

# **Отчёт по лабораторной работе №7**

**дисциплина: Архитектура компьютера**

Маслова Анна Павловна

# Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выполнение заданий для самостоятельной работы	17
4	Выводы	24
	Список литературы	25

## Список иллюстраций

2.1	Создание каталога lab07 и файла lab7-1.asm . . . . .	6
2.2	Текст программы файла lab7-1.asm . . . . .	7
2.3	Запуск файла lab7-1 . . . . .	7
2.4	Изменённый текст программы файла lab7-1.asm . . . . .	8
2.5	Запуск изменённого файла lab7-1 . . . . .	9
2.6	Повторно изменённый текст файла lab7-1.asm . . . . .	10
2.7	Запуск повторно изменённого файла lab7-1 . . . . .	10
2.8	Текст файла lab7-2.asm . . . . .	11
2.9	Текст файла lab7-2.asm . . . . .	12
2.10	Запуск файла lab7-2 . . . . .	13
2.11	Создание файла листинга для программы из файла lab7-2.asm . .	13
2.12	Файл листинга lab7-2.lst . . . . .	14
2.13	Удаление одного из двух операндов инструкции mov в файле lab7-2.asm . . . . .	15
2.14	Трансляция с получением файла листинга . . . . .	15
2.15	Файл листинга после изменений . . . . .	16
3.1	Текст файла func1.asm . . . . .	19
3.2	Запуск файла func1 . . . . .	19
3.3	Текст файла func2.asm . . . . .	22
3.4	Текст файла func2.asm . . . . .	23
3.5	Запуск файла func2 . . . . .	23

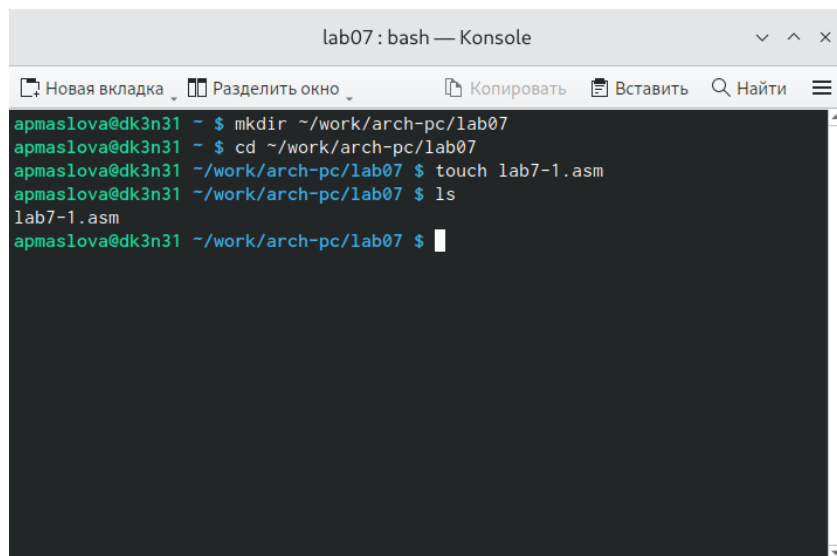
## **Список таблиц**

# 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

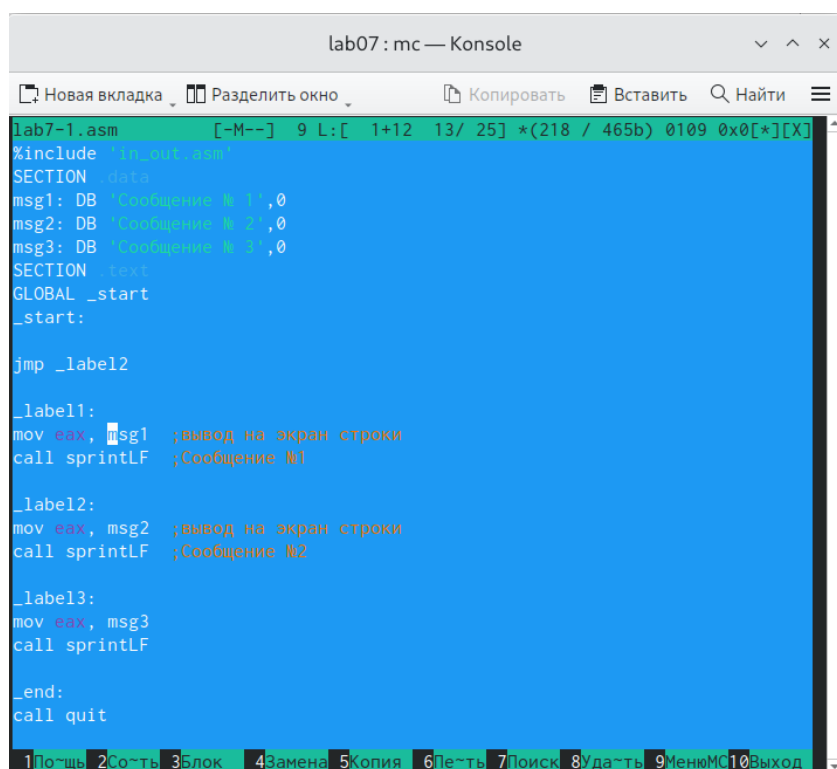
Создаём каталог для программ лабораторной работы №7, перейдём в него и создадим файл *lab7-1.asm* (рис. 2.1).



```
lab07: bash — Konsole
[+] Новая вкладка [ ] Разделить окно [ ] Копировать [ ] Вставить [ ] Найти [ ]
армаслова@dk3n31 ~ $ mkdir ~/work/arch-pc/lab07
армаслова@dk3n31 ~ $ cd ~/work/arch-pc/lab07
армаслова@dk3n31 ~/work/arch-pc/lab07 $ touch lab7-1.asm
армаслова@dk3n31 ~/work/arch-pc/lab07 $ ls
lab7-1.asm
армаслова@dk3n31 ~/work/arch-pc/lab07 $
```

