

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютеров и операционные системы

Черная София Витальевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Изучение структуры файлы листинга	13
4.2	Выполнение заданий для самостоятельной работы	14
5	Выводы	21
	Список литературы	22

Список иллюстраций

4.1	Создание директории	8
4.2	Создание файла	8
4.3	Создание копии файла	8
4.4	Ввод текста программы из листинга 7.1	9
4.5	Запуск программного кода	9
4.6	Изменение текста программы	10
4.7	Создание исполняемого файла	10
4.8	Изменение текста программы	11
4.9	Вывод программы	11
4.10	Создание файла	11
4.11	Ввод текста программы из листинга 7.3	12
4.12	Проверка работы файла	12
4.13	Создание файла листинга	13
4.14	Изучение файла листинга	13
4.15	Выбранные строки файла	13
4.16	Удаление выделенного операнда из кода	14
4.17	Получение файла листинга	14
4.18	Создание файла	14
4.19	Написание программы	15
4.20	Запуск файла и проверка его работы	15
4.21	Написание программы	17
4.22	Запуск файла и проверка его работы	18

Список таблиц

1 Цель работы

Изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

2 Задание

1. Реализация переходов в NASM.
2. Изучение структуры файлы листинга.
3. Задания 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`. 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.

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.

4 Выполнение лабораторной работы

С помощью утилиты `mkdir` создаю директорию, в которой буду создавать файлы с программами для лабораторной работы №7 (рис. [4.1]). Перехожу в созданный каталог с помощью утилиты `cd`.

```
(svchernaya@svchernaya)~$ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab07
(svchernaya@svchernaya)~$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab07
```

Рис. 4.1: Создание директории

С помощью утилиты `touch` создаю файл `lab7-1.asm` (рис. [4.2]).

```
(svchernaya@svchernaya)~$ touch ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab07/lab7-1.asm
(svchernaya@svchernaya)~$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab07
(svchernaya@svchernaya)~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab07$ ls
lab7-1.asm
```

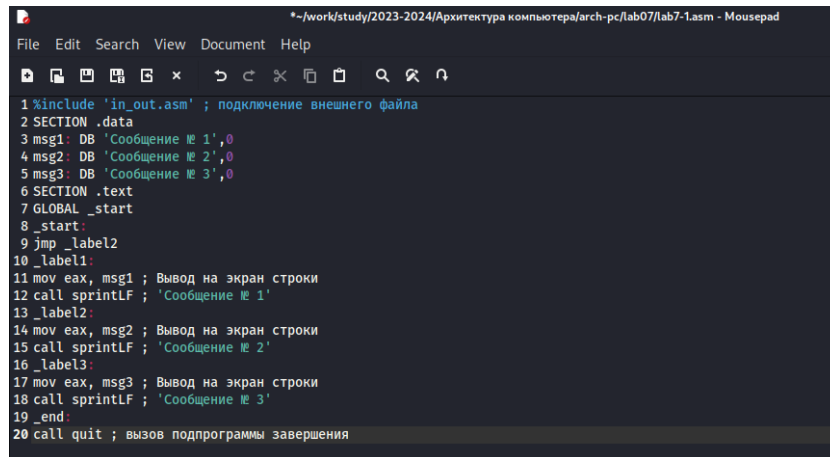
Рис. 4.2: Создание файла

Копирую в текущий каталог файл `in_out.asm` с помощью утилиты `cp`, т.к. он будет использоваться в других программах (рис. [4.3]).

```
(svchernaya@svchernaya)~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab07$ cp ~/Downloads/in_out.asm in_out.asm
(svchernaya@svchernaya)~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab07$ ls
in_out.asm  lab7-1.asm
```

Рис. 4.3: Создание копии файла

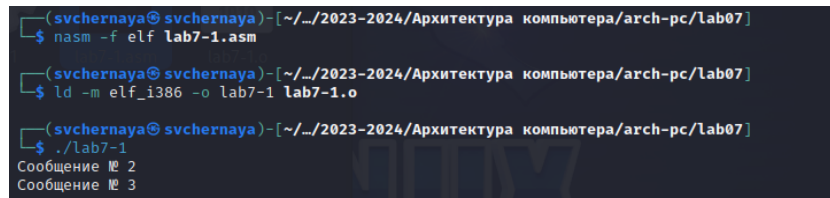
Ввожу в файл `lab7-1.asm` текст программы из листинга 7.1. (рис. [4.4]).

