

Лабораторная работа №7

**Команды безусловного и условного переходов в Nasm.
Программирование ветвлений**

Хасанов Тимур

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выполнение задания для самостоятельной работы	17
4	Выводы	22

Список иллюстраций

2.1	Создание файла lab7-1.asm	6
2.2	Запуск Midnight commander	7
2.3	Вставка кода из файла листинга 7.1	8
2.4	Копирование файла in_out.asm в рабочую директорию	9
2.5	Сборка программы из файла lab7-1.asm и её запуск	9
2.6	Изменение файла lab7-1.asm согласно листингу 7.2	10
2.7	Повторная сборка программы из файла lab7-1.asm и её запуск	10
2.8	Редактирование файла lab7-1.asm	11
2.9	Повторная сборка программы из файла lab7-1.asm и её запуск	11
2.10	Создание второго файла: lab7-2.asm	11
2.11	Запись кода из листинга 7.3 в файл lab7-2.asm	12
2.12	сборка программы из файла lab7-2.asm и её запуск	12
2.13	Создание файла листинга из файла lab7-2.asm	13
2.14	Открытие файла листинга в текстовом редакторе	13
2.15	Вид файла листинга	13
2.16	Нахождение нашей программы в файле листинга	14
2.17	Изменение исходного файла	15
2.18	Вывод ошибки при сборке объектного файла	15
2.19	Отображение ошибки в листинге	16
3.1	Создание первого файла самостоятельной работы	17
3.2	Код первого файла самостоятельной работы	18
3.3	Код первого файла самостоятельной работы (продолжение)	19
3.4	Сборка и запуск программы первого задания самостоятельной работы, а также результат выполнения	19
3.5	Создание второго файла самостоятельной работы	19
3.6	Код второго файла самостоятельной работы	20
3.7	Код второго файла самостоятельной работы (продолжение)	20
3.8	Сборка и тестирование второго файла самостоятельной работы	21

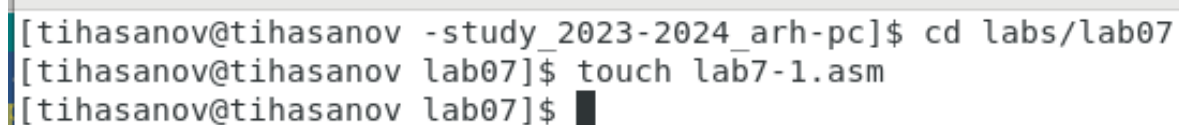
Список таблиц

1 Цель работы

Понять принцип работы условных и безусловных переходов в Ассемблере и научиться писать программы с командами, отвечающими за переходы. Научиться работать с файлами листинга и уметь их читать.

2 Выполнение лабораторной работы

Для начала выполнения лабораторной работы необходимо перейти в рабочую папку lab07 и создать файл lab7-1.asm (рис. 2.1):

A terminal window with a light gray background. The prompt is [tihasanov@tihasanov -study_2023-2024_arh-pc]\$. The first command is cd labs/lab07. The second prompt is [tihasanov@tihasanov lab07]\$. The second command is touch lab7-1.asm. The third prompt is [tihasanov@tihasanov lab07]\$.

```
[tihasanov@tihasanov -study_2023-2024_arh-pc]$ cd labs/lab07
[tihasanov@tihasanov lab07]$ touch lab7-1.asm
[tihasanov@tihasanov lab07]$
```

Рис. 2.1: Создание файла lab7-1.asm

После чего, для удобства, запустить Midnight commander (рис. 2.2):

