

Отчёт по лабораторной работе 7

Архитектура компьютеров и операционные системы

Абдулфазова Лейла Али гызы

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация переходов в NASM	8
4.2	Изучение структуры файлы листинга	14
4.3	Выполнение заданий для самостоятельной работы	17
5	Выводы	22

Список иллюстраций

4.1	Редактирование файла lab7-1.asm	9
4.2	Проверка кода lab7-1.asm	9
4.3	Редактирование файла lab7-1.asm	10
4.4	Проверка кода lab7-1.asm	11
4.5	Редактирование файла lab7-1.asm	12
4.6	Проверка кода lab7-1.asm	12
4.7	Редактирование файла lab7-2.asm	13
4.8	Проверка кода lab7-2.asm	14
4.9	Файл листинга lab7-2	15
4.10	Ошибка трансляции lab7-2	16
4.11	Файл листинга с ошибкой lab7-2	17
4.12	Редактирование файла prog-1.asm	18
4.13	Проверка кода prog-1.asm	18
4.14	Редактирование файла prog-2.asm	20
4.15	Проверка кода prog-2.asm	21

Список таблиц

1 Цель работы

Целью работы является изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

2 Задание

1. Изучение команд условного и безусловного перехода
2. Изучение файла листинга
3. Выполнение заданий, рассмотрение примеров
4. Выполнение заданий 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` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление

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

Команда условного перехода имеет вид

`j<мнемоника перехода> label`

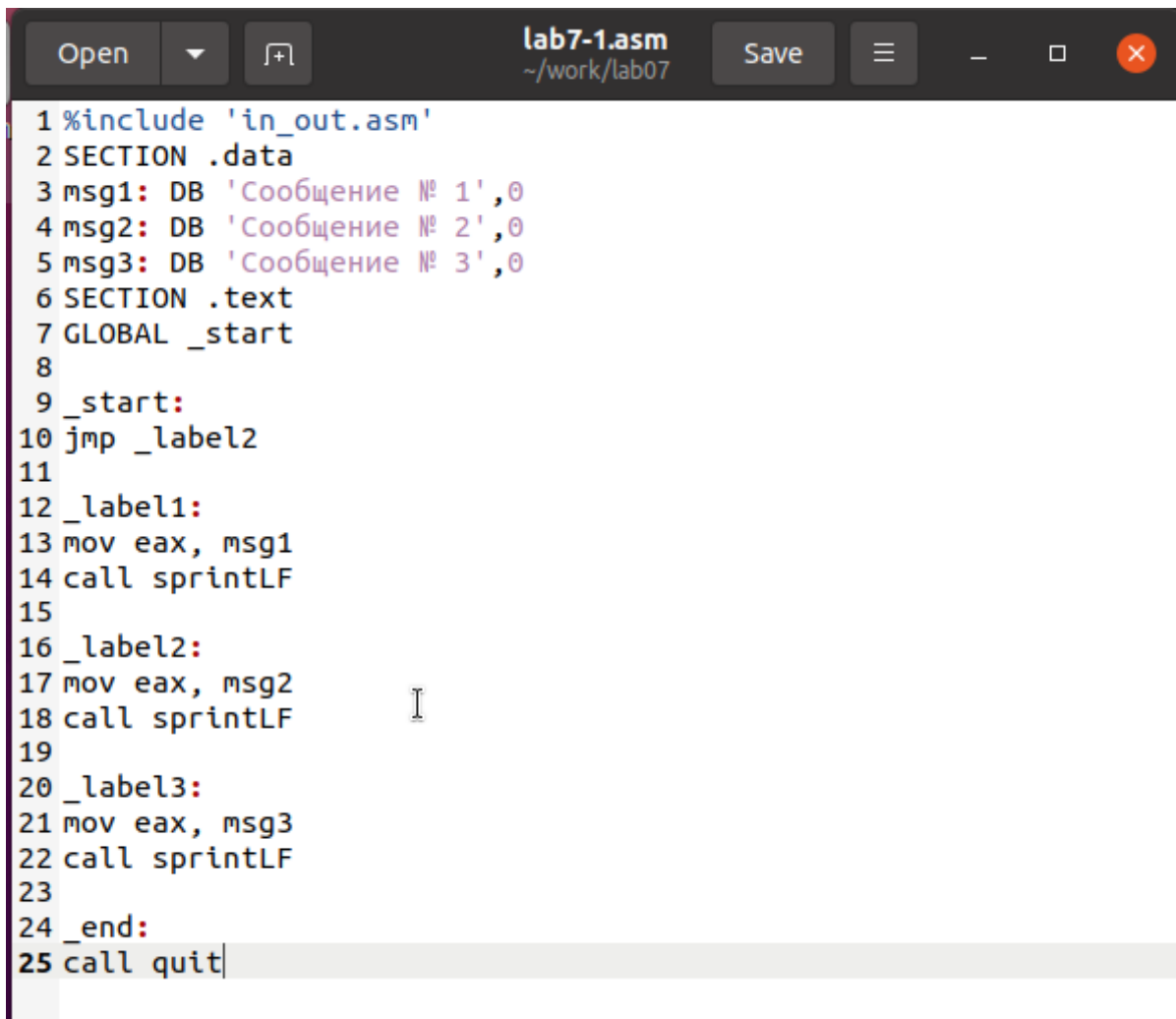
Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов.

