

Лабораторная работа №7

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

Швед Карина Дмитриевна

Содержание

1	Цель работы	5
2	Ход работы	6
3	Задание для самостоятельной работы	13
4	Выводы	16

Список иллюстраций

2.1	Код в файле lab7-1.asm.	6
2.2	работа программы в файле lab7-1.asm.	7
2.3	код программы в файле lab7-1.asm.	8
2.4	работа программы в файле lab7-1.asm.	8
2.5	код программы в файле lab7-1.asm.	9
2.6	работа программы в файле lab7-1.asm.	9
2.7	код программы в файле lab7-2.asm.	10
2.8	работа программы в файле lab7-2.asm.	10
2.9	файл листинга lab7-2	11
2.10	ошибка трансляции lab7-2	12
2.11	файл листинга lab7-2 с ошибкой	12
3.1	код программы task7-1	13
3.2	работа программы task7-1	14
3.3	код программы task7-2	15
3.4	работа программы task7-2	15

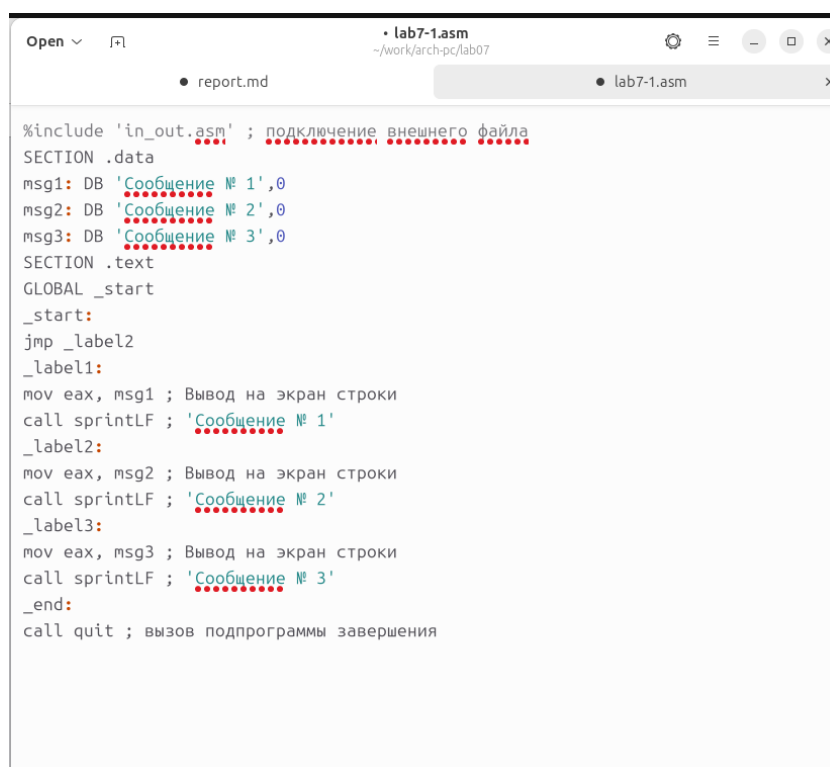
Список таблиц

1 Цель работы

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


2 Ход работы

Я создала каталог для программ лабораторной работы № 7, перешла в него и создала файл lab7-1.asm. Далее попробовала использование инструкции jmp и ввела в этот файл текст программы из листинга 7.1. (рис. 2.1). Создала исполняемый файл и запустила его. (рис. 2.2).



```
%include 'in_out.asm' ; подключение внешнего файла
