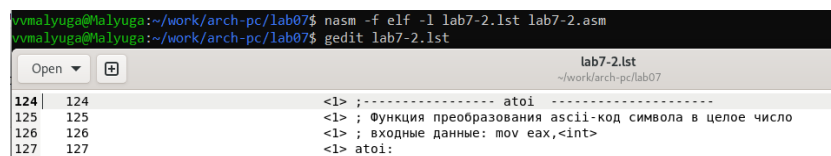


Рис. 4.9: Файл листинга lab7-2.asm

1. Строка 124: Эта строка служит в качестве разделителя, обозначающего начало блока, связанного с функцией atoi.
2. Строка 125: Эта строка предоставляет общее описание функции atoi. Она сообщает, что функция отвечает за преобразование ASCII-кода символа в целое число.
3. Строка 126: Эта строка содержит информацию о том, что входные данные для функции передаются с использованием инструкции mov eax, .

Открыла файл с программой lab7-2.asm и в инструкции с двумя операндами удалить один операнд. Выполнила трансляцию с получением файла листинга (рис. 4.10).

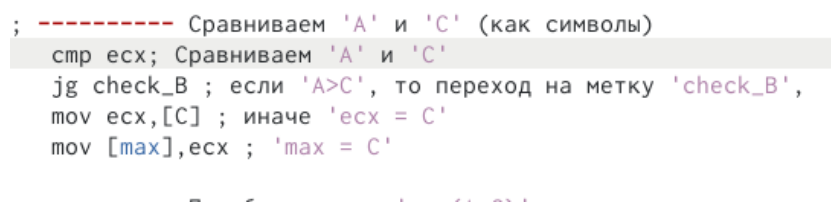


Рис. 4.10: Удаление операнда из инструкции с двумя операндами

Выполнила трансляцию с получением файла листинга. На выходе я не получила ни одного файла из-за ошибки инструкции mov,(единственная в коде содержит

два операнда) которая не может работать, имея только один операнд, из-за чего нарушается работа кода (рис. 4.11).

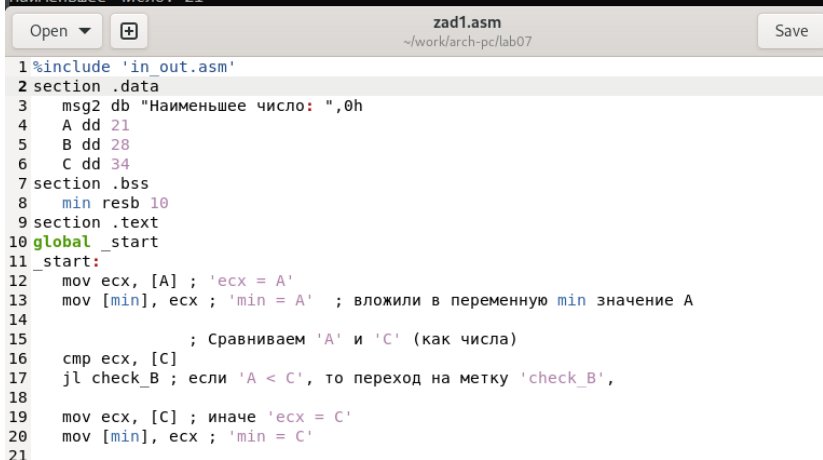
```
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:27: error: invalid combination of opcode and operands
vvmalyuga@dk8n74 ~/work/arch-pc/lab07 $
```

Рис. 4.11: Ошибка трансляции кода

4.3 Задание для самостоятельной работы

1. Написала программу для нахождения наименьшей из 3 целочисленных переменных a, b и c (вариант 11). Проверила правильность выполнения: действительно, все работает правильно (рис. 4.12).

```
vvmalyuga@Malyuga:~/work/arch-pc/lab07$ gedit zad1.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab07$ nasm -f elf zad1.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab07$ ld -m elf_i386 -o zad1 zad1.o
vvmalyuga@Malyuga:~/work/arch-pc/lab07$ ./zad1
Наименьшее число: 21
```



```
1 %include 'in_out.asm'
2 section .data
3     msg2 db "Наименьшее число: ",0h
4     A dd 21
5     B dd 28
6     C dd 34
7 section .bss
8     min resb 10
9 section .text
10 global _start
11 _start:
12     mov ecx, [A] ; 'ecx = A'
13     mov [min], ecx ; 'min = A' ; вложили в переменную min значение A
14
15     ; Сравниваем 'A' и 'C' (как числа)
16     cmp ecx, [C]
17     jl check_B ; если 'A < C', то переход на метку 'check_B',
18
19     mov ecx, [C] ; иначе 'ecx = C'
20     mov [min], ecx ; 'min = C'
21
```