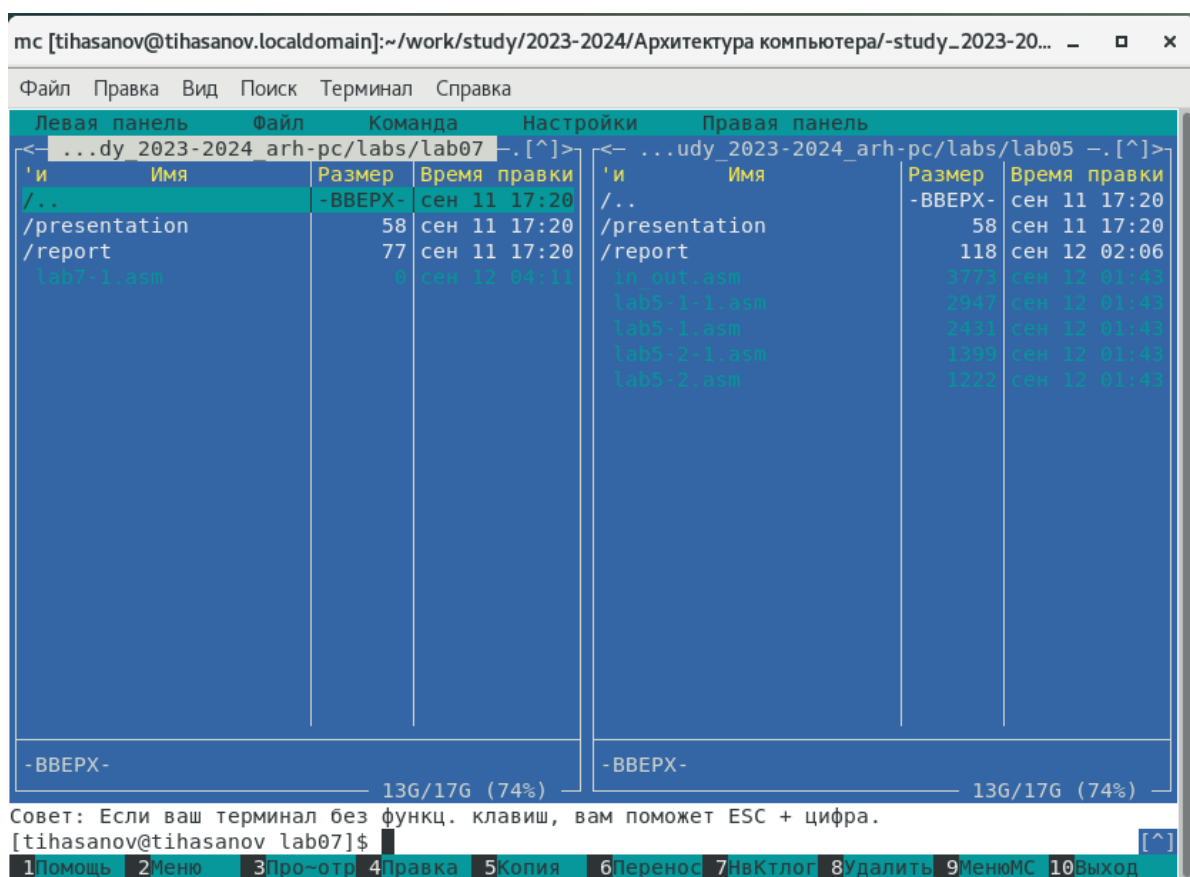


Рис. 2.2: Запуск Midnight commander

Вставим код в файл lab7-1.asm из файла листинга (рис. 2.3):

```

lab7-1.asm      [-M--] 31 L:[ 1+17 18/ 20] *(574 / 649b) 0010 0x00A
%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 sprintLF ; 'Сообщение № 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения

```

Рис. 2.3: Вставка кода из файла листинга 7.1

Теперь скопируем файл `in_out.asm` из рабочей директории прошлой лабораторной работы (рис. 2.4):

