

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютеров

Чувакина Мария Владимировна

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	12
4.3	Задания для самостоятельной работы	13
5	Выводы	19
6	Список литературы	20

Список иллюстраций

4.1 Создание директории.....	7
4.2 Создание файла	7
4.2 Ввод текста программы из листинга 7.1	8
4.4 Запуск программного кода	8
4.5 Изменение текста программы.....	9
4.6 Создание исполняемого файла	9
4.7 Изменение текста программы.....	10
4.8 Вывод программы.....	10
4.9 Создание файла	11
4.10 Ввод текста программы из листинга 7.3	11
4.11 Проверка работы файла	12
4.12 Создание файла листинга	12
4.13 Изучение файла листинга.....	12
4.14 Выбранные строки файла	13
4.15 Удаление выделенного операнда из кода	13
4.16 Получение файла листинга	13
4.17 Написание программы	14
4.18 Запуск файла и проверка его работы	14
4.19 Написание программы	16
4.20 Запуск файла и проверка его работы	17

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.

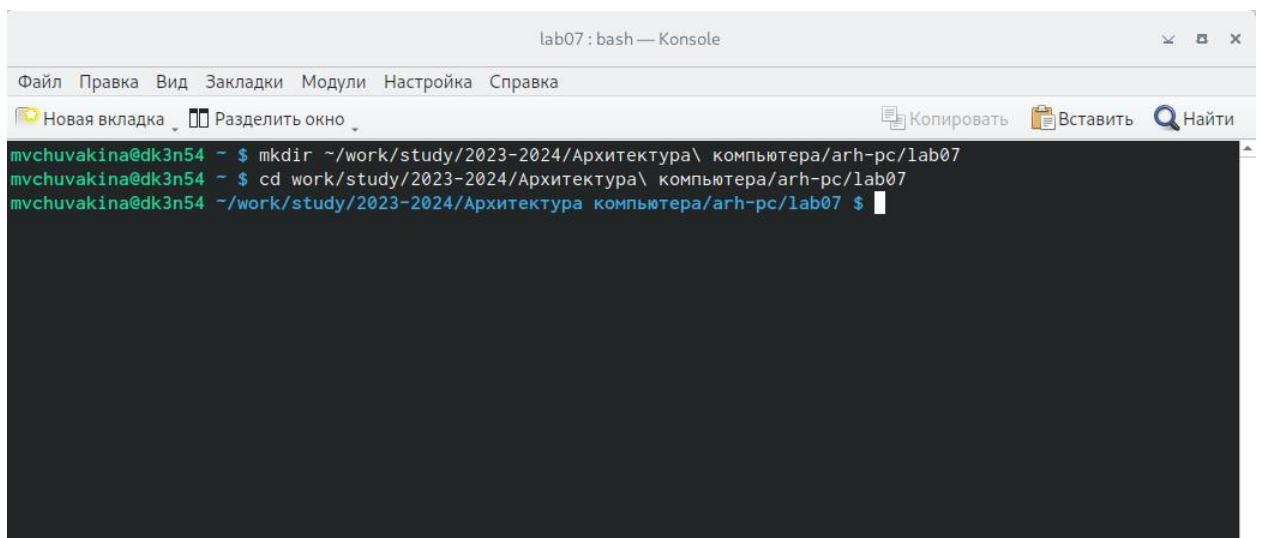
Безусловный переход выполняется инструкцией `jmp`. Инструкция `cmp` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `cmp` является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания.

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

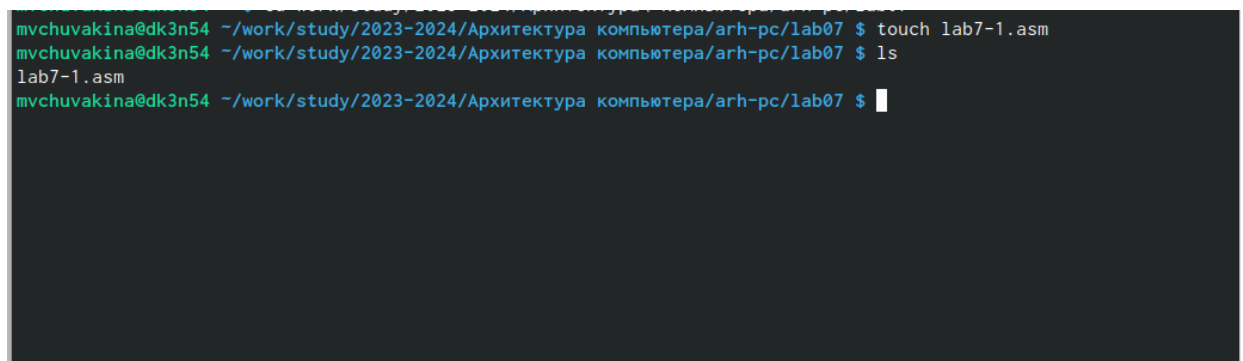
С помощью утилиты `mkdir` создаю директорию, в которой буду создавать файлы с программами для лабораторной работы №7 (рис. 4.1). Перехожу в созданный каталог с помощью утилиты `cd`.

