МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

КАФЕДРА РАДИОЭЛЕКТРОННЫХ СРЕДСТВ

Отчет по дисциплине

«Цифровые устройства и микропроцессоры»

Лабораторная работа №4

«МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ»

Вариант 19

Выполнил: студент группы ИНБб– 3301\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Е.А. Лопаткин /

Проверил:\_ к.т.н. доцент кафедры РЭС \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ М.А. Земцов /

Киров 2023

**Цель работы:** знакомство с технологией применения языка ассемблера

при разработке программного обеспечения на языках высокого уровня.

**Задание:**



**Программная реализация:**

По заданию требуется вычислить функцию y=f(xi) на интервале (0,1) с шагом 0.1. Для этого создадим массив x, состоящий из 11 элементов (0,0.1,0.2 … и тд)

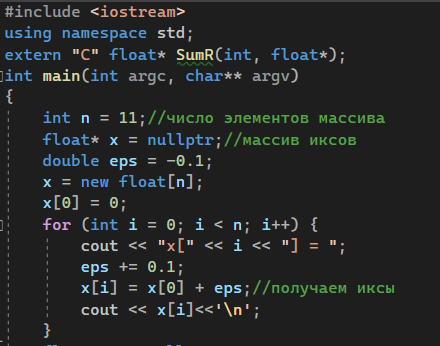
****

Рисунок 1 – Задаем массив x

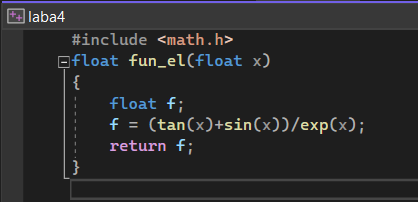
****

Рисунок 2 – Функция f(x)

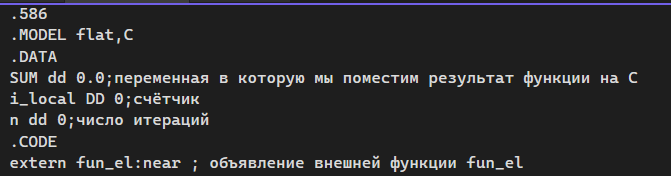


Рисунок 3 – Объявляем переменные на ассемблере

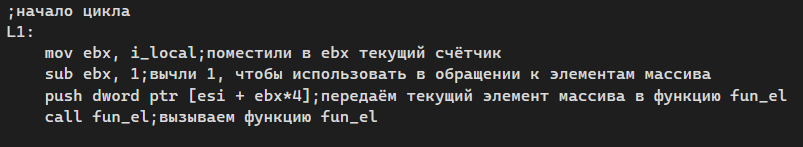


Рисунок 4 – передаем текущий элемент массива и вызываем функцию f(x)

На каждой итерации функция f(x) возвращает значения в ST0. Чтобы сохранить это значение записываем его в переменную SUM, затем в регистр ecx и затем помещаем в текущий элемент массива.

