

Отчёт по лабораторной работе 7

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

Ахмед Кусей

Содержание

1	Цель работы	5
2	Теоретическое введение	6
2.1	Команды перехода	6
2.2	Листинг	7
3	Выполнение лабораторной работы	8
3.1	Реализация переходов в NASM	8
3.2	Изучение структуры файлы листинга	15
3.3	Задание для самостоятельной работы	18
4	Выводы	22

Список иллюстраций

3.1	Программа в файле lab7-1.asm	9
3.2	Запуск программы lab7-1.asm	10
3.3	Программа в файле lab7-1.asm	11
3.4	Запуск программы lab7-1.asm	11
3.5	Программа в файле lab7-1.asm	12
3.6	Запуск программы lab7-1.asm	13
3.7	Программа в файле lab7-2.asm	14
3.8	Запуск программы lab7-2.asm	15
3.9	Файл листинга lab7-2	16
3.10	Ошибка трансляции lab7-2	17
3.11	Файл листинга с ошибкой lab7-2	18
3.12	Программа в файле task7-1.asm	19
3.13	Запуск программы task7-1.asm	19
3.14	Программа в файле task7-2.asm	20
3.15	Запуск программы task7-2.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 Теоретическое введение

2.1 Команды перехода

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.

Как отмечалось выше, для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов.

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.

2.2 Листинг

Листинг (в рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Он имеет текстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную информацию.

Итак, структура листинга:

- номер строки — это номер строки файла листинга (нужно помнить, что номер строки в файле листинга может не соответствовать номеру строки в файле с исходным текстом программы);
- адрес — это смещение машинного кода от начала текущего сегмента;
- машинный код представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности. (например, инструкция `int 80h` начинается по смещению `00000020` в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция `int 80h` ассемблируется в `CD80` (в шестнадцатеричном представлении); `CD80` — это инструкция на машинном языке, вызывающая прерывание ядра)
- исходный текст программы — это просто строка исходной программы вместе с комментариями (некоторые строки на языке ассемблера, например, строки, содержащие только комментарии, не генерируют никакого машинного кода, и поля «смещение» и «исходный текст программы» в таких строках отсутствуют, однако номер строки им присваивается)

3 Выполнение лабораторной работы

3.1 Реализация переходов в NASM

Создал каталог для программам лабораторной работы № 7 и файл lab7-1.asm

Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Написал в файл lab7-1.asm текст программы из листинга 7.1.


```
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 _label2
11
12 _label1:
13 mov eax, msg1
14 call sprintf
15
16 _label2:
17 mov eax, msg2
18 call sprintf
19
20 _label3:
21 mov eax, msg3
22 call sprintf
23
24 _end:
25 call quit
```

Рис. 3.1: Программа в файле lab7-1.asm

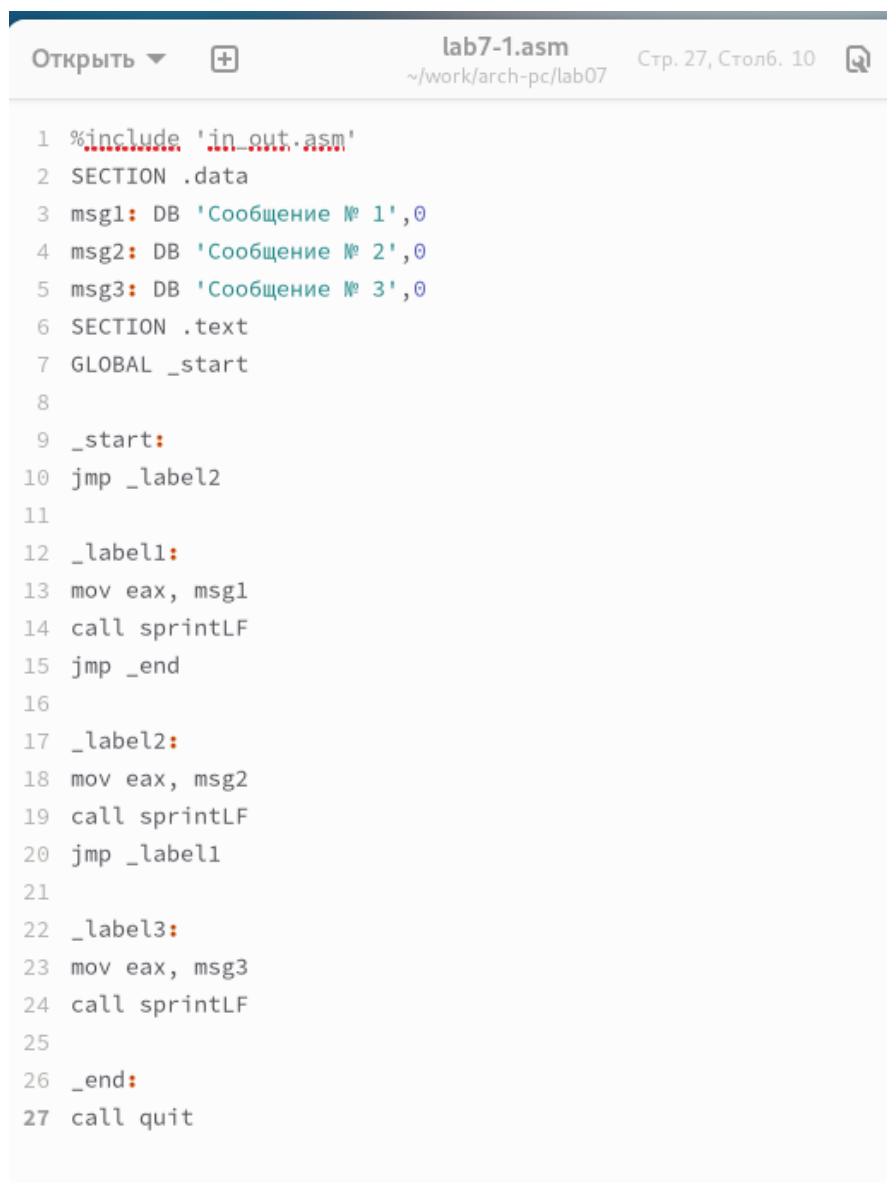
Создал исполняемый файл и запустил его.