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

Создала каталог для программ лабораторной работы No7 и файл с названием “lab7-1.asm”. Инструкция `jmp` в NASM используется для безусловных переходов.

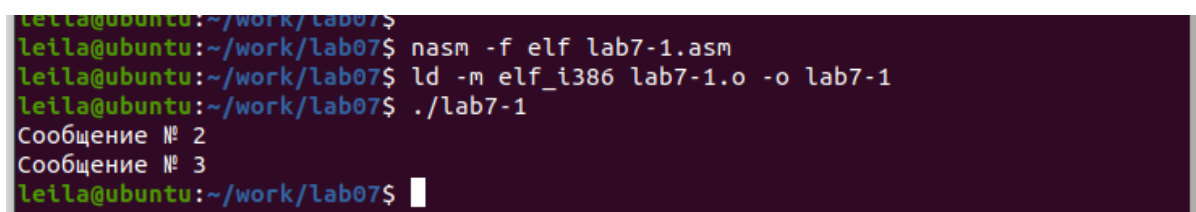
Давайте рассмотрим пример программы с использованием `jmp`. Написала текст программы из листинга 7.1 в файле “lab7-1.asm”.



```
1 %include 'in_out.asm'
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
9 _start:
10 jmp _label2
11
12 _label1:
13 mov eax, msg1
14 call sprintLF
15
16 _label2:
17 mov eax, msg2
18 call sprintLF
19
20 _label3:
21 mov eax, msg3
22 call sprintLF
23
24 _end:
25 call quit
```

Рис. 4.1: Редактирование файла lab7-1.asm

Затем создала исполняемый файл и запустила его.



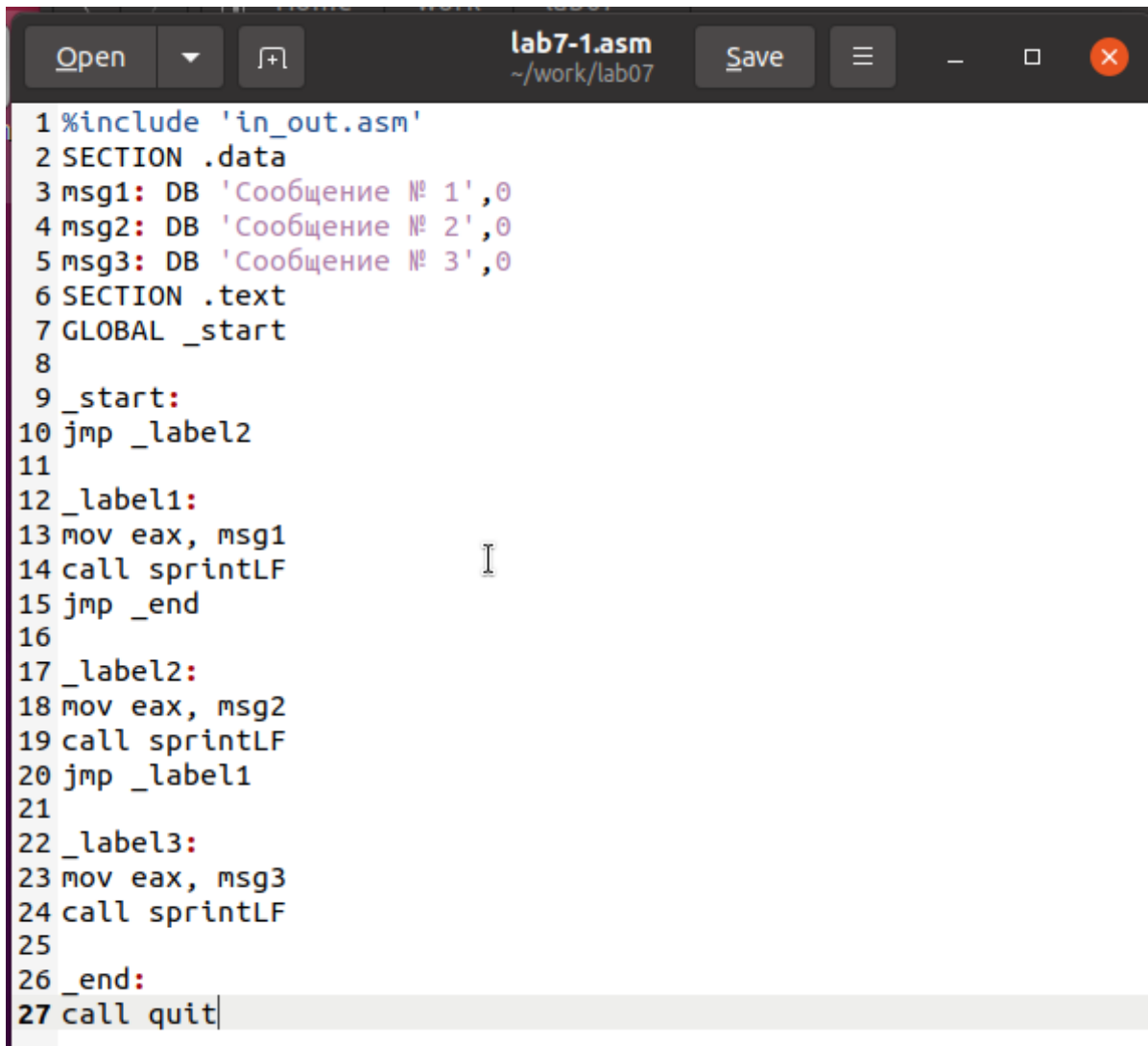
```
leila@ubuntu:~/work/lab07$
leila@ubuntu:~/work/lab07$ nasm -f elf lab7-1.asm
leila@ubuntu:~/work/lab07$ ld -m elf_i386 lab7-1.o -o lab7-1
leila@ubuntu:~/work/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
leila@ubuntu:~/work/lab07$
```