A screenshot of a text editor window titled "*/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07/lab7-1.asm - Mousepad". The editor contains assembly code for a program. The code includes a data section with three messages and a text section with labels and instructions. The instructions use the sprintf function to print the messages. The code is as follows:

```
1 %include 'in_out.asm' ; подключение внешнего файла
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 _start:
9 jmp _label2
10 _label1:
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение № 1'
13 _label2:
14 mov eax, msg2 ; Вывод на экран строки
15 call sprintf ; 'Сообщение № 2'
16 _label3:
17 mov eax, msg3 ; Вывод на экран строки
18 call sprintf ; 'Сообщение № 3'
19 _end:
20 call quit ; вызов подпрограммы завершения
```

Рис. 4.4: Ввод текста программы из листинга 7.1

Создаю исполняемый файл и запускаю его. (рис. [4.5]).

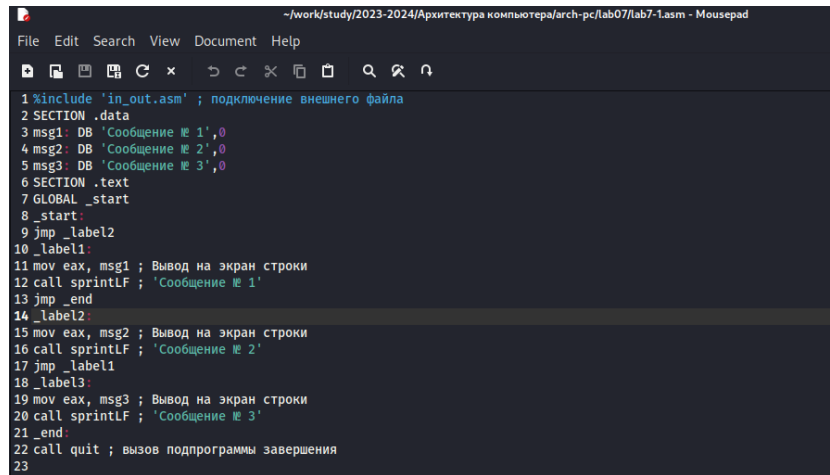
A screenshot of a terminal window showing the compilation and execution of the assembly program. The terminal output is as follows:

```
(svchernaya@svchernaya)~[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ nasm -f elf lab7-1.asm
(svchernaya@svchernaya)~[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ ld -m elf_i386 -o lab7-1 lab7-1.o
(svchernaya@svchernaya)~[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ ./lab7-1
Сообщение № 2
Сообщение № 3
```

Рис. 4.5: Запуск программного кода

Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения.

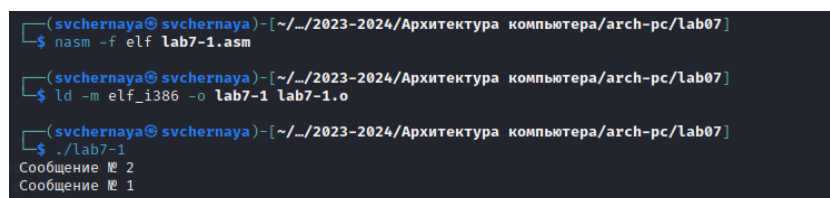
Изменяю программу таким образом, чтобы она выводила сначала 'Сообщение № 2', потом 'Сообщение № 1' и завершала работу. Для этого изменяю текст программы в соответствии с листингом 7.2. (рис. [4.6]).



```
1 %include 'in_out.asm' ; подключение внешнего файла
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 _start:
9 jmp _label2
10 _label1:
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение № 1'
13 jmp _end
14 _label2:
15 mov eax, msg2 ; Вывод на экран строки
16 call sprintf ; 'Сообщение № 2'
17 jmp _label1
18 _label3:
19 mov eax, msg3 ; Вывод на экран строки
20 call sprintf ; 'Сообщение № 3'
21 _end:
22 call quit ; вызов подпрограммы завершения
23
```

Рис. 4.6: Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. [4.7]).



