

# **Отчёт по лабораторной работе №7**

**дисциплина: Архитектура компьютера**

Аносов Даниил Игоревич

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
3.1	Реализация переходов в NASM . . . . .	7
3.2	Изучение структуры файлы листинга . . . . .	12
<b>4</b>	<b>Задание для самостоятельной работы</b>	<b>14</b>
4.1	Нахождение наименьшего из трёх чисел . . . . .	14
4.1.1	Введение . . . . .	14
4.1.2	Алгоритм работы программы. Тестирование . . . . .	15
4.1.3	Исходный код программы . . . . .	16
4.2	Вычисление значения функции . . . . .	19
4.2.1	Введение . . . . .	19
4.2.2	Алгоритм работы программы. Тестирование . . . . .	19
4.2.3	Исходный код программы . . . . .	21
<b>5</b>	<b>Выводы</b>	<b>24</b>

# Список иллюстраций

3.1	Создание каталога для программ . . . . .	7
3.2	Открытый Vim . . . . .	7
3.3	Компиляция и первый запуск программы . . . . .	8
3.4	Vim с обновленной программой . . . . .	9
3.5	Повторная компиляция и запуск программы . . . . .	9
3.6	Vim с обновленной программой . . . . .	10
3.7	Компиляция и запуск новой программы . . . . .	10
3.8	Редактирование файла <i>lab7-2.asm</i> . . . . .	11
3.9	Повторная компиляция и запуск новой программы . . . . .	12
3.10	Создание файла листинга для <i>lab7-2.asm</i> . . . . .	12
3.11	Содержимое файла листинга . . . . .	13
4.1	Копирование файла <i>lab7-2.asm</i> . . . . .	14
4.2	Редактирование файла <i>task1.asm</i> . . . . .	15
4.3	Компиляция и запуск <i>task1.asm</i> . . . . .	16
4.4	Создание файла <i>task2.asm</i> . . . . .	19
4.5	<b>Vim</b> с файлом <i>task2.asm</i> . . . . .	20
4.6	Тестирование программы <i>task2.asm</i> . . . . .	21

## **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. 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.
2. 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 № 6. Создайте исполняемый файл и проверьте его работу для значений  $x$  и  $a$  из 7.6.

## 3 Выполнение лабораторной работы

### 3.1 Реализация переходов в NASM

Откроем терминал и создадим каталог для программ лабораторной работы №7. В новом каталоге создадим файл для первой программы *lab7-1.asm*. (рис. 3.1).

```
[dianosov@arch arch-pc]$ mkdir lab07 && cd lab07
[dianosov@arch lab07]$ touch lab7-1.asm
[dianosov@arch lab07]$
```

Рис. 3.1: Создание каталога для программ

Введём в этот файл текст программы из предложенного листинга. (рис. 3.2).

```
%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. 3.2: Открытый Vim

Скомпилируем и запустим программу, предварительно скопировав из каталога предыдущей лабораторной работы вспомогательный файл с подпрограммами *in\_out.asm* (рис. 3.3).

```
[dianosov@arch lab07]$ cp ../lab06/in_out.asm .  
[dianosov@arch lab07]$ nasm -f elf lab7-1.asm  
[dianosov@arch lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o  
[dianosov@arch lab07]$ ./lab7-1  
Сообщение № 2  
Сообщение № 3  
[dianosov@arch lab07]$
```

Рис. 3.3: Компиляция и первый запуск программы

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. 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`).

Изменим текст программы в соответствии с листингом 7.2 (рис. 3.4).



```

#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'
_end:
call quit ; вызов подпрограммы завершения
~
~

```

Рис. 3.4: Vim с обновленной программой

Скомпилируем и запустим программу. (рис. 3.5).

```

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

```

Рис. 3.5: Повторная компиляция и запуск программы

Теперь изменим программу так, чтобы сообщения выводились в порядке 3,2,1. (рис. 3.6).

```

#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 _label3
_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'
jmp _label2
_end:
call quit ; вызов подпрограммы завершения
~

```

Рис. 3.6: Vim с обновленной программой

Сохраним, скомпилируем и запустим новую программу. (рис. 3.7).