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 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

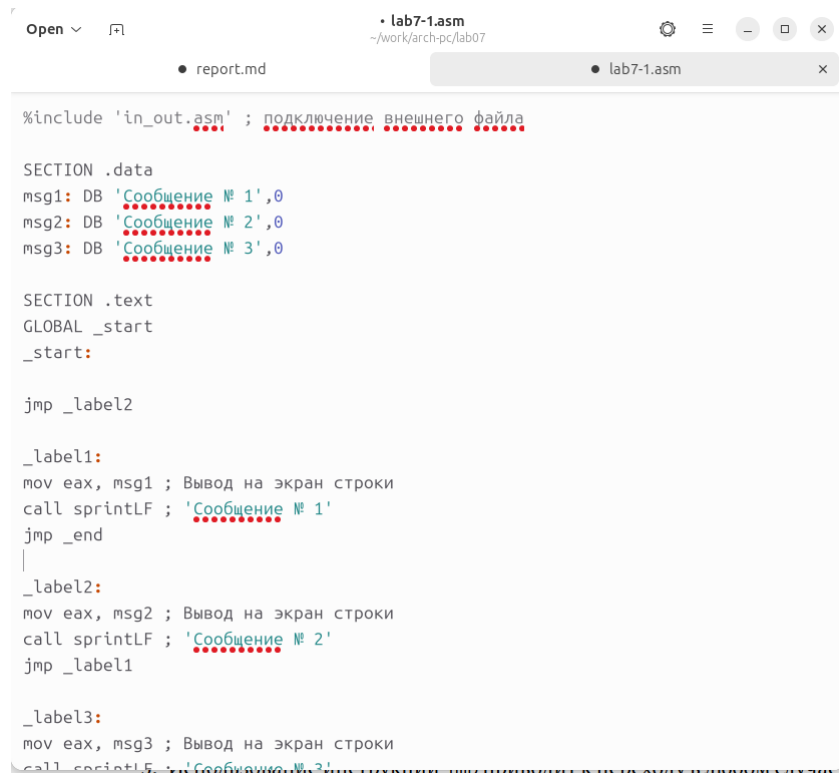
Рис. 2.1: Код в файле lab7-1.asm.



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

Рис. 2.2: работа программы в файле lab7-1.asm.

Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения. Инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. Я изменила программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого в текст программы после вывода сообщения № 2 я добавила инструкцию `jmp` с меткой `_label1` (т.е. переход к инструкциям вывода сообщения № 1) и после вывода сообщения № 1 инструкцию `jmp` с меткой `_end` (т.е. переход к инструкции `call quit`). Так я изменила текст программы в соответствии с листингом 7.2. (рис. 2.3). Создала исполняемый файл и проверила его работу (рис. 2.4)



```
%include 'in_out.asm' ; подключение внешнего файла

SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0

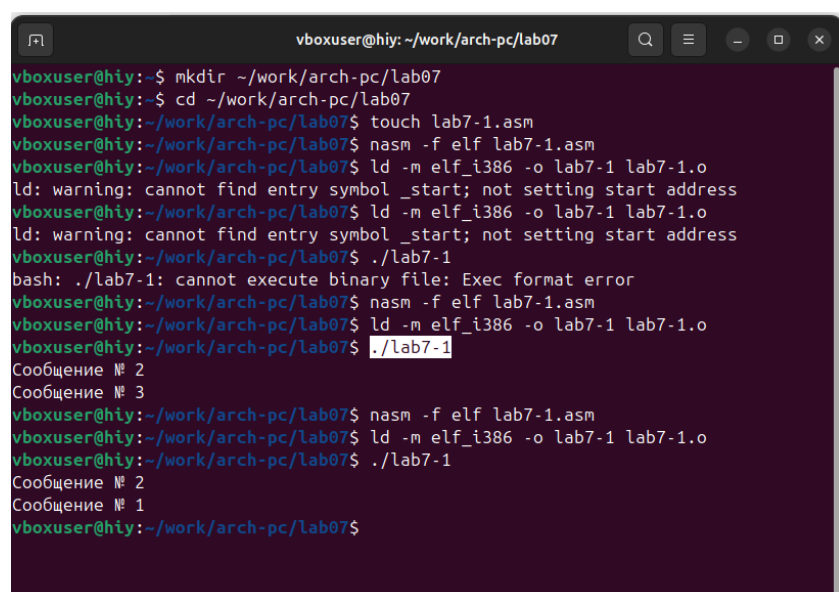
SECTION .text
GLOBAL _start
_start:

jmp _label2

_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
jmp _end
|
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label1