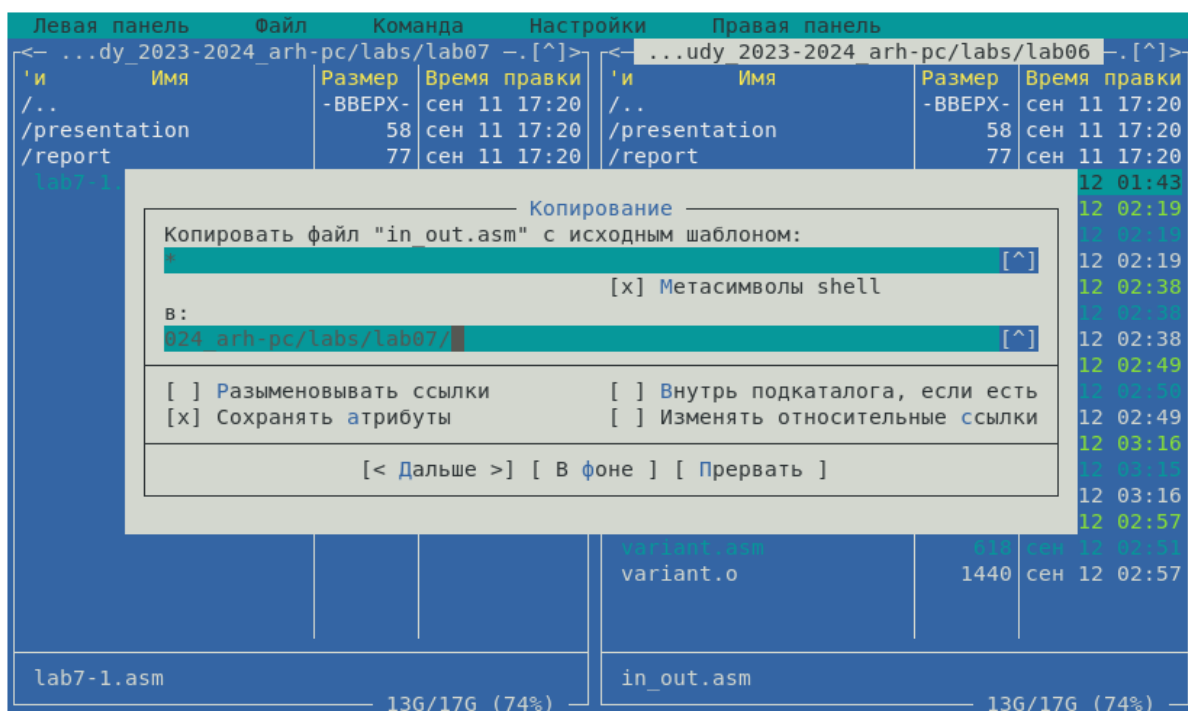


Рис. 2.4: Копирование файла in_out.asm в рабочую директорию

Теперь соберём программу из файла lab7-1.asm и запустим её (рис. 2.5):

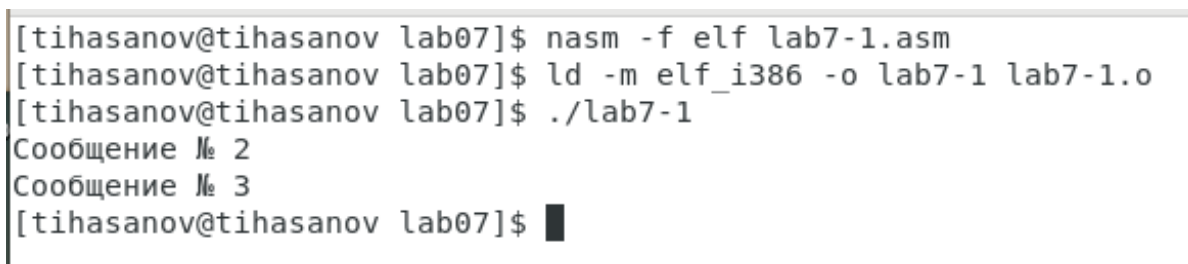


Рис. 2.5: Сборка программы из файла lab7-1.asm и её запуск

Изменим файл lab7-1.asm согласно листингу 7.2 (рис. 2.6):

```

lab7-1.asm      [-M--] 11 L:[ 1+16 17/ 22] *(487 / 670b) 0010 0x00A
%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 sprintLF ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения

```

Рис. 2.6: Изменение файла lab7-1.asm согласно листингу 7.2

Снова соберём программу и запустим её (рис. 2.7):

```

[tihasanov@tihasanov lab07]$ nasm -f elf lab7-1.asm
[tihasanov@tihasanov lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[tihasanov@tihasanov lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 1
[tihasanov@tihasanov lab07]$ █