```

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

```

Рис. 3.7: Компиляция и запуск новой программы

Как видно, программа работает корректно.

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.

Текст новой программы, взятый из листинга, введем в новый файл *lab7-2.asm*

```
%include 'in_out.asm'
section .data
msg1 db 'Введите B: ',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 B: '
mov eax,msg1
call sprint
; ----- Ввод 'B'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование '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 'max'
mov ecx,[A] ; 'ecx = A'
mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', 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 ; Вызов подпрограммы перевода символа в число
mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [max],ecx
; ----- Вывод результата
fin:
mov eax, msg2
-- INSERT --
```

2,12

Рис. 3.8: Редактирование файла *lab7-2.asm*

Создадим исполняемый файл и проверим работу программы. (рис. 3.9).

```
[dianosov@arch lab07]$ nasm -f elf lab7-2.asm
[dianosov@arch lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[dianosov@arch lab07]$ ./lab7-2
Введите B: 34
Наибольшее число: 50
[dianosov@arch lab07]$ ./lab7-2
Введите B: 100
Наибольшее число: 100
[dianosov@arch lab07]$ ./lab7-2
Введите B: 0
Наибольшее число: 50
[dianosov@arch lab07]$
```

Рис. 3.9: Повторная компиляция и запуск новой программы

Обратим внимание, в данном примере переменные A и C сравниваются как символы, а переменная B и максимум из A и C как числа (для этого используется функция `atoi` преобразования символа в число). Это сделано для демонстрации того, как сравниваются данные. Данную программу можно упростить и сравнивать все 3 переменные как символы (т.е. не использовать функцию `atoi`). Однако если переменные преобразовать из символов числа, над ними можно корректно проводить арифметические операции.

## 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. Создадим файл листинга для программы из файла `lab7-2.asm`. (рис. 3.10). Будем использовать команду:

```
nasm -f elf -l lab7-2.lst lab7-2.asm
```

```
[dianosov@arch lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
[dianosov@arch lab07]$ ls
in_out.asm lab7-1 lab7-1.asm lab7-1.o lab7-2.asm lab7-2.lst lab7-2.o
[dianosov@arch lab07]$
```

Рис. 3.10: Создание файла листинга для `lab7-2.asm`

Откроем файл листинга в редакторе **Vim** и обратим внимание на строки 45-47. (рис. 3.11).

45	00000159	B8[13000000]	mov <b>eax</b> , msg2
46	0000015E	E8ACFEFFFF	call sprint ; Вывод сообщения 'Наибольшее числ
47	00000163	A1[00000000]	mov <b>eax</b> , [max]

Машинные коды в строках 45 и 47 имеют в конце значения в квадратных скобках, так как они соответствуют вызовам инструкций с двумя операндами. Строка 46 же отвечает вызову функции sprint, поэтому её машинный код (E8ACFEFFFF) - это единая строка. Итого, машинный код, отвечающий инструкции с одним операндом - это слитная строка, а если операндов больше, чем один - то машинный код будет содержать часть в квадратных скобках.