A screenshot of a terminal window titled "lab07: bash — Konsole". The window has a menu bar with "Файл", "Правка", "Вид", "Закладки", "Модули", "Настройка", and "Справка". Below the menu bar are icons for "Новая вкладка", "Разделить окно", "Копировать", "Вставить", and "Найти". The terminal shows the following commands and output:

```
mvchuvakina@dk3n54 ~ $ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/arh-pc/lab07
mvchuvakina@dk3n54 ~ $ cd work/study/2023-2024/Архитектура\ компьютера/arh-pc/lab07
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $
```

Рис. 4.1: Создание директории

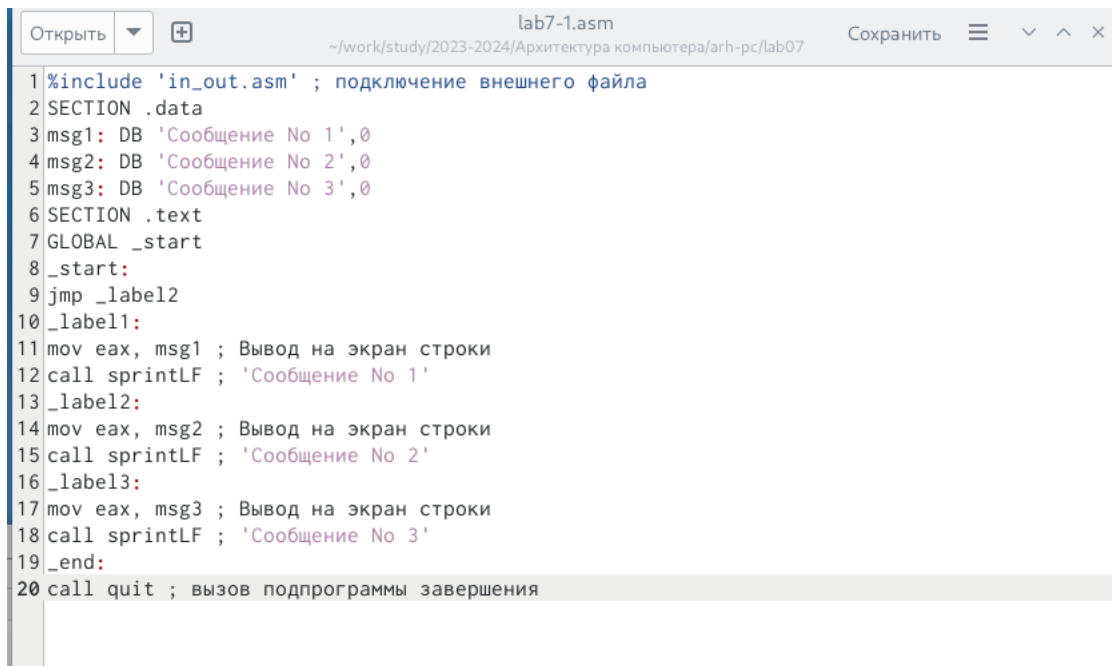
С помощью утилиты `touch` создаю файл `lab7-1.asm` (рис. 4.2).

A screenshot of a terminal window showing the creation of a file. The terminal shows the following commands and output:

```
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ touch lab7-1.asm
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ ls
lab7-1.asm
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $
```