Рис. 2.1: Создание каталога lab07 и файла lab7-1.asm

Введём в файл *lab7-1.asm* текст программы с использованием инструкции *jmp* (рис. 2.2).



```
lab7-1.asm [-M--] 9 L:[ 1+12 13/ 25] *(218 / 465b) 0109 0x0[*][X]
#include "in_out.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

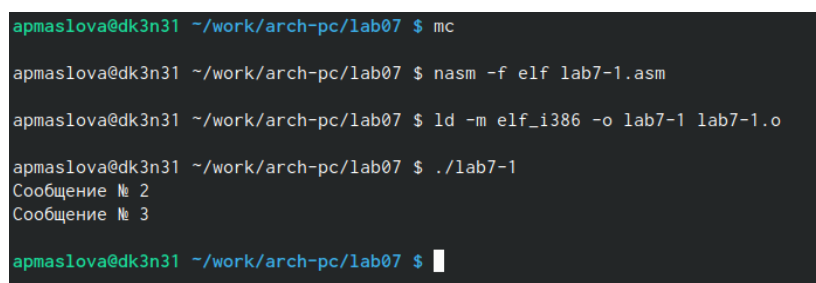
_label2:
mov eax, msg2 ;вывод на экран строки
call sprintf ;Сообщение №2

_label3:
mov eax, msg3
call sprintf

_end:
call quit
```

Рис. 2.2: Текст программы файла lab7-1.asm

Создадим исполняемый файл и запустим его (рис. 2.3).

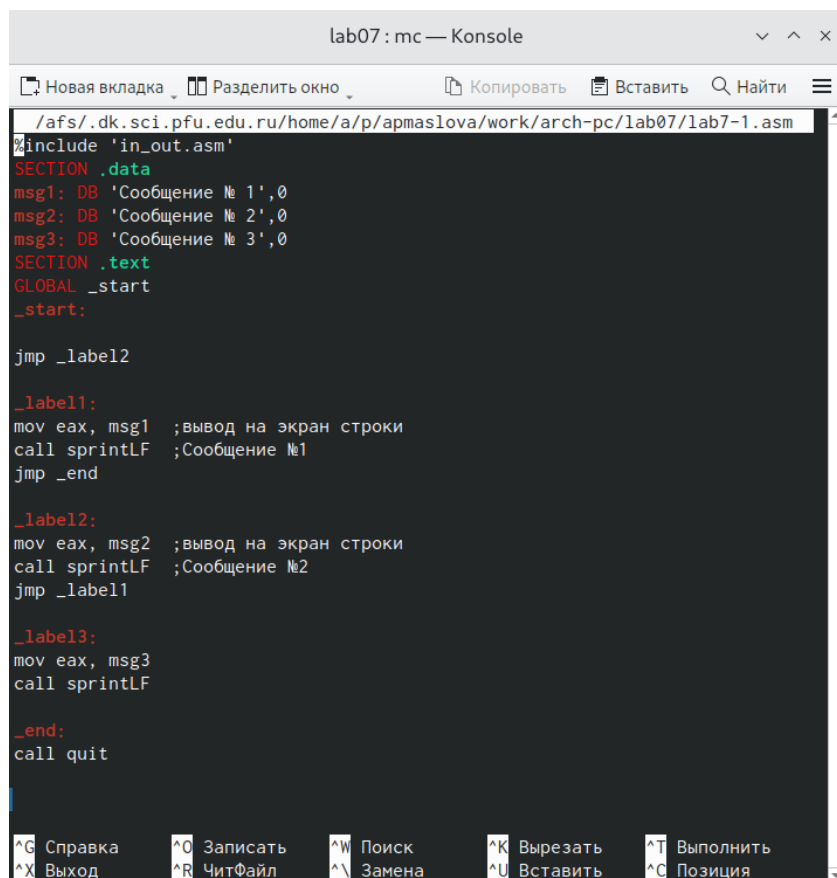


```
apmaslova@dk3n31 ~/work/arch-pc/lab07 $ mc
apmaslova@dk3n31 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
apmaslova@dk3n31 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
apmaslova@dk3n31 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 3
apmaslova@dk3n31 ~/work/arch-pc/lab07 $
```

Рис. 2.3: Запуск файла lab7-1

Мы видим, что “Сообщение №1” на экран не вывелось, но осуществился вывод “Сообщение №2” и “Сообщение №3”. Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения.