_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
```

Рис. 2.3: код программы в файле lab7-1.asm.



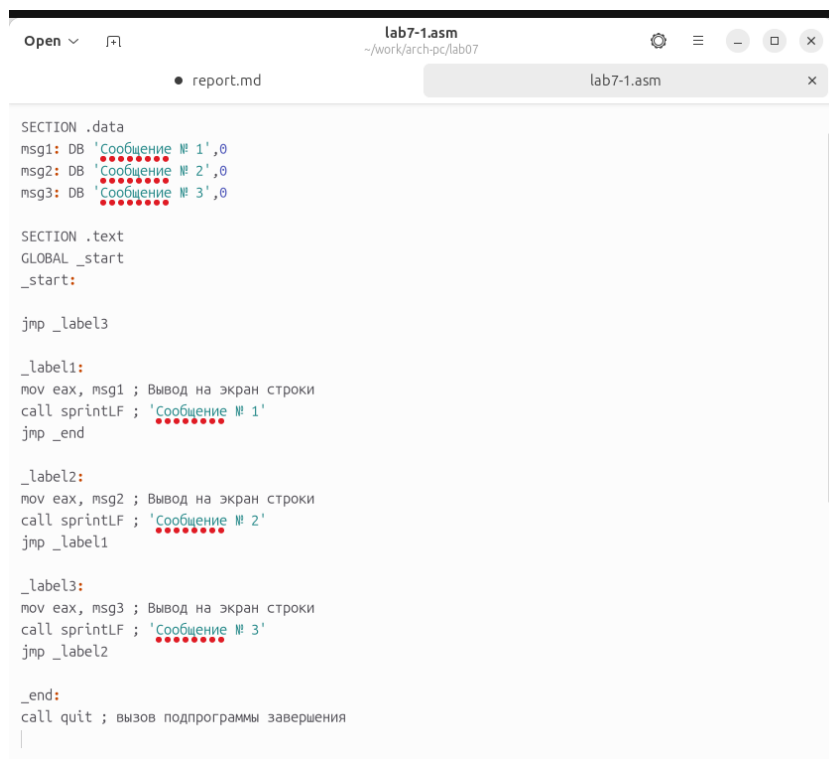
```
vboxuser@hiy: ~/work/arch-pc/lab07
vboxuser@hiy:~$ mkdir ~/work/arch-pc/lab07
vboxuser@hiy:~$ cd ~/work/arch-pc/lab07
vboxuser@hiy:~/work/arch-pc/lab07$ touch lab7-1.asm
vboxuser@hiy:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
vboxuser@hiy:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
ld: warning: cannot find entry symbol _start; not setting start address
vboxuser@hiy:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
ld: warning: cannot find entry symbol _start; not setting start address
vboxuser@hiy:~/work/arch-pc/lab07$ ./lab7-1
bash: ./lab7-1: cannot execute binary file: Exec format error
vboxuser@hiy:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
vboxuser@hiy:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
vboxuser@hiy:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
Сообщение № 2
Сообщение № 1
vboxuser@hiy:~/work/arch-pc/lab07$
```

Рис. 2.4: работа программы в файле lab7-1.asm.

Далее я изменила текст программы, чтобы вывод был следующим:


```
user@dk4n31:~$ ./lab7-1 Сообщение № 3 Сообщение № 2 Сообщение № 1
user@dk4n31:~$
```

(рис. 2.5)(рис. 2.6)



```
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label3

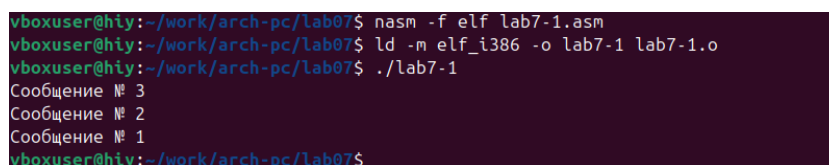
_label1:
mov eax, msg1 ; Вывод на экран строки
call printf ; 'Сообщение № 1'
jmp _end

_label2:
mov eax, msg2 ; Вывод на экран строки
call printf ; 'Сообщение № 2'
jmp _label1

_label3:
mov eax, msg3 ; Вывод на экран строки
call printf ; 'Сообщение № 3'
jmp _label2

_end:
call quit ; вызов подпрограммы завершения
```

Рис. 2.5: код программы в файле lab7-1.asm.



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

Рис. 2.6: работа программы в файле 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: А,В и С. Значения для А и С задаются в программе, значение В вводится с клавиатуры.

Создала файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. Внимательно изучила текст программы из листинга 7.3 и ввела в lab7-2.asm (рис. 2.7) Создала исполняемый файл и проверила его работу для разных значений В. (рис. 2.8)