Рис. 4.2: Создание файла

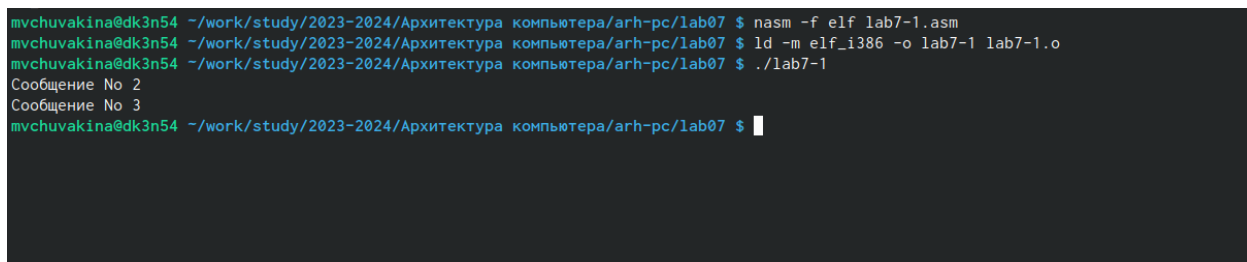
Ввожу в файл `lab7-1.asm` текст программы из листинга 7.1. (рис. 4.3).



```
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. 4.3: Ввод текста программы из листинга 7.1

Создаю исполняемый файл и запускаю его. (рис. 4.4).

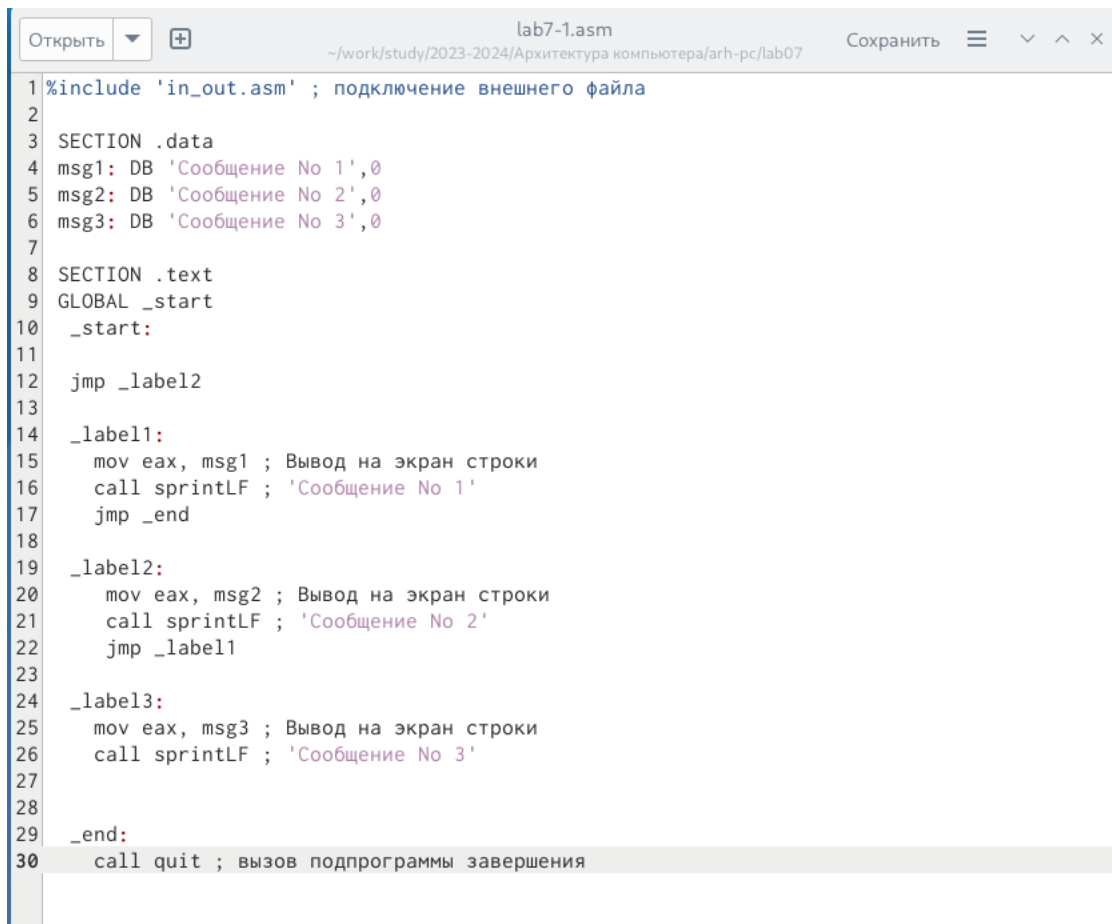


