Министерство образования Республики Беларусь

Учреждение образования   
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Отчет по лабораторным работам №7-8  
«Программирование арифметического сопроцессора»

Выполнил:  
студент гр. 153504  
Климкович Н.В.

Проверила:

Калиновская А.А.

Минск 2023

# 1. Цель работы

Ознакомиться с предназначением арифметического сопроцессора. Рассмотреть строение и основные принципы работы сопроцессора. Изучить команды, доступные при использовании сопроцессора. Научиться работать с сопроцессором в ходе выполнения лабораторного задания.

# 2. Постановка задачи

***Задание к лабораторной работе 4***

Написать программу, находящую решение квадратного уравнения

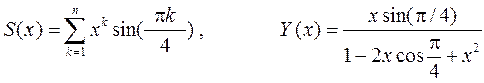
ax2 + bx + c = 0

с помощью сопроцессора.

***Задание к лабораторной работе 5***

Значение аргумента **x** изменяется от **a** до **b** с шагом h. Для каждого **x** найти значения функции Y(x), суммы S(x) и число итераций n, при котором достигается требуемая точность ε = |Y(x)-S(x)|. Результат вывести в виде таблицы. Значения **a, b, h** и **ε** вводятся с клавиатуры.

*Вариант 5:*



# 3. Теоретические сведения

### Сопроцессорные конфигурации

 Использование сопроцессора позволяет значительно ускорить работу программ, выполняющих расчеты с высокой точностью, тригонометрические вычисления и обработку информации, которая должна быть представлена в виде действительных чисел. Сопроцессор подключается к системной шине параллельно с центральным процессором (CPU) и может работать только совместно с ним. Все команды попадают в оба процессора, а выполняет каждый свои. Сопроцессор не имеет своей программы и не может осуществлять выборку команд и данных. Это делает центральный процессор. Сопроцессор перехватывает с шины данные и после этого реализует конкретные действия по выполнению команды. Два процессора работают параллельно, что повышает эффективность системы.  Но возникают ситуации, когда их работа требует синхронизации (из-за разницы во времени выполнения команд).

*Синхронизация по командам*. Когда центральный процессор выбирает для выполнения команду FPU, последний может быть занят выполнением предыдущей команды. Поэтому перед каждой командой сопроцессора в программе должна стоять специальная команда (wait), которая только проверяет текущее состояние FPU и, если он занят, переводит центральный процессор в состояние ожидания. Соответствующую команду в программу может вводить либо ассемблер, либо компилятор языка без указаний программиста.

*Синхронизация по данным.* Если выполняемая в FPU команда записывает операнд в память перед последующей командой СРU, которая обращается к этой ячейке памяти, требуется команда проверки состояния FPU. Если данные еще не были записаны, СPU должен переходить в состояние ожидания. Автоматически учесть такие ситуации довольно сложно, поэтому вводить команды, которые проверяют состояние сопроцессора и при необходимости заставляют центральный процессор ожидать, должен программист.

### Программная модель сопроцессора

В программную модель любого процессора включаются только те регистры, которые доступны программисту на уровне машинных команд. Основу программной модели FPU образует регистровый стек из восьми 80-битных регистров R0-R7. В них хранятся числа в вещественном формате. В любой момент времени 3-битное поле ST в слове состояния определяет регистр, являющийся текущей вершиной стека и обозначаемый ST (0). При занесении в стек (push) осуществляется декремент поля ST и загружаются данные в новую вершину стека. При извлечении из стека (pop) в получатель, которым чаще всего является память, передается содержимое вершины стека, а затем инкрементируется поле ST.

В организации регистрового стека FPU есть отличия от классического стека.

1.  Стек имеет кольцевую структуру. Контроль за использованием стека должен осуществлять программист. Максимальное число занесений без промежуточных извлечений равно 8.

2.    В командах FPU допускается явное или неявное обращение к регистрам с модификацией или без поля ST., например, команда fsqrt замещает число из вершины стека значением корня из него. В бинарных операциях допускается явное указание регистров. Адресация осуществляется относительно текущей вершины стека и обозначение ST (i) 0 <i <7, считая от вершины.

   3. Не все стековые команды автоматически модифицируют указатель вершины стека.

С каждым регистром стека ассоциируется 2-битный тег (признак), совокупность которых образует слово тегов. Тег регистра R0 находиться в младших битах, R7 – в старших. Тег фиксирует наличие в регистре действительного числа (код 00), истинного нуля (код 01), ненормализованного или бесконечности (код 10) и отсутствие данных (код 11).  Наличие тегов позволяет FPU быстрее обнаруживать особые случаи (попытка загрузить в непустой регистр, попытка извлечь из пустого) и обрабатывать данные.

