

# Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютеров и операционные системы

Авдадаев Джамал Геланиевич

## Содержание

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

## 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 `str` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `str` 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

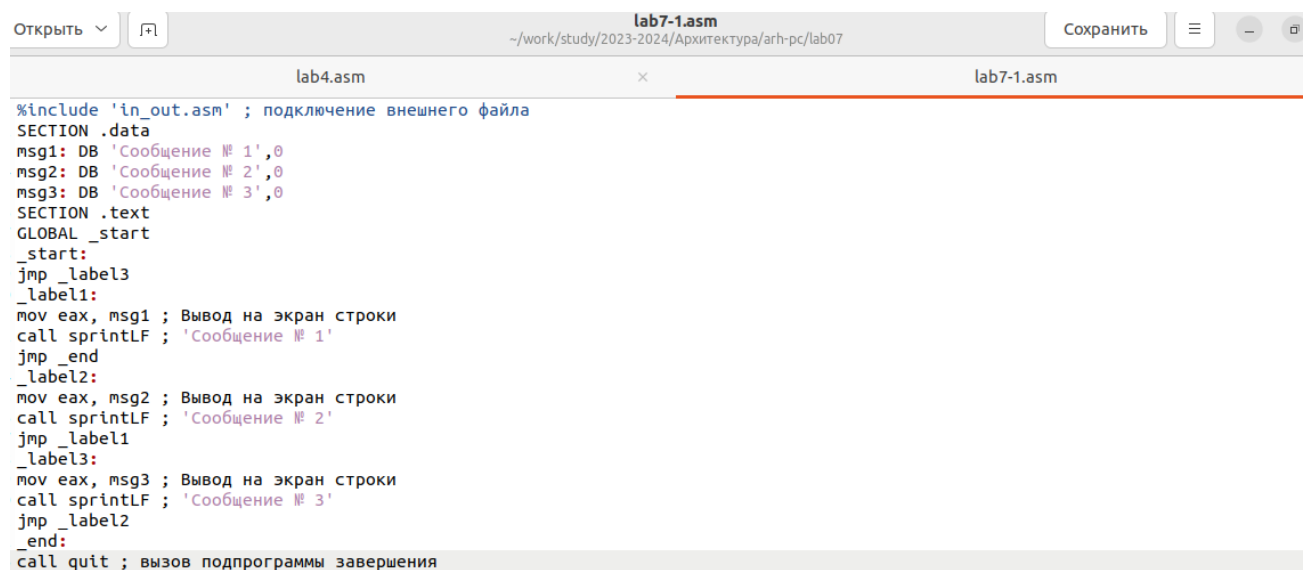
### 4.1 Реализация переходов в NASM

Создаю каталог для программ лабораторной работы № 7, перехожу в него и создаю файл `lab7-1.asm`. (рис. 19).

```
@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab04$ mkdir ~/work/study/2023-2024/Архитектура/arh-pc/lab07
@gavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab04$ cd ~/work/study/2023-2024/Архитектура/arh-pc/lab07
@gavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$ touch lab7-1.asm
```

Figure 1: Создание файлов для лабораторной работы

Ввожу в файл `lab7-1.asm` текст программы из листинга 7.1. (рис. 19).



```
lab7-1.asm
~/work/study/2023-2024/Архитектура/arh-pc/lab07
Сохранить

lab4.asm x 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 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
```