```
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ nasm -f elf lab7-1.asm
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 3
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $
```

Рис. 4.4: Запуск программного кода

Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения.

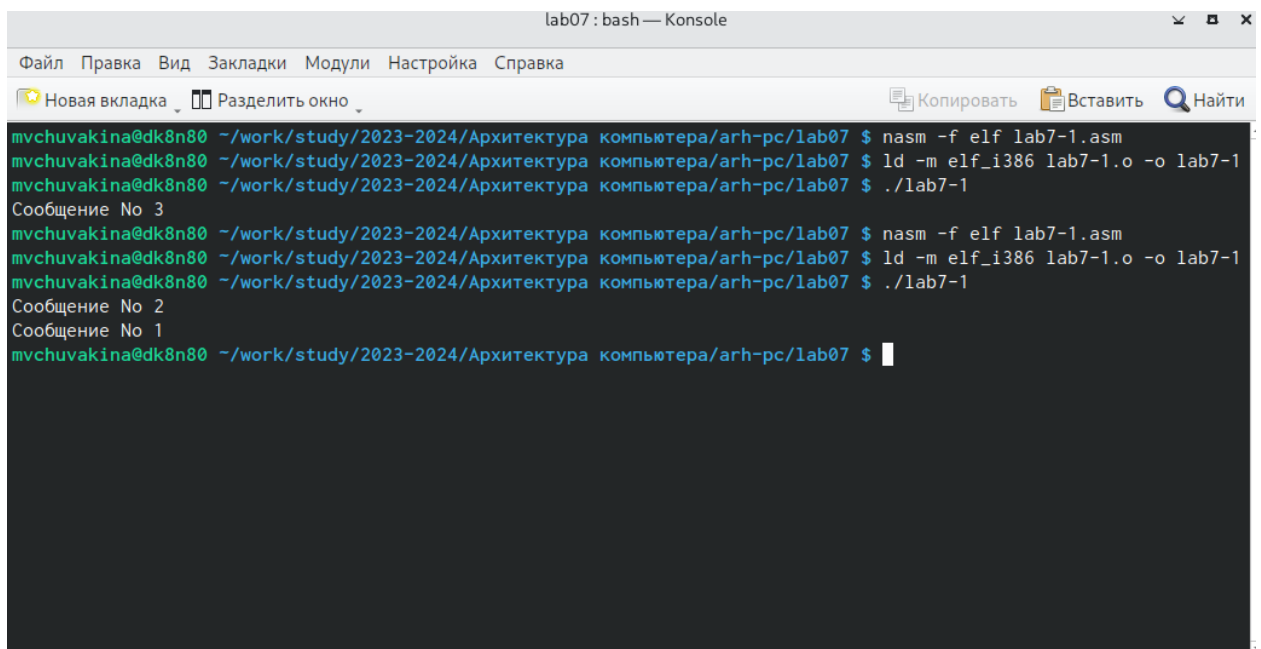
Изменяю программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого изменяю текст программы в соответствии с листингом 7.2. (рис. 4.5).



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

Рис. 4.5: Изменение текста программы

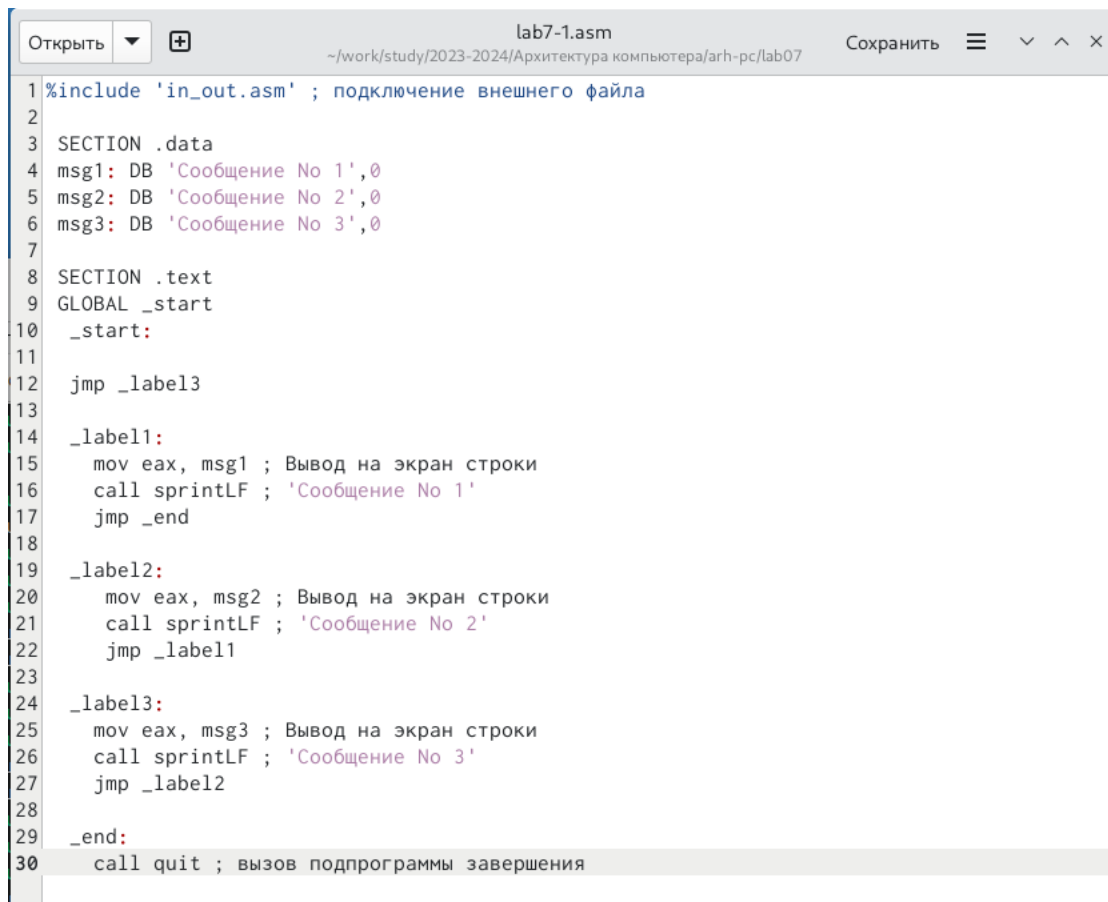
Создаю исполняемый файл и проверяю его работу. (рис. 4.6).



```
lab07: bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ nasm -f elf lab7-1.asm
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ ld -m elf_i386 lab7-1.o -o lab7-1
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ ./lab7-1
Сообщение No 3
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ nasm -f elf lab7-1.asm
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ ld -m elf_i386 lab7-1.o -o lab7-1
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 1
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $
```

Рис. 4.6: Создание исполняемого файла

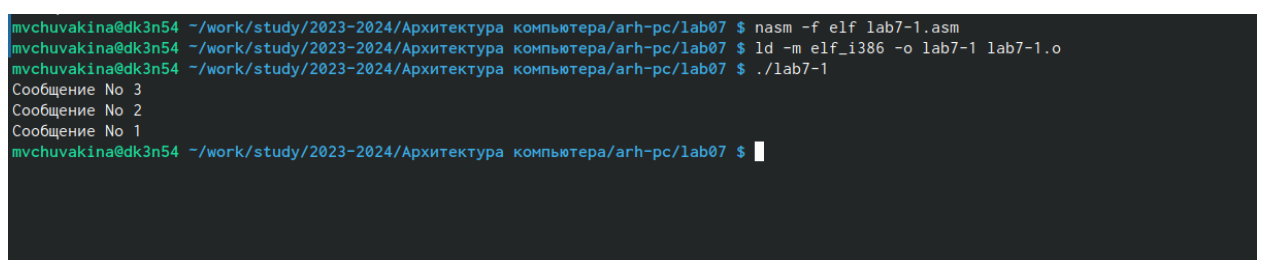
Затем изменяю текст программы, добавив в начале программы `jmp _label3`, `jmp _label2` в конце метки `jmp _label3`, `jmp _label1` добавляю в конце метки `jmp _label2`, и добавляю `jmp _end` в конце метки `jmp _label1`, (рис. 4.7).



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

Рис. 4.7: Изменение текста программы

чтобы вывод программы был следующим: (рис. 4.8).