Остальными регистрами FPU являются регистр управления, регистр состояния, два регистра состояния команды и два регистра указателя данных. Длина их всех 16 бит.

### Форматы численных данных

 Арифметический FPU К1810ВМ87 оперирует с семью форматами численных данных, образующих три класса: двоичные целые, упакованные десятичные целые и вещественные числа. Во всех форматах старший (левый) бит отведен для знака.

Форматы различаются длиной, следовательно, диапазоном допустимых чисел, способом представления (упакованный и неупакованный формат), способом кодировки (прямой и дополнительный код).

*Двоичные целые числа.*Три формата целых двоичных (целое слово (16 бит), короткое целое (32 бита), длинное целое (64 бита)) отличаются длиной, следовательно, диапазоном чисел. Только в этих форматах применяется стандартный дополнительный код. 0 имеет единственное кодирование. Наибольшее положительное число кодируется как 011…1, а Наибольшее по модулю отрицательное как 100...0.

*Упакованные десятичные целые.* Числа представлены в прямом коде и упакованном формате, т.е. в байте содержится две десятичные цифры в коде 8421. Старший бит левого байта – знак, остальные игнорируются, но при записи в них помещаются нули.  Но надо учитывать, что при наличии в тетради запрещающих комбинаций 1010 – 1111 результат операции не определен. Т.е. сопроцессор не контролирует правильность результата.

*Вещественные числа.*Различают короткие вещественные (КВ) (мантисса – 24 бита, порядок – 8 бит), длинные вещественные (ДВ) (мантисса – 53 бита, порядок – 11 бит) и временные вещественные (ВВ) (мантисса – 64 бита, порядок – 15 бит). Для них применяется формат с плавающей точкой. Значащие цифры находятся в поле мантиссы, порядок показывает фактическое положение двоичной точки в разрядах мантиссы, бит знака S определяет знак числа. Порядок дается в смещенной форме:

Е = истинный порядок + смещение

Смещение для соответствующих форматов равно 127, 1023, 16383 это упрощает операцию сравнения.  Операция с целыми числами быстрее операции над плавающей точкой. Это важно в алгоритмах.

Значение числа равно

(1) Sх 2E-смещениех F0F1F2…Fn,

где n для разных форматов равно 23,52 или 63.

Порядок имеет фиксированную длину, определяя один диапазон представимых чисел. Мантисса – правильная дробь. В коротком и длинном вещественном формате бит F0 при передаче чисел и хранении их в памяти не фигурирует. Это скрытый бит, который в нормализованных числах содержит 1. Скрытый бит не дает представить в этих форматах нуль, и он должен кодироваться как спец значение.

Числа во временном вещественном формате имеют явный бит F0. Формат повышает скорость выполнения операций благодаря простоте представления чисел, не являющихся ненормализованными. При загрузке из памяти в регистр FPU оно преобразуется во временный вещественный формат. А при записи в память – обратный формат. Временной вещественный формат – единственный, в котором абсолютно точно кодируется любые загружаемые из памяти числа.

### Режимы работы. Состояние

Сопроцессоримеет 2 доступных 16-битных регистра, содержимое которых определяет его режим работы и текущее состояние. Форматы регистров содержат слово управления CW и слово состояния SW. Регистр управления содержит 6 бит масок особых случаев. Регистр состояния – 6 бит флажков.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Регистр управления: | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| X | X | X | IC | RC | | PC | | IEM | X | PM | UM | OM | ZM | DM | IM |
| Регистр состояния: | | | | | | | | | | | | | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| B | C3 | ST | | | C2 | C1 | C0 | IR | X | PE | UE | OE | ZE | DE | IE |
| Рис 1. Форматы слова управления и слова состояния | | | | | | | | | | | | | | | |

Регистр управления содержит 6 бит масок особых случаев, а регистр состояния 6 бит флажков особых случаев:

P – потеря точности

U – антипереполнение

О – переполнение

Z - деление на нуль

D - денормализованный операнд

I - недействительный операция

*Слово управления***.** Оно определяет для FPU один из нескольких вариантов обработки численных данных. Программа центрального процессора может сформировать в памяти образ слова управления, а затем заставить сопроцессор загрузить его в регистр CW. Рассмотрим значение полей.