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

Рис. 3.2: Запуск программы lab7-1.asm

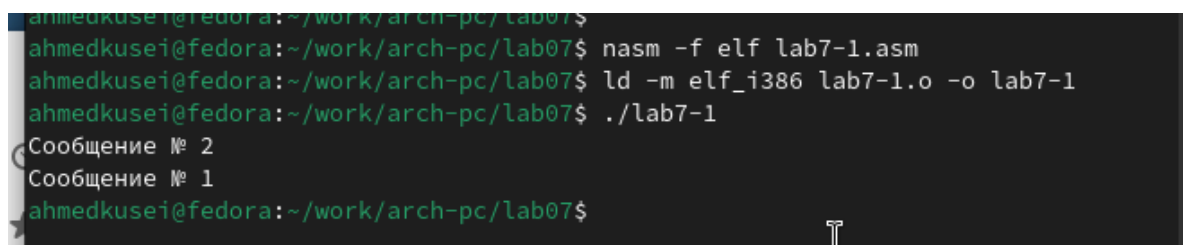
Инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. Изменим программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого в текст программы после вывода сообщения № 2 добавим инструкцию `jmp` с меткой `_label1` (т.е. переход к инструкциям вывода сообщения № 1) и после вывода сообщения № 1 добавим инструкцию `jmp` с меткой `_end` (т.е. переход к инструкции `call quit`).

Изменил 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. 3.3: Программа в файле lab7-1.asm



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

Рис. 3.4: Запуск программы lab7-1.asm

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

Рис. 3.5: Программа в файле lab7-1.asm

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

Рис. 3.6: Запуск программы lab7-1.asm

Использование инструкции `jmp` приводит к переходу в любом случае. Однако, часто при написании программ необходимо использовать условные переходы, т.е. переход должен происходить если выполнено какое-либо условие. В качестве примера рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: A, B и C. Значения для A и C задаются в программе, значение B вводится с клавиатуры.

Создал исполняемый файл и проверил его работу для разных значений B.



```
Открыть ▾ [иконка] lab7-2.asm Стр. 1, Столб. 1 [иконка] [меню] [закрыть]
~/work/arch-pc/lab07
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 sprintf
47 mov eax,[max]
48 call iprintLF
49 call quit
```

Рис. 3.7: Программа в файле lab7-2.asm

```
ahmedkusei@fedora:~/work/arch-pc/lab07$  
ahmedkusei@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm  
ahmedkusei@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-2.o -o lab7-2  
ahmedkusei@fedora:~/work/arch-pc/lab07$ ./lab7-2  
Введите В: 3  
Наибольшее число: 50  
ahmedkusei@fedora:~/work/arch-pc/lab07$ ./lab7-2  
Введите В: 52  
Наибольшее число: 52  
ahmedkusei@fedora:~/work/arch-pc/lab07$
```

Рис. 3.8: Запуск программы lab7-2.asm

3.2 Изучение структуры файлы листинга

Обычно nasm создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ `-l` и задав имя файла листинга в командной строке.

