

Отчёт по лабораторной работе 7

Архитектура компьютеров и операционные системы

Горелашвили Лия Михайловна НКАбд-02-23

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	15
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	11
4.4	Проверка кода lab7-1.asm	11
4.5	Редактирование файла lab7-1.asm	12
4.6	Проверка кода lab7-1.asm	13
4.7	Редактирование файла lab7-2.asm	14
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`

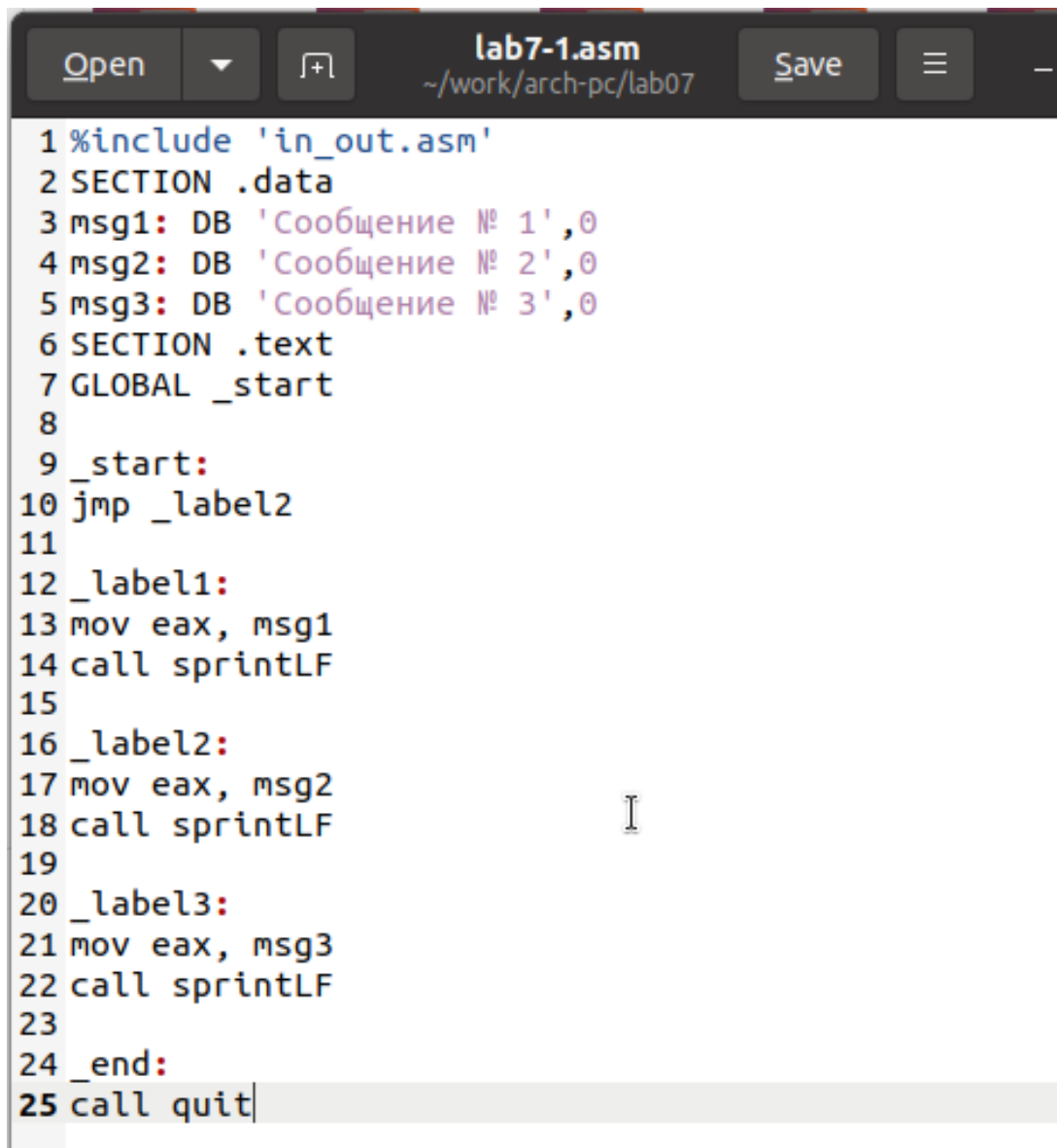
Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов.

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.