Изменим программу так, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого в текст программы после вывода сообщения № 2 добавим инструкцию `jmp` с меткой `_label1` (т.е. переход к инструкциям вывода сообщения № 1) и после вывода сообщения № 1 добавим инструкцию `jmp` с меткой `_end` (т.е. переход к инструкции `call quit`) (рис. 2.4).



```
lab07: mc — Konsole
/afs/.dk.sci.pfu.edu.ru/home/a/p/apmaslova/work/arch-pc/lab07/lab7-1.asm
%include 'in_out.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
call sprintf

_end:
call quit
```

Рис. 2.4: Изменённый текст программы файла lab7-1.asm

Создадим исполняемый файл и проверим его работу (рис. 2.5).



```
армаслова@dk3n31 ~/work/arch-pc/lab07 $ mc
армаслова@dk3n31 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
армаслова@dk3n31 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
армаслова@dk3n31 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 1
армаслова@dk3n31 ~/work/arch-pc/lab07 $
```

Рис. 2.5: Запуск изменённого файла lab7-1

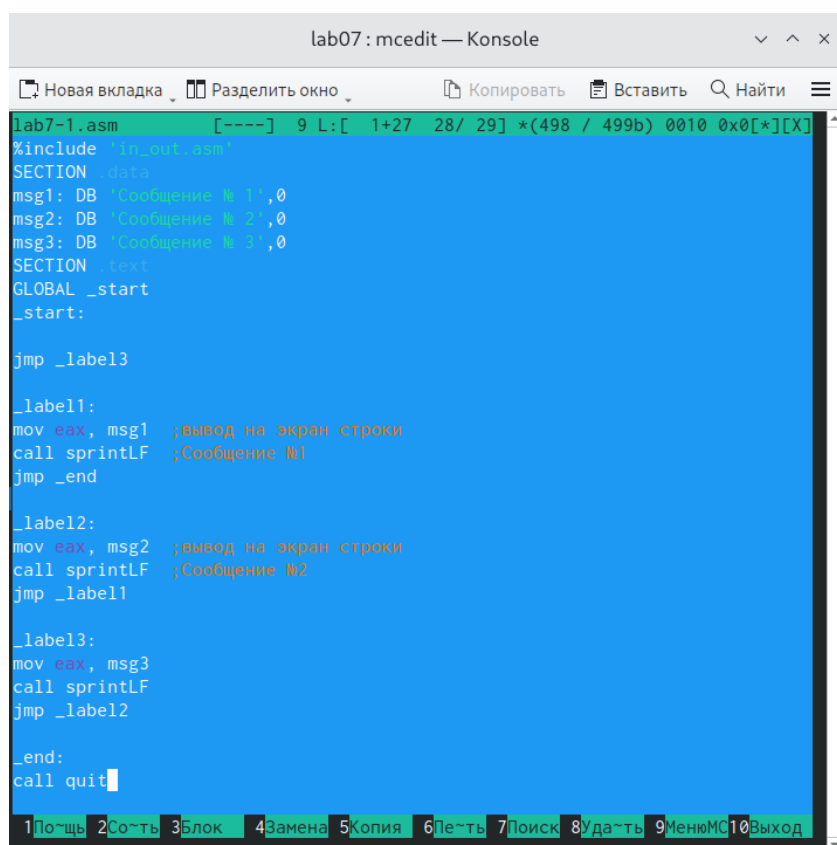
Как мы видим, программа работает корректно: выводится сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’, и работа завершилась.

Теперь изменим текст программы (рис. 2.6) так, чтобы вывод программы был следующим:

Сообщение № 3

Сообщение № 2

Сообщение № 1



```
lab7-1.asm 9 L: [ 1+27 28/ 29] *(498 / 499b) 0010 0x0[*][X]
%include 'in_out.asm'
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

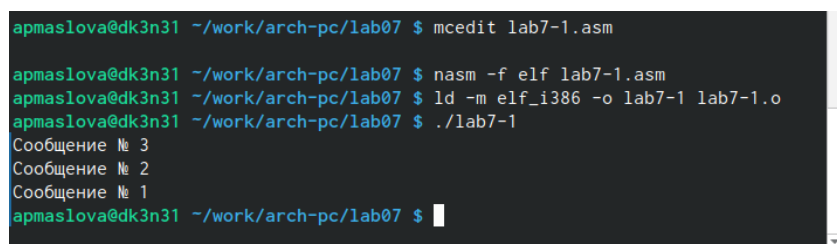
_label2:
mov eax, msg2 ;вывод на экран строки
call sprintf ;Сообщение №2
jmp _label1

_label3:
mov eax, msg3
call sprintf
jmp _label2

_end:
call quit
```

Рис. 2.6: Повторно изменённый текст файла lab7-1.asm

Создадим исполняемый файл и проверим его работу (рис. 2.7).