Создал файл листинга для программы из файла lab7-2.asm

lab7-2.asm			lab7-2.lst	
118	117	00000092 80F0	<1>	mov eax, esp
119	118	00000094 F876FFFFFF	<1>	call sprintf
120	119	00000099 58	<1>	add esp
121	120	0000009A 58	<1>	add esp
122	121	0000009B C3	<1>	ret
123	122		<1>	
124	123		<1>	;----- atoi -----
125	124		<1>	; функция преобразования ascii-код символа в целое число
126	125		<1>	; входные данные: mov eax, <int>
127	126		<1>	atoi:
128	127	0000009C 53	<1>	push ebx
129	128	0000009D 51	<1>	push ecx
130	129	0000009E 52	<1>	push edx
131	130	0000009F 56	<1>	push esi
132	131	000000A0 80C6	<1>	mov esi, eax
133	132	000000A2 B800000000	<1>	mov eax, 0
134	133	000000A7 B800000000	<1>	mov ecx, 0
135	134		<1>	
136	135		<1>	.multiplyloop:
137	136	000000AC 31D8	<1>	xor ebx, ebx
138	137	000000AE 8A1C0F	<1>	mov hl, [esi+ecx]
139	138	000000B1 80FB30	<1>	cmp hl, 48
140	139	000000B4 7C14	<1>	jl .finished
141	140	000000B6 80FB39	<1>	cmp hl, 57
142	141	000000B9 7F0F	<1>	je .finished
143	142		<1>	
144	143	000000BB 80FB30	<1>	sub hl, 48
145	144	000000BE 01D8	<1>	add eax, ebx
146	145	000000C0 B80A000000	<1>	mov ebx, 10
147	146	000000C5 F7E3	<1>	mul ebx
148	147	000000C7 41	<1>	inc ecx

Рис. 3.9: Файл листинга lab7-2

Внимательно ознакомился с его форматом и содержимым. Подробно объясню содержимое трёх строк файла листинга по выбору.

строка 203

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

строка 204

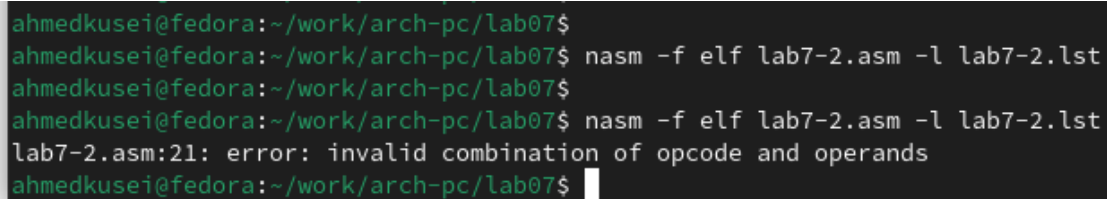
- 29 - номер строки в подпрограмме

- 00000122 - адрес
- 7F0C - машинный код
- jg check_B - код программы - если >, то переход к метке check_B

строка 205

- 30 - номер строки в подпрограмме
- 00000124 - адрес
- 8B0D[39000000] - машинный код
- mov esx,[C] - код программы - перекладывает в регистр esx значение переменной C

Открыл файл с программой lab7-2.asm и в инструкции с двумя операндами удалил один операнд. Выполнил трансляцию с получением файла листинга.



```
ahmedkusei@fedora:~/work/arch-pc/lab07$
ahmedkusei@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm -l lab7-2.lst
ahmedkusei@fedora:~/work/arch-pc/lab07$
ahmedkusei@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm -l lab7-2.lst
lab7-2.asm:21: error: invalid combination of opcode and operands
ahmedkusei@fedora:~/work/arch-pc/lab07$
```

