**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Организация ЭВМ и систем»**

Тема: «Представление и обработка целых чисел. Организация ветвящихся процессов»

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1304 |  | Ярусова Т. В.. |
| Преподаватель |  | Кирьянчиков В.А. |

Санкт-Петербург

2022

Цель работы.

Изучить представление и обработку целых чисел на языке Ассемблер, а также изучить организацию ветвящихся процессов.

**Задание.**

Разработать на языке Ассемблера программу, которая по заданным целочисленным значениям параметров a, b, i, k вычисляет:

а) значения функций i1 = f1(a,b,i) и i2 = f2(a,b,i);

b) значения результирующей функции res = f3(i1,i2,k)

Значения a, b, i, k являются исходными данными, которые должны выбираться студентом самостоятельно и задаваться в процессе исполнения программы в режиме отладки. При этом следует рассмотреть всевозможные комбинации параметров a, b и k, позволяющие проверить различные маршруты выполнения программы, а также различные знаки параметров a и b.

**Замечания:**

1. при разработке программы нельзя использовать фрагменты, представленные на ЯВУ, в частности, для ввода-вывода данных. Исходные данные должны вводиться, а результаты контролироваться в режиме отладки;
2. при вычислении функций f1 и f2 вместо операции умножения следует использовать арифметический сдвиг и, возможно, сложение;
3. при вычислении функций f1 и f2 нельзя использовать процедуры;
4. при разработке программы следует минимизировать длину кода, для чего, если надо, следует преобразовать исходные выражения для вычисления функций.

Ход работы.

**Вариант 4. Шифр студента 1.5.4**

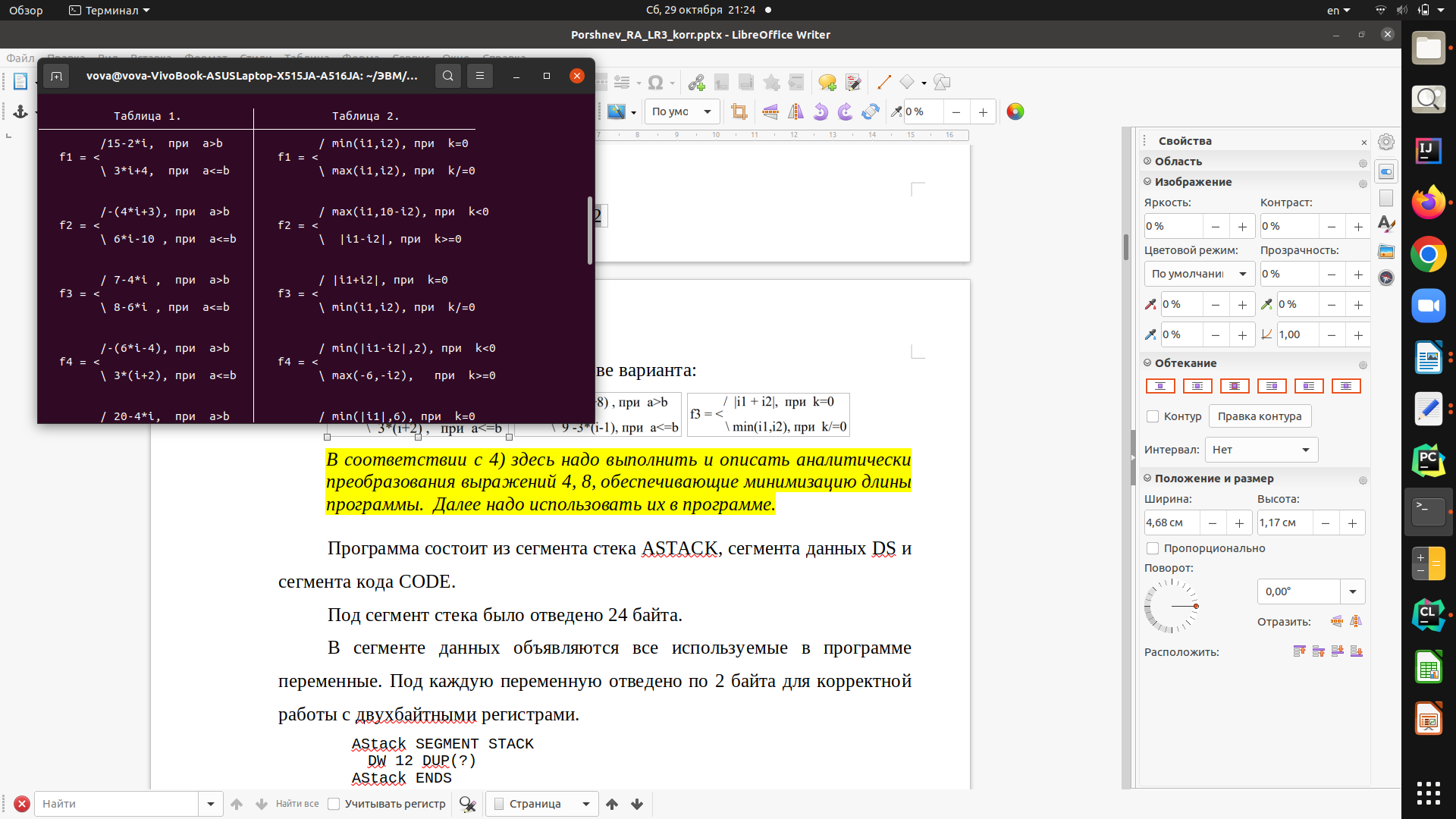
**Функции, соответствующие варианту, представлены на рисунке 1, 2, 3.**