```
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ nasm -f elf lab7-1.asm
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ ./lab7-1
Сообщение No 3
Сообщение No 2
Сообщение No 1
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $
```

Рис. 4.8: Вывод программы

Рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А,В и С. Значения для А и С задаются в про- грамме, значение В вводится с клавиатуры.

Создаю файл lab7-2.asm в каталоге ~/work/arh-pc/lab07. (рис. 4.9).

```
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ touch lab7-2.asm
mvchuvakina@dk3n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $
```

Рис. 4.9: Создание файла

Текст программы из листинга 7.3 ввожу в lab7-2.asm. (рис. 4.10).

```
lab7-2.asm      [----] 31 L:[ 1+ 3  4/ 50] *(114 /1744b)
#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
```

Рис. 4.10: Ввод текста программы из листинга 7.3

Создаю исполняемый файл и проверьте его работу. (рис. 4.11).

```
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ nasm -f elf lab7-2.asm
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ ld -m elf_i386 lab7-2.o -o lab7-2
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ ./lab7-2
Введите B: 70
Наибольшее число: 70
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $
```

Рис. 4.11: Проверка работы файла

Файл работает корректно.

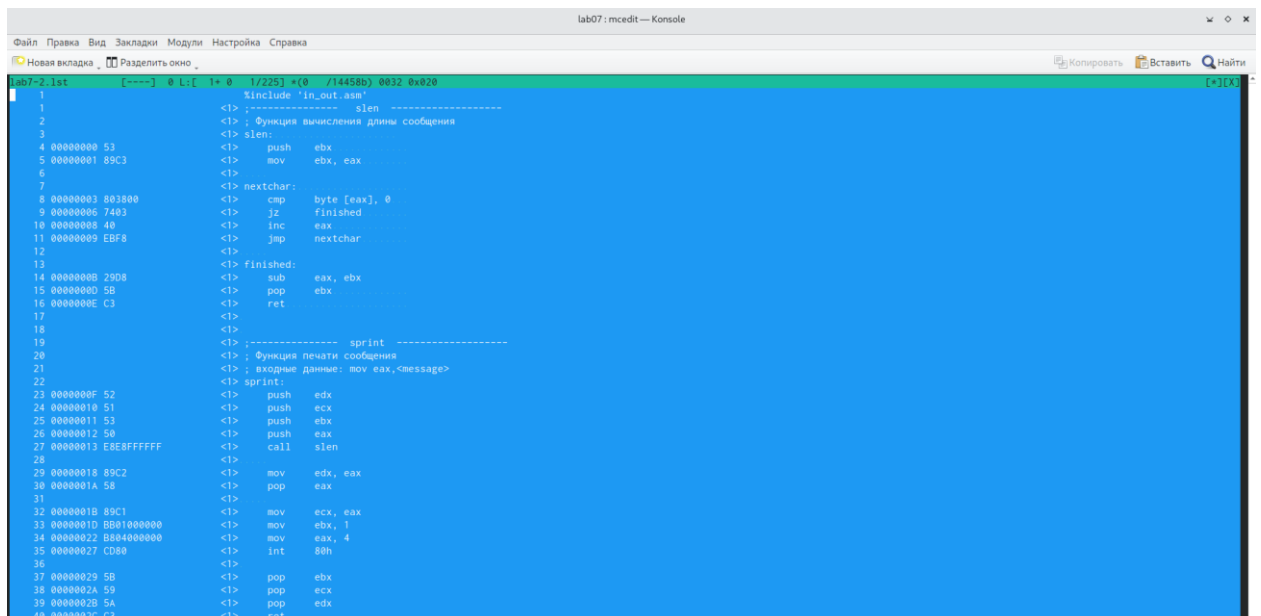
4.2 Изучение структуры файла листинга

Создаю файл листинга для программы из файла lab7-2.asm. (рис. 4.12).

```
mvchuvakina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
mvchuvakina@dk8n54 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ mcedit lab7-2.lst
```

Рис. 4.12: Создание файла листинга

Открываю файл листинга lab7-2.lst с помощью текстового редактора и внимательно изучаю его формат и содержимое. (рис. 4.13).