Рис. 3.10: Ошибка трансляции lab7-2

```

192 17 000000F2 B0[0A000000] mov ecx,B
193 18 000000F7 BA0A000000 mov edx,10
194 19 000000FC F842FFFFFF call atead
195 20 ; ----- Преобразование 'R' из символа в число
196 21 mov eax,
197 21 ***** error: invalid combination of opcode and operands
198 22 00000101 F896FFFFFF call atoi
199 23 00000106 A3[0A000000] mov [R],eax
200 24 ; ----- Записываем 'A' в переменную 'max'
201 25 0000010B 8B0D[35000000] mov ecx,[A]
202 26 00000111 890D[00000000] mov [max],ecx
203 27 ; ----- Сравниваем 'A' и 'C' (как символы)
204 28 00000117 3B0D[39000000] cmp ecx,[C]
205 29 0000011D 7F0C jg check_R
206 30 0000011F 8B0D[39000000] mov ecx,[C]
207 31 00000125 890D[00000000] mov [max],ecx
208 32 ; ----- Преобразование 'max(A,C)' из символа в число
209 33 check_R:
210 34 0000012B B8[00000000] mov eax,max
211 35 00000130 F867FFFFFF call atoi
212 36 00000135 A3[00000000] mov [max],eax
213 37 ; ----- Сравниваем 'max(A,C)' и 'R' (как числа)
214 38 0000013A 8B0D[00000000] mov ecx,[max]
215 39 00000140 3B0D[0A000000] cmp ecx,[R]
216 40 00000146 7F0C jg fin
217 41 00000148 8B0D[0A000000] mov ecx,[R]
218 42 0000014F 890D[00000000] mov [max],ecx
219 43 ; ----- Вывод результата
220 44 fin:
221 45 00000154 B8[12000000] mov eax,max

```

Рис. 3.11: Файл листинга с ошибкой lab7-2

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

3.3 Задание 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

для варианта 10 - 41,62,35

```

Открыть  lab7-2.lst  Стр. 1, Столб. 1
~/work/arch-pc/lab07

184 9 0000000A <res Ah> B resb 10
185 10 section .text
186 11 global _start
187 12 _start:
188 13 ; ----- Вывод сообщения 'Введите R: '
189 14 000000F8 B8[00000000] mov eax,msg1
190 15 000000FD F81DFFFFFF call sprint
191 16 ; ----- Ввод 'R'
192 17 000000F2 B9[0A000000] mov ecx,R
193 18 000000F7 BA0A000000 mov edx,10
194 19 000000FC F842FFFFFF call sread
195 20 ; ----- Преобразование 'R' из символа в число
196 21 mov eax,
197 21 ***** error: invalid combination of opcode and operands
198 22 00000101 F896FFFFFF call atoi
199 23 00000106 A3[0A000000] mov [R],eax
200 24 ; ----- Записываем 'A' в переменную 'max'
201 25 0000010B 8B0D[35000000] mov ecx,[A]
202 26 00000111 890D[00000000] mov [max],ecx
203 27 ; ----- Сравниваем 'A' и 'C' (как символы)
204 28 00000117 3B0D[39000000] cmp ecx,[C]
205 29 0000011D 7F0C je check_R
206 30 0000011F 8B0D[39000000] mov ecx,[C]
207 31 00000125 890D[00000000] mov [max],ecx
208 32 ; ----- Преобразование 'max(A,C)' из символа в число
209 33 check_R:
210 34 0000012B B8[00000000] mov eax,max
211 35 00000130 F867FFFFFF call atoi
212 36 00000135 A3[00000000] mov [max],eax
213 37 ; ----- Сравниваем 'max(A,C)' и 'R' (как числа)
214 38 0000013A 8B0D[00000000] mov ecx,[max]
215 39 00000140 3B0D[0A000000] cmp ecx,[R]
216 40 00000146 7F0C je fin

```

Рис. 3.12: Программа в файле task7-1.asm

```

ahmedkusei@fedora:~/work/arch-pc/lab07$
ahmedkusei@fedora:~/work/arch-pc/lab07$
ahmedkusei@fedora:~/work/arch-pc/lab07$ nasm -f elf task7-1.asm
ahmedkusei@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 task7-1.o -o task7-1
ahmedkusei@fedora:~/work/arch-pc/lab07$ ./task7-1
Input A: 41
Input B: 62
Input C: 35
Smallest: 35
ahmedkusei@fedora:~/work/arch-pc/lab07$