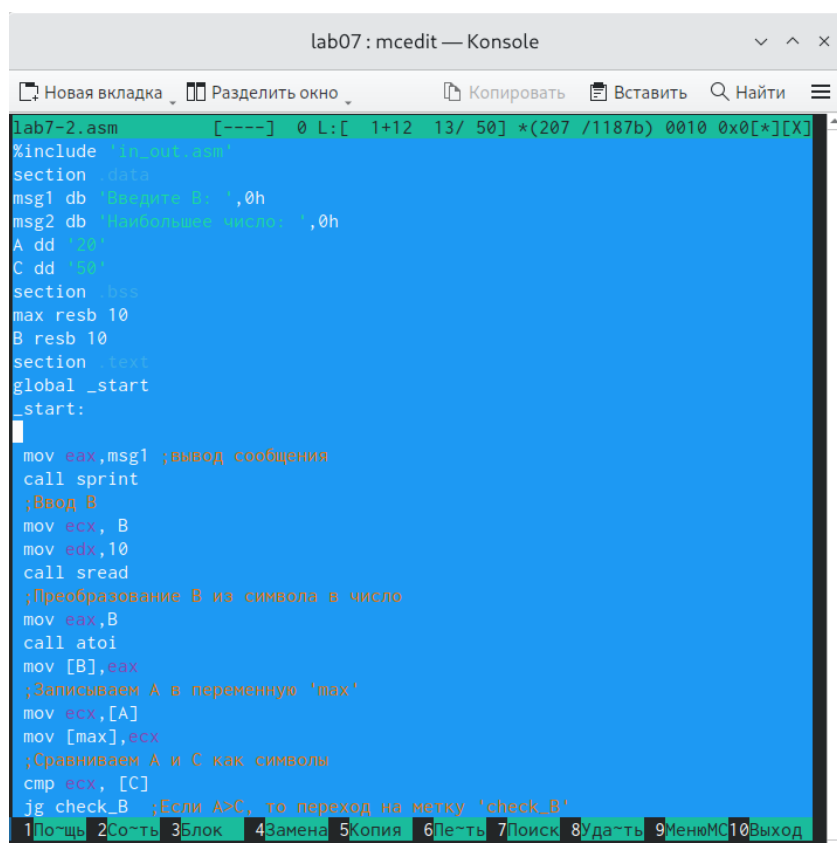
```
apmaslova@dk3n31 ~/work/arch-pc/lab07 $ mcedit lab7-1.asm
apmaslova@dk3n31 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
apmaslova@dk3n31 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
apmaslova@dk3n31 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
apmaslova@dk3n31 ~/work/arch-pc/lab07 $
```

Рис. 2.7: Запуск повторно изменённого файла lab7-1

Как мы видим, программа выводит нужные сообщения в верной последовательности.

Далее создадим файл *lab7-2.asm* в каталоге *~/work/arch-pc/lab07*. В этот файл введём текст программы, которая определяет и выводит на экран наибольшую

из 3 целочисленных переменных: А,В и С (рис. 2.8, рис. 2.9).



```
lab7-2.asm [---] 0 L: [ 1+12 13/ 50] *(207 /1187b) 0010 0x0[*][X]
#include "in_out.asm"
section .data
msg1 db "Введите В: ",0h
msg2 db "Наибольшие число: ",0h
A dd "20"
C dd "50"
section .bss
max resb 10
B resb 10
section .text
global _start
_start:
mov eax,msg1 ;вывод сообщения
call sprint
;Ввод В
mov ecx, B
mov edx,10
call sread
;Преобразование В из символа в число
mov eax,B
call atoi
mov [B],eax
;Записываем А в переменную 'max'
mov ecx,[A]
mov [max],ecx
;Сравниваем А и С как символы
cmp ecx, [C]
jg check_B ;Если А>С, то переход на метку 'check_B'
1Поч-ть 2Со-ть 3Блок 4Замена 5Копия 6Пе-ть 7Поиск 8Уда-ть 9МенюМС10Выход
```

Рис. 2.8: Текст файла lab7-2.asm

```
lab7-2.asm [----] 1 L: [ 22+28 50/ 50] *(1187/1187b) <EOF> [*][X]
call atoi
mov [B],eax
;Записываем A в переменную 'max'
mov ecx,[A]
mov [max],ecx
;Сравниваем A и C как символы
cmp ecx, [C]
jg check_B ;Если A>C, то переход на метку 'check_B'
mov ecx,[C] ;Иначе 'ecx=C'
mov [max],ecx
;Преобразование max(A,C) из символа в число
check_B:
mov eax,max
call atoi
mov [max],eax
;Сравниваем max(A,C) и B как числа
mov ecx,[max]
cmp ecx,[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 ;Вывод "Наибольшее число"
call sprintf
mov eax,[max] ;Вывод max(A,B,C)
call iprintLF
call quit
```

1Почть 2Со-ть 3Блок 4Замена 5Копия 6Те-ть 7Поиск 8Уда-ть 9МенюMC10Выход

Рис. 2.9: Текст файла lab7-2.asm

Теперь создадим исполняемый файл и проверим его работу (рис. 2.10).

```

армаслова@dk3n31 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
армаслова@dk3n31 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
армаслова@dk3n31 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 15
Наибольшее число: 50
армаслова@dk3n31 ~/work/arch-pc/lab07 $ mcedit lab7-2.asm

армаслова@dk3n31 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 70
Наибольшее число: 70
армаслова@dk3n31 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 20
Наибольшее число: 50
армаслова@dk3n31 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 50
Наибольшее число: 50
армаслова@dk3n31 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 100
Наибольшее число: 100
армаслова@dk3n31 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 0
Наибольшее число: 50
армаслова@dk3n31 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: -5
Наибольшее число: 50
армаслова@dk3n31 ~/work/arch-pc/lab07 $

```

Рис. 2.10: Запуск файла lab7-2

Как мы видим, с при разных введенных В программа выдаёт корректные результаты.