Рис. 4.2: Проверка кода lab7-1.asm

Инструкция `jmp` позволяет осуществлять переходы не только вперед, но и назад. Изменила программу так, чтобы сначала выводилось “Сообщение No2”,

потом “Сообщение No1”, а затем происходил выход. Для этого после вывода “Сообщения No2” добавила инструкцию `jmp` с меткой “_label1” (переход к выводу “Сообщения No1”). А после вывода “Сообщения No1” добавила инструкцию `jmp` с меткой “_end” (переход к инструкции `call quit`). Изменила текст программы в соответствии с листингом 7.2.

Изменила текст программы в соответствии с листингом 7.2.



```
1 %include 'in_out.asm'
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
9 _start:
10 jmp _label2
11
12 _label1:
13 mov eax, msg1
14 call sprintLF
15 jmp _end
16
17 _label2:
18 mov eax, msg2
19 call sprintLF
20 jmp _label1
21
22 _label3:
23 mov eax, msg3
24 call sprintLF
25
26 _end:
27 call quit
```

Рис. 4.3: Редактирование файла lab7-1.asm

```
leila@ubuntu:~/work/lab07$  
leila@ubuntu:~/work/lab07$ nasm -f elf lab7-1.asm  
leila@ubuntu:~/work/lab07$ ld -m elf_i386 lab7-1.o -o lab7-1  
leila@ubuntu:~/work/lab07$ ./lab7-1  
Сообщение № 2  
Сообщение № 1  
leila@ubuntu:~/work/lab07$
```