Figure 2: Ввод текста программы из листинга 7.1

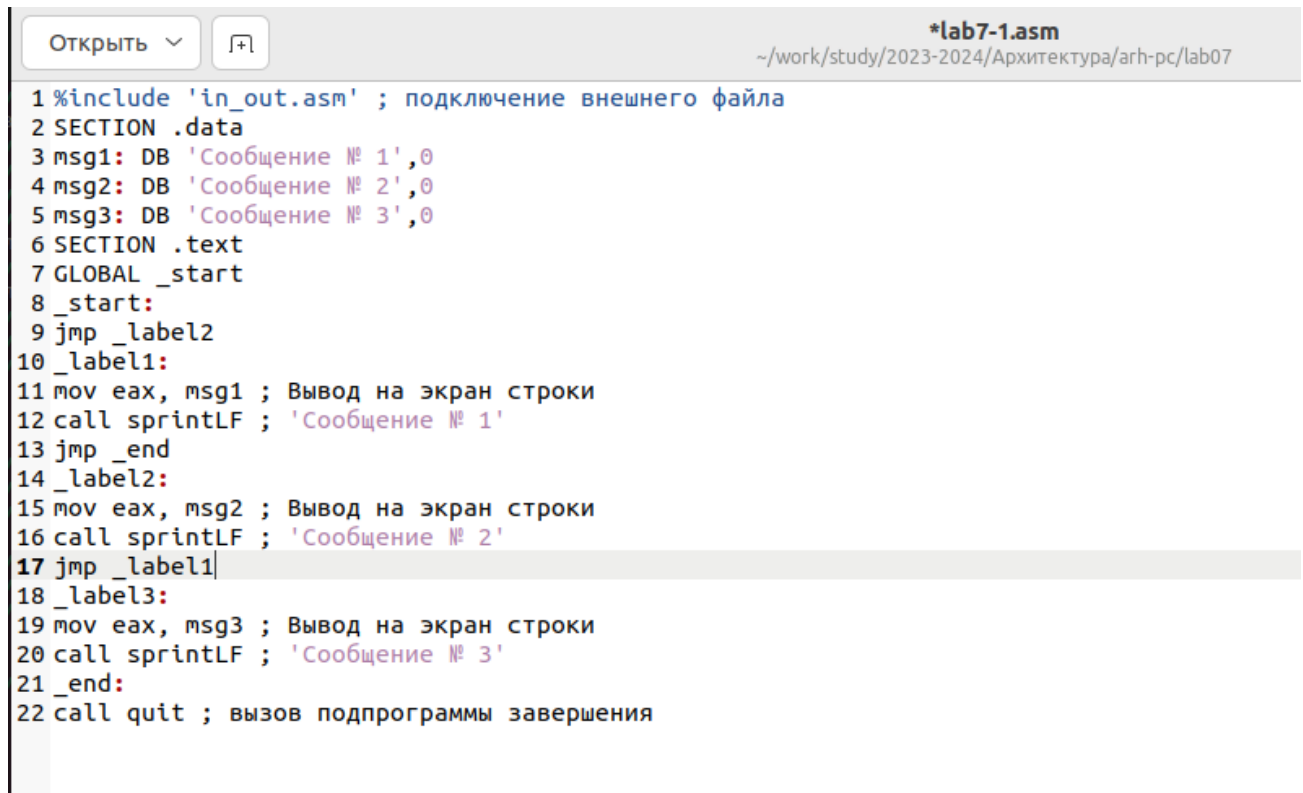
Создаю исполняемый файл и запускаю его. (рис. 19).

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

Figure 3: 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.

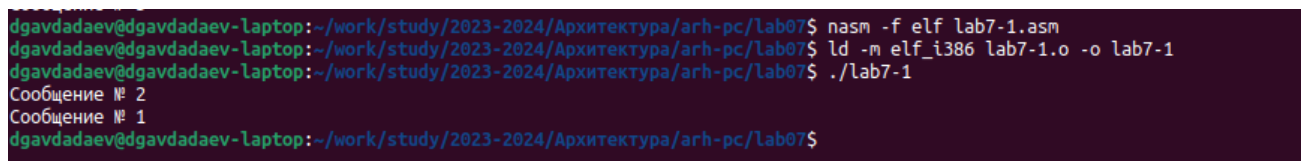
Изменяю программу таким образом, чтобы она выводила сначала 'Сообщение № 2', потом 'Сообщение № 1' и завершала работу. Для этого изменяю текст программы в соответствии с листингом 7.2. (рис. 19).



```
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 SECTION .text
7 GLOBAL _start
8 _start:
9 jmp _label2
10 _label1:
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение № 1'
13 jmp _end
14 _label2:
15 mov eax, msg2 ; Вывод на экран строки
16 call sprintf ; 'Сообщение № 2'
17 jmp _label1
18 _label3:
19 mov eax, msg3 ; Вывод на экран строки
20 call sprintf ; 'Сообщение № 3'
21 _end:
22 call quit ; вызов подпрограммы завершения
```