Теперь изучим структуру файла листинга. Создадим файл листинга для программы из файла *lab7-2.asm* и откроем этот файл с помощью редактора *mcedit* (рис. 2.11).

```

армаслова@dk3n31 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
армаслова@dk3n31 ~/work/arch-pc/lab07 $ ls
in_out.asm  lab7-1.asm  lab7-2      lab7-2.lst
lab7-1      lab7-1.o    lab7-2.asm  lab7-2.o
армаслова@dk3n31 ~/work/arch-pc/lab07 $ mcedit lab7-2.lst

```

Рис. 2.11: Создание файла листинга для программы из файла lab7-2.asm

Текст файла листинга представлен следующим образом: (рис. 2.12).

Рис. 2.12: Файл листинга lab7-2.lst

Рассмотрим подробно строки 20, 21 и 22.

- В строке 20 содержится только комментарий ;Преобразование B из символа в число. Этой строке присвоен определённый номер, однако в ней не генерируется никакой машинный код. Отсутствуют также поля с адресом и исходным текстом программы.
- В строке 21 содержится следующий текст программы: `mov eax,B`. Адрес `00000101` соответствует смещению машинного кода `B8[0a000000]` от начала текущего сегмента
- В строке 22 содержится следующий текст программы: `call atoi`. Адрес `00000106` соответствует смещению машинного кода `E891FFFFFF` от начала текущего сегмента.

Откроем файл с программой *lab7-2.asm* и в инструкции *mov* удалим один из двух операндов (рис. 2.13).

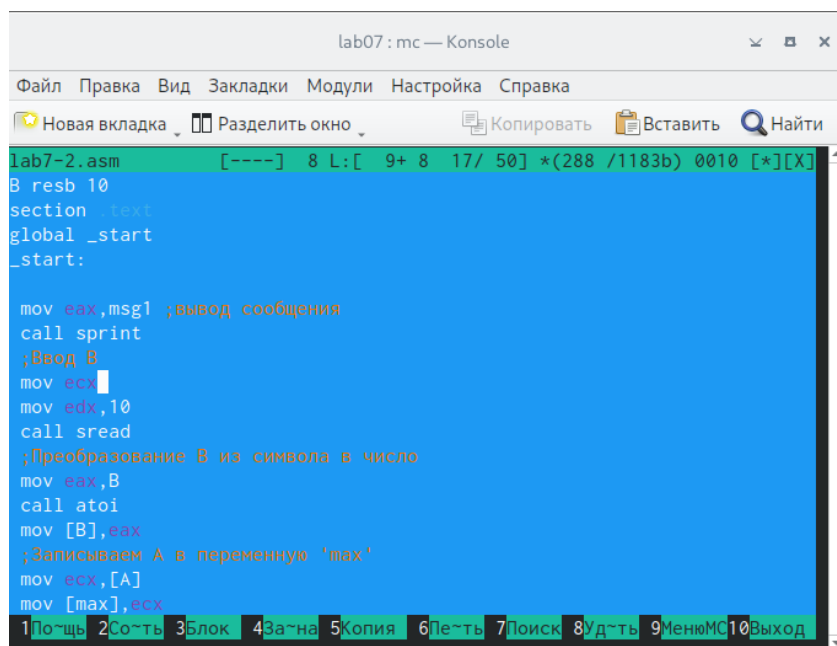


Рис. 2.13: Удаление одного из двух операндов инструкции mov в файле lab7-2.asm

Выполним трансляцию с получением файла листинга (рис. 2.14).

```

apmaslova@dk6n53 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.
asm
lab7-2.asm:17: error: invalid combination of opcode and operands

```

Рис. 2.14: Трансляция с получением файла листинга

Как мы видим, транслятор обнаружил ошибку при ассемблировании и вывел её на экран. Посмотрим, что произошло с созданным файлом листинга: (рис. 2.15)

```
lab7-2.lst [----] 0 L:[187+ 8 195/227] *(11866/14029b) 003[*][X]
12      _start:
13
14  000000E8 B8[00000000]      mov eax,msg1 ;вывод сообщения
15  000000ED E81DFFFFFF      call sprint
16                                ;Ввод B
17  mov ecx
17      *****      error: invalid combination of op
18  000000F2 BA0A000000      mov edx,10
19  000000F7 E847FFFFFF      call sread
20                                ;Преобразование B из символа в ч
21  000000FC B8[0A000000]      mov eax,B
22  00000101 E896FFFFFF      call atoi
23  00000106 A3[0A000000]      mov [B],eax
24                                ;Записываем A в переменную 'max'
25  0000010B 8B0D[36000000]      mov ecx,[A]
26  00000111 890D[00000000]      mov [max],ecx
27                                ;Сравниваем A и C как символы
28  00000117 3B0D[3A000000]      cmp ecx,[C]

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Печать 7Поиск 8Удалить 9МенюMC 10Выход
```

Рис. 2.15: Файл листинга после изменений

Как мы видим, в файле листинга также сказано, что в файле ошибка.