```

Рис. 2.7: Повторная сборка программы из файла lab7-1.asm и её запуск

Теперь сделаем так, чтобы код выводил сообщения в обратном порядке (от 3 сообщения к первому). Для этого внесём в код следующие изменения (рис. 2.8):

```

lab7-1.asm      [-M--] 41 L:[ 1+22 23/ 23] *(682 / 682b) <EOF>
%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. 2.8: Редактирование файла lab7-1.asm

И запустим её, предварительно собрав (рис. 2.9):

```

[tihasanov@tihasanov lab07]$ nasm -f elf lab7-1.asm
[tihasanov@tihasanov lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[tihasanov@tihasanov lab07]$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
[tihasanov@tihasanov lab07]$

```

Рис. 2.9: Повторная сборка программы из файла lab7-1.asm и её запуск

Теперь создадим файл lab7-2.asm (рис. 2.10):

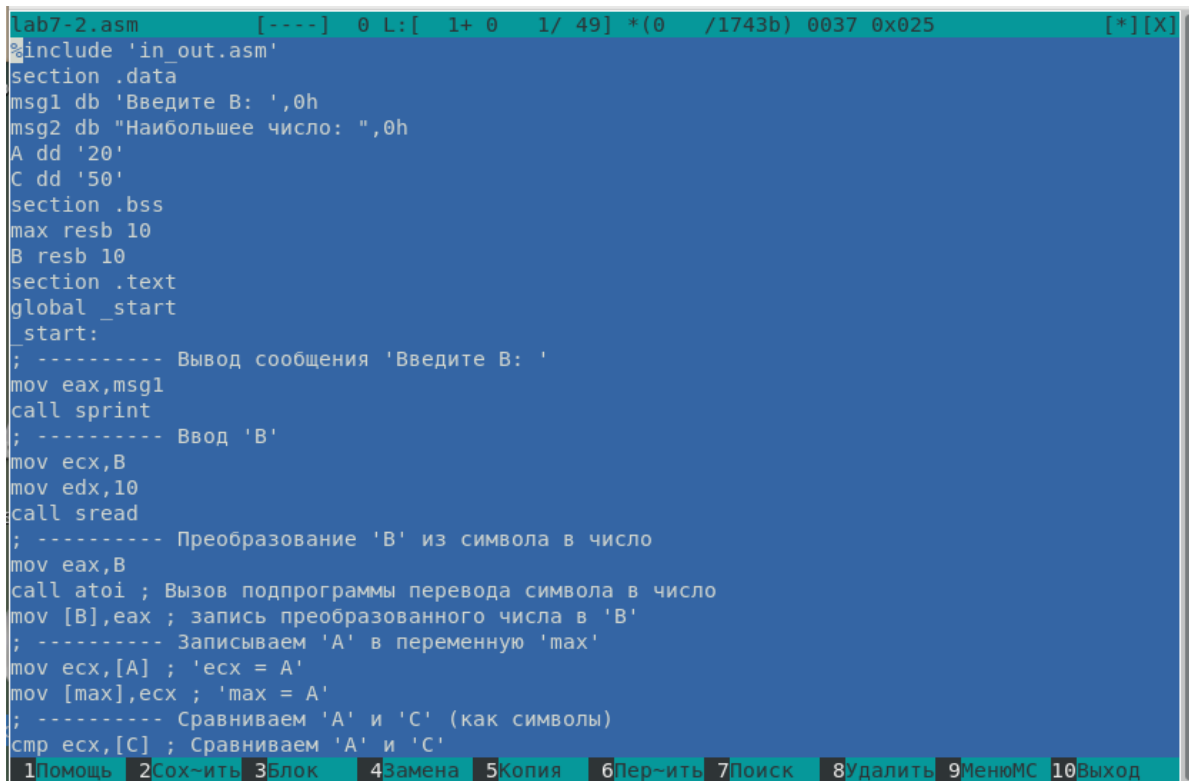
```

