

Отчёт по лабораторной работе 7

Дисциплина: архитектура компьютера

Давлетова Мадина

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	16
4.3	Выполнение заданий для самостоятельной работы	19
5	Выводы	24

Список иллюстраций

4.1	Изменение кода lab7-1.asm	9
4.2	Компиляция текста программы lab7-1.asm	10
4.3	Изменение кода lab7-1.asm	11
4.4	Компиляция текста программы lab7-1.asm	12
4.5	Изменение кода lab7-1.asm	13
4.6	Компиляция текста программы lab7-1.asm	14
4.7	Изменение кода lab7-2.asm	15
4.8	Компиляция текста программы lab7-2.asm	16
4.9	Файл листинга lab7-2	17
4.10	Ошибка трансляции lab7-2	18
4.11	Файл листинга с ошибкой lab7-2	19
4.12	Изменение кода prog-1.asm	20
4.13	Компиляция текста программы prog-1.asm	21
4.14	Изменение кода prog-2.asm	22
4.15	Компиляция текста программы prog-2.asm	23

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`

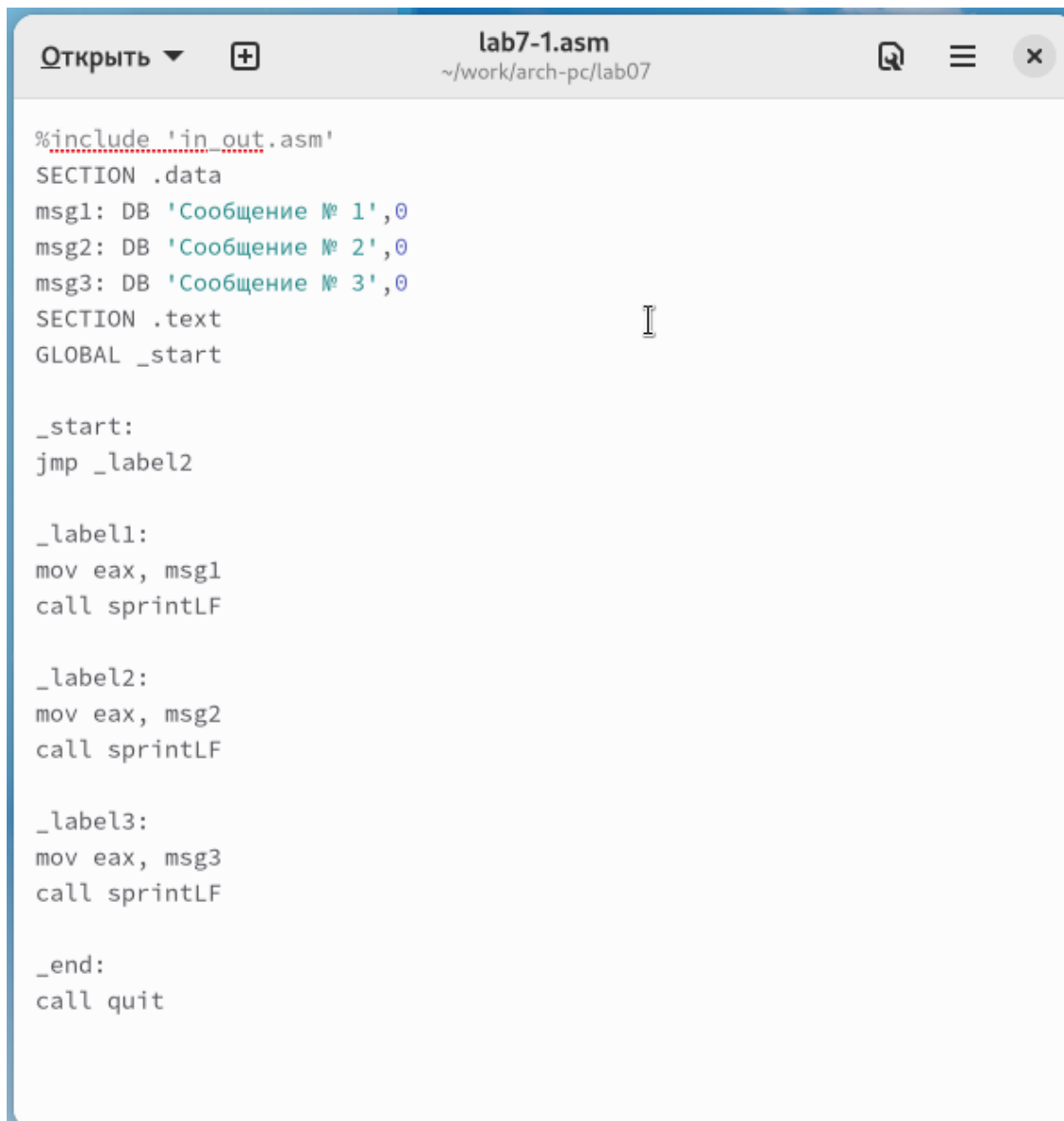
Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов.