```

%include "in_out.asm"
section .data
msg1 db "Введите В: ",0h
msg2 db "Наибольшее число: ",0h
A dd '20'
C dd '50'
section .bss
max resb 10
B resb 10
section .text
global _start
_start:
; ----- Вывод сообщения 'Введите В: '
mov eax,msg1
call sprint
; ----- Ввод 'В'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'В' из символа в число
mov eax,B
call atoi ; Вывод подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'В'
; ----- Записываем 'А' в переменную 'max'
mov ecx,[A] ; ecx = A
mov [max],ecx ; 'max' = A
; ----- Сравниваем 'А' и 'С' (как символы)
спр ecx,[C] ; Сравниваем 'А' и 'С'
jg check_B ; если 'А>С', то переход на метку 'check_B',
mov ecx,[C] ; иначе 'ecx' = C
mov [max],ecx ; 'max' = C
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,max
call atoi ; Вывод подпрограммы перевода символа в число
mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'В' (как числа)
mov ecx,[max]
спр ecx,[B] ; Сравниваем 'max(A,C)' и 'В'

```

Рис. 2.7: код программы в файле lab7-2.asm.

```

Сообщение № 1
vboxuser@hiy:~/work/arch-pc/lab07$ touch lab7-2.asm
vboxuser@hiy:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
vboxuser@hiy:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
vboxuser@hiy:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
vboxuser@hiy:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 30
Наибольшее число: 50
vboxuser@hiy:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 90
Наибольшее число: 90
vboxuser@hiy:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 3
Наибольшее число: 50
vboxuser@hiy:~/work/arch-pc/lab07$

```

Рис. 2.8: работа программы в файле lab7-2.asm.

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

командной строке. Я создала файл листинга для программы из файла lab7-2.asm и открыла его с помощью текстового редактора (рис. 2.9)

Рис. 2.9: файл листинга lab7-2

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

1) строка 192:

17 - номер строки в подпрограмме 000000F2 - адрес B9[0A000000] - машинный код mov esx,B - код программы, копирует значения из операнда B в регистр esx.

2) строка 168

167 - номер строки в подпрограмме 000000DB - адрес B9[0A000000] - машинный код mov ebx, 0 - код программы, После выполнения команды все 32 бита регистра ebx становятся равны 0

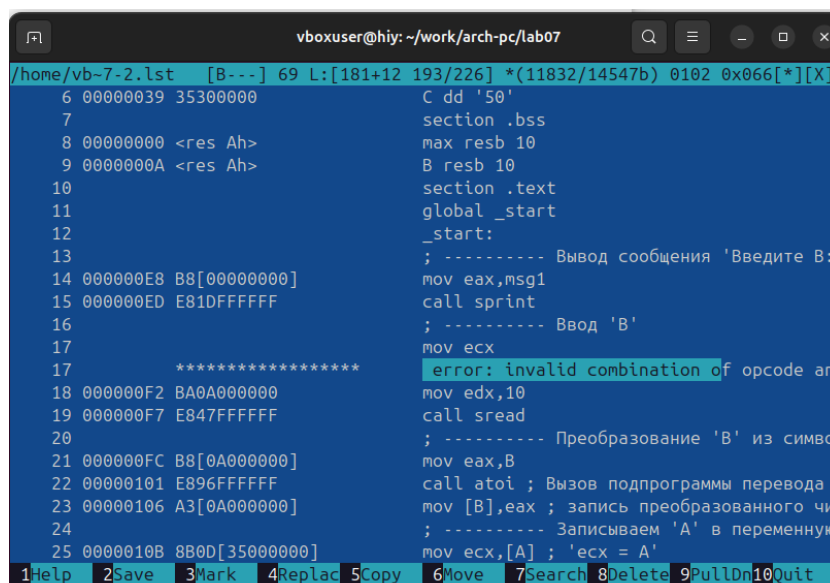
3) строка 193

18 - номер строки в подпрограмме 000000F7 - адрес BA0A000000 - машинный код mov edx,10 - код программы, инструкция записывает значение 10 (в десятичной системе) в регистр edx

Открыла файл с программой lab7-2.asm и в любой инструкции с двумя операндами удалила один операнд. Выполнила трансляцию с получением файла листинга (рис. 2.10) (рис. 2.11)