```

 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 000000F2 B9[0A000000]      mov ecx,B
18 000000F7 BA0A000000        mov edx,10
19 000000FC E842FFFFFF        call sread
20                                ; ----- Преобразование 'B' из символа в число
21 00000101 B8[0A000000]      mov eax,B
22 00000106 E891FFFFFF        call atoi ; Вызов подпрограммы перевода символа в число
23 0000010B A3[0A000000]      mov [B],eax ; запись преобразованного числа в 'B'
24                                ; ----- Записываем 'A' в переменную 'max'
25 00000110 8B0D[35000000]     mov ecx,[A] ; 'ecx = A'
26 00000116 890D[00000000]     mov [max],ecx ; 'max = A'
27                                ; ----- Сравниваем 'A' и 'C' (как символы)
28 0000011C 3B0D[39000000]     cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 00000122 7F0C               jg check_B ; если 'A>C', то переход на метку 'check_B',
30 00000124 8B0D[39000000]     mov ecx,[C] ; иначе 'ecx = C'
31 0000012A 890D[00000000]     mov [max],ecx ; 'max = C'
32                                ; ----- Преобразование 'max(A,C)' из символа в число
33                                check_B:
34 00000130 B8[00000000]      mov eax,max
35 00000135 E862FFFFFF        call atoi ; Вызов подпрограммы перевода символа в число
36 0000013A A3[00000000]      mov [max],eax ; запись преобразованного числа в 'max'
37                                ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
38 0000013F 8B0D[00000000]     mov ecx,[max]
39 00000145 3B0D[0A000000]     cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
40 0000014B 7F0C               jg fin ; если 'max(A,C)>B', то переход на 'fin',
41 0000014D 8B0D[0A000000]     mov ecx,[B] ; иначе 'ecx = B'
42 00000153 890D[00000000]     mov [max],ecx
43                                ; ----- Вывод результата
44                                fin:
45 00000159 B8[13000000]      mov eax, msg2
46 0000015E E8ACFEFFFF        call sprint ; Вывод сообщения 'Наибольшее число: '
47 00000163 A1[00000000]      mov eax,[max]
48 00000168 E819FFFFFF        call iprintLF ; Вывод 'max(A,B,C)'
49 0000016D E869FFFFFF        call quit ; Выход
~
~
~
-- INSERT --
216,75-70 Bot

```

Рис. 3.11: Содержимое файла листинга

## 4 Задание для самостоятельной работы

### 4.1 Нахождение наименьшего из трёх чисел

#### 4.1.1 Введение

Напишем программу, которая находит наименьшее из трёх введённых пользователем чисел. Для этого воспользуемся кодом программы из предыдущих примеров, и изменим его должным образом. Скопируем файл *lab7-2.asm* и дадим новой программе имя *task1.asm*. (рис. 4.1, 4.2).

```
[dianosov@arch lab07]$ ls
in_out.asm lab7-1 lab7-1.asm lab7-1.o lab7-2 lab7-2.asm lab7-2.lst lab7-2.o
[dianosov@arch lab07]$ cp lab7-2.asm task1.asm
[dianosov@arch lab07]$ ls
in_out.asm lab7-1 lab7-1.asm lab7-1.o lab7-2 lab7-2.asm lab7-2.lst lab7-2.o task1.asm
[dianosov@arch lab07]$
```

Рис. 4.1: Копирование файла *lab7-2.asm*

```

call atoi
mov [C],eax
; ----- Записываем A+B+C в переменную 'min'
; сумма значений точно больше каждого из них
mov ecx,[A]
add ecx,[B]
add ecx,[C]
mov [min],ecx

jmp cmp_a_b ; к сравнению AvB

min_below_c:
jmp fin

c_below_min:
mov eax,[C]
mov [min],eax
jmp fin

a_below_b:
mov eax,[A]
mov [min],eax ; min(A, B) = A
jmp cmp_min_c ; к сравнению min v C

b_below_a:
mov eax,[B]
mov [min],eax ; min(A, B) = B
jmp cmp_min_c ; к сравнению min v C

cmp_min_c: ; min(A, B) v C
mov eax,[C]
cmp [min],eax
jb min_below_c ; min(A, B) < C
jg c_below_min ; min(A, B) > C

cmp_a_b: ; AvB
mov eax,[B]
cmp [A],eax
jb a_below_b ; A < B
jg b_below_a ; A > B

; ----- Вывод результата
fin:
mov eax,msg1
call sprintf ; Вывод сообщения 'Наименьшее число: '

```

77,8

Рис. 4.2: Редактирование файла *task1.asm*

### 4.1.2 Алгоритм работы программы. Тестирование

Алгоритм работы программы: сравниваем A и B, минимальное из них кладём в переменную min. Сравниваем min с C. Минимальное из них - минимальное из всех чисел - результат работы программы. Пример на рис. 4.3. Скомпилируем и запустим новую программу.

```
[dianosov@arch lab07]$ nasm -f elf task1.asm
[dianosov@arch lab07]$ ld -m elf_i386 -o task1 task1.o
[dianosov@arch lab07]$ ./task1
Введите A: 79
Введите B: 83
Введите C: 41
Наименьшее число: 41
[dianosov@arch lab07]$
```

Рис. 4.3: Компиляция и запуск *task1.asm*

Тестовый запуск был проведён со значениями 79, 83, 41, соответствующими варианту №6, полученному в предыдущей лабораторной работе.

### 4.1.3 Исходный код программы

Исходный текст *task1.asm* приведён ниже.

