ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)»

Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе № 3

по дисциплине «Организация ЭВМ и систем»

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

Студент гр. 1303 _____ Ягодаров М.А.

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 $i_1 = f_1(a, b, i)$ и $i_2 = f_2(a, b, i)$;

b) значения результирующей функции $res = f_3(i_1, i_2, k)$, где вид функций f_1 и f_2 определяется из табл. 2, а функции f_3 — из табл. 3 по цифрам шифра индивидуального задания (n_1, n_2, n_3), приведённым 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, k, позволяющие проверить различные маршруты выполнения программы, а также различные знаки параметров a и b.

$$f_1 = \begin{cases} 15 - 2 \cdot i, & \text{при } a > b \\ 3 \cdot i + 4, & \text{при } a \leq b \end{cases}$$

$$f_4 = \begin{cases} -(6 \cdot i - 4), & \text{при } a > b \\ 3 \cdot (i + 2), & \text{при } a \leq b \end{cases}$$

$$f_3 = \begin{cases} |i_1 + i_2|, & \text{при } k = 0 \\ \min(i_1, i_2), & \text{при } k \neq 0 \end{cases}$$

Выполнение работы.

1. Из таблицы получен вариант набора функций, которые необходимо реализовать.

2. Программа протранслирована с различными значениями переменных, результат выполнения набора функций зафиксирован в таблице.

3. Для выполнения задания были использованы такие команды общего назначения как:

- Команды передачи данных:
  - `mov` — присваивание
- Двоичные арифметические команды:
  - `add` — сложение
  - `sub` — вычитание
  - `cmp` — сравнение
  - `neg` — инвертирование знака
- Команды побитового сдвига:
  - `sal` — арифметический сдвиг влево
- Команды передачи управления:
  - `jmp` — безусловный переход
  - `jg` — переход, если при вызове `cmp` первый операнд большего второго операнда
  - `jge` — переход, если при вызове `cmp` первый операнд больше или равен второму операнду
  - `jle` — переход, если при вызове `cmp` первый операнд меньше или равен второму операнду
  - `je` — переход, если при вызове `cmp` первый операнд равен второму операнду
  - `jne` — переход, если при вызове `cmp` первый операнд не равен второму операнду

4. Программа выполнена в пошаговом режиме под управлением отладчика с фиксацией значений используемых переменных.

№	Тестируемый случай	Функции для данного случая	Данные	
			Входные	Выходные
1	$a > b$ k = 0	$f_1 = 15 - 2 \cdot i$ $f_2 = -(6 \cdot i - 4)$ $f_3 = \lvert f_1 + f_2 \rvert$	$a = 7,\ b = 3$ $k = 0,\ i = 2$	$f_1 = 11 = 000B$ $f_2 = -8 = FFF8$ $f_3 = 3 = 0003$
2	$a > b$ $k \neq 0$	$f_1 = 15 - 2 \cdot i$ $f_2 = -(6 \cdot i - 4)$ $f_3 = \min(f_1, f_2)$	$a = 7,\ b = 3$ $k = 1,\ i = 3$	$f_1 = 9 = 0009$ $f_2 = -14 = FFF2$ $f_3 = -14 = FFF2$
3	$a \leq b$ $k = 0$	$f_1 = 3 \cdot i + 4$ $f_2 = 3 \cdot (i + 2)$ $f_3 = \lvert f_1 + f_2 \rvert$	$a = 5,\ b = 5$ $k = 0,\ i = 2$	$f_1 = 10 = 000A$ $f_2 = 12 = 000C$ $f_3 = 22 = 0016$
4	$a \leq b$ $k \neq 0$	$f_1 = 3 \cdot i + 4$ $f_2 = 3 \cdot (i + 2)$ $f_3 = \min(f_1, f_2)$	$a = 3,\ b = 5$ $k = 1,\ i = 3$	$f_1 = 13 = 000D$ $f_2 = 15 = 000F$ $f_3 = 13 = 000D$

Выводы

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

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММ

Название файла: lab3.asm

```
assume ss:my_stack, cs:my_code, ds:my_data

my_stack segment stack
   dw 12 dup('?')
my_stack ends

my_data segment

   i dw 0
   a dw 0
   b dw 0
   k dw 0

   i1 dw 0
   i2 dw 0
   res dw 0

my_data ends

my_code segment

f1_f2 proc near
   mov ax, a
   cmp ax, b
   jg greater

less_or_equal:
   mov ax, i
   sal ax, 1 ; 2 * i
   add ax, i ; 3 * i
   add ax, 4 ; 4 + 3 * i
   mov i1, ax ; i1 = 3 * i + 4

   add ax, 2 ; 6 + 3 * i
   jmp end_f1_f2

greater:
   mov cx, i
   sal cx, 1 ; 2 * i
   mov ax, 15 ; 15 + 2 * i
   sub ax, cx ; 15 - 2 * i
   mov i1, ax ; i1 = 15 - 2 * i

   add cx, i ; 3 * i
   add cx, cx ; 6 * i
```

```
   mov ax, 4
   sub ax, cx ; 4 - 6 * i

end_f1_f2:
   mov i2, ax

   ret

f1_f2 endp

f3 proc near
   mov ax, k
   cmp ax, 0
   je equal_zero

not_equal_zero:
   mov ax, i1
   cmp ax, i2
   jle end_f3 ; i1 ⩽ i2
   mov ax, i2
   jmp end_f3 ; i1 > i2

equal_zero:
   mov ax, i1
   add ax, i2
   cmp ax, 0
   jge end_f3 ; i1 + i2 ⩾ 0

   abs: ; i1 + i2 < 0
      neg ax

end_f3:
   mov res, ax

   ret

f3 endp

main proc far
   push ds
   xor ax, ax
   push ax

   mov ax, my_data
   mov ds, ax

   call f1_f2
   call f3

   mov ax, i1
   mov bx, i2
```

```
   mov cx, res

   ret
main endp

my_code ends

end main
```