[tihasanov@tihasanov lab07]$ touch lab7-2.asm
[tihasanov@tihasanov lab07]$

```

Рис. 2.10: Создание второго файла: lab7-2.asm

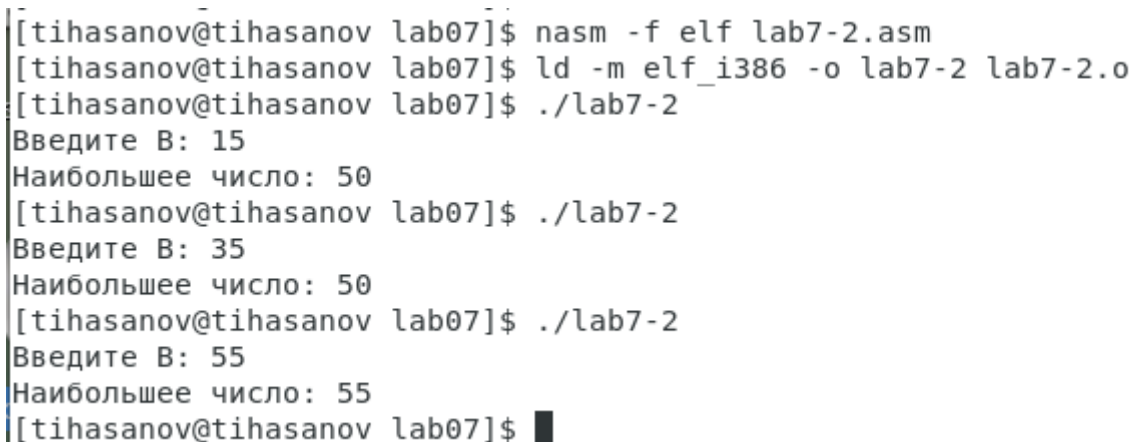
Запишем код из листинга 7.3 в файл lab7-2.asm (рис. 2.11):



```
lab7-2.asm  [----]  0 L:[ 1+ 0  1/ 49] *(0  /1743b) 0037 0x025  [*][X]
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. 2.11: Запись кода из листинга 7.3 в файл lab7-2.asm

И запустим его, предварительно собрав (рис. 2.12):



```
[tihasanov@tihasanov lab07]$ nasm -f elf lab7-2.asm
[tihasanov@tihasanov lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[tihasanov@tihasanov lab07]$ ./lab7-2
Введите B: 15
Наибольшее число: 50
[tihasanov@tihasanov lab07]$ ./lab7-2
Введите B: 35
Наибольшее число: 50
[tihasanov@tihasanov lab07]$ ./lab7-2
Введите B: 55
Наибольшее число: 55
[tihasanov@tihasanov lab07]$
```

Рис. 2.12: сборка программы из файла lab7-2.asm и её запуск

Теперь попробуем создать файл листинга при сборке файла lab7-2.asm (рис.

2.13):

```
[tihasanov@tihasanov lab07]$ nasm -f elf -l lab7-2.l lab7-2.asm  
[tihasanov@tihasanov lab07]$
```

Рис. 2.13: Создание файла листинга из файла lab7-2.asm

Теперь посмотрим, как выглядит файл листинга изнутри. Для этого откроем его в mcedit (рис. 2.14):

```
[tihasanov@tihasanov lab06]$ mcedit lab7-2.l
```

Рис. 2.14: Открытие файла листинга в текстовом редакторе

Открыв его, мы видим следующую картину (рис. 2.15):