```
lab7-2.lst [-----] 0 L: [ 1/225] *0 /14458b 0032 0x020
1      <I> ;include 'in_out.asm'
2      <I> ; Функция вычисления длины сообщения
3      <I> slen:
4 00000000 53      <I>      push    ebx
5 00000001 85C3     <I>      mov     ebx, eax
6      <I>
7      <I> nextchar:
8 00000003 803800     <I>      cmp     byte [eax], 0
9 00000006 7403     <I>      jz      finished
10 00000008 40      <I>      inc     eax
11 00000009 EBF8     <I>      jmp     nextchar
12      <I>
13      <I> finished:
14 0000000B 2908     <I>      sub     eax, ebx
15 0000000D 5B      <I>      pop     ebx
16 0000000E C3      <I>      ret
17      <I>
18      <I>
19      <I> ;----- sprint -----
20      <I> ; Функция печати сообщения
21      <I> ; входные данные: mov eax, <message>
22      <I> sprint:
23 0000000F 52      <I>      push    edx
24 00000010 51      <I>      push    ecx
25 00000011 53      <I>      push    ebx
26 00000012 58      <I>      push    eax
27 00000013 EBEBFFFF <I>      call    slen
28      <I>
29 00000018 85C2     <I>      mov     edx, eax
30 0000001A 5B      <I>      pop     eax
31      <I>
32 0000001B 85C1     <I>      mov     ecx, eax
33 0000001D B801000000 <I>      mov     ebx, 1
34 00000022 B804000000 <I>      mov     eax, 4
35 00000027 CD80     <I>      int     80h
36      <I>
37 00000029 5B      <I>      pop     ebx
38 0000002A 59      <I>      pop     ecx
39 0000002B 5A      <I>      pop     edx
40 0000002C C3      <I>      ret
```

Рис. 4.13: Изучение файла листинга

В представленных трех строчках содержатся следующие данные: (рис. 4.14).

```

2          <I> ; Функция вычисления длины сообщения
3          <I> slen
4 00000000 53      <I> push     ebx

```

Рис. 4.14: Выбранные строки файла

“2” - номер строки кода, “; Функция вычисления длины сообщения” - комментарий к коду, не имеет адреса и машинного кода.

“3” - номер строки кода, “slen” - название функции, не имеет адреса и машинного кода.

“4” - номер строки кода, “00000000” - адрес строки, “53” - машинный код, “push ebx” - исходный текст программы, инструкция “push” помещает операнд “ebx” в стек.

Открываю файл с программой lab7-2.asm и в выбранной мной инструкции с двумя операндами удаляю выделенный операнд. (рис. 4.15).

```

; ----- Сравниваем 'A' и 'C' (как символы)
; стр. есх, [C] ; Сравниваем 'A' и 'C'

```

Рис. 4.15: Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга. (рис. 4.16).

```

mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $

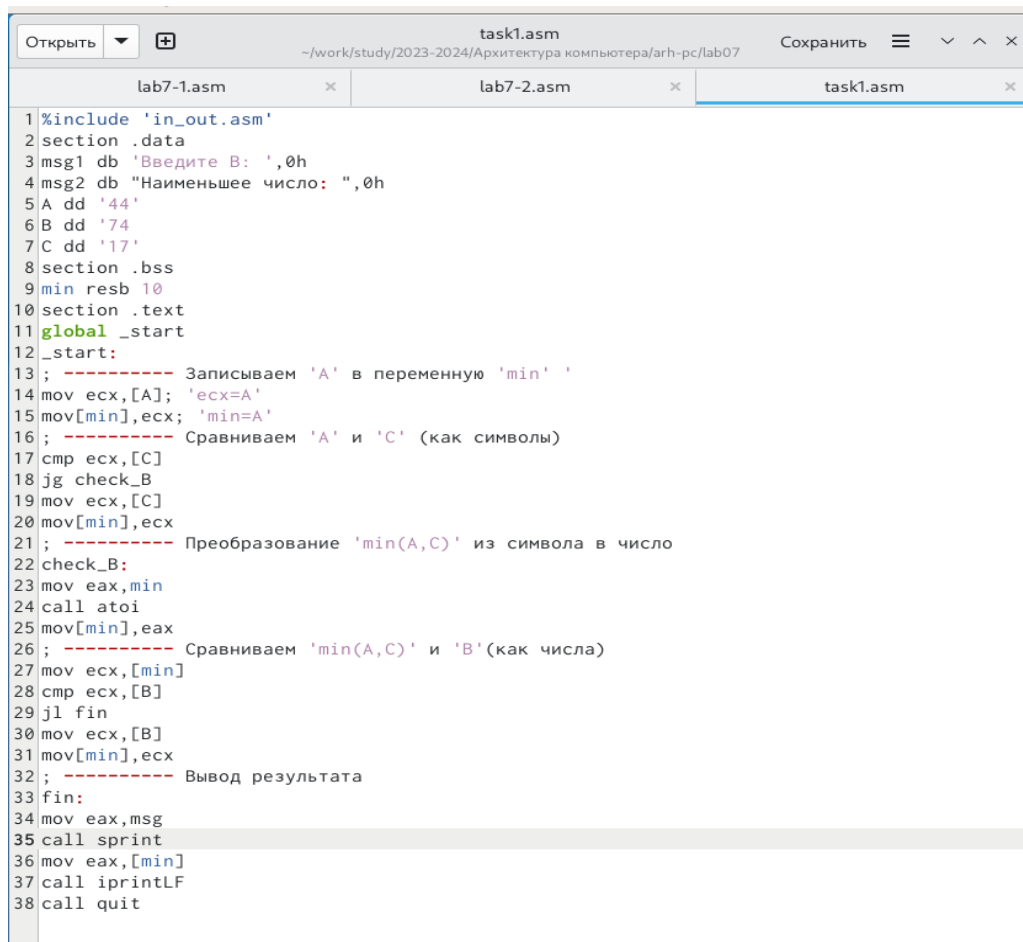
```

Рис. 4.16: Получение файла листинга

На выходе я не получаю ни одного файла из-за ошибки: инструкция mov (единственная в коде содержит два операнда) не может работать, имея только один операнд, из-за чего нарушается работа кода.

4.3 Задания для самостоятельной работы

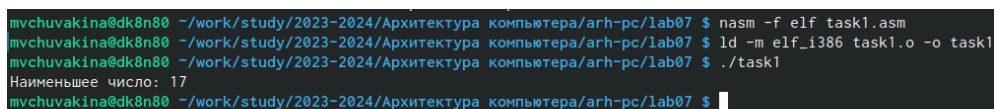
1. Пишу программу нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения переменных выбираю из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы №6. Мой вариант под номером 16, поэтому мои значения - 44, 74 и 17. (рис. 4.17).