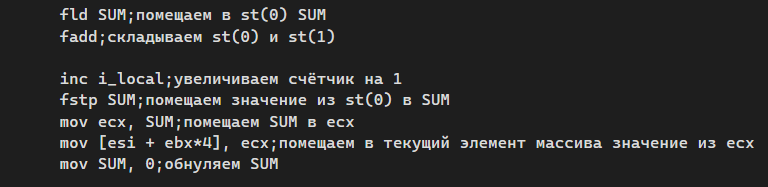


Рисунок 5 – Записываем значение функции в массив

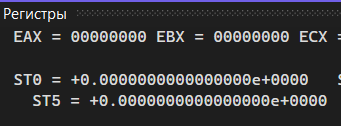


Рисунок 6 – Значение функции при x=0

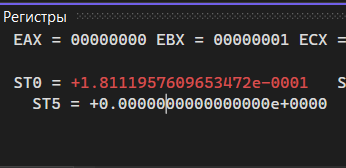


Рисунок 7 – Значение функции при x=0.1

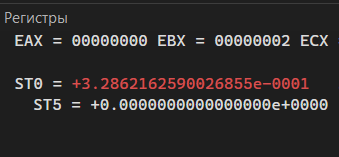


Рисунок 8 – Значение функции при x=0.2

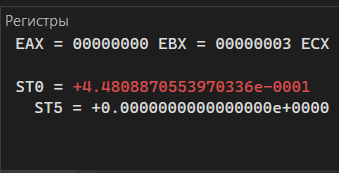


Рисунок 9 – Значение функции при x=0.3

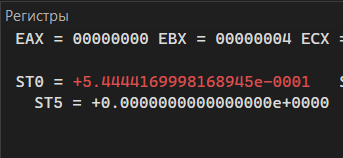


Рисунок 10 – Значение функции при x=0.4

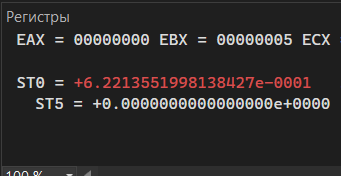


Рисунок 11 – Значение функции при x=0.5

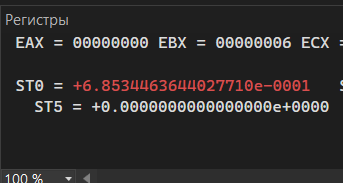


Рисунок 12 – Значение функции при x=0.6

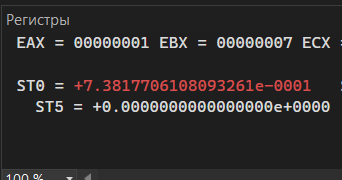


Рисунок 13 – Значение функции при x=0.7

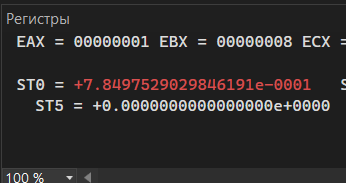


Рисунок 14 – Значение функции при x=0.8

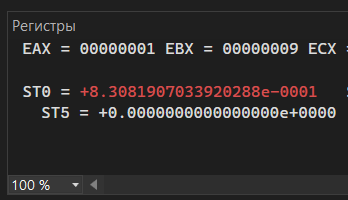


Рисунок 15 – Значение функции при x=0.9

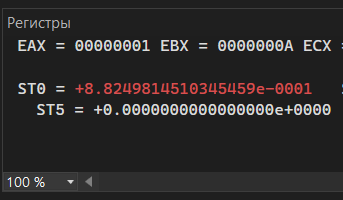


Рисунок 16 – Значение функции при x=1

Цикл заканчивается, когда текущее значение счетчика ebx совпадет с n

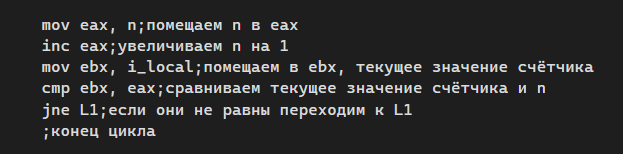


Рисунок 17 – Условие выхода из цикла

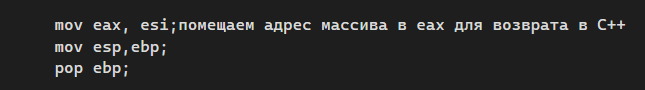


Рисунок 18 – Возврат значений в main

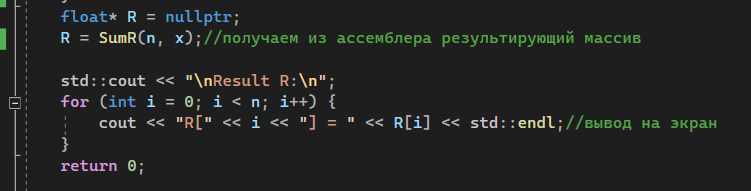
****

Рисунок 19 – Вывод значений на экран

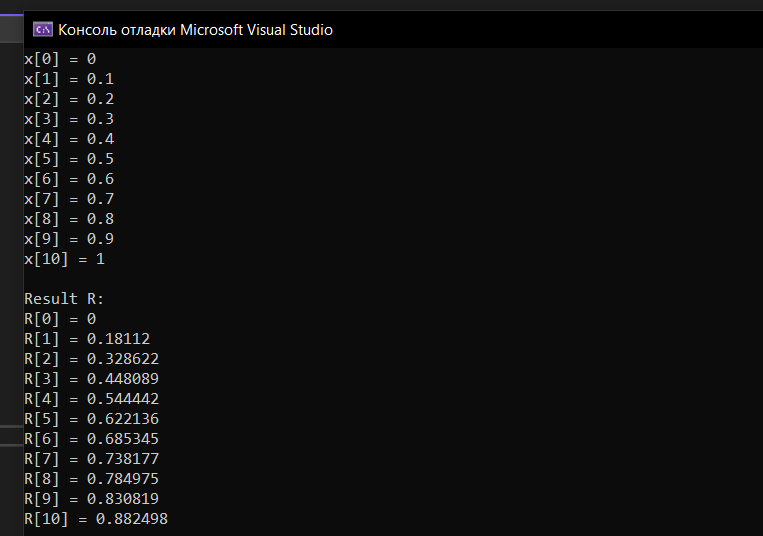
****

Рисунок 20 – Результат работы программы

**Код программы:**

**Main.cpp**

#include <iostream>

using namespace std;

extern "C" float\* SumR(int, float\*);

int main(int argc, char\*\* argv)

{

int n = 11;//число элементов массива

float\* x = nullptr;//массив иксов

double eps = -0.1;

x = new float[n];

x[0] = 0;

for (int i = 0; i < n; i++) {

cout << "x[" << i << "] = ";

eps += 0.1;

x[i] = x[0] + eps;//получаем иксы

cout << x[i]<<'\n';

}

float\* R = nullptr;