Шесть младших бит слова управления - индивидуальные маски особых случаев. Т.е. особых ситуаций, обнаруженных FPU при выполнении команд. Если бит=1, то не будет вызвано прерывание СPU. Иначе FPU устанавливает в 1 бит запроса прерывания в слове состояния и при общем разрешении прерываний генерирует сигнал *int* прерывания СPU.

Бит 7 слова управления содержит маску управления прерыванием IEM, которая разрешает (IEM=0) или запрещает (IEM=1) прерывание центрального процессора.

Двухбитное поле управления точностью (PC) определяет точность вычислений в 24 бита (РС=00), 53 бита (РС=10) или 64 бита (РС=11). По умолчанию вводиться режим с максимальной точностью в 64 бита.

Двухбитное поле управления округлением RC определяет один из четырех возможных вариантов округления результатов операций сопроцессора.

Бит 12 управляет режимом бесконечности IC. Когда IC=0, сопроцессор обрабатывает два специальных значения “плюс бесконечность” и “минус бесконечность” как одно и то же значение “бесконечность”, не имеющее знака.

*Слово состояния.*В нем младшие 6 бит отведены для регистрации особых случаев. Бит 7 – запроса прерывания (IR), устанавливается в 1 при возникновении любого незамаскированного особого случая. Бит С3-С0 фиксирует код условия в операциях сравнения, проверки условия и анализа. Три бита ST указатели стека. Стековые операции сопровождаются модификацией поля ST. Наконец, флажок занятости «В» устанавливается в состояние 1 когда численное операционные устройство выполняет операцию. Большую роль играют биты кода условия, которые фиксируют особенности результата (табл. 1.). Коды условия привлекаются для реализации условных переходов. Сопроцессор самостоятельно не может влиять на ход выполнения программ. Поэтому для условных переходов по результатам операций сопроцессора приходится сначала передавать код условия в память, а затем загружать один из регистров центрального процессора. После этого код условия передается в регистр флагов, производится условный переход.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 1. Интерпретация кода условия в операциях сравнения и проверки (FCOM, FCOMP, FTST) | | | | |
| C3 | C2 | C1 | C0 | Описание |
| 0 | X | X | 0 | (ST)>источника (src) |
| 0 | X | X | 1 | (ST)<источника (src) |
| 1 | X | X | 0 | (ST)=источнику (src) |
| 1 | X | X | 1 | Не сравнимы |

**Система команд**

Система команд сопроцессора содержит 6 групп команд: команды передач данных, арифметические команды, команды сравнения, трансцендентных операций, команды загрузки констант и управления сопроцессором. Операнды некоторых команд определяются неявно. Другие команды допускают или требуют явного задания операнда.

# 4. Программная реализация

В качестве языка для написания программы с использованием арифметического сопроцессора был выбран язык C++.

В ходе работы программы производится проверка на равенство нулю   
коэффициента в квадратном уравнении. По формуле вычисляется дискриминант и в зависимости от ответа выполняются дальнейшие вычисления.

На рисунках представлен результат работы программы. Демонстрируется нахождение корней квадратного уравнения и нахождение суммы S(x), функции Y(x) и число итераций n при различных значениях аргумента. Также показана обработка исключительных ситуаций.

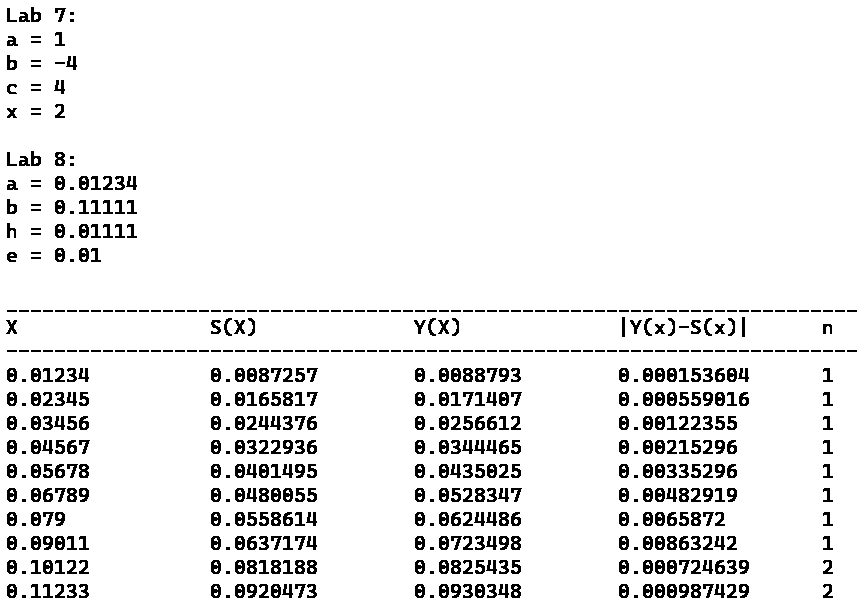


Рис.15 Пример 1.