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

Я создала каталог для программ лабораторной работы No7 и файл lab7-1.asm.

Инструкция `jmp` в NASM используется для безусловных переходов. Давайте рассмотрим пример программы с использованием `jmp`. Я написала текст программы из листинга 7.1 в файле lab7-1.asm (рис. [4.1]).



```
Открыть ▾ + lab7-1.asm ~/work/arch-pc/lab07
%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
call sprintLF

_label2:
mov eax, msg2
call sprintLF

_label3:
mov eax, msg3
call sprintLF

_end:
call quit
```

Рис. 4.1: Изменение кода lab7-1.asm

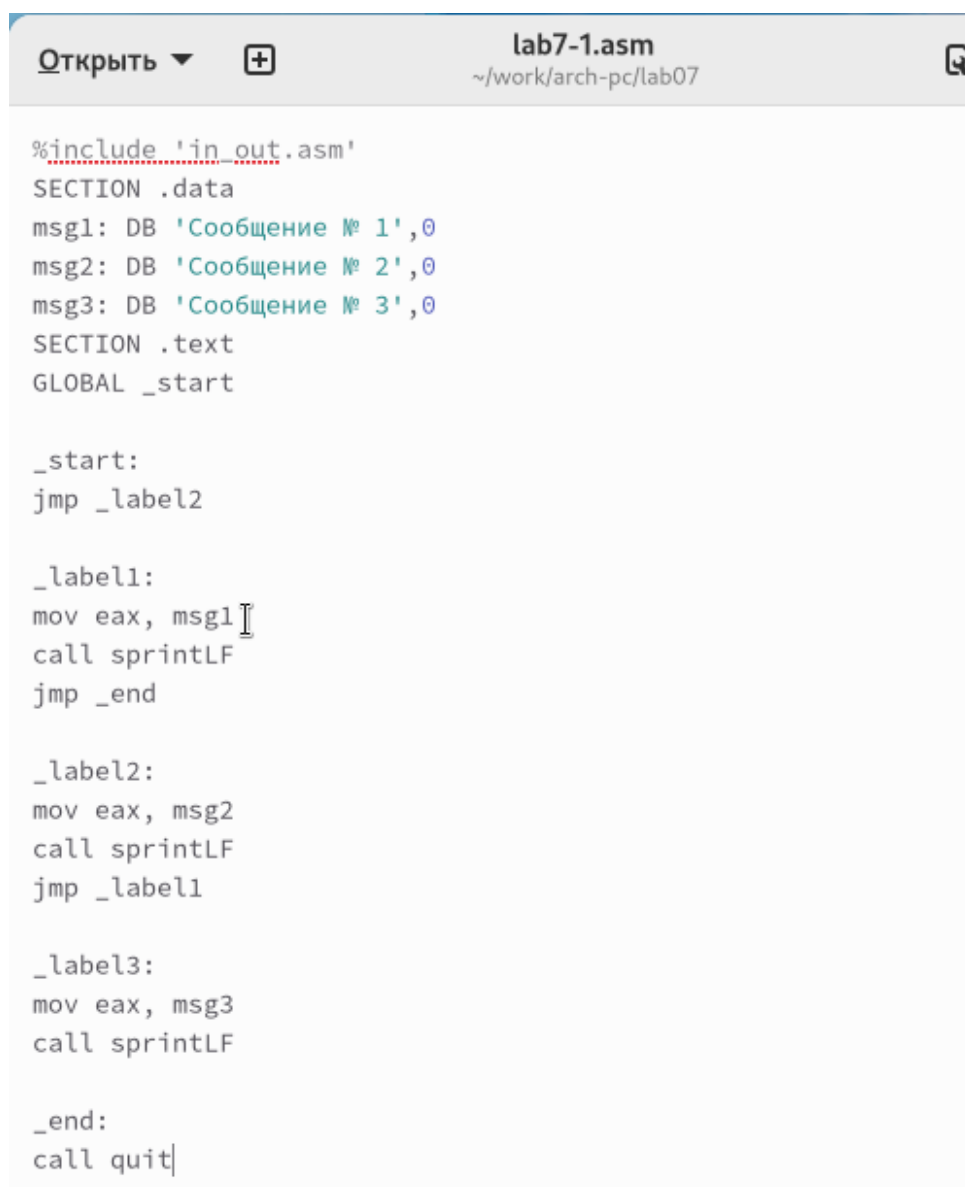
Затем создала исполняемый файл и запустила его (рис. [4.2]).