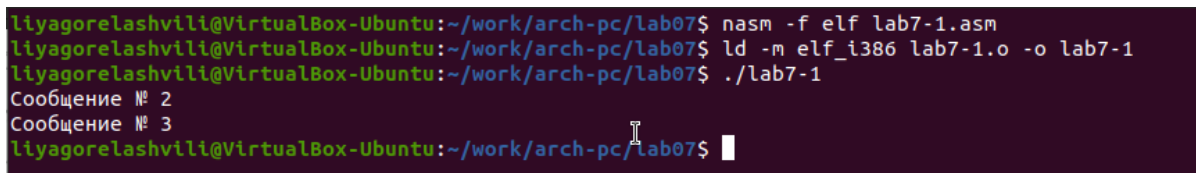
Давайте рассмотрим пример программы с использованием `jmp`. Написала текст программы из листинга 7.1 в файле “lab7-1.asm”.



```
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.

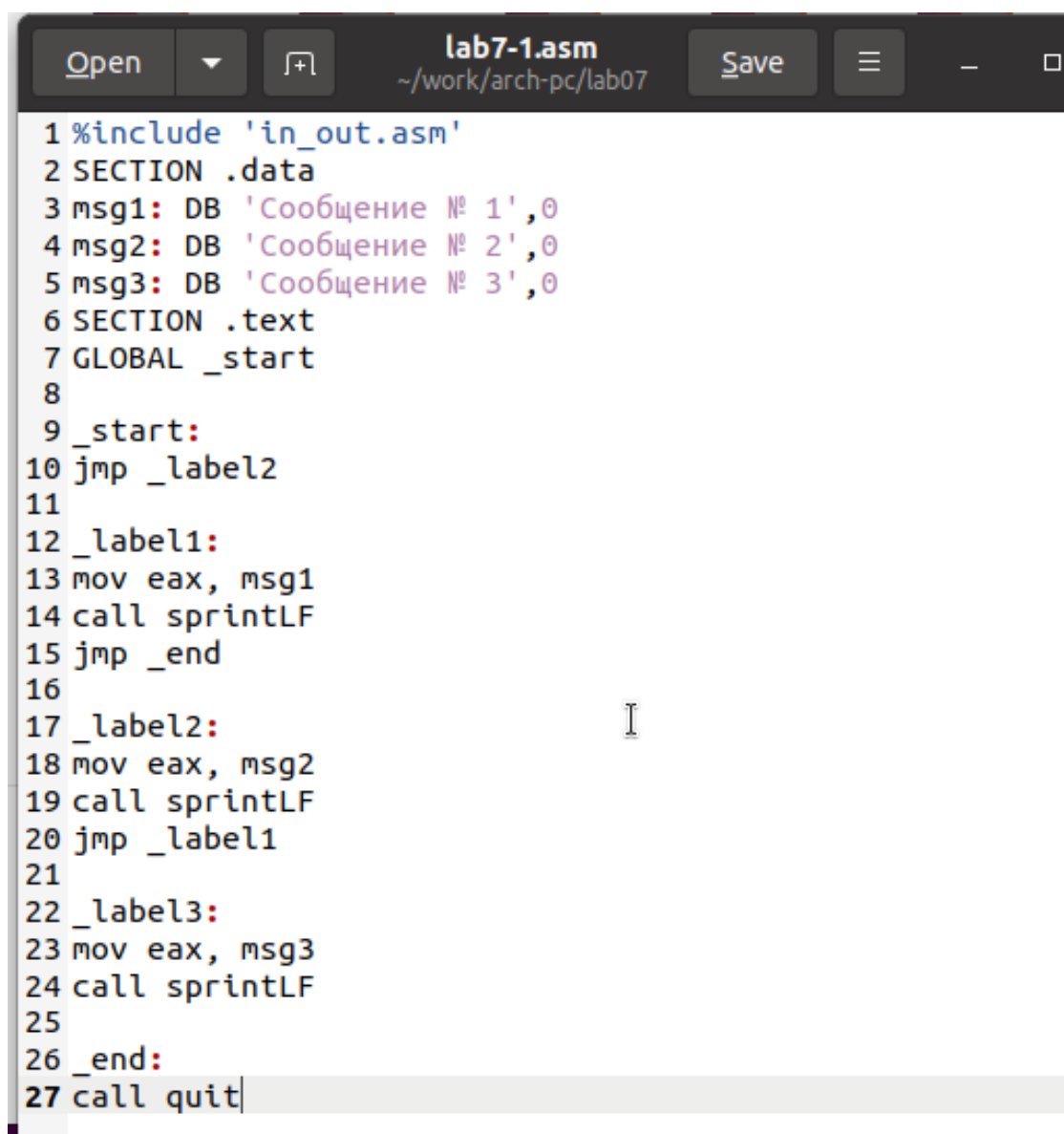