Рисунок 1 – Функция f1(a, b, i)

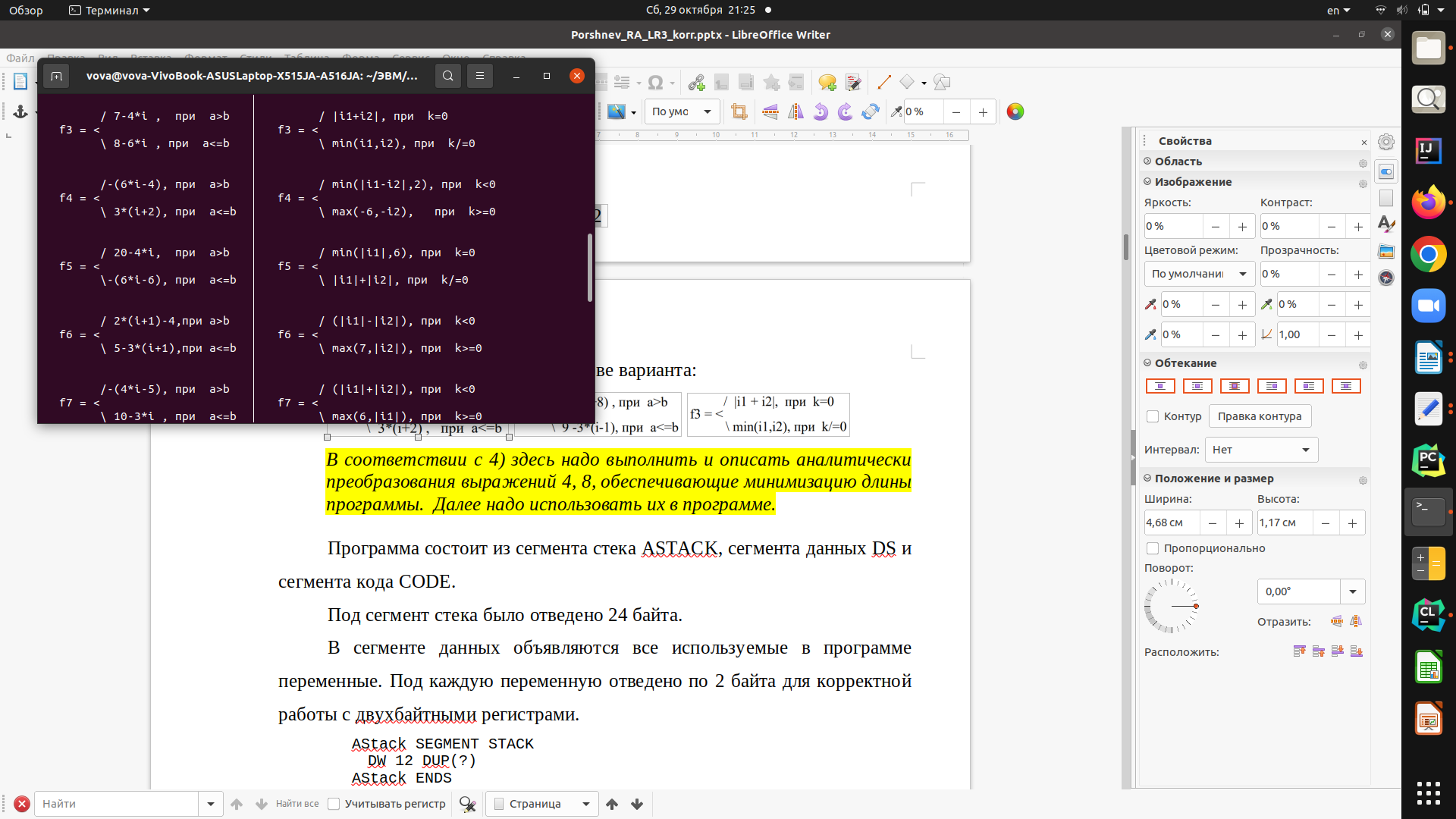


Рисунок 2 – Функция f2(a, b, i)