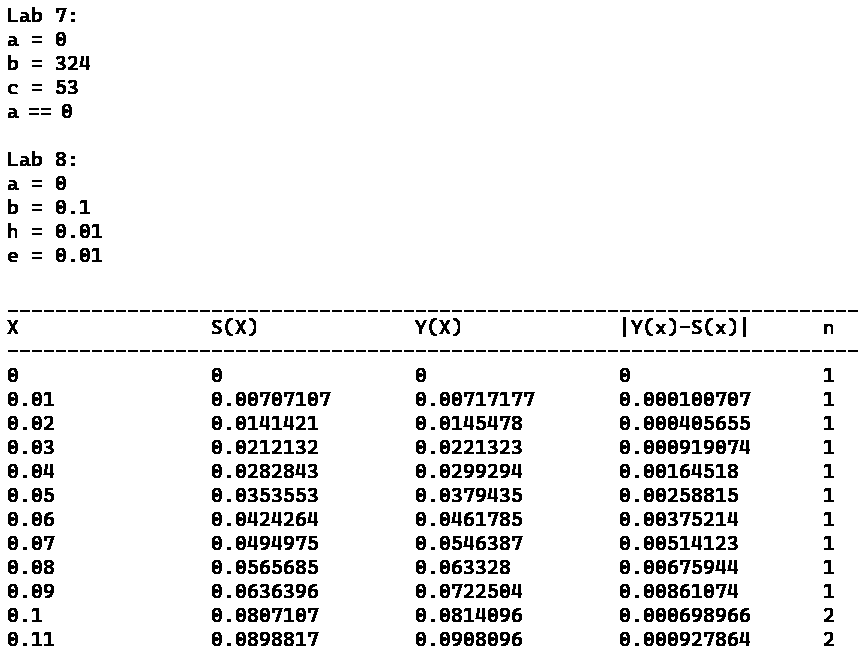


Рис.15 Пример 2.

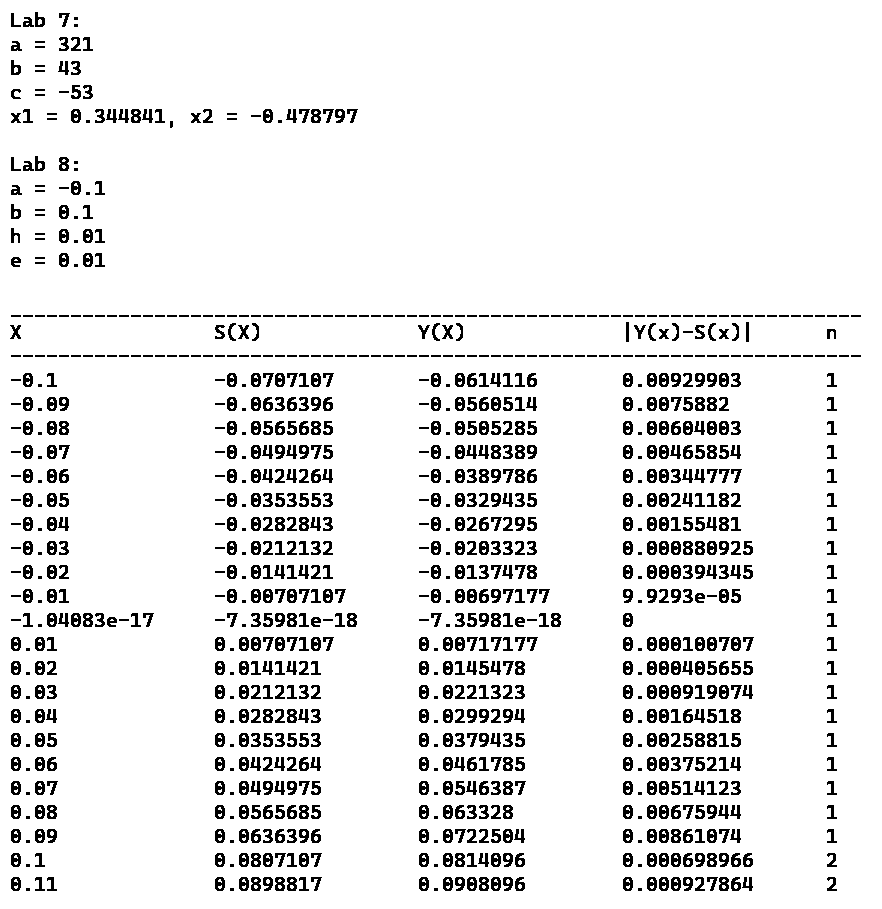


Рис.15 Пример 3.