```
%include 'in_out.asm'

section .data

msga db 'Введите A: ',0h
msgb db 'Введите B: ',0h
msgc db 'Введите C: ',0h
msg1 db "Наименьшее число: ",0h

section .bss

min resb 10
A resb 10 ; резервируем место не
B resb 10 ; только под B,
C resb 10 ; но и под A и C

section .text
global _start
_start:
; ----- Ввод A, B, C

mov eax,msga
call sprint
```



```

mov ecx,A
mov edx, 10
call sread
mov eax,msgb
call sprint
mov ecx,B
call sread
mov eax,msgc
call sprint
mov ecx,C
call sread
; ----- Преобразование A,B,C из символов в числа
mov eax,A
call atoi ; Вызов подпрограммы перевода символа в число
mov [A],eax ; запись преобразованного числа в 'A'
mov eax,B
call atoi
mov [B],eax
mov eax,C
call atoi
mov [C],eax
; ----- Записываем A+B+C в переменную 'min'
; сумма значений точно больше каждого из них
mov ecx,[A]
add ecx,[B]
add ecx,[C]
mov [min],ecx

jmp cmp_a_b ; к сравнению A и B

```

min\_below\_c:

jmp fin

c\_below\_min:

mov eax, [C]

mov [min], eax

jmp fin

a\_below\_b:

mov eax, [A]

mov [min], eax ;  $\min(A, B) = A$

jmp cmp\_min\_c ; к сравнению min и C

b\_below\_a:

mov eax, [B]

mov [min], eax ;  $\min(A, B) = B$

jmp cmp\_min\_c ; к сравнению min и C

cmp\_min\_c: ;  $\min(A, B) \neq C$

mov eax, [C]

cmp [min], eax

jb min\_below\_c ;  $\min(A, B) < C$

jg c\_below\_min ;  $\min(A, B) > C$

cmp\_a\_b: ;  $A \neq B$

mov eax, [B]

cmp [A], eax

jb a\_below\_b ;  $A < B$

```

jg b_below_a      ; A>B

; ----- 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

```

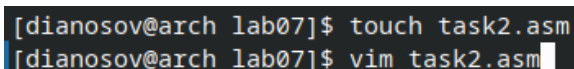
## 4.2 Вычисление значения функции

### 4.2.1 Введение

Требуется написать программу, вычисляющую значение функции в зависимости от значений  $x, a$ . Я получил вариант №6 в процессе выполнения заданий предыдущей лабораторной работы. Этому варианту соответствует

следующая функция: 
$$f(x) = \begin{cases} x + a, & x = a \\ 5x, & x \neq a \end{cases} \quad (x_1, a_1) = (2, 2), (x_2, a_2) = (2, 1)$$

Для написания программы создадим файл *task2.asm*, откроем его в редакторе **Vim**. (рис. 4.4).



```

[dianosov@arch lab07]$ touch task2.asm
[dianosov@arch lab07]$ vim task2.asm

```

Рис. 4.4: Создание файла *task2.asm*

### 4.2.2 Алгоритм работы программы. Тестирование

Программа работает по следующей логике: Сравниваем значения  $a$  и  $x$ . Если  $a = x$ , то переходим к метке *a\_equals\_x*. В этой части программы приравниваем результат к  $x$  и прибавляем  $a$  к нему, а затем переходим к метке завершения *fin*.

Иначе, если значения не равны, переходим к метке `a_nequals_x`: приравниваем результат к `x`, умножаем его на 5 и переходим к завершению (`fin`). Под меткой `fin` выводим сообщение с результатом и вызываем сигнал прерывания.

Фрагмент кода приведён на рис. 4.5.