Figure 4: Изменение текста программы

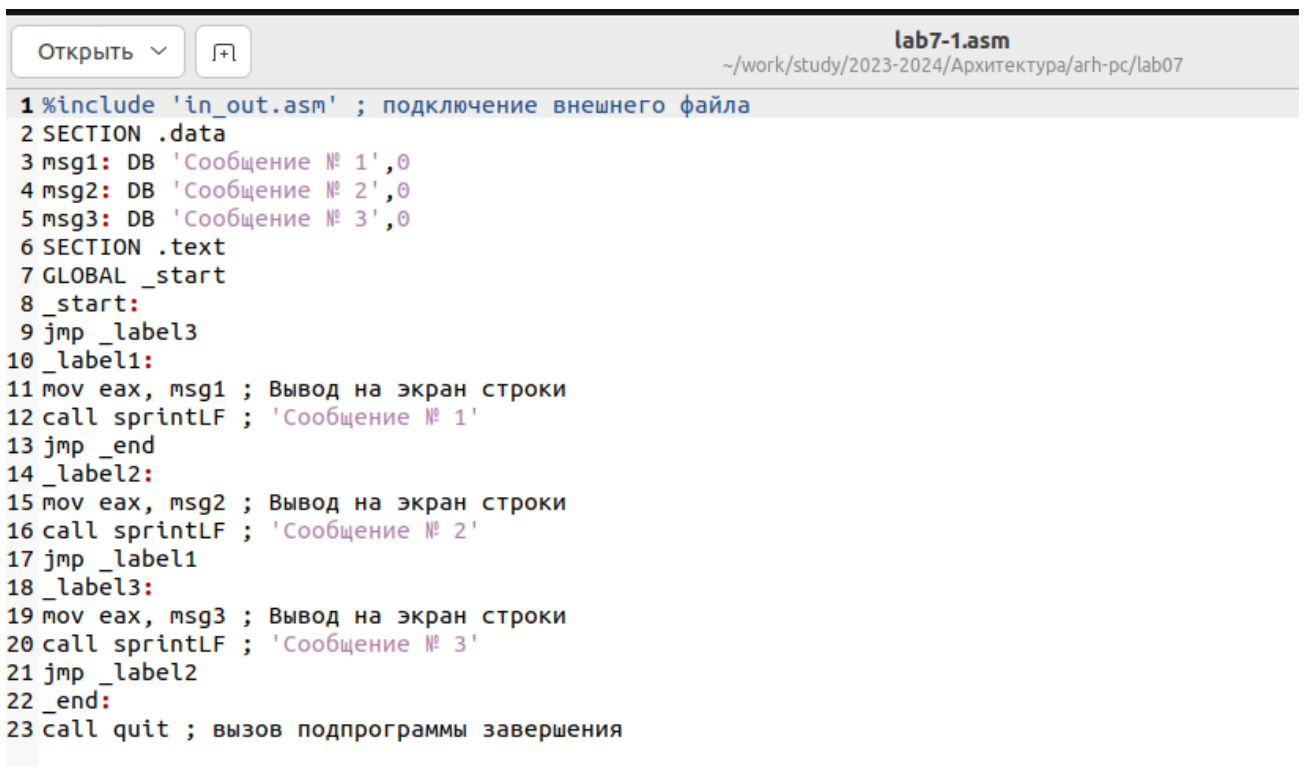
Создаю исполняемый файл и проверяю его работу. (рис. 19).



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

Figure 5: 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. 19).

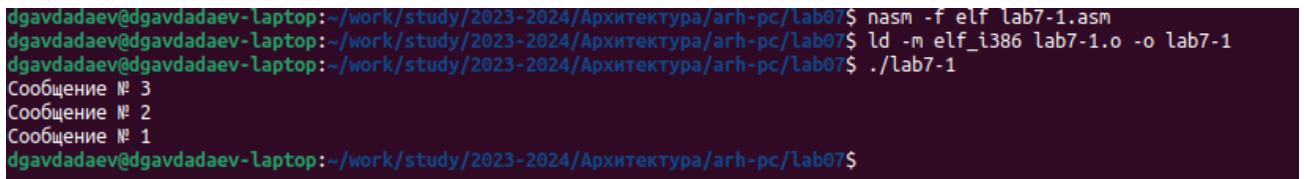


```
lab7-1.asm
~/work/study/2023-2024/Архитектура/arch-pc/lab07

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 SECTION .text
7 GLOBAL _start
8 _start:
9 jmp _label3
10 _label1:
11 mov eax, msg1 ; Вывод на экран строки
12 call printf ; 'Сообщение № 1'
13 jmp _end
14 _label2:
15 mov eax, msg2 ; Вывод на экран строки
16 call printf ; 'Сообщение № 2'
17 jmp _label1
18 _label3:
19 mov eax, msg3 ; Вывод на экран строки
20 call printf ; 'Сообщение № 3'
21 jmp _label2
22 _end:
23 call quit ; вызов подпрограммы завершения
```

Figure 6: Изменение текста программы

чтобы вывод программы был следующим: (рис. 19).



```
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arch-pc/lab07$ nasm -f elf lab7-1.asm
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arch-pc/lab07$ ld -m elf_i386 lab7-1.o -o lab7-1
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arch-pc/lab07$
```

Figure 7: 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 программе, значение В вводится с клавиатуры.