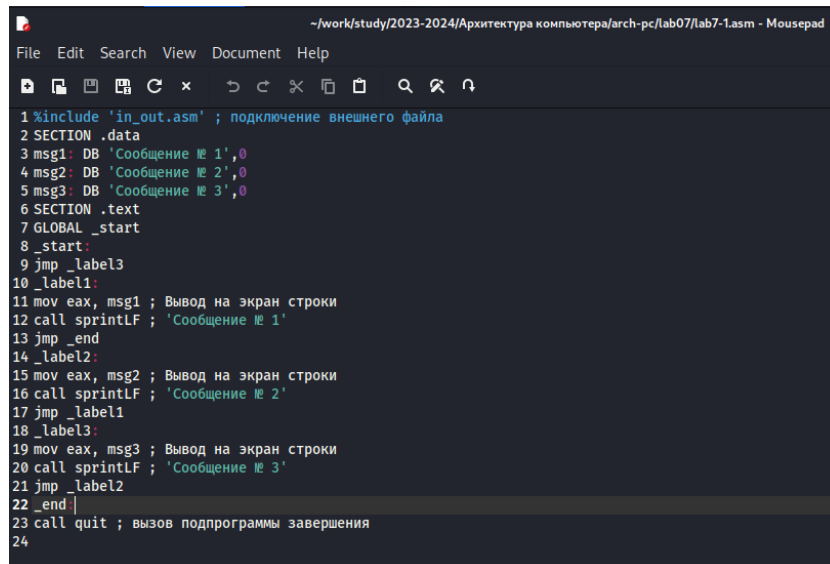
```
(svchernaya@svchernaya)~[/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ nasm -f elf lab7-1.asm

(svchernaya@svchernaya)~[/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ ld -m elf_i386 -o lab7-1 lab7-1.o

(svchernaya@svchernaya)~[/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ ./lab7-1
Сообщение № 2
Сообщение № 1
```

Рис. 4.7: Создание исполняемого файла

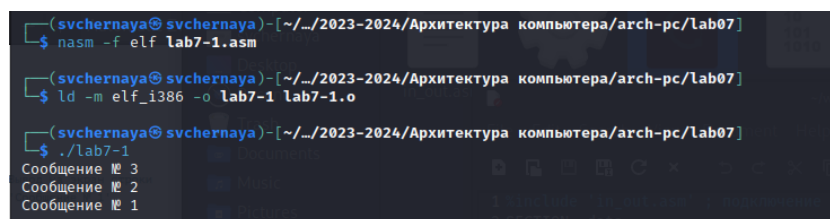
Затем изменяю текст программы, добавив в начале программы `jmp _label3`, `jmp _label2` в конце метки `jmp _label3`, `jmp _label1` добавляю в конце метки `jmp _label2`, и добавляю `jmp _end` в конце метки `jmp _label1`, (рис. [4.8]).



```
1 %include 'in_out.asm' ; подключение внешнего файла
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 _start:
9 jmp _label3
10 _label1:
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение № 1'
13 jmp _end
14 _label2:
15 mov eax, msg2 ; Вывод на экран строки
16 call sprintf ; 'Сообщение № 2'
17 jmp _label1
18 _label3:
19 mov eax, msg3 ; Вывод на экран строки
20 call sprintf ; 'Сообщение № 3'
21 jmp _label2
22 _end:
23 call quit ; вызов подпрограммы завершения
24
```

Рис. 4.8: Изменение текста программы

чтобы вывод программы был следующим: (рис. [4.9]).