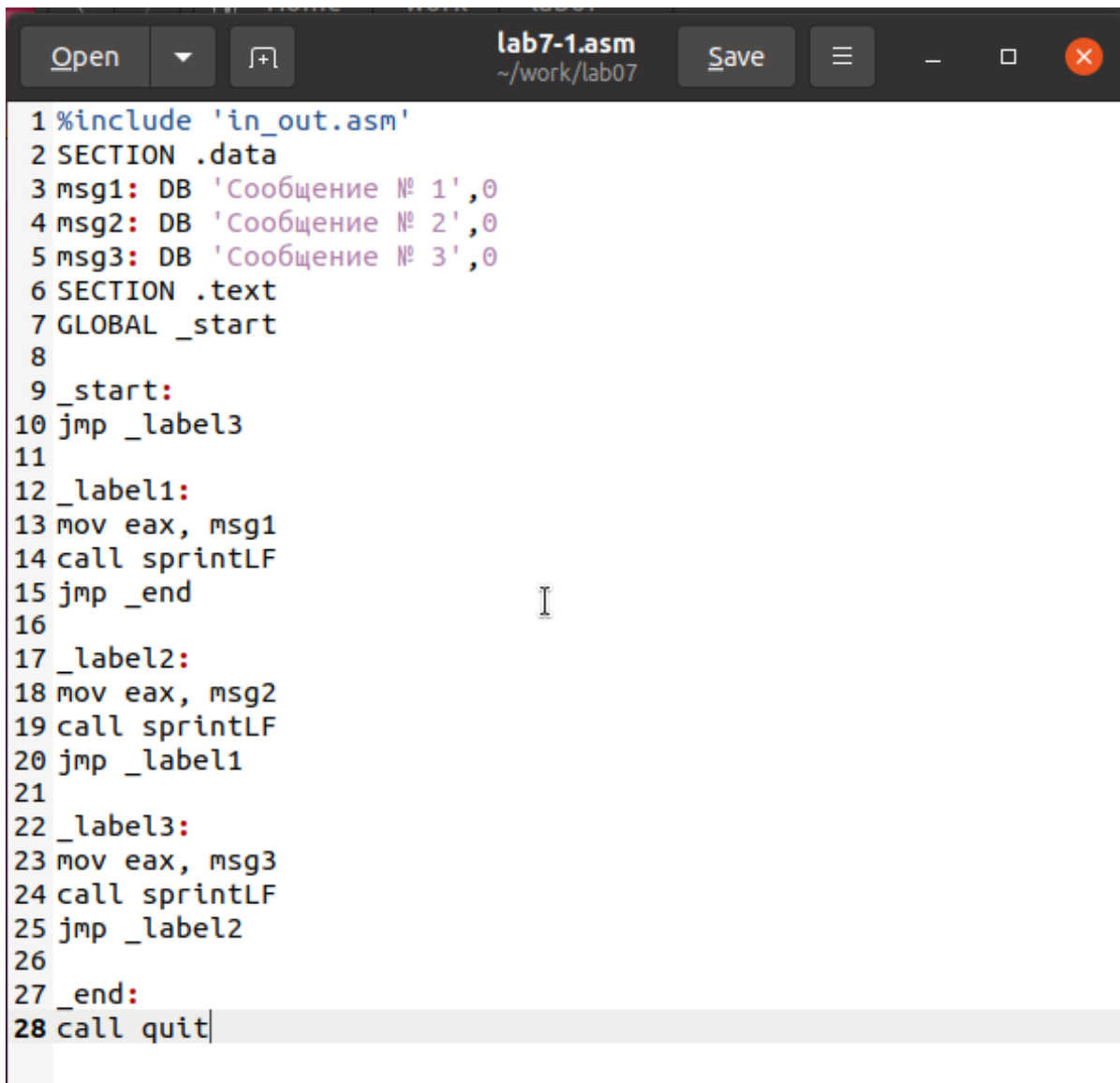
Рис. 4.4: Проверка кода lab7-1.asm

Изменила текст программы, изменив инструкции jmp, чтобы вывод программы был следующим:

Сообщение № 3

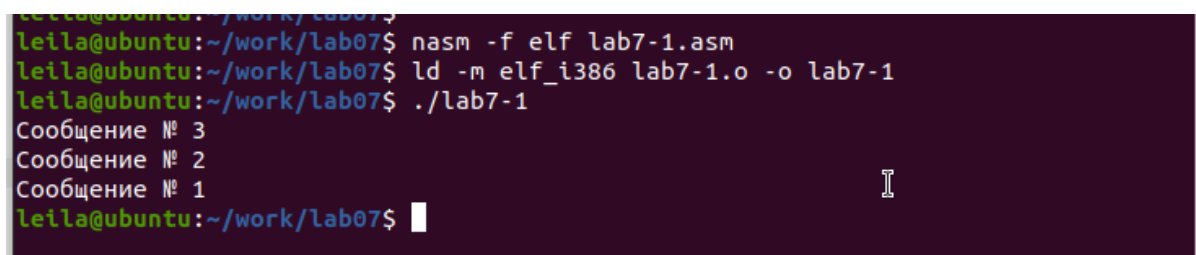
Сообщение № 2

Сообщение № 1



```
1 %include 'in_out.asm'
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
9 _start:
10 jmp _label3
11
12 _label1:
13 mov eax, msg1
14 call sprintLF
15 jmp _end
16
17 _label2:
18 mov eax, msg2
19 call sprintLF
20 jmp _label1
21
22 _label3:
23 mov eax, msg3
24 call sprintLF
25 jmp _label2
26
27 _end:
28 call quit
```

Рис. 4.5: Редактирование файла lab7-1.asm



```
leila@ubuntu:~/work/lab07$ nasm -f elf lab7-1.asm
leila@ubuntu:~/work/lab07$ ld -m elf_i386 lab7-1.o -o lab7-1
leila@ubuntu:~/work/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
leila@ubuntu:~/work/lab07$
```