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

Рис. 4.2: Компиляция текста программы lab7-1.asm

Инструкция `jmp` позволяет осуществлять переходы не только вперед, но и назад. Я 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 `call quit`).

Изменила текст программы в соответствии с листингом 7.2. (рис. [4.3] [4.4])



```
%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
call sprintLF
jmp _end

_label2:
mov eax, msg2
call sprintLF
jmp _label1

_label3:
mov eax, msg3
call sprintLF

_end:
call quit
```

Рис. 4.3: Изменение кода lab7-1.asm

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


Рис. 4.4: Компиляция текста программы lab7-1.asm

Изменила текст программы (рис. [4.5] [4.6]), изменив инструкции `jmp`, чтобы вывод программы был следующим:

Сообщение № 3

Сообщение № 2

Сообщение № 1

Открыть ▾ 

lab7-1.asm
~/work/arch-pc/lab07

```
%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
call sprintf
jmp _end

_label2:
mov eax, msg2
call sprintf
jmp _label1

_label3:
mov eax, msg3
call sprintf
jmp _label2

_end:
call quit
```

Рис. 4.5: Изменение кода lab7-1.asm

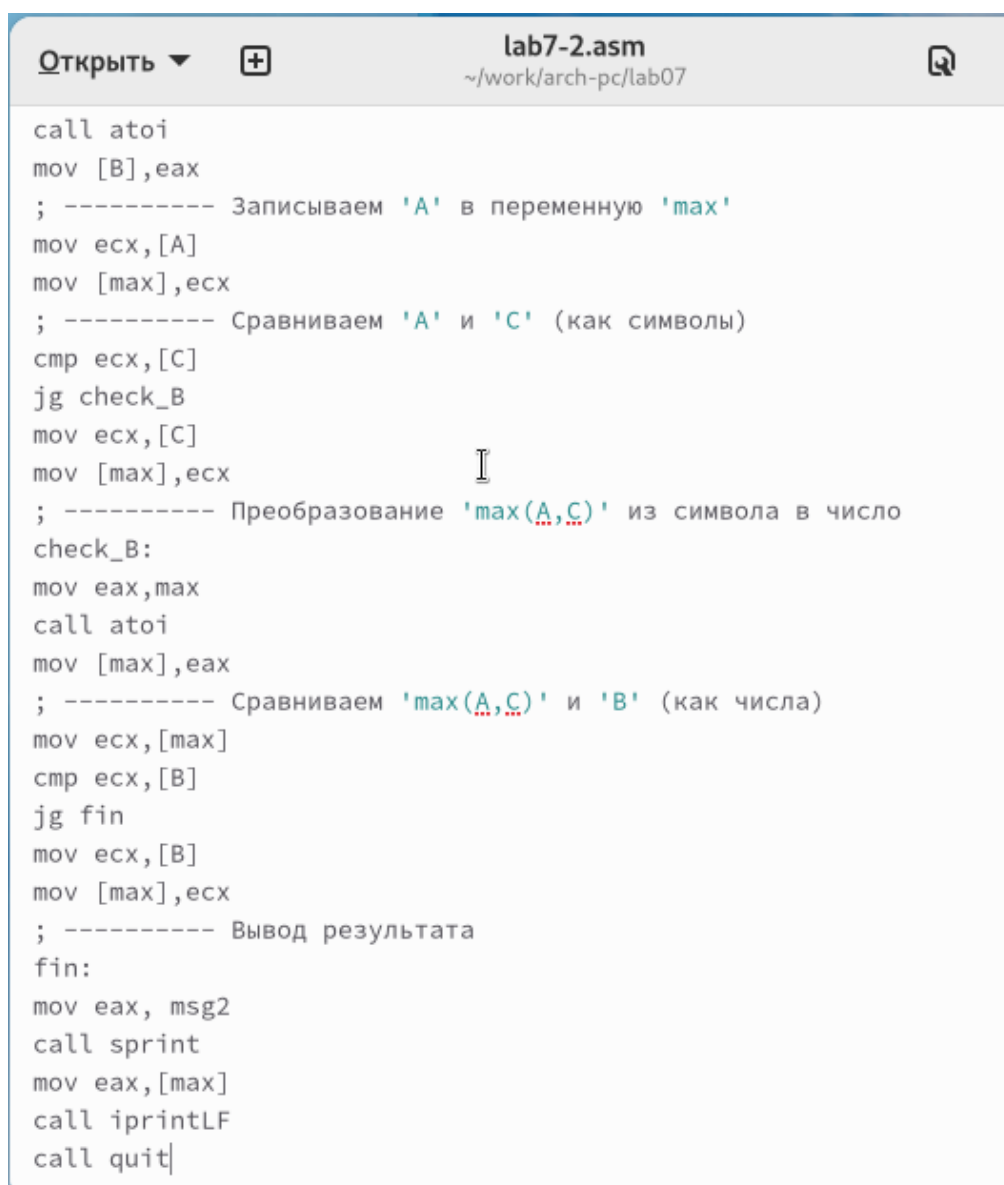
```
[mdavletova@fedora lab07]$  
[mdavletova@fedora lab07]$ nasm -f elf lab7-1.asm  