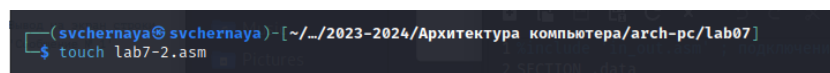


```
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ nasm -f elf lab7-1.asm
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ ld -m elf_i386 -o lab7-1 lab7-1.o
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Рис. 4.9: Вывод программы

Рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А,В и С. Значения для А и С задаются в программе, значение В вводится с клавиатуры.

Создаю файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. (рис. [4.10]).



```
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ touch lab7-2.asm
```

Рис. 4.10: Создание файла

Текст программы из листинга 7.3 ввожу в lab7-2.asm. (рис. [4.11]).

```

1 %include 'in_out.asm'
2 section .data
3 msg1 db 'Введите B: ',0h
4 msg2 db "Наибольшее число: ",0h
5 A dd '20'
6 C dd '50'
7 section .bss
8 max resb 10
9 B resb 10
10 section .text
11 global _start
12 _start:
13 ; ----- Вывод сообщения 'Введите B: '
14 mov eax,msg1
15 call sprint
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 ; Вызов подпрограммы перевода символа в число
23 mov [B],eax ; запись преобразованного числа в 'B'
24 ; ----- Записываем 'A' в переменную 'max'
25 mov ecx,[A] ; 'ecx = A'
26 mov [max],ecx ; 'max = A'
27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 jg check_B ; если 'A>C', то переход на метку 'check_B',
30 mov ecx,[C] ; иначе 'ecx = C'
31 mov [max],ecx ; 'max = C'
32 ; ----- Преобразование 'max(A,C)' из символа в число
33 check_B:
34 mov eax,max
35 call atoi ; Вызов подпрограммы перевода символа в число
36 mov [max],eax ; запись преобразованного числа в 'max'
37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
38 mov ecx,[max]
39 cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
40 jg fin ; если 'max(A,C)>B', то переход на 'fin',
41 mov ecx,[B] ; иначе 'ecx = B'
42 mov [max],ecx
43 ; ----- Вывод результата
44 fin:
45 mov eax, msg2
46 call sprint ; Вывод сообщения 'Наибольшее число: '
47 mov eax,[max]
48 call iprintLF ; Вывод 'max(A,B,C)'
49 call quit ; Выход

```

Рис. 4.11: Ввод текста программы из листинга 7.3

Создаю исполняемый файл и проверьте его работу. (рис. [4.12]).

```

(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ nasm -f elf lab7-2.asm

(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ ld -m elf_i386 -o lab7-2 lab7-2.o

(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ ./lab7-2
Введите B: 100
Наибольшее число: 100

```

Рис. 4.12: Проверка работы файла

Файл работает корректно.

4.1 Изучение структуры файлы листинга

Создаю файл листинга для программы из файла lab7-2.asm. (рис. [4.13]).

```
(svchernaya@svchernaya)~[~/2023-2024/Архитектура компьютера/arch-pc/lab07]  
$ nasm -f elf -l lab7-2.lst lab7-2.asm
```

Рис. 4.13: Создание файла листинга

Открываю файл листинга lab7-2.lst с помощью текстового редактора и внимательно изучаю его формат и содержимое. (рис. [4.14]).

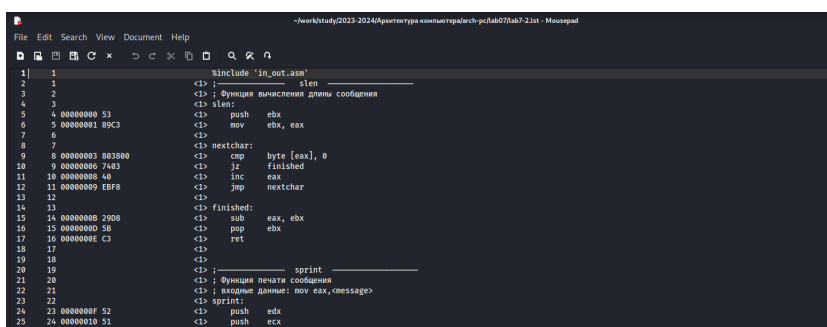


Рис. 4.14: Изучение файла листинга

В представленных трех строчках содержатся следующие данные: (рис. [4.15]).