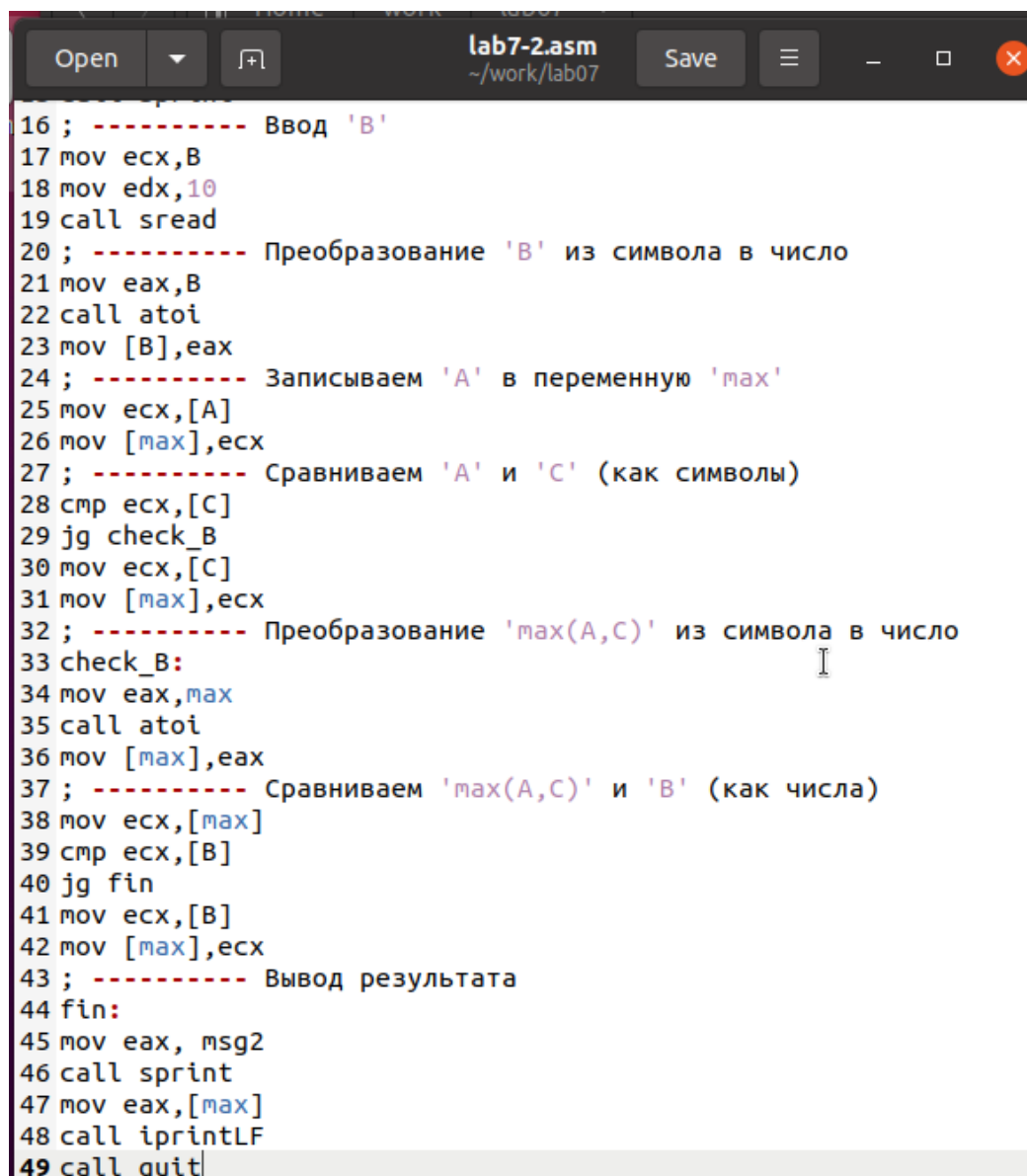
Рис. 4.6: Проверка кода lab7-1.asm

Использование инструкции `jmp` приводит к переходу в любом случае. Однако,

часто при написании программ необходимо использовать условные переходы, то есть переход должен происходить, если выполнено какое-либо условие.

Я рассмотрела программу, которая определяет и выводит наибольшее из трех чисел: A, B и C. Значения для A и C задаются в коде, а значение B вводится с клавиатуры.

Создала исполняемый файл и проверила его работу для разных значений B.



```
lab7-2.asm
~/work/lab07

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
23 mov [B],eax
24 ; ----- Записываем 'A' в переменную 'max'
25 mov ecx,[A]
26 mov [max],ecx
27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 cmp ecx,[C]
29 jg check_B
30 mov ecx,[C]
31 mov [max],ecx
32 ; ----- Преобразование 'max(A,C)' из символа в число
33 check_B:
34 mov eax,max
35 call atoi
36 mov [max],eax
37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
38 mov ecx,[max]
39 cmp ecx,[B]
40 jg fin
41 mov ecx,[B]
42 mov [max],ecx
43 ; ----- Вывод результата
44 fin:
45 mov eax, msg2
46 call sprint
47 mov eax,[max]
48 call iprintLF
49 call quit
```

Рис. 4.7: Редактирование файла lab7-2.asm

```
leila@ubuntu:~/work/lab07$ nasm -f elf lab7-2.asm
leila@ubuntu:~/work/lab07$ ld -m elf_i386 lab7-2.o -o lab7-2
leila@ubuntu:~/work/lab07$ ./lab7-2
Введите В: 40
Наибольшее число: 50
leila@ubuntu:~/work/lab07$ ./lab7-2
Введите В: 60
Наибольшее число: 60
leila@ubuntu:~/work/lab07$
```

Рис. 4.8: Проверка кода lab7-2.asm

4.2 Изучение структуры файлы листинга

Обычно nasm создаёт только объектный файл после ассемблирования. Чтобы получить файл листинга, нужно указать ключ -l и задать имя файла листинга в командной строке.