```
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-1.o -o lab7-1
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$
```

Рис. 4.2: Проверка кода lab7-1.asm

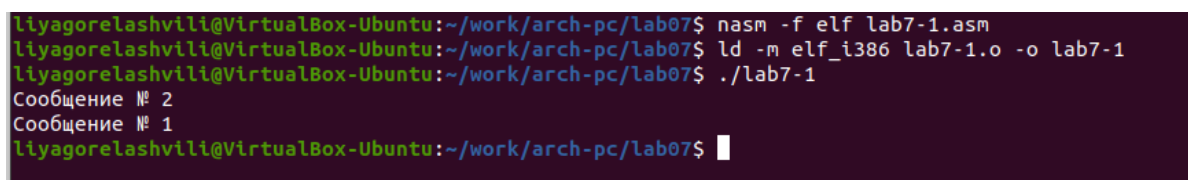
Инструкция `jmp` позволяет осуществлять переходы не только вперед, но и назад. Изменила программу так, чтобы сначала выводилось “Сообщение No2”, потом “Сообщение No1”, а затем происходил выход. Для этого после вывода “Сообщения No2” добавила инструкцию `jmp` с меткой `_label1` (переход к выводу “Сообщения 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



```
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-1.o -o lab7-1
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$
```

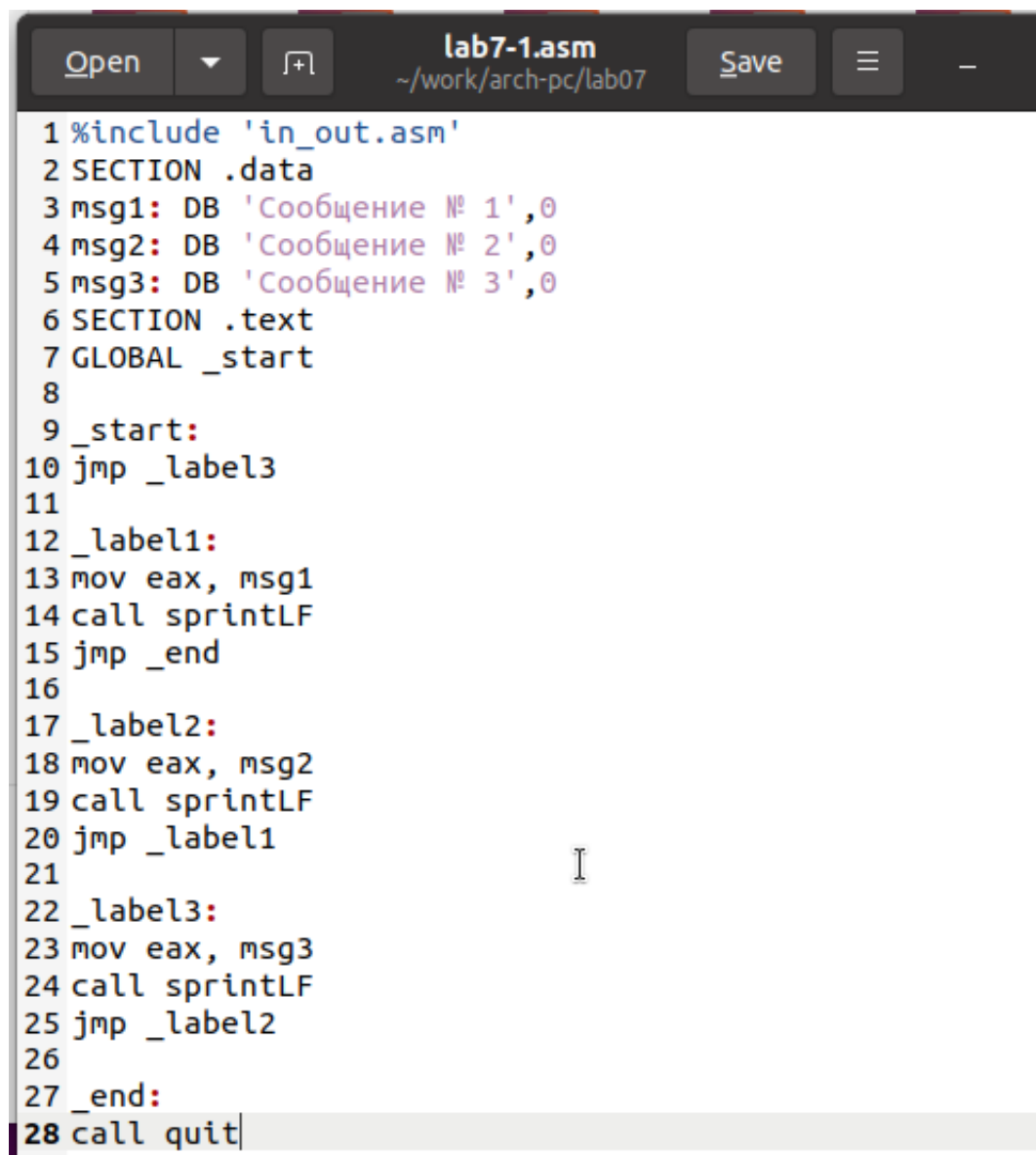
Рис. 4.4: Проверка кода lab7-1.asm

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 sprintfLF
15 jmp _end
16
17 _label2:
18 mov eax, msg2
19 call sprintfLF
20 jmp _label1
21
22 _label3:
23 mov eax, msg3
24 call sprintfLF
25 jmp _label2
26
27 _end:
28 call quit
```