```
2                                <1> ; Функция вычисления длины сообщения  
3                                <1> slen:  
4 00000000 53                   <1>      push    ebx
```

Рис. 4.15: Выбранные строки файла

“2” - номер строки кода, “; Функция вычисления длинны сообщения” - комментарий к коду, не имеет адреса и машинного кода.

“3” - номер строки кода, “slen” - название функции, не имеет адреса и машинного кода.

“4” - номер строки кода, “00000000” - адрес строки, “53” - машинный код, “push ebx” - исходный текст программы, инструкция “push” помещает операнд “ebx” в стек.

Открываю файл с программой lab7-2.asm и в выбранной мной инструкции с двумя операндами удаляю выделенный операнд. (рис. [4.16]).

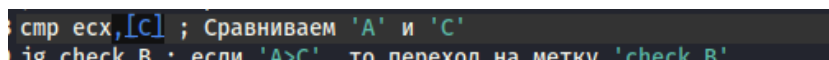


Рис. 4.16: Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга. (рис. [4.17]).

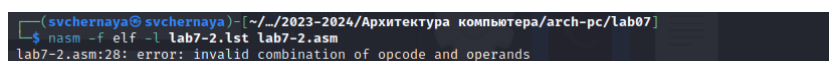


Рис. 4.17: Получение файла листинга

На выходе я не получаю ни одного файла из-за ошибки: инструкция mov (единственная в коде содержит два операнда) не может работать, имея только один операнд, из-за чего нарушается работа кода.

4.2 Выполнение заданий для самостоятельной работы

Создаю файл task1.asm с помощью утилиты touch (рис. [4.18]).

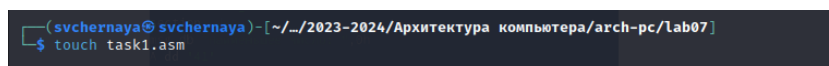


Рис. 4.18: Создание файла

Пишу программу нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения переменных выбираю из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Мой вариант под номером 4, поэтому мои значения - 8, 88 и 68. (рис. [4.19]).

```
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07/task1.asm - Mousepad
File Edit Search View Document Help
1 %include 'in_out.asm'
2 section .data
3 msg1 db "Наименьшее число: ",0h
4 A dd '8'
5 B dd '88'
6 C dd '68'
7 section .bss
8 min resb 10
9 section .text
10 global _start
11 _start:
12 ; ----- Записываем 'B' из символа в число
13 mov eax,B
14 call atoi ; Вызов подпрограммы перевода символа в число
15 mov [B],eax ; запись преобразованного числа в 'B'
16 ; ----- Записываем 'A' в переменную 'min'
17 mov ecx,[A] ; 'ecx = A'
18 mov [min],ecx ; 'min = A'
19 ; ----- Сравниваем 'A' и 'C' как символы
20 cmp ecx,[C] ; Сравниваем 'A' и 'C'
21 jl check_B ; если 'A>C', то переход на метку 'check_B',
22 mov ecx,[C] ; иначе 'ecx = C'
23 mov [min],ecx ; 'min = C'
24 ; ----- Преобразование 'min(A,C)' из символа в число
25 check_B:
26 mov eax,min
27 call atoi ; Вызов подпрограммы перевода символа в число
28 mov [min],eax ; запись преобразованного числа
29 ; ----- Сравниваем 'min(A,C)' и 'B' как числа
30 mov ecx,[min]
31 cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
32 jl fin ; если 'min(A,C)>B', то переход на 'fin',
33 mov ecx [B] ; иначе 'ecx = B'
```

Рис. 4.19: Написание программы

Создаю исполняемый файл и проверяю его работу, подставляя необходимые значение. (рис. [4.20]).