[mdavletova@fedora lab07]$ ld -m elf_i386 lab7-1.o -o lab7-1  
[mdavletova@fedora lab07]$ ./lab7-1  
Сообщение № 3  
Сообщение № 2  
Сообщение № 1  
[mdavletova@fedora lab07]$  
[mdavletova@fedora lab07]$
```

Рис. 4.6: Компиляция текста программы 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. (рис. [4.7])

Создала исполняемый файл и проверила его работу для разных значений В. (рис. [4.8])



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

Открыть ▾ +

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
mov 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
mov ecx,[B]
mov [max],ecx
; ----- Вывод результата
fin:
mov eax, msg2
call sprint
mov eax,[max]
call iprintLF
call quit
```

Рис. 4.7: Изменение кода lab7-2.asm

```
[mdavletova@fedora lab07]$ nasm -f elf lab7-2.asm
[mdavletova@fedora lab07]$ ld -m elf_i386 lab7-2.o -o lab7-2
[mdavletova@fedora lab07]$ ./lab7-2
Введите В: 40
Наибольшее число: 50
[mdavletova@fedora lab07]$ ./lab7-2
Введите В: 70
Наибольшее число: 70
[mdavletova@fedora lab07]$ ./lab7-2
Введите В: 100
Наибольшее число: 100
[mdavletova@fedora lab07]$
```

Рис. 4.8: Компиляция текста программы 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` (рис. [4.9]).