```

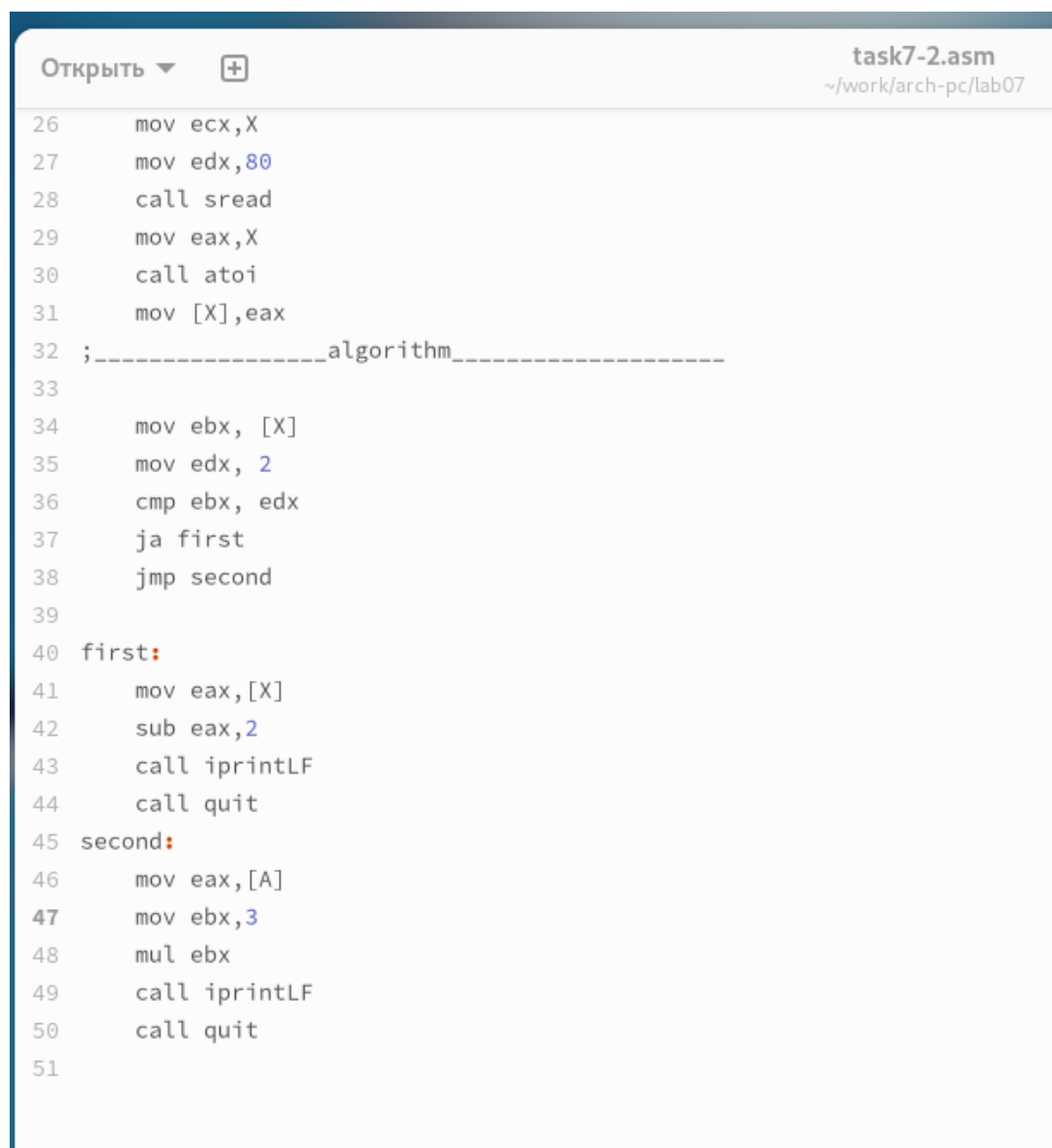
Рис. 3.13: Запуск программы task7-1.asm

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.

для варианта 10

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



```
task7-2.asm
~/work/arch-pc/lab07

26     mov ecx,X
27     mov edx,80
28     call sread
29     mov eax,X
30     call atoi
31     mov [X],eax
32     ;-----algorithm-----
33
34     mov ebx, [X]
35     mov edx, 2
36     cmp ebx, edx
37     ja first
38     jmp second
39
40 first:
41     mov eax,[X]
42     sub eax,2
43     call iprintLF
44     call quit
45 second:
46     mov eax,[A]
47     mov ebx,3
48     mul ebx
49     call iprintLF
50     call quit
51
```

Рис. 3.14: Программа в файле task7-2.asm

```
ahmedkusei@fedora:~/work/arch-pc/lab07$  
ahmedkusei@fedora:~/work/arch-pc/lab07$ nasm -f elf task7-2.asm  
ahmedkusei@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 task7-2.o -o task7-2  
ahmedkusei@fedora:~/work/arch-pc/lab07$ ./task7-2  
Input A: 0  
Input X: 3  
1  
ahmedkusei@fedora:~/work/arch-pc/lab07$ ./task7-2  
Input A: 2  
Input X: 1  
6  
ahmedkusei@fedora:~/work/arch-pc/lab07$
```

Рис. 3.15: Запуск программы task7-2.asm

4 Выводы

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