```
(svchernaya@svchernaya)~[/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ nasm -f elf task1.asm
(svchernaya@svchernaya)~[/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ ld -m elf_i386 task1.o -o task1
(svchernaya@svchernaya)~[/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ ./task1
Наименьшее число: 8
```

Рис. 4.20: Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```
%include 'in_out.asm'
section .data
msg1 db "Наименьшее число: ",0h
A dd '8'
B dd '88'
```

```

C dd '68'

section .bss
min resb 10

section .text
global _start
_start:
; ----- Записываем '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 '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:
mov eax,min
call atoi ; Вызов подпрограммы перевода символа в число
mov [min],eax ; запись преобразованного числа
; ----- 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, msg1
call sprint ; Вывод сообщения 'Наименьшее число: '
mov eax,[min]
call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход

```

Пишу программу, которая для введенных с клавиатуры значений x и a вычисляет значение и выводит результат вычислений заданной для моего варианта функции $f(x)$:

$2 * x - a$, если $a \neq 0$
 $2 * x + 1$, если $a = 0$
 (рис. [4.21]).

```

~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07/task2.asm - Mousepad
File Edit Search View Document Help
1 %include 'in_out.asm'
2 section .data
3
4 msg1 DB 'Введите x: ',0h
5 msg2 DB 'Введите a: ',0h
6 otv DB 'F(x)=',0h
7
8 section .bss
9 x: RESB 10
10 a: RESB 10
11 res: RESB 10
12
13 section .text
14 global _start
15 _start:
16
17 mov eax, msg1
18 call sprint
19 mov ecx, x
20 mov edx, 10
21 call sread
22 mov eax, x
23 call atoi
24 mov [x], eax
25
26 mov eax, msg2
27 call sprint
28 mov ecx, a
29 mov edx, 10
30 call sread
31 mov eax, a
32 call atoi
33 mov [a], eax
34
35 mov ecx, [a]

```

Рис. 4.21: Написание программы

Создаю исполняемый файл и проверяю его работу для значений x и a соответственно: $(3;0)$, $(3;2)$. (рис. [4.22]).

```
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ nasm -f elf task2.asm
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ ld -m elf_1386 task2.o -o task2
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ ./task2
Введите x: 3
Введите a: 0
F(x)=7
(svchernaya@svchernaya)-[~/2023-2024/Архитектура компьютера/arch-pc/lab07]
$ ./task2
Введите x: 3
Введите a: 2
F(x)=8
```

Рис. 4.22: Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```
%include 'in_out.asm'

section .data

msg1 DB 'Введите x: ',0h
msg2 DB "Введите a: ",0h
otv: DB 'F(x)=',0h

section .bss
x: RESB 10
a: RESB 10
res: RESB 10

section .text
global _start
_start:

mov eax, msg1
call sprint
mov ecx, x
mov edx, 10
```

```

call sread
mov eax,x
call atoi
mov [x],eax

mov eax,msg2
call sprint
mov ecx,a
mov edx,10
call sread
mov eax,a
call atoi
mov [a],eax

mov ecx, [a]
cmp ecx, 0
je x_is_0
mov eax, [x]
mov ebx,2
mul ebx
add eax, ecx
jmp calc_res
x_is_0:
mov ebx,2
mov eax, [x]
mul ebx
inc eax
calc_res:
mov [res],eax

```

```
fin:
mov eax,otv
call sprint
mov eax,[res]
call iprintLF
call quit
```

5 Выводы

По итогам данной лабораторной работы я изучила команды условного и безусловного переходов, приобрела навыки написания программ с использованием переходов и ознакомилась с назначением и структурой файла листинга, что поможет мне при выполнении последующих лабораторных работ.

Список литературы