```
1 %include 'in_out.asm'
2 section .data
3 msg1 db 'Введите B: ',0h
4 msg2 db "Наименьшее число: ",0h
5 A dd '44'
6 B dd '74'
7 C dd '17'
8 section .bss
9 min resb 10
10 section .text
11 global _start
12 _start:
13 ; ----- Записываем 'A' в переменную 'min'
14 mov ecx,[A]; 'ecx=A'
15 mov[min],ecx; 'min=A'
16 ; ----- Сравниваем 'A' и 'C' (как символы)
17 cmp ecx,[C]
18 jg check_B
19 mov ecx,[C]
20 mov[min],ecx
21 ; ----- Преобразование 'min(A,C)' из символа в число
22 check_B:
23 mov eax,min
24 call atoi
25 mov[min],eax
26 ; ----- Сравниваем 'min(A,C)' и 'B' (как числа)
27 mov ecx,[min]
28 cmp ecx,[B]
29 jl fin
30 mov ecx,[B]
31 mov[min],ecx
32 ; ----- Вывод результата
33 fin:
34 mov eax,msg
35 call sprint
36 mov eax,[min]
37 call iprintLF
38 call quit
```

Рис. 4.17: Написание программы

Создаю исполняемый файл и проверяю его работу, подставляя необходимые значение. (рис. 4.18).



```
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ nasm -f elf task1.asm
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ ld -m elf_i386 task1.o -o task1
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $ ./task1
Наименьшее число: 17
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/arh-pc/lab07 $
```

Рис. 4.18: Запуск файла и проверка его работы

Программа работает корректно. Код программы:

```
%include 'in_out.asm'
section .data
```

```

msg db "Наименьшее число: ",0h
A dd '44'
B dd '74'
C dd '17'
section .bss
min resb 10
section .text
global _start
_start:
; ----- 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'
jg 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:
mov eax,min
call atoi ; Вызов подпрограммы перевода символа в число
mov [min],eax ; запись преобразованного числа в `min`
; ----- 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, msg
call sprint ; Вывод сообщения 'Наименьшее число: '

```

```

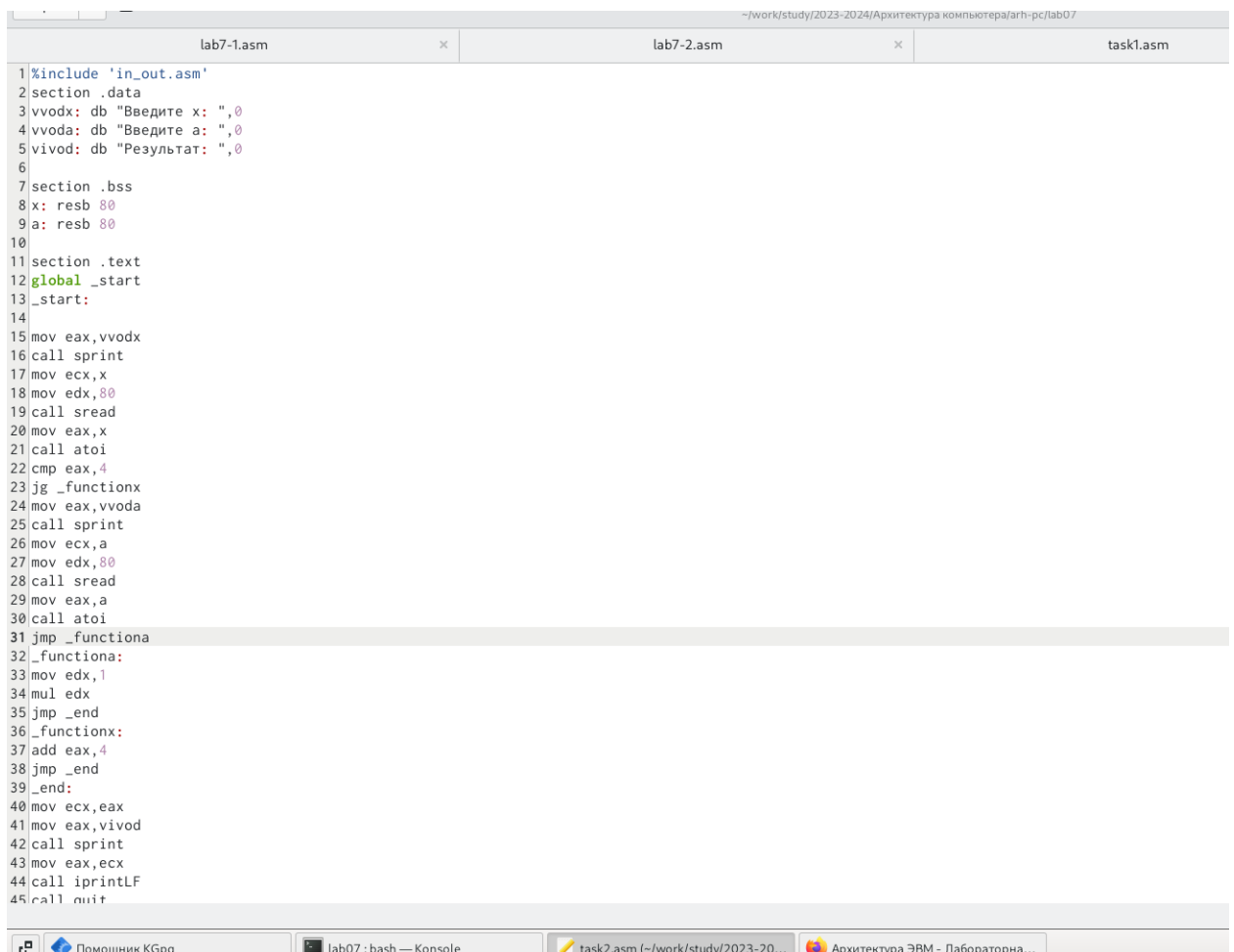
mov eax,[min]
call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход

```

2. Пишу программу, которая для введенных с клавиатуры значений x и a вычисляет значение и выводит результат вычислений заданной для моего варианта функции $f(x)$:

$x + 4$, если $x < 4$

$a * x$, если $x \geq 4$ (рис. 4.1)



