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

ОТЧЕТ

по лабораторной работе №8
по дисциплине «Организация ЭВМ»
Тема: Обработка вещественных чисел. Программирование математического сопроцессора.

Студент гр. 9383 _____ Чумак М.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Изучить работу с математическим сопроцессором и написать программу, обрабатывающую вещественные числа при помощи него.

Задание.

Разработать на языке Ассемблера фрагмент программы, обеспечивающий вычисление заданной математической функции с использованием математического сопроцессора, который включается по принципу in-line в программу, разработанную на языке С.

ВАРИАНТ 2.

Name cosh - hyperbolic function:

Usage double cosh(double x);

Prototype in math.h

Description cosh computes the hyperbolic cosine of the input value.

cosh (x) = (exp(x) + exp(-x)) / 2

cosh is more accurately calculated by the polynomial (1 + x^2/2) when x is tiny (|x| < 2^-13).

Выполнение работы.

Была разработана программа, которая вычисляет значение cosh — гиперболический косинус.

В функции *main()* идёт считывание значения числа x с консоли, а затем вычисляется значение гиперболического косинуса при помощи функции *cosh()* и выводит результат на экран. Функция *cosh()* проверяет, насколько мало число *x*. В зависимости от этого идёт дальнейшее вычисление по разным формулам: при маленьком *x (|x| < 2^-13)* формула - *(1 + x^2/2)*, иначе - *(exp(x) + exp(-x)) / 2*.

Исходный код программы представлен в приложении А.

Выводы.

Была изучена работа с математическим сопроцессором и написана программа, обрабатывающая вещественные числа при помощи него.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММ

Файл lab8.cpp

```
#include <iostream>

using namespace std;
int main() {
    setlocale(LC_ALL, "Rus");
    double x;
    cout << "Enter the value of x: ";
    cin >> x;
    cout << "The result is " << cosh(x) << endl;
}

double cosh(double x) {
    double result;
    int degree = -13; // для вычисления по другой формуле
    _asm {
        fld x;              //st(0) = x
        fabs;               //st(0) = |x|
        fild degree;        //st(0) = -13
                            //st(1) = |x|
        fld1;               //st(0) = 1
                            //st(1) = -13
                            //st(2) = |x|
        fscale;             //st(0) = 1 * 2 ** (|x|)
                            //st(1) = |x|
        fcomip st, st(1);   //st(0) = |x|
        fstp st(0);         //стэк пустой
         jl tiny_x          //если |x| слишком мал, то вычисляем по формуле 1+
(x^2)/2
```

```
fld x;              //st(0) = x
fldl2e;             //st(0) = log2e
                    //st(1) = x
fmul;               //st(0) = x * log2e
fld st;             //st(0) = x * log2e
                    //st(1) = x * log2e
frndint;            //st(0) = [x * log2e]
                    //st(1) = x * log2e
fsub st(1), st;     //st(0) = [x * log2e]
                    //st(1) = x * log2e - [x * log2e]
fxch st(1);         //st(0) = x * log2e - [x * log2e]
                    //st(1) = [x * log2e]
f2xm1;              //st(0) = 2 ** (x * log2e - [x * log2e]) - 1
                    //st(1) = [x * log2e]
fld1;               //st(0) = 1
                    //st(1) = 2 ^ (x * log2e - [x * log2e]) - 1
                    //st(2) = [x * log2e]
fadd;               //st(0) = 2 ^ (x * log2e - [x * log2e])
                    //st(1) = [x * log2e]
fscale;             //st(0) = e ^ x st(1) = [x * log2e]
fstp st(1);         //st(0) = e ^ x

fld x;              //st(0) = x
fchs;               //st(0) = -x
fldl2e;             //st(0) = log2e
                    //st(1) = -x
fmul;               //st(0) = -x * log2e
fld st;             //st(0) = -x * log2e
                    //st(1) = -x * log2e
```

```
    frndint;            //st(0) = [-x * log2e]
                        //st(1) = -x * log2e
    fsub st(1), st;     //st(0) = [-x * log2e]
                        //st(1) = -x * log2e - [-x * log2e]
    fxch st(1);         //st(0) = -x * log2e - [-x * log2e]
                        //st(1) = [-x * log2e]
    f2xm1;              //st(0) = 2 ^ (-x * log2e - [-x * log2e]) - 1
                        //st(1) = [-x * log2e]
    fld1;               //st(0) = 1
                        //st(1) = 2 ^ (-x * log2e - [-x * log2e]) - 1
                        //st(2) = [-x * log2e]
    fadd;               //st(0) = 2 ^ (-x * log2e - [-x * log2e])
                        //st(1) = [-x * log2e]
    fscale;             //st(0) = e ^ -x
                        //st(1) = [-x * log2e]
    fstp st(1);         //st(0) = e ^ -x

    fadd;               //st(0) = e ^ x + e ^ (-x)
    fld1;               //st(0) = 1
                        //st(1) = e ^ x + e ^ (-x)
    fld1;               //st(0) = 1
                        //st(1) = 1
                        //st(2) = e ^ x + e ^ (-x)
    fadd;               //st(0) = 2
                        //st(1) = e ^ x + e ^ (-x)
    fdiv;               //st(0) =  (e ^ x + e ^ (-x)) / 2
    jmp end_func;

tiny_x:
    fld1;               //st(0) =  1
```

```
        fld x;              //st(0) =  x
                            //st(1) =  1
        fld x;              //st(0) =  x
                            //st(1) =  x
                            //st(2) =  1
        fmul;               //st(0) =  x ^ 2
                            //st(1) =  1
        fld1;               //st(0) = 1
                            //st(1) =  x ^ 2
                            //st(2) =  1
        fld1;               //st(0) = 1
                            //st(1) = 1
                            //st(2) =  x ^ 2
                            //st(3) =  1
        fadd;               //st(0) = 2
                            //st(1) =  x ^ 2
                            //st(2) =  1
        fdiv;               //st(0) = (x ^ 2) / 2
                            //st(1) =  1
        fadd;               //st(0) = ((x ^ 2) / 2) + 1
        jmp end_func;
    end_func:
        fstp result;
    }
    return result;
}
```