```
vboxuser@hiy:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:17: error: invalid combination of opcode and operands
vboxuser@hiy:~/work/arch-pc/lab07$
```

Рис. 2.10: ошибка трансляции lab7-2



```

/home/vb-7-2.lst [B---] 69 L:[181+12 193/226] *(11832/14547b) 0102 0x066[*][X]
6 00000039 35300000          C dd '50'
7                                section .bss
8 00000000 <res Ah>         max resb 10
9 0000000A <res Ah>         B resb 10
10                               section .text
11                               global _start
12                               _start:
13                               ; ----- Вывод сообщения 'Введите B:
14 000000E8 B8[00000000]     mov eax,msg1
15 000000ED E81DFFFFFF       call sprint
16                               ; ----- Ввод 'B'
17                               mov ecx
17                               ***** error: invalid combination of opcode and
18 000000F2 BA0A000000       mov edx,10
19 000000F7 E847FFFFFF       call sread
20                               ; ----- Преобразование 'B' из симво
21 000000FC B8[0A000000]     mov eax,B
22 00000101 E896FFFFFF       call atoi ; Вызов подпрограммы перевода
23 00000106 A3[0A000000]     mov [B],eax ; запись преобразованного чи
24                               ; ----- Записываем 'A' в переменную
25 0000010B 8B0D[35000000]   mov ecx,[A] ; 'ecx = A'
1Help 2Save 3Mark 4Replac 5Copy 6Move 7Search 8Delete 9PullDn10Quit
```

Рис. 2.11: файл листинга lab7-2 с ошибкой

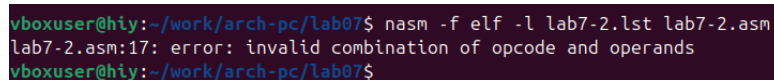
Объектный файл не создан из-за ошибки.Получился листинг, где выделено место ошибки

3 Задание для самостоятельной работы

- 1) Написать программу нахождения наименьшей из 3 целочисленных переменных a,b,c. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создать исполняемый файл и проверить его работу.

Для варианта 5: 54,62,87

Для выполнения задания я создала файл task7-1.asm, написала код (рис. 3.1) и создала исполняемый файл. Проверила правильность работы программы (рис. 3.2)



```
vboxuser@hiy:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:17: error: invalid combination of opcode and operands
vboxuser@hiy:~/work/arch-pc/lab07$
```

Рис. 3.1: код программы task7-1