```

190 15 000000ED E81DFFFFFF call sprint
191 16 ; ----- Ввод 'B'
192 17 000000F2 B9[0A000000] mov ecx,B
193 18 000000F7 8A0A000000 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

```

Рис. 4.9: Файл листинга lab7-2

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

строка 203

- 28 - номер строки в подпрограмме
- 0000011C - адрес
- 3B0D[39000000] - машинный код
- str ecx,[C] - код программы - сравнивает ecx и C

строка 204

- 29 - номер строки в подпрограмме

- 00000122 - адрес
- 7F0C - машинный код
- `jb check_B` - код программы - если больше переходит к метке `check_B`

строка 205

- 30 - номер строки в подпрограмме
- 00000124 - адрес
- 8B0D[39000000] - машинный код
- `mov ecx,[C]` - код программы - копирует C в ecx

Открыла файл с программой `lab7-2.asm` и в инструкции с двумя операндами удалила один операнд. Выполнила трансляцию с получением файла листинга. (рис. [4.10]) (рис. [4.11])

```
[mdavletova@fedora lab07]$  
[mdavletova@fedora lab07]$ nasm -f elf lab7-2.asm -l lab7-2.lst  
[mdavletova@fedora lab07]$  
[mdavletova@fedora lab07]$ nasm -f elf lab7-2.asm -l lab7-2.lst  
lab7-2.asm:28: error: invalid combination of opcode and operands  
[mdavletova@fedora lab07]$  
[mdavletova@fedora lab07]$
```

Рис. 4.10: Ошибка трансляции `lab7-2`

```
lab7-2.asm
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 cmp ecx,
204 28 ***** error: invalid combination of opcode and operands
205 29 0000011C 7F0C ig check_B
206 30 0000011E 8B0D[39000000] mov ecx,[C]
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 ig 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 E8B2FFFFFF call sprint
223 47 0000015D A1[00000000] mov eax,[max]
224 48 00000162 E81FFFFFFF call iprintLF
225 49 00000167 E86FFFFFFF call quit
```

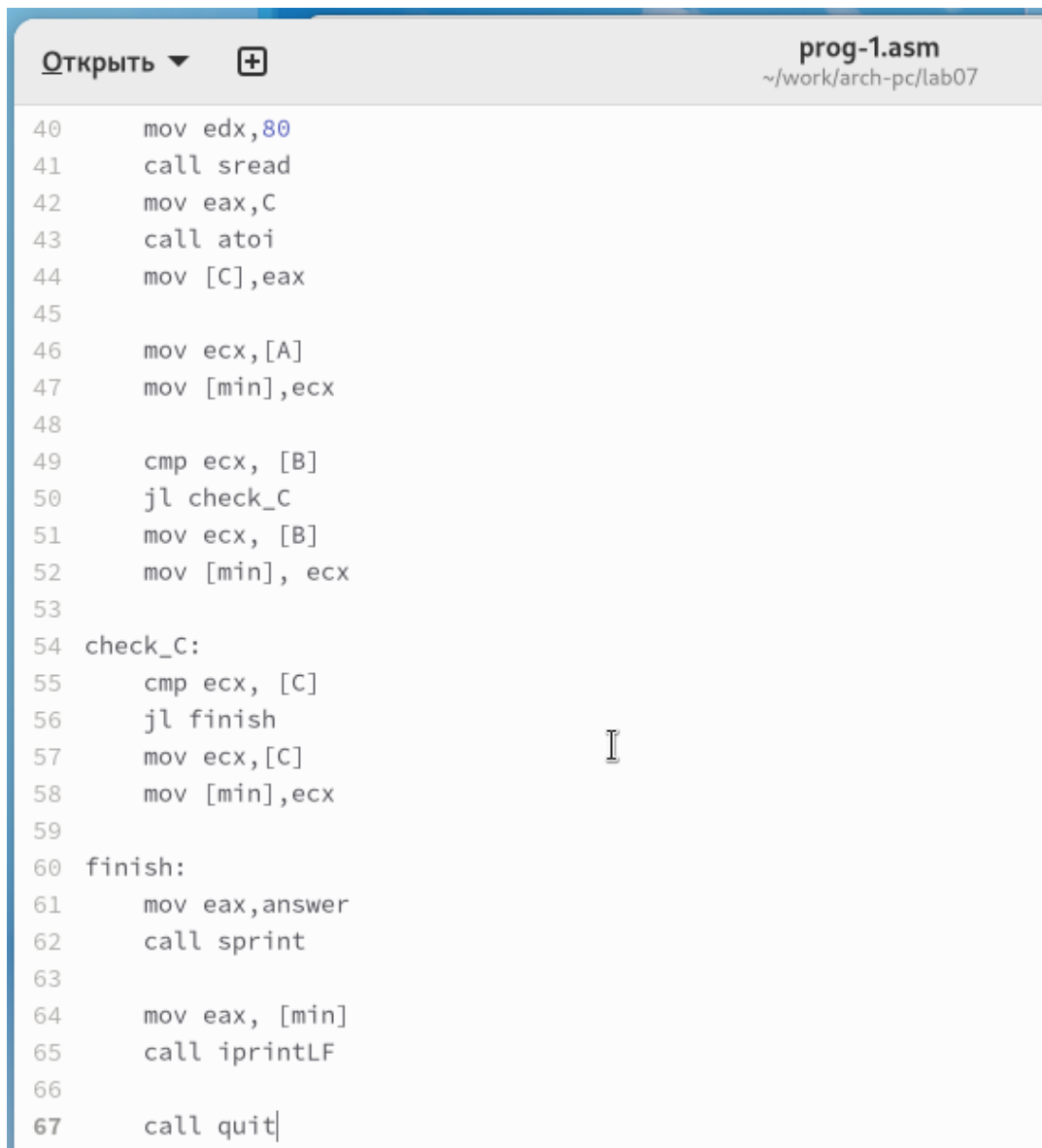
Рис. 4.11: Файл листинга с ошибкой lab7-2


Объектный файл не смог 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 (рис. [4.12]) (рис. [4.13])

Мой вариант 7 - числа: 45, 67, 15