Создаю файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. (рис. 19).



```
@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arch-pc/lab07$ touch lab7-2.asm
```

Figure 8: Создание файла

Текст программы из листинга 7.3 ввожу в lab7-2.asm. (рис. 19).

Открыть ▾

lab7-2.asm  
~/work/study/2023-2024/Архитектура/arh-pc/lab07

```
11 global _start
12 _start:
13 ; ----- Вывод сообщения 'Введите B: '
14 mov eax,msg1
15 call sprint
16 ; ----- Ввод 'B'
17 mov ecx,B
18 mov edx,10
19 call sread
20 ; ----- Преобразование 'B' из символа в число
21 mov eax,B
22 call atoi ; Вызов подпрограммы перевода символа в число
23 mov [B],eax ; запись преобразованного числа в 'B'
24 ; ----- Записываем 'A' в переменную 'max'
25 mov ecx,[A] ; 'ecx = A'
26 mov [max],ecx ; 'max = A'
27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 jg check_B ; если 'A>C', то переход на метку 'check_B',
30 mov ecx,[C] ; иначе 'ecx = C'
31 mov [max],ecx ; 'max = C'
32 ; ----- Преобразование 'max(A,C)' из символа в число
33 check_B:
34 mov eax,max
35 call atoi ; Вызов подпрограммы перевода символа в число
36 mov [max],eax ; запись преобразованного числа в 'max'
37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
38 mov ecx,[max]
39 cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
40 jg fin ; если 'max(A,C)>B', то переход на 'fin',
41 mov ecx,[B] ; иначе 'ecx = B'
42 mov [max],ecx
43 ; ----- Вывод результата
44 fin:
45 mov eax, msg2
46 call sprint ; Вывод сообщения 'Наибольшее число: '
47 mov eax,[max]
48 call iprintLF ; Вывод 'max(A,B,C)'
49 call quit ; Выход
```

Figure 9: Ввод текста программы из листинга 7.3

Создаю исполняемый файл и проверьте его работу. (рис. 19).

```
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$ nasm -f elf lab7-2.asm
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$ ld -m elf_i386 lab7-2.o -o lab7-2
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$ ./lab7-2
Введите B: 90
Наибольшее число: 90
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$
```

Figure 10: Проверка работы файла

Файл работает корректно.

4.2 Изучение структуры файлы листинга

Создаю файл листинга для программы из файла lab7-2.asm. (рис. 19).

```
@dgavdadaev-laptop: ~/work/study/2023-2024/Архитектура/арх-пс/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
```

Figure 11: Создание файла листинга

Открываю файл листинга lab7-2.lst с помощью текстового редактора и внимательно изучаю его формат и содержимое. (рис. 19).

Открыть ▾

lab7-2.lst

~/work/study/2023-2024/Архитектура/арх-пс/lab07

lab7-2.asm

lab7-2.lst

1	1	%include 'in_out.asm'
2	2	<1> ;----- slen -----
3	3	<1> ; Функция вычисления длины сообщения
4	4	<1> slen:
5	5 00000000 53	<1> push ebx
6	6 00000001 89C3	<1> mov ebx, eax
7	7	<1>
8	8	<1> nextchar:
9	9 00000003 803800	<1> cmp byte [eax], 0
10	10 00000006 7403	<1> jz finished
11	11 00000008 40	<1> inc eax
12	12 00000009 EBF8	<1> jmp nextchar
13	13	<1>
14	14	<1> finished:
15	15 0000000B 29D8	<1> sub eax, ebx
16	16 0000000D 5B	<1> pop ebx
17	17 0000000E C3	<1> ret
18	18	<1>
19	19	<1>
20	20	<1> ;----- sprint -----
21	21	<1> ; Функция печати сообщения
22	22	<1> ; входные данные: mov eax,<message>
23	23	<1> sprint:
24	24 0000000F 52	<1> push edx
25	25 00000010 51	<1> push ecx
26	26 00000011 53	<1> push ebx
27	27 00000012 50	<1> push eax
28	28 00000013 E8E8FFFFFF	<1> call slen
29	29	<1>
30	30 00000018 89C2	<1> mov edx, eax
31	31 0000001A 58	<1> pop eax
32	32	<1>
33	33 0000001B 89C1	<1> mov ecx, eax
34	34 0000001D BB01000000	<1> mov ebx, 1
35	35 00000022 B804000000	<1> mov eax, 4
36	36 00000027 CD80	<1> int 80h
37	37	<1>

Текст Шрифт: Табуляция: 8

Figure 12: Изучение файла листинга

В представленных трех строчках содержатся следующие данные: (рис. 19).