R = SumR(n, x);//получаем из ассемблера результирующий массив

std::cout << "\nResult R:\n";

for (int i = 0; i < n; i++) {

cout << "R[" << i << "] = " << R[i] << std::endl;//вывод на экран

}

return 0;

}

**func.c**

#include <math.h>

float fun\_el(float x)

{

float f;

f = (tan(x)+sin(x))/exp(x);

return f;

}

**laba.asm**

.586

.MODEL flat,C

.DATA

SUM dd 0.0;переменная в которую мы поместим результат функции на C

i\_local DD 0;счётчик

n dd 0;число итераций

.CODE

extern fun\_el:near ; объявление внешней функции fun\_el

public SumR

SumR proc C

push ebp;

mov ebp,esp;

mov i\_local,1;помещаем в счётчик 1

mov ecx, dword ptr [ebp+8];помещаем в ecx число итераций цикла n

mov n, ecx;поместили в локальную переменную "n" число итераций из ecx

mov esi, [ebp + 12];поместили адрес массива в esi

;начало цикла

L1:

mov ebx, i\_local;поместили в ebx текущий счётчик

sub ebx, 1;вычли 1, чтобы использовать в обращении к элементам массива

push dword ptr [esi + ebx\*4];передаём текущий элемент массива в функцию fun\_el

call fun\_el;вызываем функцию fun\_el

fld SUM;помещаем в st(0) SUM

fadd;складываем st(0) и st(1)

inc i\_local;увеличиваем счётчик на 1

fstp SUM;помещаем значение из st(0) в SUM

mov ecx, SUM;помещаем SUM в ecx

mov [esi + ebx\*4], ecx;помещаем в текущий элемент массива значение из ecx

mov SUM, 0;обнуляем SUM

mov eax, n;помещаем n в eax

inc eax;увеличиваем n на 1

mov ebx, i\_local;помещаем в ebx, текущее значение счётчика

cmp ebx, eax;сравниваем текущее значение счётчика и n

jne L1;если они не равны переходим к L1

;конец цикла

mov eax, esi; помещаем адрес массива в eax для возврата в C++

mov esp,ebp;

pop ebp;

ret

SumR endp

End

**Вывод:**

Код, написанный на C++, C и MASM (Microsoft Macro Assembler), может быть быстрее, чем код на C++, по нескольким причинам:

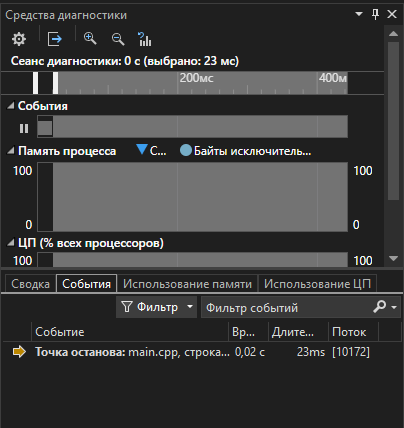
Прямой доступ к аппаратным ресурсам: Код, написанный на C и MASM, предоставляет прямой доступ к аппаратным ресурсам компьютера, таким как регистры процессора, специальные инструкции и прерывания. Это позволяет более эффективно использовать ресурсы и оптимизировать код для конкретной архитектуры процессора.

Управление низкоуровневыми деталями: Низкоуровневые языки программирования, такие как C и MASM, позволяют более точно контролировать каждую инструкцию и операцию в программе. Это позволяет оптимизировать код до мельчайших деталей, что может привести к повышению производительности.

Отсутствие накладных расходов компилятора: Компиляторы высокоуровневых языков, таких как C++, вносят некоторые накладные расходы при трансляции кода в машинный код. Они выполняют оптимизации, добавляют дополнительный код для обработки исключений, управления памятью и других функций. Низкоуровневые языки, такие как C и MASM, могут быть лишены этих накладных расходов, что позволяет более эффективно использовать ресурсы и увеличивает производительность.

Однако, стоит отметить, что написание кода на низкоуровневых языках требует больше времени, усилий и экспертизы по сравнению с использованием языков программирования высокого уровня, таких как C++. Кроме того, современные компиляторы C++ обладают мощными оптимизациями, которые могут значительно улучшить производительность кода.

В целом, для большинства приложений и задач использование языка программирования высокого уровня, такого как C++, является предпочтительным. Он обеспечивает удобство разработки, читаемость кода, переносимость и мощные инструменты оптимизации, предоставляемые современными компиляторами.



#include <iostream>

#include <cmath>

int main() {

setlocale(LC\_ALL, "Russian");

// Задаем начальное,конечное значения интервала, шаг

double start = 0.0;

double end = 1.0;

double step = 0.1;

// Вычисляем и выводим значения функции на каждом шаге

for (double x = start; x <= end; x += step) {

double result = (tan(x) + sin(x)) / exp(x);

std::cout << "Значение x: " << x << " Значение f(x): " << result << std::endl;

}

return 0;

}