Рис. 4.12: Программа для нахождения наименьшей из 3 переменных

Прилагаю код:

```
%include 'in_out.asm'

section .data
```

```

msg2 db "Наименьшее число: ",0h
A dd 21
B dd 28
C dd 34
section .bss
    min resb 10
section .text
global _start
_start:
    mov ecx, [A] ; 'ecx = A'
    mov [min], ecx ; 'min = A' ; вложили в переменную min значение A

                    ; Сравниваем 'A' и 'C' (как числа)
    cmp ecx, [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]
    jl fin ; если 'min(A,C) < B', то переход на 'fin',

    mov ecx, [B] ; иначе 'ecx = B'
    mov [min], ecx

fin:

```

```

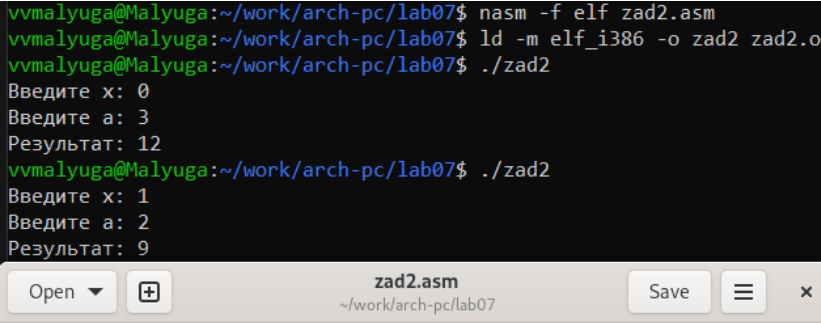
; Вывод результата
mov eax, msg2
call sprint ; Вывод сообщения 'Наименьшее число: '

mov eax, [min]
call iprintLF ; Вывод 'min(A,B,C)'

call quit ; Выход

```

2. Написала программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений (вариант 11). Проверила правильность выполнения: действительно, все работает правильно (рис. 4.13).



```

vvmalyuga@Malyuga:~/work/arch-pc/lab07$ nasm -f elf zad2.asm
vvmalyuga@Malyuga:~/work/arch-pc/lab07$ ld -m elf_i386 -o zad2 zad2.o
vvmalyuga@Malyuga:~/work/arch-pc/lab07$ ./zad2
Введите x: 0
Введите a: 3
Результат: 12
vvmalyuga@Malyuga:~/work/arch-pc/lab07$ ./zad2
Введите x: 1
Введите a: 2
Результат: 9

```

```

1 %include 'in_out.asm'
2
3 SECTION .data
4     msgX: db 'Введите x: ', 0h ;
5     msgA: db 'Введите a: ', 0h ;
6     result: db 'Результат: ', 0h
7
8 SECTION .bss
9     x: resb 80 ; Буфер для ввода x
10    a: resb 80 ; Буфер для ввода a
11    res: resb 80 ; Буфер для результата вычислений
12
13 SECTION .text
14     GLOBAL _start
15     _start:
16
17     mov eax, msgX
18     call sprint
19
20     ; Ввод x
21

```

Рис. 4.13: Программа вычисления функции

Прилагаю код:

```
%include 'in_out.asm'
```

SECTION .data

```
msgX: db 'Введите x: ', 0h ;  
msgA: db 'Введите a: ', 0h ;  
result: db 'Результат: ', 0h
```

SECTION .bss

```
x: resb 80 ; для ввода x  
a: resb 80 ; для ввода a  
res: resb 80 ; для результата вычислений
```

SECTION .text

```
GLOBAL _start
```

```
_start:
```

```
mov eax, msgX
```

```
call sprint
```

```
; Ввод x
```

```
mov ecx, x
```

```
mov edx, 80
```

```
call sread
```

```
mov eax, x
```

```
call atoi
```

```
mov [x], eax
```

```
mov eax, msgA
```

```

call sprint

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

; Перемещение x в регистр ecx для дальнейших вычислений
mov ecx, [x]

; сравнение x с 0
cmp ecx, 0
je _xisnull ; "jump if equals" (x = 0)
jne _xisnotnull ; "jump if not equals" (x != 0)

_xisnull:
; вычисление f(x)= 4a, если x = 0
mov eax, [a] ; Загрузка значения a в регистр eax
mov ebx, 4 ; Загрузка константы 4 в регистр ebx
imul eax, ebx
mov [res], eax
jmp _fin ; Переход к fin для завершения программы

_xisnotnull:
; вычисление f(x)= 4a + x, если x != 0
mov eax, [a] ; Загрузка значения a в регистр eax

```

```
mov ebx, 4      ; Загрузка константы 4 в регистр ebx
imul eax, ebx   ; Умножение a на 4
add eax, ecx    ; Прибавление x к результату
mov [res], eax  ; Сохранение результата в res
jmp _fin
```

```
_fin:
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit
```

5 Выводы

По итогам данной лабораторной работы я изучила команды условного и безусловного переходов, приобрела навыки написания программ с использованием переходов и ознакомилась с назначением и структурой файла листинга, что поможет мне при выполнении последующих лабораторных работ.

Список литературы

::: https://esystem.rudn.ru/pluginfile.php/2089087/mod_resource/content/0/Лабораторная%20работа%20N°7.%20Команды%20безусловного%20и%20условного%20переходов%20в%20Nasm.%20Программирование%20ветвлений..pdf