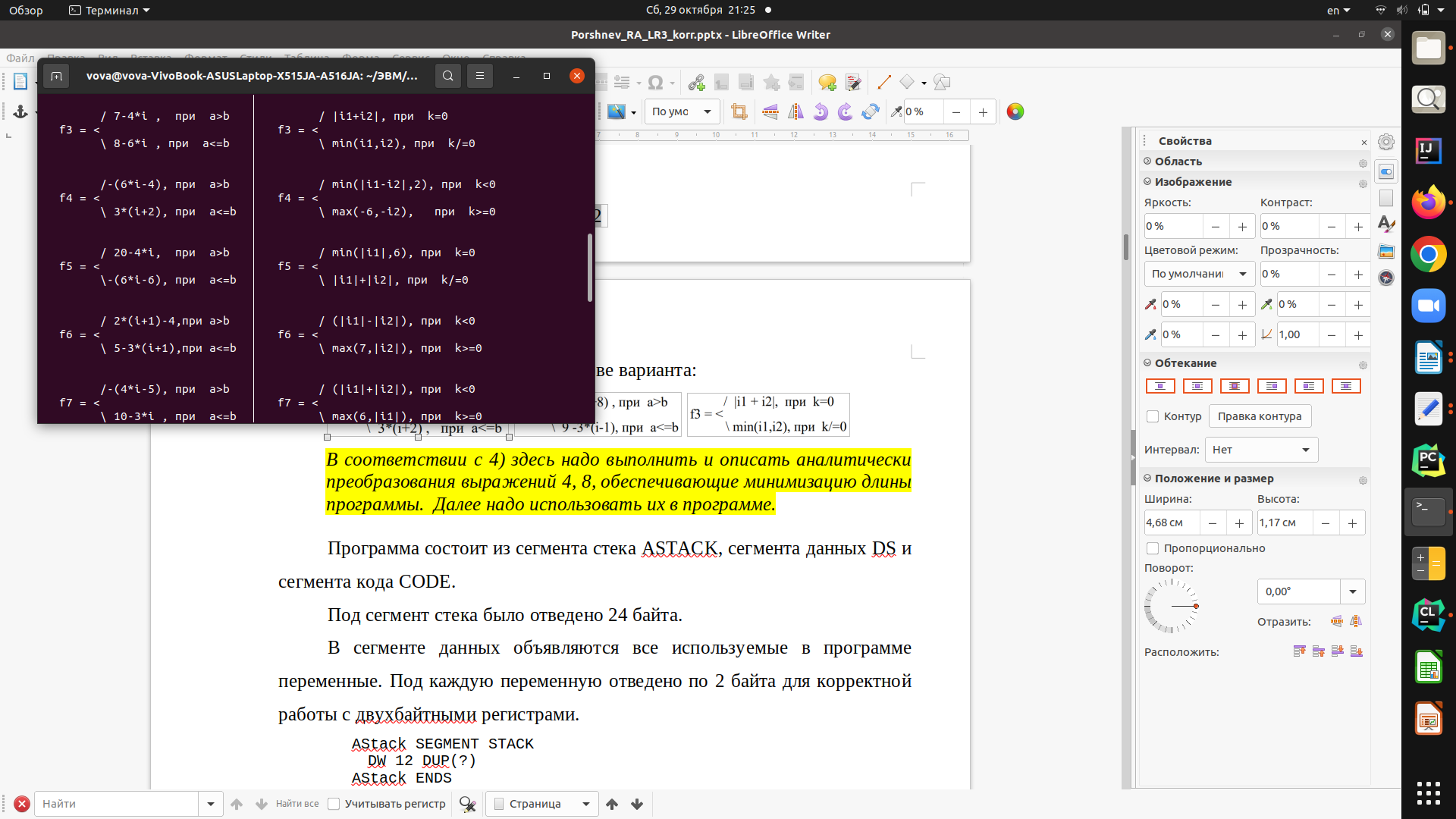


Рисунок 3 – Функция f3(i1, i2, k)

# Программа состоит из сегмента данных(DATA SEGMENT), сегмента стека(ASTACK) и сегмента кода(CODE).

# Под сегмент стека отведено 24 байта.

# В сегменте стека объявлены все необходимые 2-ух байтовые слова.

# AStack SEGMENT STACK

# DW 12 DUP(?)

# AStack ENDS

# DATA SEGMENT

# a DW 0

# b DW 0

# i DW 0

# k DW 0

# 

# i1 DW 0

# i2 DW 0

# res DW 0

# DATA ENDS

# CODE SEGMENT

# ASSUME CS:CODE, DS:DATA, SS:AStack

# После объявления главной процедуры *Main* происходит сохранение адреса начала *PSP* в стек и сохранение адреса той команды *(ip = 0000),* которая будет выполнена после команды *ret*, то есть, *int 20h.* Далее в регистр *ds* загружается адрес начала сегмента данных.

# push ds

# sub ax, ax

# push ax

# mov ax, DATA

# mov ds, ax

# В основной части программы в регистрах bx, *cx, dx* хранится -*i, -2i, -4i* соответственно, в регистре *ax* будет храниться *a*. Происходит сравнение регистра *ax* и *b*.

# mov bx, i

# neg bx ; -i

# mov cx, bx

# sal cx, 1 ; -2\*i

# mov dx, cx

# add dx, dx ; -4\*i

# 

# mov ax, a

# CMP ax, b

# jle a\_less

# Если данные, хранящиеся в регистре *ax* меньше или