```

lab7-1.asm lab7-2.asm task1.asm
1 %include 'in_out.asm'
2 section .data
3 vvodx: db "Введите x: ",0
4 vvoda: db "Введите a: ",0
5 vivod: db "Результат: ",0
6
7 section .bss
8 x: resb 80
9 a: resb 80
10
11 section .text
12 global _start
13 _start:
14
15 mov eax,vvodx
16 call sprint
17 mov ecx,x
18 mov edx,80
19 call sread
20 mov eax,x
21 call atoi
22 cmp eax,4
23 jg _functionx
24 mov eax,vvoda
25 call sprint
26 mov ecx,a
27 mov edx,80
28 call sread
29 mov eax,a
30 call atoi
31 jmp _functiona
32 _functiona:
33 mov edx,1
34 mul edx
35 jmp _end
36 _functionx:
37 add eax,4
38 jmp _end
39 _end:
40 mov ecx,eax
41 mov eax,vivod
42 call sprint
43 mov eax,ecx
44 call iprintLF
45 call quit

```

Рис. 4.19: Написание программы

Создаю исполняемый файл и проверяю его работу для значений x и a соответственно. (рис. 4.20).


```

Результат: 13
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/agh-pc/lab07 $ nasm -f elf task2.asm
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/agh-pc/lab07 $ ld -m elf_i386 task2.o -o task2
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/agh-pc/lab07 $ ./task2
Введите x: 1
Введите a: 6
Результат: 6
mvchuvakina@dk8n80 ~/work/study/2023-2024/Архитектура компьютера/agh-pc/lab07 $

```

Рис. 4.20: Запуск файла и проверка его работы

Код программы:

```

#include 'in_out.asm'

section .data
vvodx: db "Введите x: ",0
vvoda: db "Введите a: ",0
vivod: db "Результат: ",0

```

```

section .bss
x: resb 80
a: resb 80

```

```

section .text
global _start
_start:

```

```

mov eax,vvodx
call sprint
mov ecx,x
mov edx,80
call sread
mov eax,x
call atoi
cmp eax,4
jg _functionx
mov eax,vvoda
call sprint
mov ecx,a

```

```
mov edx,80
call sread
mov eax,a
call atoi
jmp _functiona
_functiona:
mov edx,1
mul edx
jmp _end
_functionx:
add eax,4
jmp _end
_end:
mov ecx,eax
mov eax,vivod
call sprint
mov eax,ecx
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.

6 Список литературы

1. GDB:TheGNUProjectDebugger.—URL:<https://www.gnu.org/software/gdb/>.
2. GNUBashManual.—2016.—URL:<https://www.gnu.org/software/bash/manual/>.
3. MidnightCommanderDevelopmentCenter.—2021.—URL:<https://midnight-commander.org/>.
4. NASMAssemblyLanguageTutorials.—2021.—URL:<https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. TheNASMdocumentation.—2021.—URL:<https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. КолдаевВ.Д.,ЛупинС.А.АрхитектураЭВМ.—М.:Форум,2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. НовожиловО.П.АрхитектураЭВМисистем.—М.:Юрайт,2016.
12. Расширенныйассемблер:NASM.—2021.—
URL:<https://www.opennet.ru/docs/RUS/nasm/>.
13. РобачевскийА.,НемнюгинС.,СтефикО.ОперационнаясистемаUNIX.—2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. СтоляроваА.ПрограммированиеязыкеассемблераNASMдляОСUnix.—2-

е изд. — М. : МАКС Пресс, 2011. — URL:
http://www.stolyarov.info/books/asm_unix.

21

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб.: Питер, 2013. — 874 с.
— (Классика Computer Science).

16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. —
СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).