```
msgx db 'Введите x: ',0h
msga db 'Введите a: ',0h
msg1 db 'Результат: ',0h

section .bss
x resb 10 ; резервируем место под x
a resb 10 ; под a
res resb 10 ; и под результат

section .text
global _start
_start:
; ----- Ввод значений x, a
mov eax,msgx
call sprint
mov ecx,x
mov edx,10
call sread
mov eax,msga
call sprint
mov ecx,a
call sread
; ----- Преобразование x, a в числа
mov eax,x
call atoi ; вызов подпрограммы перевода символа в число
mov [x],eax ; запись преобразованного числа в 'x'
mov eax,a
call atoi
mov [a],eax
; ----- Логическая часть программы
mov eax,[a]
mov ebx,[x]
cmp ebx,eax ; сравнение a и x
je a_equals_x
jne a_nequals_x
a_equals_x:
mov ecx,[x]
mov [res],ecx ; ecx := x
mov ecx,[a]
add [res],ecx ; res := res + a = x + a
jmp fin
a_nequals_x:
mov ecx,[x]
```

Рис. 4.5: **Vim** с файлом *task2.asm*

Проверим работу программы для входных данных  $(x_1, a_1) = (2, 2)$ ,  $(x_2, a_2) = (2, 1)$  (рис. 4.6).

```

[dianosov@arch lab07]$ nasm -f elf task2.asm
[dianosov@arch lab07]$ ld -m elf_i386 -o task2 task2.o
[dianosov@arch lab07]$ ./task2
Введите x: 2
Введите a: 2
Результат: 4
[dianosov@arch lab07]$ ./task2
Введите x: 2
Введите a: 1
Результат: 10
[dianosov@arch lab07]$

```

Рис. 4.6: Тестирование программы *task2.asm*

Как видим, программа работает корректно.

### 4.2.3 Исходный код программы

Исходный код программы *task2.asm* приведён ниже.

```

%include 'in_out.asm'

section .data
msgx db 'Введите x: ',0h
msga db 'Введите a: ',0h
msg1 db "Результат: ",0h

section .bss
x resb 10 ; резервируем место под x
a resb 10 ; под a
res resb 10 ; и под результат

section .text
global _start
_start:
; ----- Ввод значений x, a
mov eax,msgx
call sprint
mov ecx,x

```

```

mov edx,10
call sread
mov eax,msga
call sprint
mov ecx,a
call sread
; ----- Преобразование x, a в числа
mov eax,x
call atoi ; вызов подпрограммы перевода символа в число
mov [x],eax ; запись преобразованного числа в 'x'
mov eax,a
call atoi
mov [a],eax
; ----- Логическая часть программы
mov eax, [a]
mov ebx, [x]
cmp ebx, eax ; сравнение a и x
je a_equals_x
jne a_nequals_x

a_equals_x:
mov ecx, [x]
mov [res], ecx ; ecx := x
mov ecx, [a]
add [res], ecx ; res := res + a = x + a
jmp fin

a_nequals_x:
mov ecx, [x]

```

```

mov [res], ecx ; res := x
mov ecx, 5
mov eax, [res] ; eax := res
mul ecx        ; eax := res * 5 = 5x
mov [res], eax ; res := eax = 5x
jmp fin

fin:
mov eax, msg1
call sprint ; Вывод сообщения 'Результат: '
mov eax, [res]
call iprintLF ; Вывод res
call quit ; Выход

```

Задание выполнено, загрузим новую версию проекта курса на GitHub.

```

[dianosov@arch arch-pc]$ git add .
[dianosov@arch arch-pc]$ git commit -am "add files for lab07"
On branch master
nothing to commit, working tree clean
[dianosov@arch arch-pc]$ git push origin master
Enumerating objects: 63, done.
Counting objects: 100% (63/63), done.
Delta compression using up to 12 threads
Compressing objects: 100% (53/53), done.
Writing objects: 100% (53/53), 2.04 MiB | 3.14 MiB/s, done.
Total 53 (delta 19), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (19/19), completed with 9 local objects.
To https://github.com/exterminateddd/pc-course-2024-2025
   d2edaa7..734d02f  master -> master
[dianosov@arch arch-pc]$

```

## **5 Выводы**

В ходе выполнения лабораторной работы были изучены команды условного и безусловного переходов. Приобретены навыки написания программ с использованием переходов.