Рис. 4.5: Редактирование файла lab7-1.asm

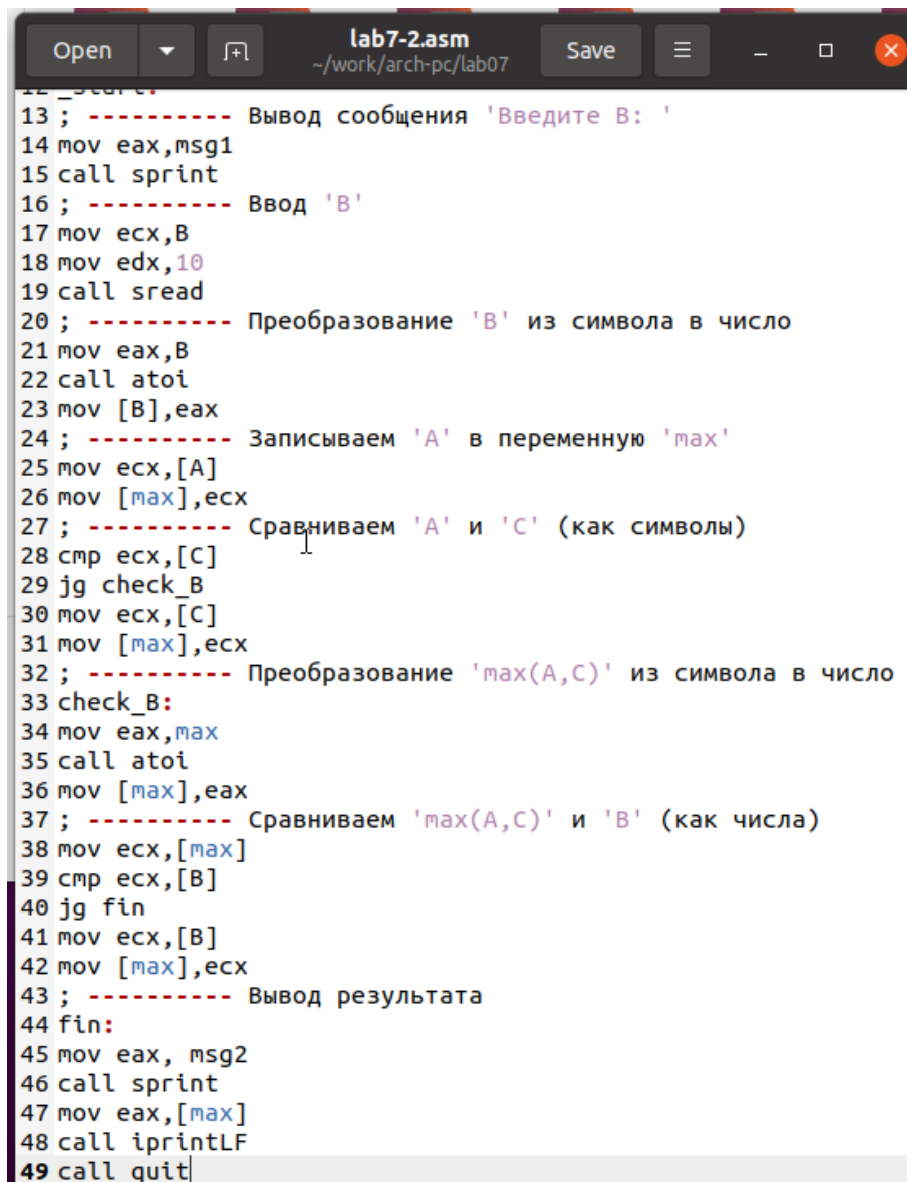
```
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-1.o -o lab7-1
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ █
```

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: А, В и С. Значения для А и С задаются в коде, а значение В вводится с клавиатуры.