Я создала файл листинга для программы из lab7-2.asm.

```

lab7-2.lst
~/work/lab07

190 15 000000ED E81DFFFFFF call sprint
191 16 ; ----- Ввод 'B'
192 17 000000F2 B9[0A000000] mov ecx,B
193 18 000000F7 BA0A000000 mov edx,10
194 19 000000FC E842FFFFFF call sread
195 20 ; ----- Преобразование 'B' из символа в число
196 21 00000101 B8[0A000000] mov eax,B
197 22 00000106 E891FFFFFF call atoi
198 23 0000010B A3[0A000000] mov [B],eax
199 24 ; ----- Записываем 'A' в переменную 'max'
200 25 00000110 8B0D[35000000] mov ecx,[A]
201 26 00000116 890D[00000000] mov [max],ecx
202 27 ; ----- Сравниваем 'A' и 'C' (как символы)
203 28 0000011C 3B0D[39000000] cmp ecx,[C]
204 29 00000122 7F0C jg check_B
205 30 00000124 8B0D[39000000] mov ecx,[C]
206 31 0000012A 890D[00000000] mov [max],ecx
207 32 ; ----- Преобразование 'max(A,C)' из символа в
число
208 33 check_B:
209 34 00000130 B8[00000000] mov eax,max
210 35 00000135 E862FFFFFF call atoi
211 36 0000013A A3[00000000] mov [max],eax
212 37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
213 38 0000013F 8B0D[00000000] mov ecx,[max]
214 39 00000145 3B0D[0A000000] cmp ecx,[B]
215 40 0000014B 7F0C jg fin
216 41 0000014D 8B0D[0A000000] mov ecx,[B]
217 42 00000153 890D[00000000] mov [max],ecx
218 43 ; ----- Вывод результата
219 44 fin:
220 45 00000159 B8[13000000] mov eax,msg2
221 46 0000015E E8ACFEFFFF call sprint
222 47 00000163 A1[00000000] mov eax,[max]

```

Рис. 4.9: Файл листинга lab7-2

Внимательно ознакомилась с его форматом и содержимым. Подробно объясню содержимое трёх строк этого листинга.

строка 209

- 34 - номер строки в подпрограмме
- 00000130 - адрес
- B8[00000000] - машинный код
- mov eax,max - код программы - копирует max в eax

строка 210

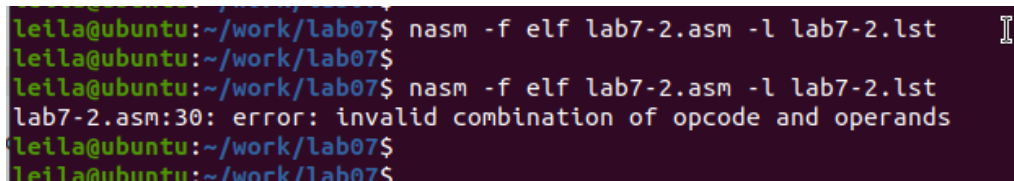
- 35 - номер строки в подпрограмме

- 00000135 - адрес
- E862FFFFFF - машинный код
- call atoi - код программы - вызов подпрограммы atoi

строка 211

- 36 - номер строки в подпрограмме
- 0000013A - адрес
- A3[00000000] - машинный код
- mov [max],eax - код программы - копирует eax в max

Открыла файл с программой lab7-2.asm и в инструкции с двумя операндами удалила один операнд. Выполнила трансляцию с получением файла листинга.



```
leila@ubuntu:~/work/lab07$ nasm -f elf lab7-2.asm -l lab7-2.lst
leila@ubuntu:~/work/lab07$
leila@ubuntu:~/work/lab07$ nasm -f elf lab7-2.asm -l lab7-2.lst
lab7-2.asm:30: error: invalid combination of opcode and operands
leila@ubuntu:~/work/lab07$
leila@ubuntu:~/work/lab07$
```

Рис. 4.10: Ошибка трансляции lab7-2


```

lab7-2.lst
~/work/lab07

lab7-2.asm
lab7-2.lst

195 20 ; ----- Преобразование 'B' из символа в число
196 21 00000101 B8[0A000000] mov eax,B
197 22 00000106 E891FFFFFF call atoi
198 23 0000010B A3[0A000000] mov [B],eax
199 24 ; ----- Записываем 'A' в переменную 'max'
200 25 00000110 8B0D[35000000] mov ecx,[A]
201 26 00000116 890D[00000000] mov [max],ecx
202 27 ; ----- Сравниваем 'A' и 'C' (как символы)
203 28 0000011C 3B0D[39000000] cmp ecx,[C]
204 29 00000122 7F06 jg check_B
205 30 mov ecx,
206 30 *****
      |
      |Error: invalid combination of opcode and operands
207 31 00000124 890D[00000000] mov [max],ecx
208 32 ; ----- Преобразование 'max(A,C)' из символа в
      |
      |число
209 33 check_B:
210 34 0000012A B8[00000000] mov eax,max
211 35 0000012F E868FFFFFF call atoi
212 36 00000134 A3[00000000] mov [max],eax
213 37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
214 38 00000139 8B0D[00000000] mov ecx,[max]
215 39 0000013F 3B0D[0A000000] cmp ecx,[B]
216 40 00000145 7F0C jg fin
217 41 00000147 8B0D[0A000000] mov ecx,[B]
218 42 0000014D 890D[00000000] mov [max],ecx
219 43 ; ----- Вывод результата
220 44 fin:
221 45 00000153 B8[13000000] mov eax, msg2
222 46 00000158 E8B2FEFFFF call sprint
223 47 0000015D A1[00000000] mov eax,[max]
224 48 00000162 E81FFFFFFF call iprintLF
225 49 00000167 E86FFFFFFF call quit