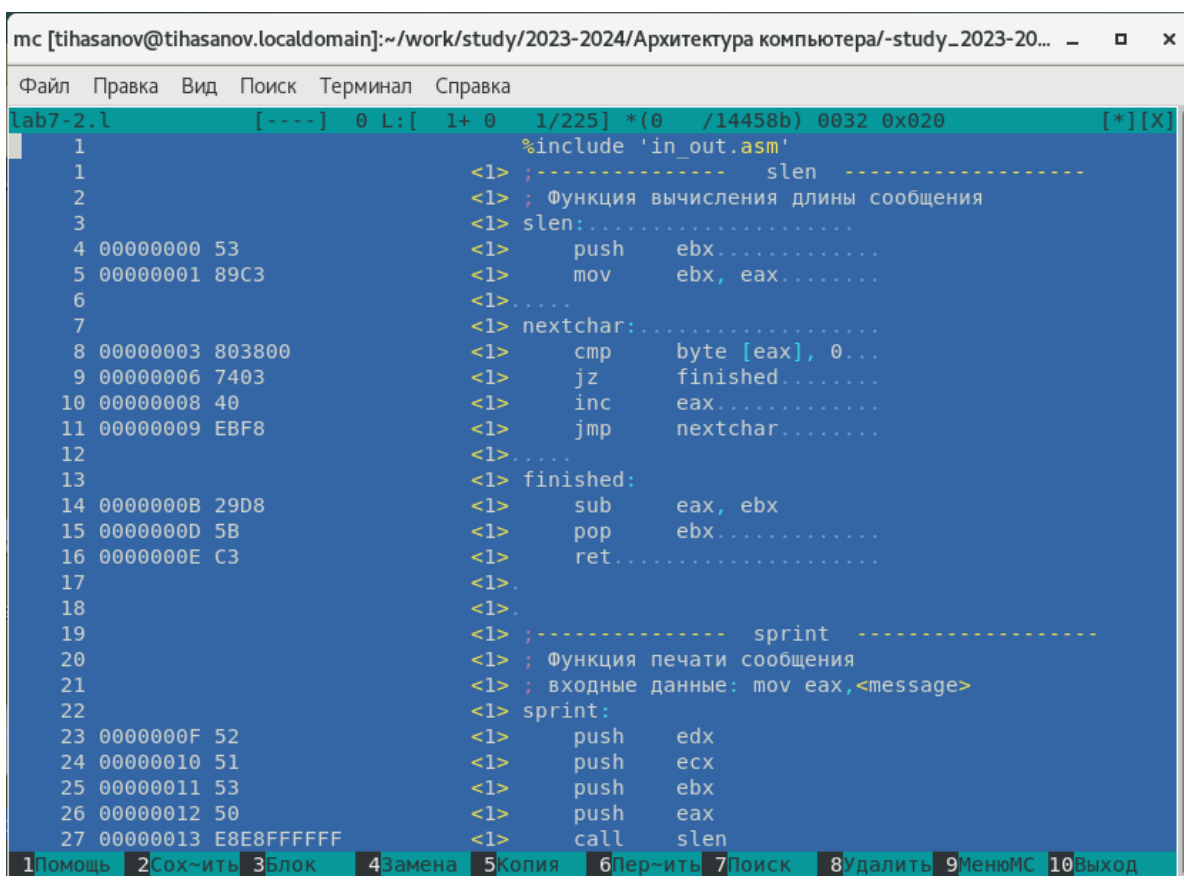


Рис. 2.15: Вид файла листинга

Наша программа находится чуть ниже (рис. 2.16):

```
lab7-2.l      [----]  0 L:[171+ 0 171/225] *(10588/14458b) 0032 0x020  [*][X]
170 000000E7 C3          <1>      ret
2                                section .data
3 00000000 D092D0B2D0B5D0B4D0-    msg1 db 'Введите В: ',0h
3 00000009 B8D182D0B520423A20-
3 00000012 00.....
4 00000013 D09DD0B0D0B8D0B1D0-    msg2 db "Наибольшее число: ",0h
4 0000001C BED0BBD18CD188D0B5-
4 00000025 D0B520D187D0B8D181-
4 0000002E D0BBD0BE3A2000....
5 00000035 32300000          A dd '20'
6 00000039 35300000          C dd '50'
7                                section .bss
8 00000000 <res Ah>          max resb 10
9 0000000A <res Ah>          B resb 10
10                               section .text
11                               global _start
12                               _start:
13                               ; ----- Вывод сообщения 'Введите В: '
14 000000E8 B8[00000000]      mov eax,msg1
15 000000ED E81DFFFFFF        call sprint
16                               ; ----- Ввод 'В'
17 000000F2 B9[0A000000]      mov ecx,B
18 000000F7 BA0A000000        mov edx,10
19 000000FC E842FFFFFF        call sread
20                               ; ----- Преобразование 'В' из символа в число
21 00000101 B8[0A000000]      mov eax,B
22 00000106 E891FFFFFF        call atoi ; Вызов подпрограммы перевода символа в ч
23 0000010B A3[0A000000]      mov [B],eax ; запись преобразованного числа в 'В'
```