```
Открыть ▾  prog-1.asm
~/work/arch-pc/lab07

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
67     call quit|
```

Рис. 4.12: Изменение кода prog-1.asm

```

[mdavletova@fedora lab07]$
[mdavletova@fedora lab07]$ nasm -f elf prog-1.asm
[mdavletova@fedora lab07]$ ld -m elf_i386 prog-1.o -o prog-1
[mdavletova@fedora lab07]$ ./prog-1
Input A: 45
Input B: 67
Input C: 15
Smallest: 15
[mdavletova@fedora lab07]$
[mdavletova@fedora lab07]$

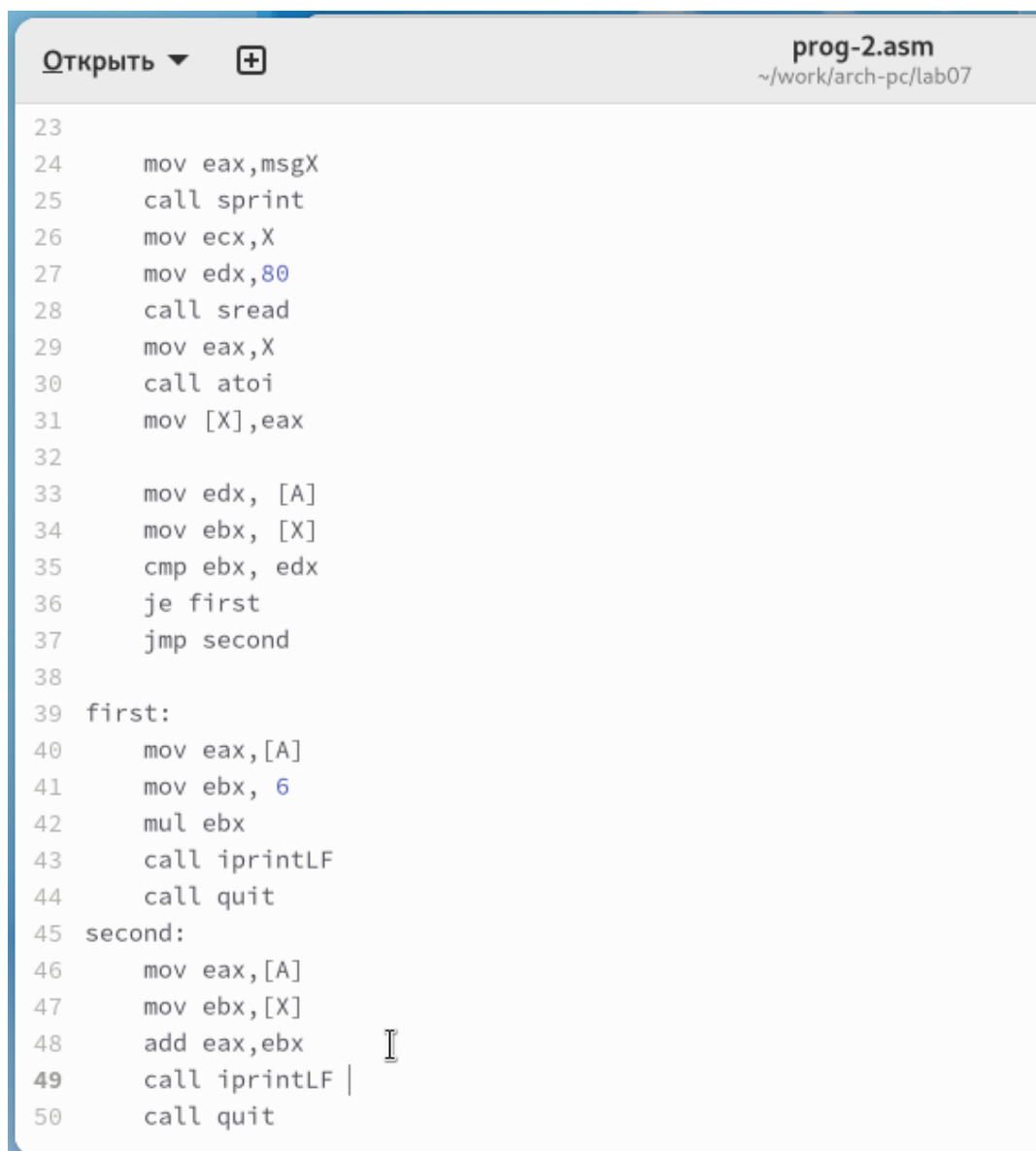
```

Рис. 4.13: Компиляция текста программы 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 7

$$\begin{cases} 6a, x = a \\ a + x, x \neq a \end{cases}$$




```
Открыть ▾  prog-2.asm  
~/work/arch-pc/lab07  
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, [A]  
34     mov ebx, [X]  
35     cmp ebx, edx  
36     je first  
37     jmp second  
38  
39 first:  
40     mov eax,[A]  
41     mov ebx, 6  
42     mul ebx  
43     call iprintLF  
44     call quit  
45 second:  
46     mov eax,[A]  
47     mov ebx,[X]  
48     add eax,ebx  
49     call iprintLF |  
50     call quit
```

Рис. 4.14: Изменение кода prog-2.asm

```
[mdavletova@fedora lab07]$ nasm -f elf prog-2.asm
[mdavletova@fedora lab07]$ ld -m elf_i386 prog-2.o -o prog-2
[mdavletova@fedora lab07]$ ./prog-2
Input A: 1
Input X: 1
6
[mdavletova@fedora lab07]$ ./prog-2
Input A: 1
Input X: 2
3
[mdavletova@fedora lab07]$
```

Рис. 4.15: Компиляция текста программы prog-2.asm

5 Выводы

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