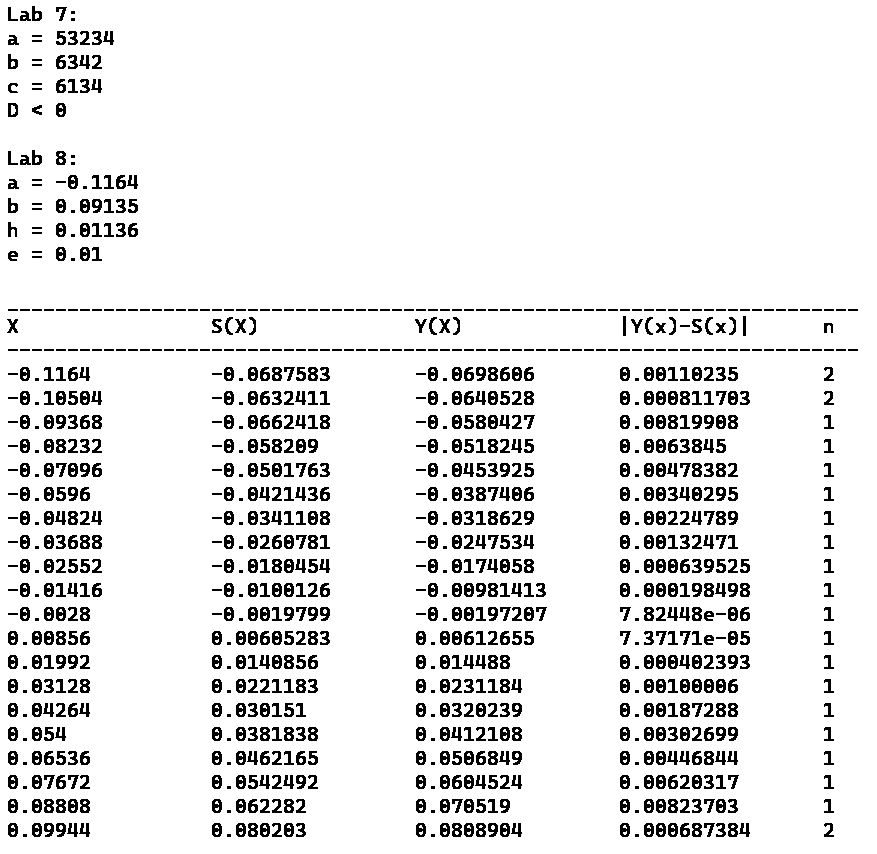


Рис.15 Пример 4.

# 5. Выводы

В ходе лабораторной работы были изучены понятия арифметического сопроцессора, сопроцессорных конфигураций, программной модели сопроцессора. Рассмотрены режимы работы и состояния арифметического сопроцессора, система команд сопроцессора, особенности задания команд, различные группы команд сопроцессора. Также были изучены различные форматы представления численных данных: двоичные целые числа, упакованные десятичные числа, вещественные числа.

# Список литературы

1. Волорова Н. А. Лабораторный практикум по курсу «Архитектура вычислительных систем» для студентов специальности «Информатика» / 93-444-487-2- Мн.: БГУИР, 2003. – 32с.:ил.

# Приложение 1. Текст программы

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <windows.h>

#include <string>

#include <cmath>

#include <iomanip>

using namespace std;

int main()

{

double a, b, c, d, x1, x2, \_4 = 4, \_2 = 2, \_1 = 1;

cout << "Lab 7:\n";

cout << "a = "; cin >> a;

cout << "b = "; cin >> b;

cout << "c = "; cin >> c;

if (a == 0) {

cout << "a==0";

}

else {

// Нахождение D

\_\_asm {

FLD b // Загрузка в стек b

FLD b // Загрузка в стек b

FMUL // Умножение ST(0) и ST(1)

FLD \_4 // Загрузка в стек 4

FLD a // Загрузка в стек a

FLD c // Загрузка в стек c

FMUL // Умножение ST(0) и ST(1)