Рис. 2.16: Нахождение нашей программы в файле листинга

Разберём несколько строк файла листинга:

1. Строка под номером 14 перемещает содержимое msg1 в регистр eax. Адрес указывается сразу после номера. Следом идёт машинный код, который представляет собой исходную ассемблированную строку в виде шестнадцатиричной системы. Далее идёт исходный код

2. 15-ая строка отвечает за вызов функции sprint. Она также имеет адрес и машинный код

3. Строка 17 отвечает за запись переменной В в регистр ecx. Как видно, все строки имеют номер, адрес, машинный код и исходный код.

Теперь попробуем намеренно допустить ошибку в нашем коде, убрав у команды move 1 операнд (рис. 2.17):

```

lab7-2.asm      [-M--]  7 L:[ 1+17 18/ 49] *(338 /1740b) 0010 0x00A
#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,
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. 2.17: Изменение исходного файла

И попробуем собрать файл с ошибкой, генерируя файл листинга (рис. 2.18):

```

[tihasanov@tihasanov lab07]$ nasm -f elf -l lab7-2.l lab7-2.asm
lab7-2.asm:18: error: invalid combination of opcode and operands
[tihasanov@tihasanov lab07]$ █

```

Рис. 2.18: Вывод ошибки при сборке объектного файла

Мы видим, что объектный файл не создался, однако появился файл листинга. Теперь зайдём в файл листинга, и посмотрим, отображается ли в нём ошибка (рис. 2.19):

```

lab7-2.1      [----]  0 L:[171+ 0 171/226] *(10588/14546b) 0032 0x020      [*][X]
170 000000E7 C3          <1>      ret
2                          section .data
3 00000000 D092D0B2D0B5D0B4D0-    msg1 db 'Введите B: ',0h
3 00000009 B8D182D0B520423A20-
3 00000012 00.....
4 00000013 D09DD0B0D0B8D0B1D0-    msg2 db "Наибольшее число: ",0h
4 0000001C BED0BBD18CD188D0B5-
4 00000025 D0B520D187D0B8D181-
4 0000002E D0BBD0BE3A2000.....
5 00000035 32300000          A dd '20'
6 00000039 35300000          C dd '50'
7                          section .bss
8 00000000 <res Ah>          max resb 10
9 0000000A <res Ah>          B resb 10
10                         section .text
11                         global _start
12                         _start:
13                         ; ----- Вывод сообщения 'Введите B: '
14 000000E8 B8[00000000]      mov eax,msg1
15 000000ED E81DFFFFFF      call sprint
16                         ; ----- Ввод 'B'
17 000000F2 B9[0A000000]      mov ecx,B
18                         mov edx
18                         *****
19 000000F7 E847FFFFFF      error: invalid combination of opcode and operands
20                         call sread
21                         ; ----- Преобразование 'B' из символа в число
21 000000FC B8[0A000000]      mov eax,B
22 00000101 E896FFFFFF      call atoi ; Вызов подпрограммы перевода символа в ч

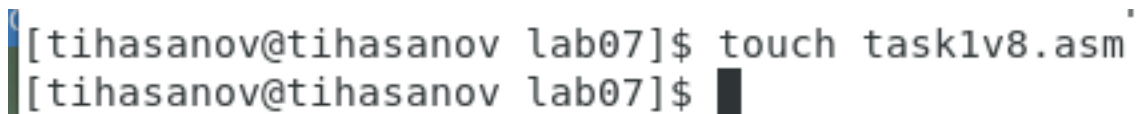
```

Рис. 2.19: Отображение ошибки в листинге

Как видим, в листинге прописана ошибка

3 Выполнение задания для самостоятельной работы

Создадим файл для выполнения самостоятельной работы. Мой вариант - 8 (рис. 3.1):