равны *b*, то с помощью команды *jle* (меньше или равно) происходит переход к метке *a\_less* (для сокращения кода сначала было вычислено выражение 6 – 6 \* i, а из него далее было получено выражение 4 + 3\*i),

# a\_less:

add cx, dx ; -6i

mov ax, 6

add ax, cx ; 6-6\*i

mov i2, ax

sar ax, 1 ; 3 - 3\*i

neg ax ; 3\*i -3

add ax, 7 ;3\*i + 4

mov i1, ax

# иначе программа продолжается.

# a\_greater:

# add cx, 15

# mov i1, ax

# add dx, 20

# mov i2, dx

# jmp f\_3

# 

# В конце блока *a\_greater* с помощью команды *jmp* (безусловный переход) происходит переход к метке *f\_3*.

# В блоке *f\_3* проверяем условие k >= 0 или k < 0.

# f\_3:

# mov ax, k

# cmp ax, 0

# jge k\_greater

# Если k >= 0, то с помощью команды *jge (*больше или равно) переходим к метке *k\_greater*, в которой есть вложенное условие, в котором проверяется какое из значений больше: -6 или –i2,

# k\_greater:

mov ax, i2

neg ax

cmp ax, -6

jge max\_1; -i2 >= -6

max\_2:

mov res, -6

jmp end\_cmp

max\_1:

mov res, ax ; -i2

# иначе программа продолжается в блоке *k\_less*, в котором есть два вложенных условия. Блоки *abs\_less* и *abs\_greater* заканчиваются командой *jmp,* которая переходит к метке *end\_cmp*.