## 3 Выполнение заданий для самостоятельной работы

Теперь напишем программу нахождения наименьшей из 3 целочисленных переменных  $a, b$  и  $c$ . Значения переменных возьмём в соответствии с вариантом №15:  $a = 32, b = 6, c = 54$ . Создадим файл *func1.asm* и введём в него текст программы из листинга 7.1 (рис. 3.1).

**Листинг 7.1. Программа нахождения наименьшей из 3 целочисленных переменных  $a, b$  и  $c$**

```
%include 'in_out.asm'

section .data
msg db 'Наименьшее число: ',0h
a dd 32
b dd 6
c dd 54

section .bss
min resb 10

section .text
global _start
_start:

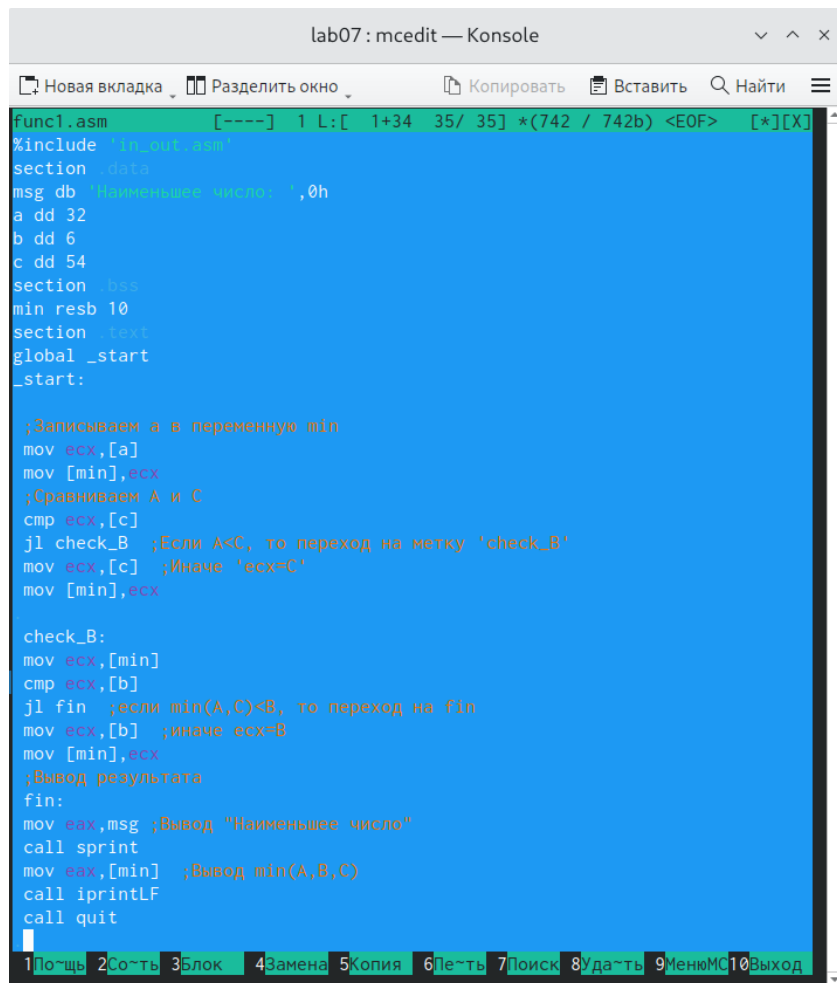
;Записываем a в переменную min
```

```

mov ecx,[a]
mov [min],ecx
;Сравниваем A и C
cmp ecx,[c]
jl check_B ;Если A<C, то переход на метку 'check_B'
mov ecx,[c] ;Иначе 'ecx=C'
mov [min],ecx

check_B:
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
;Вывод результата
fin:
mov eax,msg ;Вывод "Наименьшее число"
call sprint
mov eax,[min] ;Вывод min(A,B,C)
call iprintLF
call quit

```



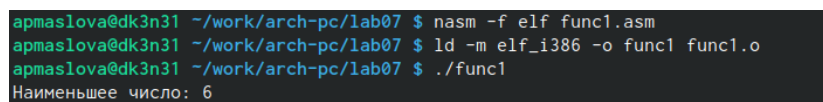
```
func1.asm [----] 1 L: [ 1+34 35/ 35] *(742 / 742b) <EOF> [*][X]
#include "in_out.asm"
section .data
msg db "Наименьшее число: ",0h
a dd 32
b dd 6
c dd 54
section .bss
min resb 10
section .text
global _start
_start:

;Записываем a в переменную min
mov ecx,[a]
mov [min],ecx
;Сравниваем A и C
cmp ecx,[c]
jl check_B ;Если A<C, то переход на метку 'check_B'
mov ecx,[c] ;Иначе 'ecx=c'
mov [min],ecx

check_B:
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
;Вывод результата
fin:
mov eax,msg ;Вывод "Наименьшее число"
call sprint
mov eax,[min] ;Вывод min(A,B,C)
call iprintLF
call quit
```