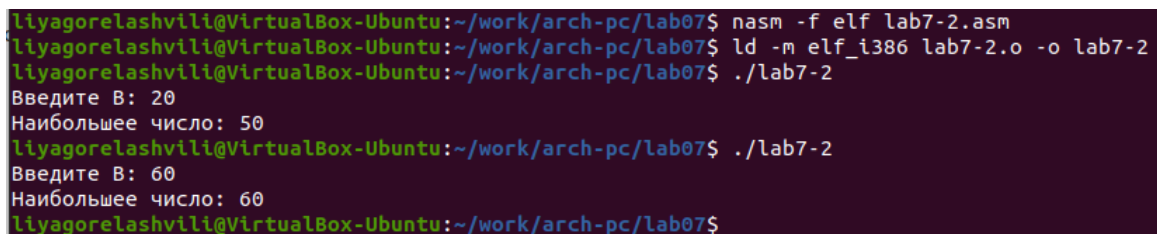
Создала исполняемый файл и проверила его работу для разных значений В.



```
lab7-2.asm
~/work/arch-pc/lab07
Save

12 ; ----- Вывод сообщения 'Введите В: '
13 ; ----- Вывод сообщения 'Введите В: '
14 mov eax,msg1
15 call sprint
16 ; ----- Ввод 'В'
17 mov ecx,B
18 mov edx,10
19 call sread
20 ; ----- Преобразование 'В' 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 'В' (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



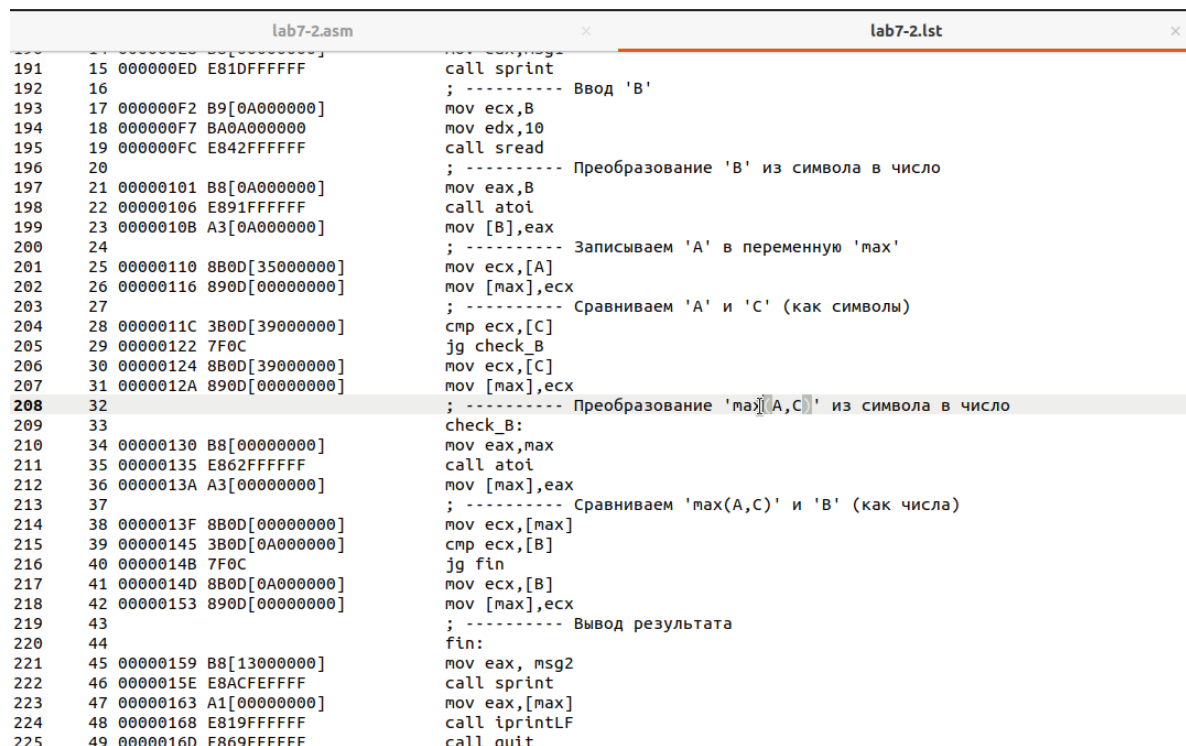
```
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-2.o -o lab7-2
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 20
Наибольшее число: 50
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 60
Наибольшее число: 60
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/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.asm x lab7-2.lst x
191 15 000000ED E81DFFFFFF call sprint
192 16 ; ----- Ввод 'B'
193 17 000000F2 B9[0A000000] mov ecx,B
194 18 000000F7 BA0A000000 mov edx,10
195 19 000000FC E842FFFFFF call sread
196 20 ; ----- Преобразование 'B' из символа в число
197 21 00000101 B8[0A000000] mov eax,B
198 22 00000106 E891FFFFFF call atoi
199 23 0000010B A3[0A000000] mov [B],eax
200 24 ; ----- Записываем 'A' в переменную 'max'
201 25 00000110 8B0D[35000000] mov ecx,[A]
202 26 00000116 890D[00000000] mov [max],ecx
203 27 ; ----- Сравниваем 'A' и 'C' (как символы)
204 28 0000011C 3B0D[39000000] cmp ecx,[C]
205 29 00000122 7F0C jg check_B
206 30 00000124 8B0D[39000000] mov ecx,[C]
207 31 0000012A 890D[00000000] mov [max],ecx
208 32 ; ----- Преобразование 'max[A,C]' из символа в число
209 33 check_B:
210 34 00000130 B8[00000000] mov eax,max
211 35 00000135 E862FFFFFF call atoi
212 36 0000013A A3[00000000] mov [max],eax
213 37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
214 38 0000013F 8B0D[00000000] mov ecx,[max]
215 39 00000145 3B0D[0A000000] cmp ecx,[B]
216 40 0000014B 7F0C jg fin
217 41 0000014D 8B0D[0A000000] mov ecx,[B]
218 42 00000153 890D[00000000] mov [max],ecx
219 43 ; ----- Вывод результата
220 44 fin:
221 45 00000159 B8[13000000] mov eax,msg2
222 46 0000015E E8ACFEFFFF call sprint
223 47 00000163 A1[00000000] mov eax,[max]
224 48 00000168 E819FFFFFF call iprintlnLF
225 49 0000016D E869FFFFFF call quit
```

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.