# k\_less:

# mov ax, i1

# sub ax, i2

# L:

# neg ax

# js L

# cmp ax,2

# jg abs\_greater

# 

# abs\_less:

# mov res, ax

# jmp end\_cmp

# 

# abs\_greater:

# mov res, 2

# jmp end\_cmp

# 

# Метка *end\_cmp* завршает программу.

# 

# end\_cmp:

# ret

# Разработанный код см. Приложение

# Тестирование.

Результаты тестирования представлены в табл. 1. Тестирование проводилось в отладчике AFD.

Таблица 1 - Тестирование

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | a = 7  b = 1  i = 9  k = 5 | i1 = FFFD16 = (-3)10;  i2 = FFF016 = (-16)10;  res = 001016 = 1610; | Проверяется случай a > b,  K > 0,  - i2 > - 6 |
|  | a = 5  b = -3  i = -2  k = 1 | i1 = 001316 = 1910;  i2 = 001C16 = 2810;  res = FFFA16 = (-6)10; | Проверяется случай a > b,  K > 0,  - i2 < - 6 |
|  | a = -3  b = 7  i = 0  k = -3 | i1 = 000416 = 410;  i2 = 000616 = 610;  res = 000216 = 210; | Проверяется случай a < b,  K < 0,  |i1- i2| > 2 |
|  | a = -3  b = 5  i = 3  k = -1 | i1 = 000D16 = 1310;  i2 = FFF416 = (-12)10;  res = 000216 = 210; | Проверяется случай a < b,  K < 0,  |i1- i2| < 2. |
|  | a = 5  b = 5  i =0  k = 0 | i1 = 000416 = 410;  i2 = 000616 = 610;  res = FFFA16 = (-6)10; | Проверяются граничные случаи k=0,a=b. |

**Вывод**

В ходе данной лабораторной работы была изучена организация ветвящихся процессов на языке Ассемблер.

**ПРИЛОЖЕНИЕ**

;Стек программы

AStack SEGMENT STACK

DW 12 DUP(?)

AStack ENDS

;Сегмент Данных

DATA SEGMENT

a DW 0

b DW 0

i DW 0

k DW 0

i1 DW 0

i2 DW 0

res DW 0

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA, SS:AStack

Main PROC FAR

push ds

sub ax, ax

push ax

mov ax, DATA

mov ds, ax

mov bx, i

neg bx ; -i

mov cx, bx

sal cx, 1 ; -2\*i

mov dx, cx

add dx, dx ; -4\*i

mov ax, a

mov bx, b

CMP ax, bx

jle a\_less ; Если а <= б переходим по метке

a\_greater: ; Опишем ситуацию, когда a > b

mov ax, 15

add ax, cx ; 15 - 2\*i

mov i1, ax ; i1 = 15 - 2\*i

mov ax, 20

add ax, dx ; 20 - 4\*i

mov i2, ax

jmp f\_3 ; перейдем в функцию f3

a\_less: ; Если a <= b

add cx, dx ; -6i

mov ax, 6

add ax, cx ; 6-6\*i

mov i2, ax

sar ax, 1 ; 3 - 3\*i

neg ax ; 3\*i -3

add ax, 7 ;3\*i + 4

mov i1, ax

f\_3:

mov ax, k

cmp ax, 0

jge k\_greater ; Если k >=0

k\_less: ;Если k < 0

mov ax, i1

sub ax, i2

L:

neg ax

js L ; Если i1 - i2 < 0

cmp ax,2

jg abs\_greater ; Если abs(i1-i2) > 2

abs\_less:

mov res, ax ; abs(i1-i2)

jmp end\_cmp ; Завершаем сравнени

abs\_greater:

mov res, 2 ; 2

jmp end\_cmp ; Завершаем сравнени

k\_greater:

mov ax, i2

neg ax

cmp ax, -6

jge max\_1; -i2 >= -6

max\_2:

mov res, -6

jmp end\_cmp

max\_1:

mov res, ax ; -i2

end\_cmp:

ret

Main ENDP

CODE ENDS

END Main