Рис. 3.1: Текст файла func1.asm

Создадим исполняемый файл и запустим программу (рис. 3.2).



```
армаслова@dk3n31 ~/work/arch-pc/lab07 $ nasm -f elf func1.asm
армаслова@dk3n31 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o func1 func1.o
армаслова@dk3n31 ~/work/arch-pc/lab07 $ ./func1
Наименьшее число: 6
```

Рис. 3.2: Запуск файла func1

На экран программа вывела наименьшую из трёх переменных, равную 6. Программа работает корректно.

Теперь напишем программу, которая для введенных с клавиатуры значений  $x$  и  $a$  вычисляет значение заданной функции  $f(x)$  и выводит результат вычислений.

Вид функции  $f(x)$ , соответствующий варианту №15, следующий:

$$f(x) = a + 10, x < a$$

$$f(x) = x + 10, x \geq a$$

Создадим файл *func2.asm* и введём в него текст программы из листинга 7.2 (рис. 3.3, рис. 3.4).

#### Листинг 7.2. Программа вычисления значения функции

```
%include 'in_out.asm'
SECTION .data
msg1: DB 'Введите x: ',0
msg2: DB 'Введите a: ',0
msg3: DB 'f(x) = ',0
section .bss
x resb 10
a resb 10
f resb 10
SECTION .text
GLOBAL _start
_start:
;Вывод сообщения и ввод x
mov eax,msg1
call sprint
mov ecx,x
mov edx,10
call sread
;Вывод сообщения и ввод a
```

```

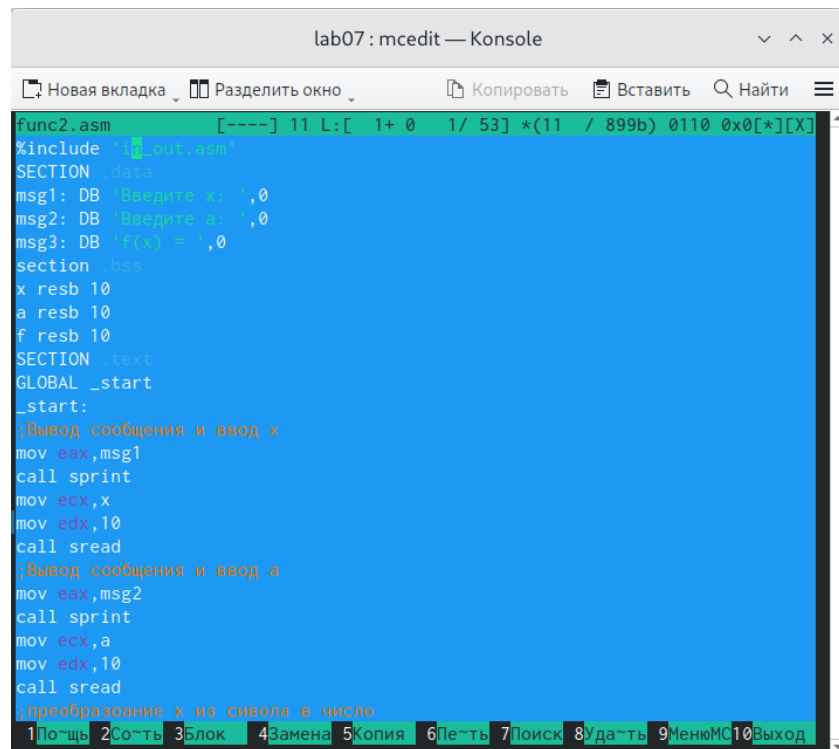
mov eax,msg2
call sprint
mov ecx,a
mov edx,10
call sread
;преобразоание x из сивола в число
mov eax,x
call atoi
mov [x],eax
;преобразование a из симвла в число
mov eax,a
call atoi
mov [a],eax
;Сравниваем x и a
mov ecx,[x]
cmp ecx,[a]
jl _label ;если x меньше a
mov eax,[x] ;иначе
jmp fun

_label:
mov eax,[a]
jmp fun

fun:
add eax,10
mov [f],eax
;Вывод сообщения и результата
mov eax,msg3

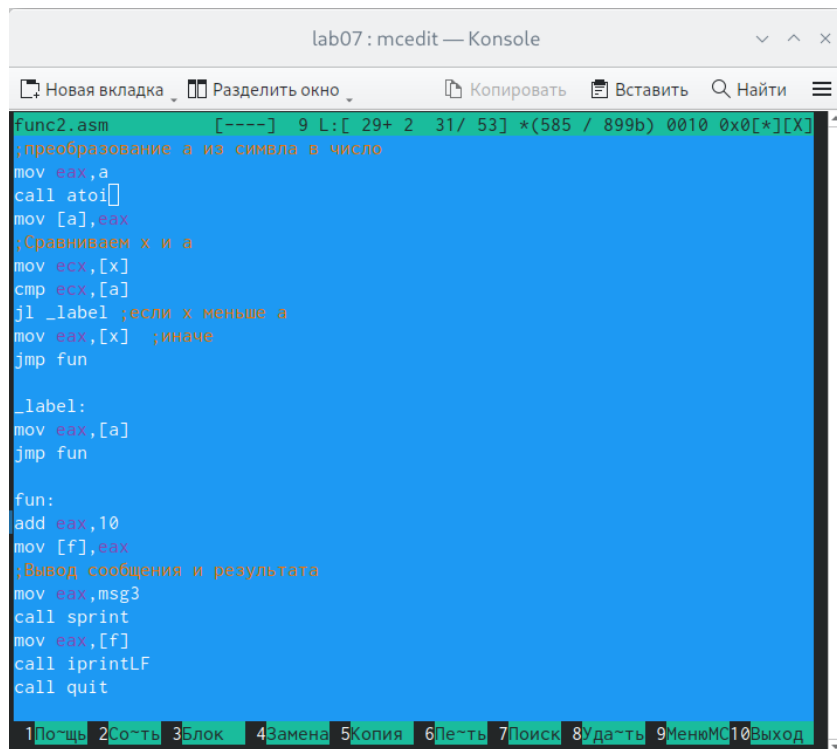
```

```
call sprint
mov eax,[f]
call iprintLF
call quit
```



```
func2.asm [----] 11 L:[ 1+ 0 1/ 53] *(11 / 899b) 0110 0x0[*][X]
#include "i_out.asm"
SECTION .data
msg1: DB "Введите x:",0
msg2: DB "Введите a:",0
msg3: DB "f(x) = ",0
section .bss
x resb 10
a resb 10
f resb 10
SECTION .text
GLOBAL _start
_start:
;Вывод сообщения и ввод x
mov eax,msg1
call sprint
mov ecx,x
mov edx,10
call sread
;Вывод сообщения и ввод a
mov eax,msg2
call sprint
mov ecx,a
mov edx,10
call sread
;преобразование x из символа в число
1Почть 2Со-ть 3Блок 4Замена 5Копия 6Печть 7Поиск 8Уда-ть 9МенюMC10Выход
```