```
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm -l lab7-2.lst
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm -l lab7-2.lst
lab7-2.asm:28: error: invalid combination of opcode and operands
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$
```

Рис. 4.10: Ошибка трансляции `lab7-2`


```

lab7-2.asm
lab7-2.lst
192 16 ; ----- Ввод 'B'
193 17 000000F2 B9[0A000000] mov ecx,B
194 18 000000F7 BA0A000000 mov edx,10
195 19 000000FC E842FFFFFF call sread
196 20 ; ----- Преобразование 'B' из символа в число
197 21 00000101 B8[0A000000] mov eax,B
198 22 00000106 E891FFFFFF call atoi
199 23 0000010B A3[0A000000] mov [B],eax
200 24 ; ----- Записываем 'A' в переменную 'max'
201 25 00000110 8B0D[35000000] mov ecx,[A]
202 26 00000116 890D[00000000] mov [max],ecx
203 27 ; ----- Сравниваем 'A' и 'C' (как символы)
204 28 cmp ecx,
205 28 *****
206 29 0000011C 7F0C jg check_B
207 30 0000011E 8B0D[39000000] mov ecx,[C]
208 31 00000124 890D[00000000] mov [max],ecx
209 32 ; ----- Преобразование 'max(A,C)' из символа в число
210 33 check_B:
211 34 0000012A B8[00000000] mov eax,max
212 35 0000012F E868FFFFFF call atoi
213 36 00000134 A3[00000000] mov [max],eax
214 37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
215 38 00000139 8B0D[00000000] mov ecx,[max]
216 39 0000013F 3B0D[0A000000] cmp ecx,[B]
217 40 00000145 7F0C jg fin
218 41 00000147 8B0D[0A000000] mov ecx,[B]
219 42 0000014D 890D[00000000] mov [max],ecx
220 43 ; ----- Вывод результата
221 44 fin:
222 45 00000153 B8[13000000] mov eax, msg2
223 46 00000158 E8B2FFFFFF call sprint
224 47 0000015D A1[00000000] mov eax,[max]
225 48 00000162 E81FFFFFFF call iprintf
226 49 00000167 E86FFFFFFF call quit

```

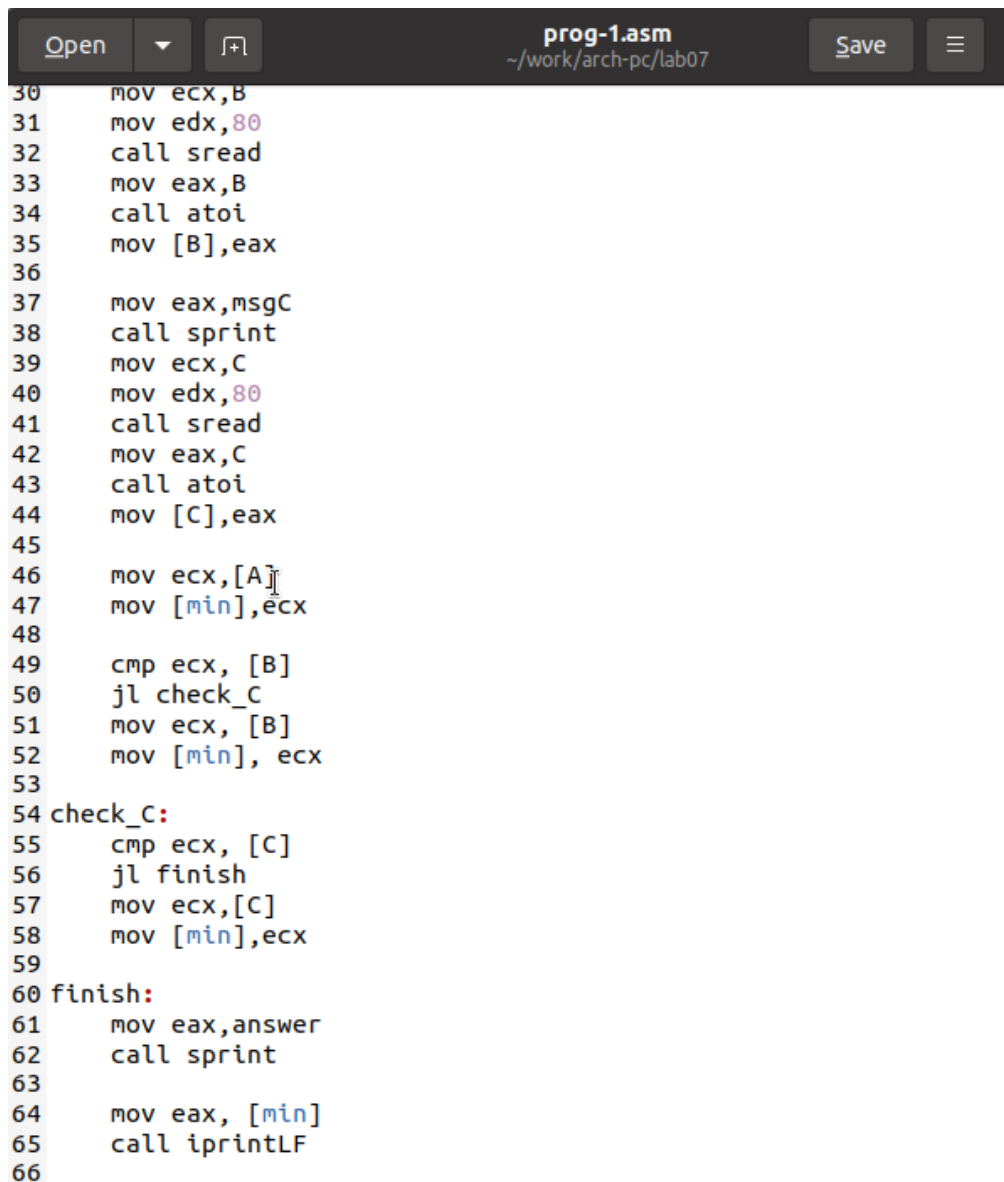
Рис. 4.11: Файл листинга с ошибкой lab7-2

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

4.3 Выполнение заданий для самостоятельной работы

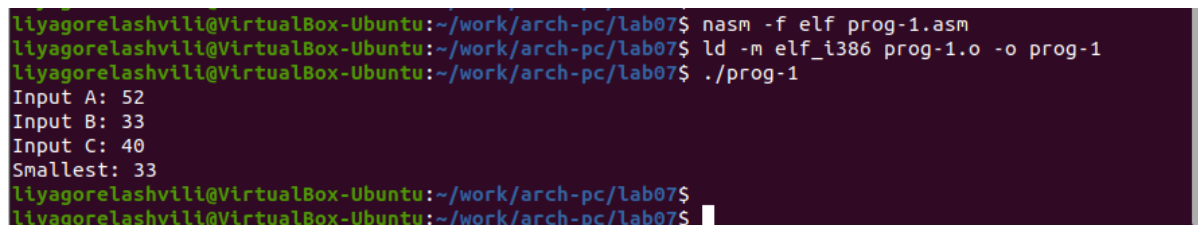
Напишите программу нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Создайте исполняемый файл и проверьте его работу.

Мой вариант 8 - числа: 52,33,40