```

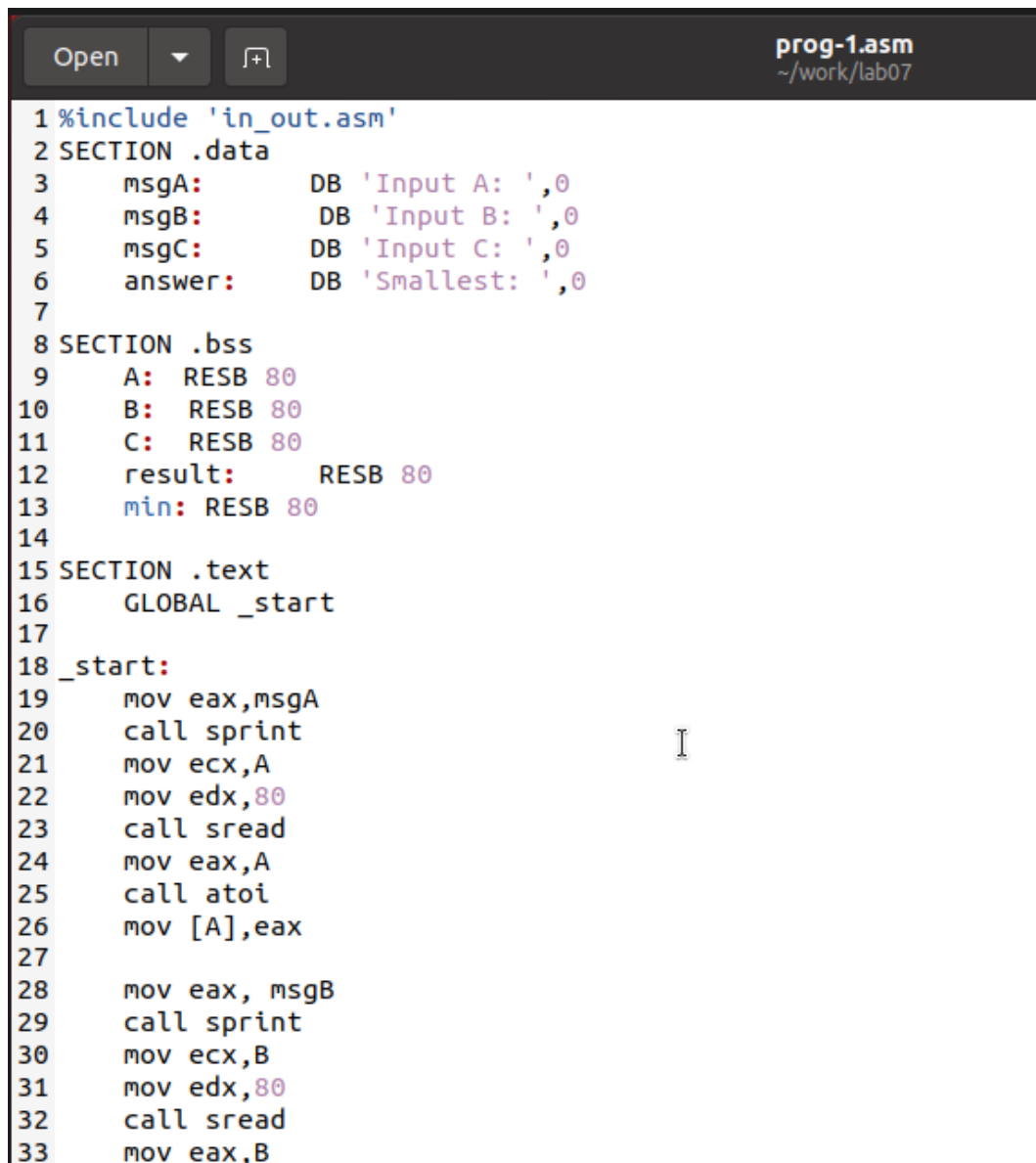
Рис. 4.11: Файл листинга с ошибкой lab7-2

Объектный файл не смог создаться из-за ошибки. Но получился листинг, где выделено место ошибки.

4.3 Выполнение заданий для самостоятельной работы

Напишите программу нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Создайте исполняемый файл и проверьте его работу.

Мой вариант 10 - числа: 41, 62, 35



```
1 %include 'in_out.asm'
2 SECTION .data
3     msgA:      DB 'Input A: ',0
4     msgB:      DB 'Input B: ',0
5     msgC:      DB 'Input C: ',0
6     answer:    DB 'Smallest: ',0
7
8 SECTION .bss
9     A:  RESB 80
10    B:  RESB 80
11    C:  RESB 80
12    result:  RESB 80
13    min: RESB 80
14
15 SECTION .text
16     GLOBAL _start
17
18 _start:
19     mov eax,msgA
20     call sprint
21     mov ecx,A
22     mov edx,80
23     call sread
24     mov eax,A
25     call atoi
26     mov [A],eax
27
28     mov eax, msgB
29     call sprint
30     mov ecx,B
31     mov edx,80
32     call sread
33     mov eax,B
```