FMUL // Умножение ST(0) и ST(1)

FSUB // Вычитание ST(0) и ST(1)

FSTP d // Запись ST(0) в d

}

if (d < 0) {

cout << "D < 0";

}

else {

if (d == 0) {

// Вычисление x

\_\_asm {

FLD b // Загрузка в стек b

FCHS // Инверсия знака ST(0)

FLD \_2 // Загрузка в стек 2

FLD a // Загрузка в стек a

FMUL // Умножение ST(0) и ST(1)

FDIV // Деление ST(0) и ST(1)

FSTP x1 // Запись ST(0) в x1

}

cout << "x = " << x1;

}

else {

// Вычисление x1, x2

\_\_asm {

FLD b // Загрузка в стек b

FCHS // Инверсия знака ST(0)

FLD d // Загрузка в стек d

FSQRT // Квадратный корень ST(0)

FADD // Сумма ST(0) из ST(1)

FLD \_2 // Загрузка в стек 2

FLD a // Загрузка в стек a

FMUL // Умножение ST(0) и ST(1)

FDIV // Деление ST(0) и ST(1)

FSTP x1 // Запись ST(0) в x1

}

\_\_asm {

FLD b // Загрузка в стек b

FCHS // Инверсия знака ST(0)

FLD d // Загрузка в стек d

FSQRT // Квадратный корень ST(0)

FSUB // Вычитание ST(0) из ST(1)

FLD \_2 // Загрузка в стек 2

FLD a // Загрузка в стек a

FMUL // Умножение ST(0) и ST(1)

FDIV // Деление ST(0) и ST(1)

FSTP x2 // Запись ST(0) в x2

}

cout << "x1 = " << x1 << ", x2 = " << x2;

}

}

}

double h, e, pi = M\_PI;

bool fail = false;

cout << "\n\nLab 8:\n";

cout << "a = "; cin >> a;

cout << "b = "; cin >> b;

cout << "h = "; cin >> h;

cout << "e = "; cin >> e;

// Вариант 5

cout << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "X S(X) Y(X) |Y(x)-S(x)| n\n";

cout << "-----------------------------------------------------------------------\n";

for (double x = a; x < b + h;x += h) {

double y;

\_\_asm {

FLD x // Загрузка в стек x

FLD pi // Загрузка в стек pi

FLD \_4 // Загрузка в стек 4

FDIV // Деление ST(0) и ST(1)

FSIN // Синус ST(0)

FMUL // Умножение ST(0) и ST(1)

FLD \_1 // Загрузка в стек 1

FLD x // Загрузка в стек x

FLD x // Загрузка в стек x

FMUL // Умножение ST(0) и ST(1)

FLD \_2 // Загрузка в стек 2

FLD x // Загрузка в стек x

FLD pi // Загрузка в стек pi

FLD \_4 // Загрузка в стек 4

FDIV // Деление ST(0) и ST(1)

FCOS // Косинус ST(0)

FMUL // Умножение ST(0) и ST(1)

FMUL // Умножение ST(0) и ST(1)

FSUB // Вычитание ST(0) и ST(1)

FADD // Сумма ST(0) и ST(1)

FDIV // Деление ST(0) и ST(1)

FSTP y // Запись ST(0) в у

}

double n = 1;

double s = 0;

double power = x;

do {

for (double o = 1;o < n;o++) {

\_\_asm {

FLD x // Загрузка в стек x

FLD power // Загрузка в стек power

FMUL // Умножение ST(0) и ST(1)

FSTP power // Запись ST(0) в power

}

}

\_\_asm {

FLD s // Загрузка в стек s

FLD power // Загрузка в стек power

FLD pi // Загрузка в стек pi

FLD n // Загрузка в стек n

FMUL // Умножение ST(0) и ST(1)

FLD \_4 // Загрузка в стек 4

FDIV // Деление ST(0) и ST(1)

FSIN // Синус ST(0) и ST(1)

FMUL // Умножение ST(0) и ST(1)

FADD // Сумма ST(0) и ST(1)

FSTP s // Запись ST(0) в s

}

n++;

if (n == 1000) {

fail = true;

break;

}

} while (abs(y - s) > e);

if (!fail)

cout << setiosflags(ios::left) << setw(17) << x << setw(17) << s << setw(17) << y <<

setw(17) << abs(y - s) << setw(5) << n - 1 << '\n';

else

cout << "--------------------------------ERROR----------------------------------\n";

}

return 0;

}