Рис. 3.3: Текст файла func2.asm



```
func2.asm  [----]  9  L:[ 29+ 2 31/ 53]  *(585 / 899b)  0010 0x0[*][X]
;преобразование a из символа в число
mov  eax,a
call atoi
mov  [a],eax
;Сравниваем x и a
mov  ecx,[x]
cmp  ecx,[a]
jl  _label ;если x меньше a
mov  eax,[x] ;иначе
jmp  fun

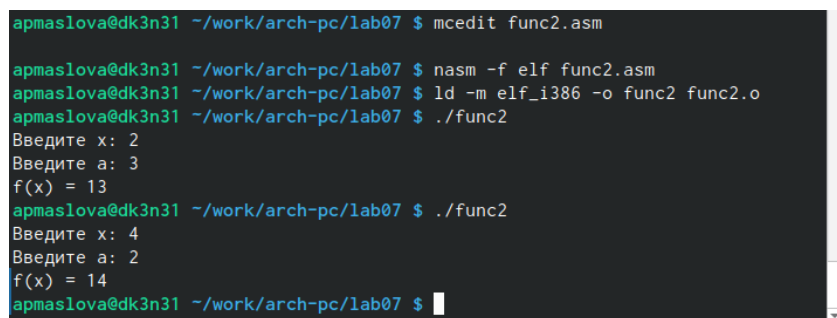
_label:
mov  eax,[a]
jmp  fun

fun:
add  eax,10
mov  [f],eax
;Вывод сообщения и результата
mov  eax,msg3
call sprint
mov  eax,[f]
call iprintLF
call quit

1По-щ-ь 2Со-ть 3Блок 4Замена 5Копия 6Де-ть 7Поиск 8Уда-ть 9МенюMC10Выход
```

Рис. 3.4: Текст файла func2.asm

Создадим исполняемый файл и проверим его работу для заданных значений:  
\$ x\_1 , a\_1 = (2;3), x\_2 , a\_2 = (4;2) \$ (рис. 3.5)



```
apmaslova@dk3n31 ~/work/arch-pc/lab07 $ mcedit func2.asm
apmaslova@dk3n31 ~/work/arch-pc/lab07 $ nasm -f elf func2.asm
apmaslova@dk3n31 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o func2 func2.o
apmaslova@dk3n31 ~/work/arch-pc/lab07 $ ./func2
Введите x: 2
Введите a: 3
f(x) = 13
apmaslova@dk3n31 ~/work/arch-pc/lab07 $ ./func2
Введите x: 4
Введите a: 2
f(x) = 14
apmaslova@dk3n31 ~/work/arch-pc/lab07 $
```

Рис. 3.5: Запуск файла func2

После проверки убедились, что программа работает верно.

## 4 Выводы

Мы познакомились с командами условного и безусловного переходов языка ассемблера NASM и научились писать программы с их использованием. Также изучили назначение и структуру файла листинга.



## 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,
- 11.