3	3	<1> ; Функция вычисления длины сообщения
4	4	<1> slen:
5	5 00000000 53	<1> push ebx

Figure 13: 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. 19).

```
24 ; ----- Записываем 'A' в переменную 'max'
25 mov ecx,[A] ; 'ecx = A'
26 mov [max],ecx ; 'max = A'
27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 jg check_B ; если 'A>C', то переход на метку 'check_B',
30 mov ecx,[C] ; иначе load 'C'
```

Figure 14: Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга. (рис. 19).

```
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/арх-пс/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/арх-пс/lab07$
```

Figure 15: 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 № 7. Мой вариант под номером 10, поэтому мои значения - 41, 62 и 35. (рис. 19).



Открыть ▾

task1.asm  
~/work/study/2023-2024/Архитектура/arh-pc/lab07

```
1 %include 'in_out.asm'
2 section .data
3 msg db "Наименьшее число: ",0h
4 A dd '41'
5 B dd '62'
6 C dd '35'
7 section .bss
8 min resb 10
9 section .text
10 global _start
11 _start:
12 ; ----- Записываем 'A' в переменную 'min'
13 mov ecx,[A] ; 'ecx = A'
14 mov [min],ecx ; 'min = A'
15 ; ----- Сравниваем 'A' и 'C' (как символы)
16 cmp ecx,[C] ; Сравниваем 'A' и 'C'
17 jg check_B ; если 'A<C', то переход на метку 'check_B',
18 mov ecx,[C] ; иначе 'ecx = C'
19 mov [min],ecx ; 'min = C'
20 ; ----- Преобразование 'min(A,C)' из символа в число
21 check_B:
22 mov eax,min
23 call atoi ; Вызов подпрограммы перевода символа в число
24 mov [min],eax ; запись преобразованного числа в 'min'
25 ; ----- Сравниваем 'min(A,C)' и 'B' (как числа)
26 mov ecx,[min]
27 cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
28 jl fin ; если 'min(A,C)<B', то переход на 'fin',
29 mov ecx,[B] ; иначе 'ecx = B'
30 mov [min],ecx
31 ; ----- Вывод результата
32 fin:
33 mov eax, msg
34 call sprint ; Вывод сообщения 'Наименьшее число: '
35 mov eax,[min]
36 call iprintLF ; Вывод 'min(A,B,C)'
37 call quit ; Выход
```

Figure 16: Написание программы

Создаю исполняемый файл и проверяю его работу, подставляя необходимые значения (рис. 19).

```
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$ touch task1.asm
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$ nasm -f elf task1.asm
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$ ld -m elf_i386 task1.o -o task1
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$ ./task1
Наименьшее число: 35
```

Figure 17: Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

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

```
section .data
```

```
msg db "Наименьшее число:",0h
```



```

A dd '41'
B dd '62'
C dd '35'
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
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)'
```

