МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра Математического обеспечения электронно-вычислительных машин

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Организация ЭВМ и систем»

ТЕМА: ПРЕДСТАВЛЕНИЕ И ОБРАБОТКА ЦЕЛЫХ ЧИСЕЛ. ОРГАНИЗАЦИЯ ВЕТВЯЩИХСЯ ПРОЦЕССОВ.

Вариант 13

Студентка гр. 0382 _____ Рубежова Н.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить организацию ветвящихся процессов и отработать на практике, разработав программу, вычисляющую значение функций по заданным целочисленным параметрам, на языке Ассемблер.

Задание.

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

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

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

где вид функций f1 и f2 определяется из табл. 2, а функции f3 - из табл.3 по цифрам шифра индивидуального задания (n1,n2,n3), приведенным в табл.4.

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.

Задание, соответствующее варианту 13:

$$f1 = \begin{cases} -(4*i+3), & \text{при } a>b \\ 6*i-10, & \text{при } a<=b \end{cases} \qquad f2 = \begin{cases} -(6*i+8), & \text{при } a>b \\ 9-3*(i-1), & \text{при } a<=b \end{cases}$$

$$f3 = \begin{cases} |i1 + i2|, & \text{при } k=0 \\ \min(i1,i2), & \text{при } k/=0 \end{cases}$$

Ход выполнения.

1. В сегменте данных DATA объявим двухбайтовые переменные var_a, var_b, var_i, var_k, var_i1, var_i2, var_res.
2. В сегменте кода CODE реализуем головную процедуру Main, вычисляющую искомые значения функций. Внутри процедуры будем пользоваться условными и безусловными переходами для организации ветвящихся процессов.

3. Команда условного перехода *jg case1* позволит перейти к участку кода по метке *case1* в случае, если при предшествующем сравнении первый операнд будет больше второго(*a>b*).
4. Команда условного перехода *je abs_sum* позволит перейти к участку кода по метке *abs_sum* в случае, если при предшествующем сравнении первый операнд будет равен второму(*k==0*).
5. Команда условного перехода *jng set_min_i1* позволит перейти к участку кода по метке *set_min_i1* в случае, если при предшествующем сравнении первый операнд будет не больше второго(*i1<=i2*).
6. Также в программе используется команда безусловного перехода *jmp,* для организации порядка выполнения команд.

Тестирование.

Значения переменных в тестах записывались в соответствующие ячейки памяти в процессе отладки программы.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	a=8, b=5, i=1, k=0	i1= $F9_{16}$= -7, i2= $F2_{16}$= -14, res = 15_{16}= 21	Результаты корректны
2.	a=8, b=5, i=1, k=1	i1=$F9_{16}$= -7, i2= $F2_{16}$= -14, res = $F2_{16}$= -14	Результаты корректны
3.	a=3, b=4, i=2, k=0	i1=2, i2= 6, res = 8	Результаты корректны
4.	a=3, b=4, i=2, k=1	i1=2, i2= 6, res = 2	Результаты корректны

Выводы.

В результате работы была изучена, а также отработана на практике организация ветвящихся процессов посредством разработки программы, вычисляющей значение функций по заданным целочисленным параметрам, на языке Ассемблер.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.asm

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

AStack    SEGMENT  STACK
          DW 12 DUP('?')
AStack    ENDS

DATA      SEGMENT
        var_a     DW 8
        var_b     DW 5
        var_i     DW 1
        var_k     DW 0
        var_i1     DW 0
        var_i2     DW 0
        var_res     DW 0
DATA      ENDS

CODE      SEGMENT

Main      PROC  FAR
          push  DS
          sub   AX,AX
          push  AX
          mov AX,DATA
          mov DS,AX

          mov ax, var_i ;ax=i (need for case1 and case2)
          shl ax,1    ; ax=2i (need for case1 and case2)
          mov bx,var_a
          cmp bx, var_b
          jg case1    ; jmp if a>b
case2:
        add ax, var_i ; ax=3i
        mov bx,12     ; bx=12
        sub bx,ax
        mov var_i2,bx ; i2=12-3i
        shl ax,1 ; ax=6i
        sub ax, 10
        mov var_i1,ax ; i1=6i-10
        jmp f3
case1:
        shl ax,1 ; ax=4i
        mov bx,ax     ; bx=4i
        add ax,3h     ; ax=4i+3
        neg ax
        mov var_i1,ax ; i1=-(4i+3)
        mov ax,var_i
        shl ax,1 ; ax=2i
        add ax,bx     ; ax=6i
        add ax,8h     ; ax=6i+8
        neg ax
        mov var_i2,ax ; i2=-(6i+8)
f3:
```

```
        cmp var_k,0h
        je abs_sum    ; if k==0 then jmp
        mov ax, var_i1 ; ax=i1
        cmp ax, var_i2
        jng set_min_i1 ; if i1<=i2 then jmp
        mov ax, var_i2
        jmp set_res
abs_sum:
       mov ax, var_i1 ; ax=i1
       add ax, var_i2 ; ax=i1+i2
       cmp ax,0h
       jl neg_sum
       jmp set_res
neg_sum:
       neg ax
       jmp set_res
set_min_i1:
       mov ax, var_i1
set_res:
       mov var_res, ax
          ret
Main      ENDP
CODE      ENDS
          END Main
```