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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Организация ЭВМ и систем»
Тема: Представление и обработка целых чисел. Организация ветвящихся процессов.

Студентка гр. 9383 _____ Карпекина А.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Изучить представление и обработку целых чисел на языке Ассемблер. Научиться строить программы с условными переходами.

Текст задания.

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.

Исходные данные.

Вариант 5

Таблица 1 — Исходные данные.

Имя функции	Функция
f1	15-2*i , при a>b
	3*i+4 , при a<=b
f6	2*(i+1) -4 , при a>b
	5 - 3*(i+1), при a<=b
f5	min(|i1|, 6), при k=0
	|i1|+|i2|, при k/=0

Ход работы.

Была разработана программа, которая вычисляет значение функции по заданным целочисленным параметрам. Правильность записи исходных и входных данных была проверена с помощью отладчика.

Были использованы следующие операнды:

add – для суммирования;

sub – для вычитания;

shl – для логического сдвига влево, что равнозначно умножению на два;

neg – для смены знака

Результаты записывались по заранее заданным адресам переменных i1, i2 и res.

Для реализации условных переходов были использованы операнды:

cmp – для сравнения двух чисел.

jle – условный переход, срабатывающий, если левый аргумент выражения в cmp был меньше или равен второму.

je – короткий переход, если первый операнд равен второму операнду при выполнении операции сравнения с помощью команды cmp.

jl – условный переход, срабатывающий, если левый аргумент выражения в cmp был меньше второго.

jmp – безусловный переход без дополнительных проверок.

Исходный код программы представлен в приложении А.

Примеры работы программы.

Таблица 2 — Примеры работы программы.

№	a	b	i	k	i1	i2	res
1	1	2	3	4	D	FFFE	14
2	4	3	2	1	B	2	D
3	1	4	2	0	A	FFFC	6

Выводы.

Была построена программа с условными переходами, которая считает значения функций с заданными целочисленными параметрами.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb3.asm

```
AStack SEGMENT STACK
        DW 32 DUP(?)
AStack ENDS

DATA SEGMENT
a       DW   1
b       DW   4
i       DW   2
k       DW   0
i1      DW   ?
i2      DW   ?
res     DW   ?
DATA ENDS

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

Main PROC FAR
        mov ax, DATA
        mov ds, ax

    mov ax, a
        cmp ax, b
        jle f1_b ;jump if a <= b

f1_a:   ;a > b
```

```
        mov ax, i
        shl ax, 1
    neg ax
        mov bx, ax
    add ax, 15
    mov i1, ax

        jmp f6

f1_b:  ;a <= b
        mov ax, i
        shl ax, 1
    add ax, i
        mov bx, ax
    add ax, 4
    mov i1, ax

    jmp f6


f6:
     mov ax, a
     cmp ax, b
     jle f6_b          ; a <= b
     mov ax, 1          ; кладем в ax 1
     add ax, i          ; ax = 1 + i
     shl ax, 1          ; домножим ax на 2
     sub ax, 4          ; ax = ax - 4
     mov i2, ax
     jmp f5
```

```
f6_b:
    mov ax, 1          ; кладем в ax 1
    add ax, i          ; ax = 1 + i
    mov bx, ax         ; кладем в bx i + 1
    shl ax, 1          ; домножим ax на 2
    add ax, bx         ; прибавим i + 1 и получим (i+1)*3
    mov bx, 5          ; кладем в bx 5
    sub bx, ax         ; bx - ax = 5 - (1 + i) * 3
    mov i2, bx
    jmp f5
f5:
        mov ax, k
        cmp k, 0
        je f5_cmp_6        ;k = 0
        jmp f5_sum_1       ;k != 0

f5_cmp_6:              ;проверяем i1 на знак
        mov bx, i1
        cmp bx, 0
        jl f5_neg
        jmp f5_cmp_6_1

f5_neg:                ;если i1 < 0, меняем знак по модулю
        neg bx
        jmp f5_cmp_6_1

f5_cmp_6_1:            ;основная функция сравнения
        mov cx, 6
        cmp bx, cx
```

```
        jl res_i1
        jmp res_6

f5_sum_1:               ;проверка на положительность i1
        mov bx, i1
        cmp bx, 0
        jl f5_neg_sum
        jmp f5_sum_2
f5_sum_2:
        mov cx, i2          ;проверка на положительность i2
        cmp cx, 0
        jl f5_neg_sum_2
        jmp f5_res_sum

f5_neg_sum:
        neg bx
        jmp f5_sum_2
                ;две функции для смены знака
f5_neg_sum_2:
        neg cx
        jmp f5_res_sum

f5_res_sum:             ;сложение
        mov ax, bx
        add ax, cx
        mov res, ax
        jmp f_end



res_6:
```

```
        mov res, 6
        jmp f_end

                ;запись результата функции при k = 0
res_i1:
        mov res, bx
        jmp f_end

f_end:
        mov ah, 4ch
        int 21h

Main  ENDP
CODE        ENDS
        END Main
```