Рис. 4.12: Редактирование файла prog-1.asm



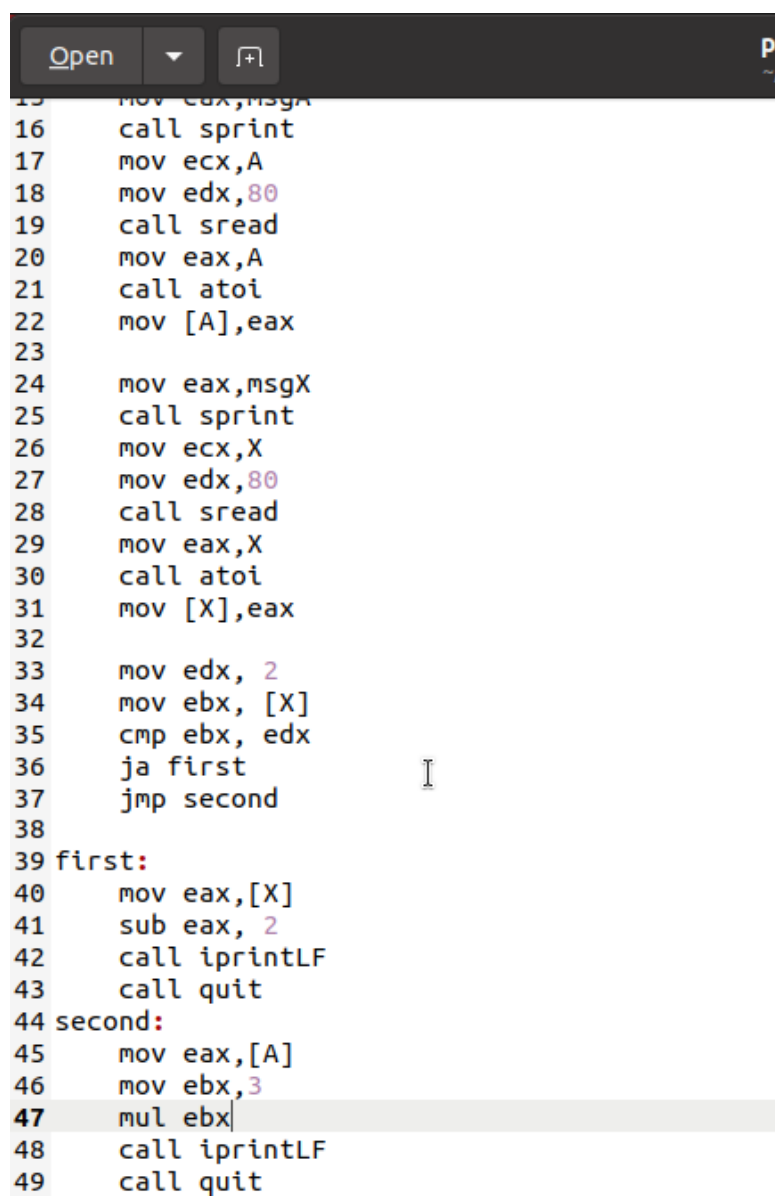
```
leila@ubuntu: ~/work/lab07
leila@ubuntu:~/work/lab07$ nasm -f elf prog-1.asm
leila@ubuntu:~/work/lab07$ ld -m elf_i386 prog-1.o -o prog-1
leila@ubuntu:~/work/lab07$ ./prog-1
Input A: 41
Input B: 62
Input C: 35
Smallest: 35
leila@ubuntu:~/work/lab07$
```

Рис. 4.13: Проверка кода prog-1.asm

Напишите программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений. Вид функции $f(x)$ выбрать из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу для значений X и a из 7.6. (рис. 4.14) (рис. 4.15)

Мой вариант 10

$$\begin{cases} x - 2, x > 2 \\ 3a, x \leq 2 \end{cases}$$



```
15  mov ecx,msgA
16  call sprint
17  mov ecx,A
18  mov edx,80
19  call sread
20  mov eax,A
21  call atoi
22  mov [A],eax
23
24  mov eax,msgX
25  call sprint
26  mov ecx,X
27  mov edx,80
28  call sread
29  mov eax,X
30  call atoi
31  mov [X],eax
32
33  mov edx, 2
34  mov ebx, [X]
35  cmp ebx, edx
36  ja first
37  jmp second
38
39 first:
40  mov eax,[X]
41  sub eax, 2
42  call iprintLF
43  call quit
44 second:
45  mov eax,[A]
46  mov ebx,3
47  mul ebx
48  call iprintLF
49  call quit
```

Рис. 4.14: Редактирование файла prog-2.asm

```
leila@ubuntu:~/work/lab07$  
leila@ubuntu:~/work/lab07$ nasm -f elf prog-2.asm  
leila@ubuntu:~/work/lab07$ ld -m elf_i386 prog-2.o -o prog-2  
leila@ubuntu:~/work/lab07$ ./prog-2  
Input A: 0  
Input X: 3  
1  
leila@ubuntu:~/work/lab07$ ./prog-2  
Input A: 2  
Input X: 1  
6  
leila@ubuntu:~/work/lab07$
```

Рис. 4.15: Проверка кода prog-2.asm

5 Выводы

Изучили команды условного и безусловного переходов, познакомились с фалом листинга.