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.

Студент гр. 1383 _____ Депрейс А.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2022

Цель работы.

Разработать на языке Ассемблера программу, которая по заданным целочисленным значениям параметров вычисляет значения функций.

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.

$$f1 = \begin{cases} 15-2*i, & \text{при } a>b \\ 3*i+4, & \text{при } a<=b \end{cases}$$

$$f8 = \begin{cases} -(6*i+8), & \text{при } a>b \\ 9-3*(i-1), & \text{при } a<=b \end{cases}$$

$$f7 = \begin{cases} |i1| + |i2|, & \text{при } k<0 \\ \max(6, |i1|), & \text{при } k>=0 \end{cases}$$

Выполнение работы.

В соответствии с выбранным вариантом были реализованы заданные функции. Программа протранслирована с различными текстовыми данными.

В ходе выполнения лабораторной работы были использованы команды:

Для передачи данных:

- mov – присваивание

Арифметические команды:

1) add - сложение
2) sub - вычитание
3) cmp - сравнение
4) neg – смена знака

Сдвига:

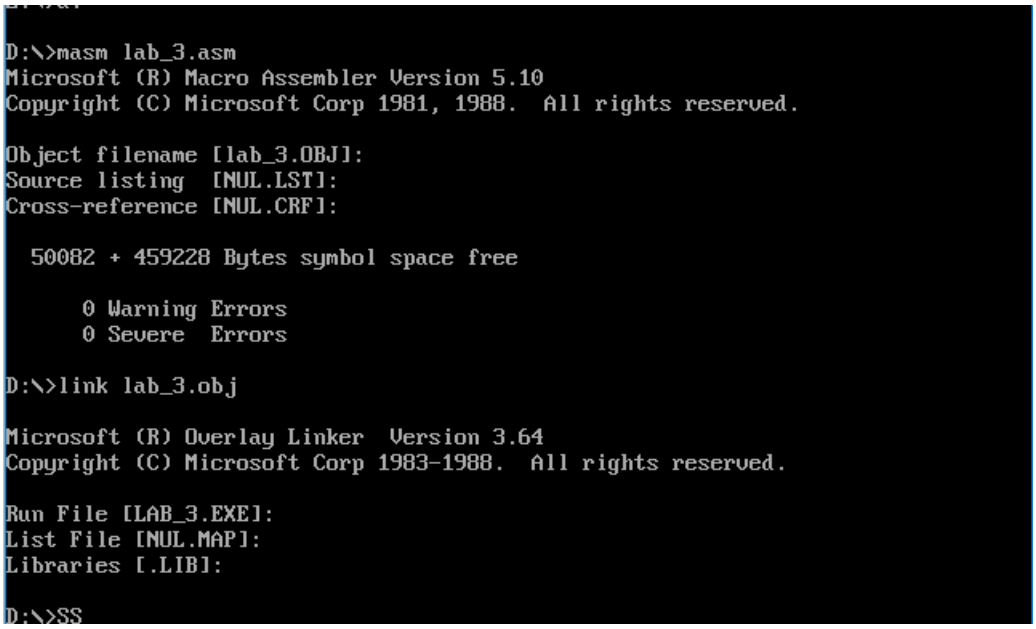
- sal – сдвиг влево

Передача управления:

1) jmp – безусловный переход
2) jg – короткий переход, при условии что первый операнд больше второго при сравнении командой cmp.
3) jl – короткий переход, при условии что первый операнд меньше второго при сравнении командой cmp.
4) jge – короткий переход, при условии что первый операнд больше или равен второму при сравнении командой cmp.

В работе наблюдалась явна необходимость реализации ветвления, которое на языке Ассемблера возможно представить с помощью меток – символьных имен, содержащих определенные команды. Переходы между метками обеспечиваются с помощью описанных ранее команд передачи управления.

Трансляция и линковка программы:



Выводы.

В ходе лабораторной работы были получены навыки разработки программы с заданными целочисленными значениями на языке программирования Ассемблер.

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

```
ASSUME SS:Stack, CS:Code, DS:Data

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

Data SEGMENT
    a DW 2
    b DW 1
    i DW 1
    k DW -1

    i1 DW ?
    i2 DW ?
    res DW ?
Data ENDS

Code SEGMENT

f1_f2 PROC NEAR
    mov ax, a
    cmp ax, b
    jg greater

less_or_equal:

    mov ax, i ; i
    sal ax, 1 ; i*2
    add ax, i ; i*3
    add ax, 4 ; i*3 + 4 / 3*(i - 1) + 7
    mov i1, ax ; i1 = i*3 + 4

    sub ax, 16 ; 3*(i - 1) - 9
    neg ax ; 9 - 3*(i - 1)
    jmp f1_f2_end
```

```
greater:

    mov ax, i ; i
    sal ax, 1 ; 2*i
    mov bx, ax
    sub ax, 15 ; 2*i - 15
    neg ax ; 15 - 2*i
    mov i1, ax ; i1 = 15 - 2*i

    mov ax, bx ; 2*i
    sal ax,1 ; 4*i
    add ax, bx ; 6*i
    add ax, 8 ; 6*i + 8
    neg ax ; -(6*i + 8)

f1_f2_end:
    mov i2, ax
    ret
f1_f2 ENDP


f3 PROC NEAR
    mov ax, k
    cmp ax, 0
    jl k_less_zero

k_not_less_zero:

    mov ax, i1 ; i1
    cmp ax, 0 ; i1 >= 0 ?
    jge i1_not_below_zero_k_upper

i1_below_zero_k_upper:

    neg ax ; -(i1)

i1_not_below_zero_k_upper:
```

```
    cmp ax, 6 ; |i1| >= 6 ?
    jg abs_i1_more_six


abs_i1_lower_six:

    mov ax, 6

abs_i1_more_six:

    jmp f3_end

k_less_zero:

    mov ax, i1 ; i1
    cmp ax, 0 ; i1 >= 0 ?
    jge i1_not_below_zero_k_lower

i1_below_zero_k_lower:

    neg ax ; -(i1)

i1_not_below_zero_k_lower:

    mov bx, i2 ; i2
    cmp bx, 0 ; i1 >= 0 ?
    jge i2_not_below_zero

i2_below_zero:

    neg bx ; -(i2)

i2_not_below_zero:

    add ax, bx ; (|i1| + |i2|)

f3_end:
```

```
    mov res, ax
    ret

f3 ENDP

main PROC FAR
    push  ds
    sub   ax,ax
    push  ax

    mov ax, Data
    mov ds, ax

    call f1_f2
    call f3


    ret
main ENDP

Code ENDS

END main
```