```
[tihasanov@tihasanov lab07]$ touch task1v8.asm  
[tihasanov@tihasanov lab07]$
```

Рис. 3.1: Создание первого файла самостоятельной работы

Напишем код для выполнения задания. Код выглядит так (рис. 3.2 и рис. 3.3):

```

tasklv8.asm      [-M~-]  0 L:[ 1+ 0  1/ 48] *(0  /1744b) 0037 0x025
%include 'in_out.asm'
section .data
msg2 db "Наименьшее число: ",0h
A dd '52'
B dd '33'
C dd '40'
section .bss
min resb 10
section .text
global _start
_start:
; ----- 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'

mov eax,A
call atoi ; Вызов подпрограммы перевода символа в число
mov [A],eax ; запись преобразованного числа в 'A'

mov eax,C
call atoi ; Вызов подпрограммы перевода символа в число
mov [C],eax ; запись преобразованного числа в 'C'

; ----- 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'
jl 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:
; ----- Сравниваем 'min(A,C)' и 'B' (как числа)

```

Рис. 3.2: Код первого файла самостоятельной работы

```

; ----- 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, msg2
call sprint ; Вывод сообщения 'Наименьшее число: '
mov eax,[min]
call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход

```

Рис. 3.3: Код первого файла самостоятельной работы (продолжение)

Соберём, запустим его и посмотрим на результат (рис. 3.4):

```

[tihasanov@tihasanov lab07]$ nasm -f elf task1v8.asm
[tihasanov@tihasanov lab07]$ ld -m elf_i386 -o task1v8 task1v8.o
[tihasanov@tihasanov lab07]$ ./task1v8
Наименьшее число: 33
[tihasanov@tihasanov lab07]$ █

```

Рис. 3.4: Сборка и запуск программы первого задания самостоятельной работы, а также результат выполнения

Теперь создадим второй файл самостоятельной работы для второго задания (рис. 3.5):

```

[tihasanov@tihasanov lab07]$ touch task2v8.asm
[tihasanov@tihasanov lab07]$ █

```

Рис. 3.5: Создание второго файла самостоятельной работы

Код будет выглядеть так (рис. 3.6 и рис. 3.7):

```

task2v8.asm      [----]  9 L:[  1+ 0   1/ 49] *(9   / 583b) 0039 0x027
%include 'in_out.asm'
section .data
msg1 DB "Введите X: ",0h
msg2 DB "Введите A: ",0h
msg3 DB "Ответ=",0h
section .bss
x: RESB 80
a: RESB 80
ans: RESB 80
section .text
global _start
_start:
mov eax, msg1
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
mov [x], eax

mov eax, msg2
call sprint
mov ecx, a
mov edx, 80
call sread
mov eax, a
call atoi
mov [a], eax

mov eax, [a]
cmp eax, 3
jl calc_3a

```

Рис. 3.6: Код второго файла самостоятельной работы

```

mov eax, [x]
add eax, 1
jmp ansv

calc_3a:
mov ebx, 3
mul ebx

ansv:
mov [ans], eax
mov eax, msg3
call sprint
mov eax, [ans]
call iprintLF
call quit

```

Рис. 3.7: Код второго файла самостоятельной работы (продолжение)

Соберём исполняемый файл и запустим его (рис. 3.8):

```
[tihasanov@tihasanov lab07]$ nasm -f elf task2v8.asm
[tihasanov@tihasanov lab07]$ ld -m elf_i386 -o task2v8 task2v8.o
[tihasanov@tihasanov lab07]$ ./task2v8
Введите X: 1
Введите A: 4
Ответ=2
[tihasanov@tihasanov lab07]$ ./task2v8
Введите X: 1
Введите A: 2
Ответ=6
[tihasanov@tihasanov lab07]$ █
```

Рис. 3.8: Сборка и тестирование второго файла самостоятельной работы

Как видим, программа всё посчитала правильно

4 Выводы

В результате работы над лабораторной работой были написаны программы, которые используют команды условных и безусловных переходов, были получены навыки работы с этими командами, а также были созданы и успешно прочитаны листинги для некоторых из программ.