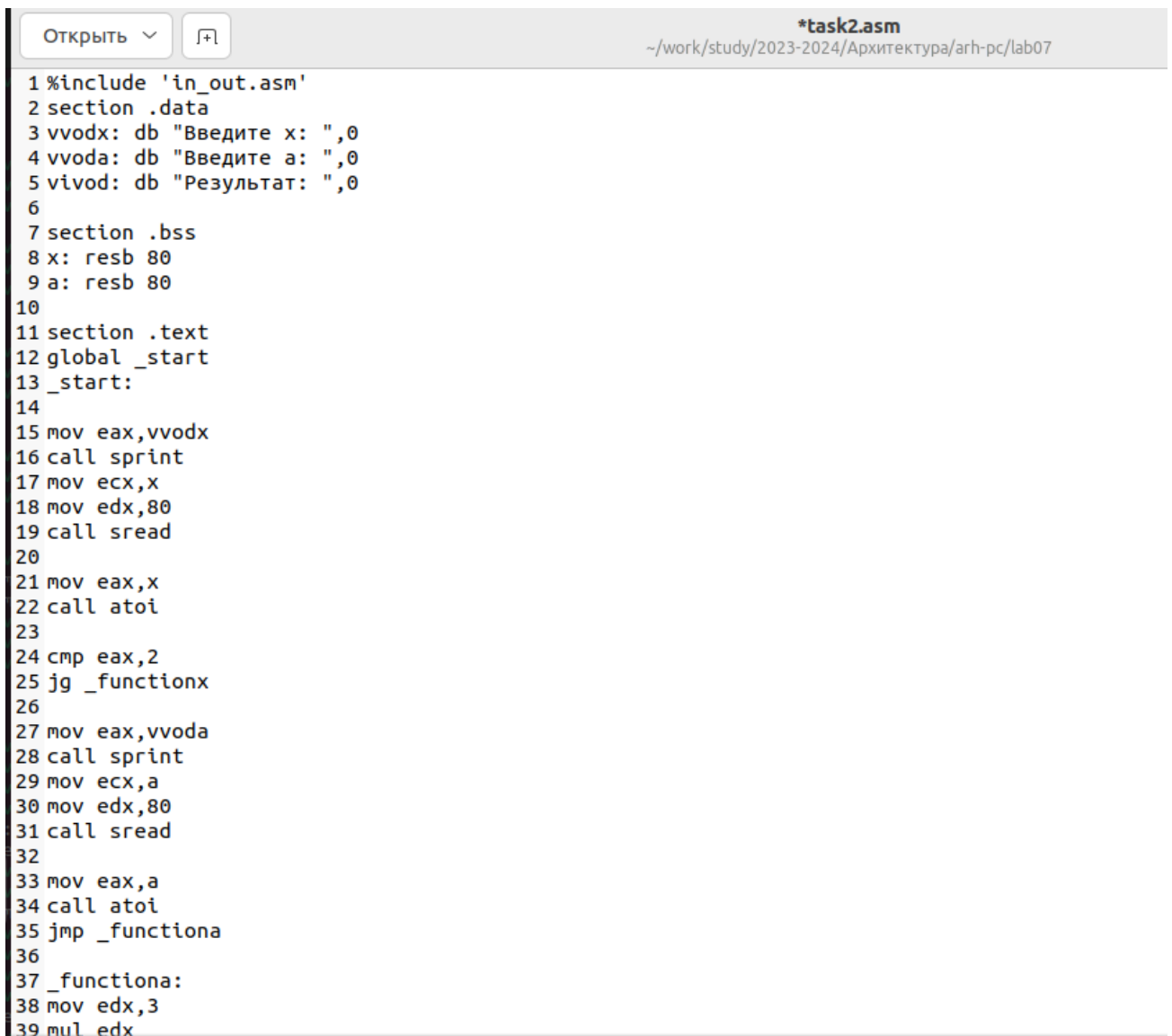
```
call quit ; Выход
```

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

$x - 2$ , если  $x > 2$

$3 \cdot a$ , если  $x \leq 2$

(рис. 19).



```
Открыть  ↕  ~/work/study/2023-2024/Архитектура/arh-pc/lab07 *task2.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
21 mov eax,x
22 call atoi
23
24 cmp eax,2
25 jg _functionx
26
27 mov eax,vvoda
28 call sprint
29 mov ecx,a
30 mov edx,80
31 call sread
32
33 mov eax,a
34 call atoi
35 jmp _functiona
36
37 _functiona:
38 mov edx,3
39 mul edx
```

Figure 18: Написание программы

Создаю исполняемый файл и проверяю его работу для значений x и a соответственно: (3;0), (1;2). (рис. 19).



```
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$ touch task2.asm
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$ nasm -f elf task2.asm
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$ ld -m elf_i386 task2.o -o task2
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$ ./task2
Введите x: 1
Введите a: 2
Результат: 6
dgavdadaev@dgavdadaev-laptop:~/work/study/2023-2024/Архитектура/arh-pc/lab07$ ./task2
Введите x: 3
Результат: 1
```

Figure 19: Запуск файла и проверка его работы

Программа работает корректно.

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,2
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,3
mul edx
jmp _end
_functionx:
add eax,-2
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 изучил команды условного и безусловного переходов, приобрел навыки написания программ с использованием переходов и ознакомился с назначением и структурой файла листинга, что поможет мне при выполнении последующих лабораторных работ.

## 6 Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 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. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.

10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: [http://www.stolyarov.info/books/asm\\_unix](http://www.stolyarov.info/books/asm_unix).
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).