```
30    mov ecx,B
31    mov edx,80
32    call sread
33    mov eax,B
34    call atoi
35    mov [B],eax
36
37    mov eax,msgC
38    call sprint
39    mov ecx,C
40    mov edx,80
41    call sread
42    mov eax,C
43    call atoi
44    mov [C],eax
45
46    mov ecx,[A]
47    mov [min],ecx
48
49    cmp ecx, [B]
50    jl check_C
51    mov ecx, [B]
52    mov [min], ecx
53
54 check_C:
55    cmp ecx, [C]
56    jl finish
57    mov ecx,[C]
58    mov [min],ecx
59
60 finish:
61    mov eax,answer
62    call sprint
63
64    mov eax, [min]
65    call iprintLF
66
```

Рис. 4.12: Редактирование файла prog-1.asm



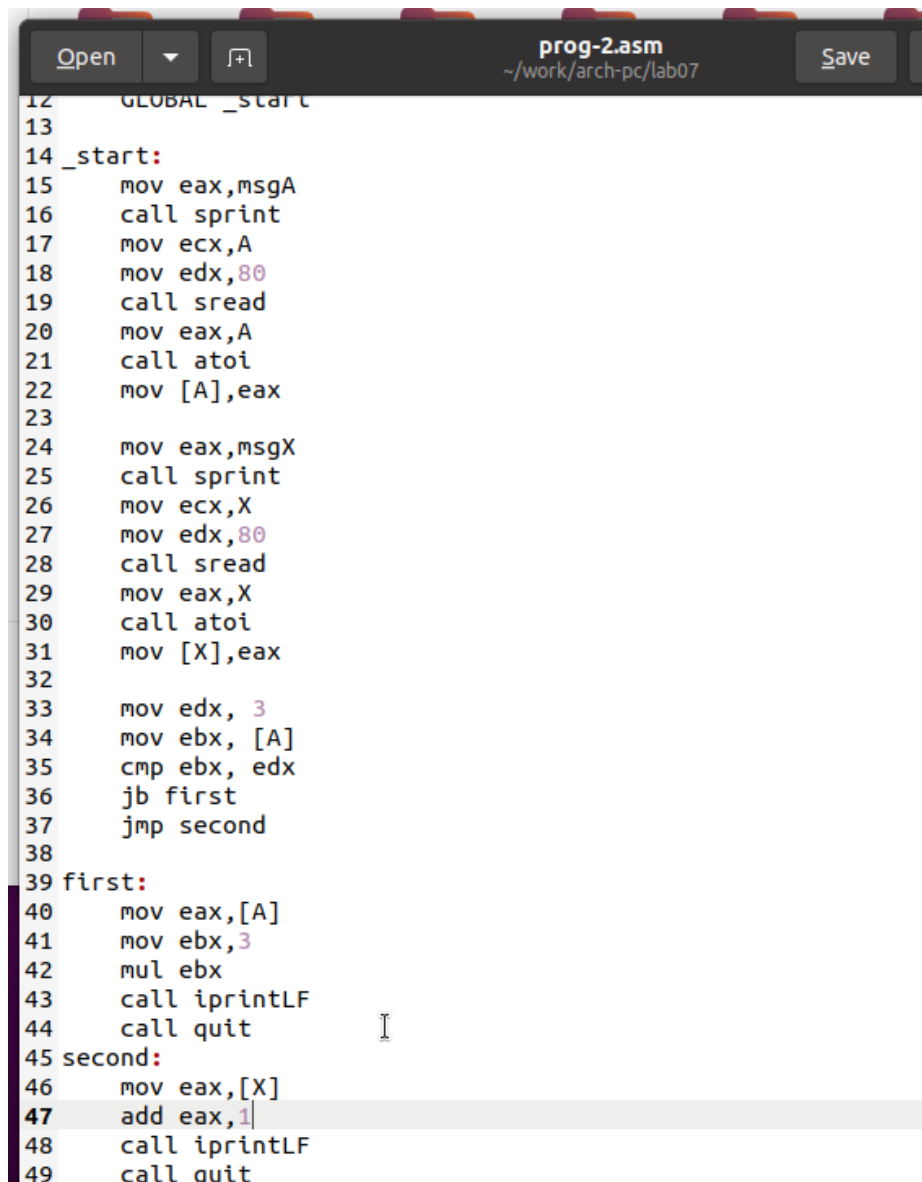
```
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ nasm -f elf prog-1.asm
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 prog-1.o -o prog-1
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ./prog-1
Input A: 52
Input B: 33
Input C: 40
Smallest: 33
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/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 8

$$\begin{cases} 3a, a < 3 \\ x + 1, a \geq 3 \end{cases}$$



```
12 GLOBAL _start
13
14 _start:
15     mov eax,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, 3
34     mov ebx, [A]
35     cmp ebx, edx
36     jb first
37     jmp second
38
39 first:
40     mov eax,[A]
41     mov ebx,3
42     mul ebx
43     call iprintLF
44     call quit
45 second:
46     mov eax,[X]
47     add eax,1
48     call iprintLF
49     call quit
```

Рис. 4.14: Редактирование файла prog-2.asm

```
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ nasm -f elf prog-2.asm
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ld -m elf_i386 prog-2.o -o prog-2
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ./prog-2
Input A: 4
Input X: 1
2
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$ ./prog-2
Input A: 2
Input X: 1
6
liyagorelashvili@VirtualBox-Ubuntu:~/work/arch-pc/lab07$
```

Рис. 4.15: Проверка кода prog-2.asm

5 Выводы

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