```

/home/vb-7-2.lst [B---] 69 L:[181+12 193/226] *(11832/14547b) 0102 0x066[*][X]
6 00000039 35300000 C dd '50'
7 section .bss
8 00000000 <res Ah> max resb 10
9 0000000A <res Ah> B resb 10
10 section .text
11 global _start
12 _start:
13 ; ----- Вывод сообщения 'Введите B:
14 000000E8 B8[00000000] mov eax,msg1
15 000000ED E81DFFFFFF call sprint
16 ; ----- Ввод 'B'
17 mov ecx
error: invalid combination of opcode and operand
18 000000F2 BA0A000000 mov edx,10
19 000000F7 E847FFFFFF call sread
20 ; ----- Преобразование 'B' из символа
21 000000FC B8[0A000000] mov eax,B
22 00000101 E896FFFFFF call atoi ; Вызов подпрограммы перевода
23 00000106 A3[0A000000] mov [B],eax ; запись преобразованного числа
24 ; ----- Записываем 'A' в переменную
25 0000010B 8B0D[35000000] mov ecx,[A] ; 'ecx = A'
1Help 2Save 3Mark 4Replac 5Copy 6Move 7Search 8Delete 9PullDn10Quit

```

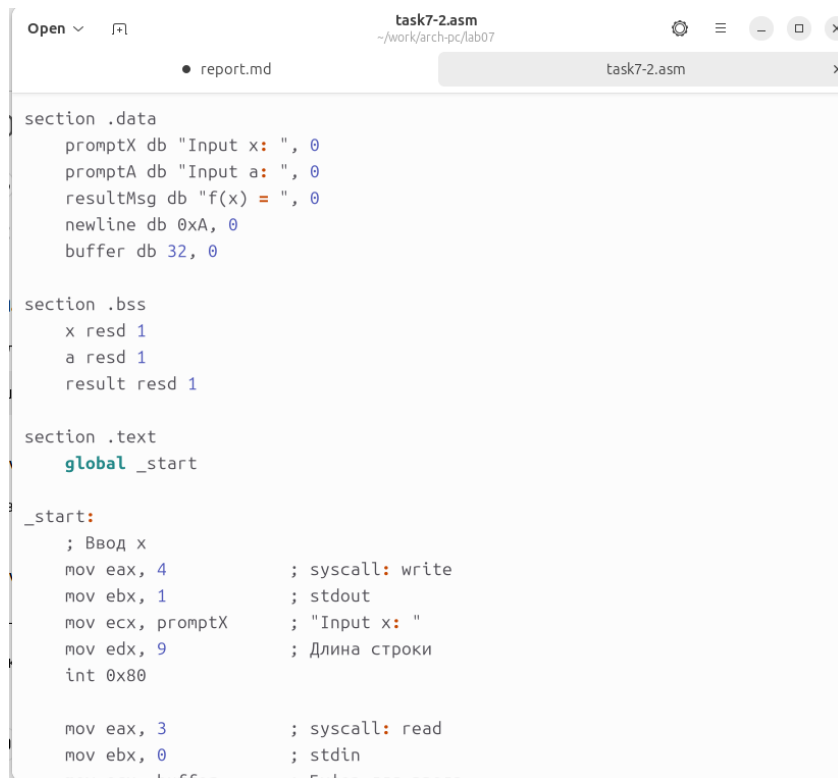
Рис. 3.2: работа программы task7-1

2) Написать 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. Создать исполняемый файл и проверить его работу для значений x и a из 7.6.

Для варианта 5: $f(x) = 2(x - a)$, $x > 15$, $x \leq 15$

Если подставить $x=1$ $a=2$ получается 15 Если подставить $x=2$ $a=1$ получается $2(2-1)=2$

Для выполнения задания я создала task7-2 и ввела код. (рис. 3.3) Создала исполняемый файл и проверила его работу (рис. 3.4)



```
section .data
    promptX db "Input x: ", 0
    promptA db "Input a: ", 0
    resultMsg db "f(x) = ", 0
    newline db 0xA, 0
    buffer db 32, 0

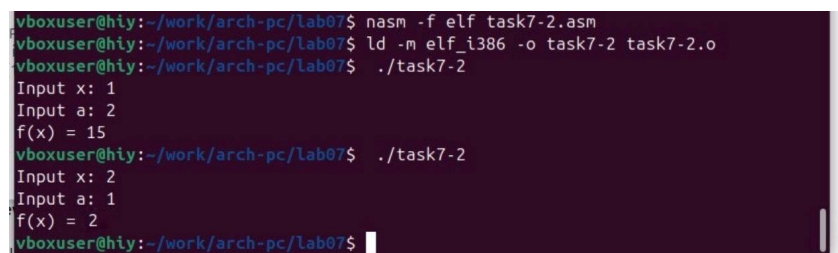
section .bss
    x resd 1
    a resd 1
    result resd 1

section .text
    global _start

_start:
    ; Ввод x
    mov eax, 4          ; syscall: write
    mov ebx, 1          ; stdout
    mov ecx, promptX    ; "Input x: "
    mov edx, 9          ; Длина строки
    int 0x80

    mov eax, 3          ; syscall: read
    mov ebx, 0          ; stdin
    mov ecx, buffer     ; Buffer для чтения
```

Рис. 3.3: код программы task7-2



```
vboxuser@hiy:~/work/arch-pc/lab07$ nasm -f elf task7-2.asm
vboxuser@hiy:~/work/arch-pc/lab07$ ld -m elf_i386 -o task7-2 task7-2.o
vboxuser@hiy:~/work/arch-pc/lab07$ ./task7-2
Input x: 1
Input a: 2
f(x) = 15
vboxuser@hiy:~/work/arch-pc/lab07$ ./task7-2
Input x: 2
Input a: 1
f(x) = 2
vboxuser@hiy:~/work/arch-pc/lab07$
```

Рис. 3.4: работа программы task7-2

4 Выводы

Я изучила команды условного и безусловного переходов, а также приобрела навыки написания программ с использованием переходов. Познакомилась с назначением и структурой файла листинга