МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное  
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет  
имени академика С.П. Королева»  
(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики

Кафедра программных систем

**ОТЧЕТ**

по лабораторному практикуму по дисциплине

«Организация ЭВМ и вычислительных систем»

Студент В.Д. Гижевская

Руководитель Л.С. Зеленко

Самара 2019

СОДЕРЖАНИЕ

[Лабораторная работа 1 «Арифметические и логические команды в ассемблере» 4](#_Toc8580559)

[1.1 Теоретические основы лабораторной работы 4](#_Toc8580560)

[1.2 Задание 5](#_Toc8580561)

[1.3 Схема алгоритма 5](#_Toc8580562)

[1.4 Решение 7](#_Toc8580563)

[1.5 Результаты тестирования 8](#_Toc8580564)

[Лабораторная работа 2 «Арифметические команды и команды переходов в ассемблере» 10](#_Toc8580565)

[2.1 Теоретические основы лабораторной работы 10](#_Toc8580566)

[2.2 Задание 10](#_Toc8580567)

[2.3 Схема алгоритма 11](#_Toc8580568)

[2.4 Решение 13](#_Toc8580569)

[2.5 Результаты тестирования 15](#_Toc8580570)

[Лабораторная работа 3 «Команды работы с массивами и стеком» 16](#_Toc8580571)

[3.1 Теоретические основы лабораторной работы 16](#_Toc8580572)

[3.2 Задание 16](#_Toc8580573)

[3.3 Схема алгоритма 17](#_Toc8580574)

[3.4 Решение 17](#_Toc8580575)

[3.5 Результаты тестирования 19](#_Toc8580576)

[Лабораторная работа 4 «Изучение работы математического сопроцессора в среде Assembler» 21](#_Toc8580577)

[4.1 Теоретические основы лабораторной работы 21](#_Toc8580578)

[4.2 Задание 22](#_Toc8580579)

[4.3 Схема алгоритма 23](#_Toc8580580)

[4.4 Решение 23](#_Toc8580581)

[4.5 Результаты тестирования 25](#_Toc8580582)

[Лабораторная работа 5 «Нахождение корней уравнения методом Ньютона на языке ассемблера» 27](#_Toc8580583)

[5.1 Теоретические основы лабораторной работы 27](#_Toc8580584)

[5.2 Задание 27](#_Toc8580585)

[5.3 Схема алгоритма 28](#_Toc8580586)

[5.4 Решение 28](#_Toc8580587)

[5.5 Результаты тестирования 31](#_Toc8580588)

[Лабораторная работа 6 «Вычисление определенного интеграла методом Симпсона на языке ассемблера» 33](#_Toc8580589)

[6.1 Теоретические основы лабораторной работы 33](#_Toc8580590)

[6.2 Задание 33](#_Toc8580591)

[6.3 Схема алгоритма 34](#_Toc8580592)

[6.4 Решение 34](#_Toc8580593)

[6.5 Результаты тестирования 36](#_Toc8580594)

[Лабораторная работа 7 «Вычисление суммы ряда на языке ассемблера» 38](#_Toc8580595)

[7.1 Теоретические основы лабораторной работы 38](#_Toc8580596)

[7.2 Задание 38](#_Toc8580597)

[7.3 Схема алгоритма 39](#_Toc8580598)

[7.4 Решение 39](#_Toc8580599)

[7.5 Результаты тестирования 41](#_Toc8580600)

[Список использованных источников 42](#_Toc8580601)

Лабораторная работа 1 «Арифметические и логические команды в ассемблере»

1.1 Теоретические основы лабораторной работы

1.1.1 Команда MOV

MOV используется для копирования значения из одного места в другое (mov dst, src) [1]. «Местом» может быть регистр, ячейка памяти или непосредственное значение:

mov eax 5; eax = 5, eax – регистр.

1.1.2 Арифметические команды [1]:

* ADD – сложение (add dst, src);
* SUB – вычитание (sub dst, src);
* IMUL – знаковое умножение (imul src; первый операнд – в EAX, результат – в EDX:EAX);
* IDIV – знаковое деление (idiv src; первый операнд – в EDX:EAX, результат – в EAX).

1.1.3 Команды преобразования знака:

* CDQ – обеспечивает преобразование двойного слова в учетверенное слово (копирует знаковый бит регистра ЕАХ на все биты регистра EDX).

1.1.4 Логические команды:

* SHR – логический (беззнаковый) сдвиг вправо (shr opr, cnt);
* SHL – логический (беззнаковый) сдвиг влево (shl opr, cnt).

Команды сдвига используются для умножения и деления на степени двойки: сдвиг влево на разрядов соответствует умножению на , сдвиг вправо – делению на .

1.1.5 Команды переходов:

* JE/JZ – переход по указанному адресу, если равно или при нуле в результате (je addr/jz addr);
* JMP – безусловный переход по указанному адресу (jmp addr).

1.1.6 Команды работы со стеком:

* PUSH – помещает содержимое источника в стек (push src);
* POP – помещает в приемник значение из вершины стека (pop dst). POP выполняет действие, полностью обратное PUSH.

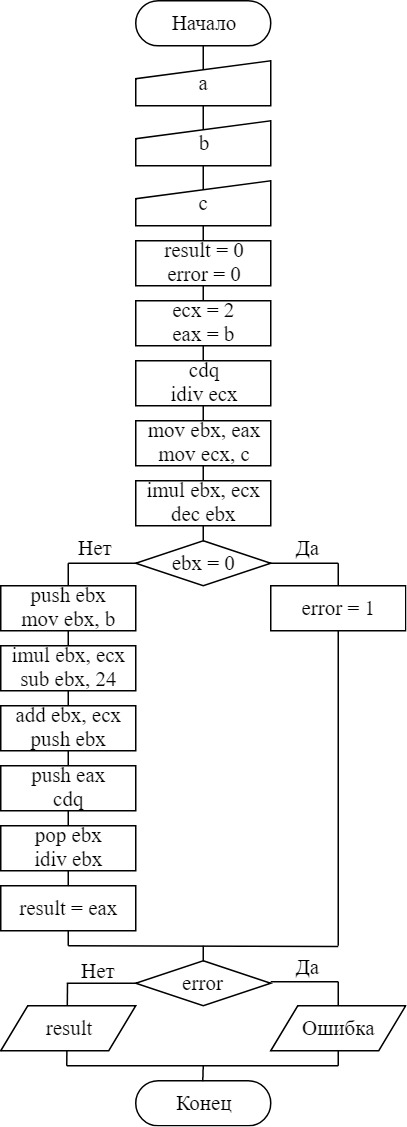
Команда PUSH практически всегда используется в паре с POP.

1.2 Задание

1. В программе необходимо реализовать функцию вычисления целочисленного выражения (с \* b – 24 + a) / (b / 2 \* c – 1) на встроенном ассемблере MASM в среде Microsoft Visual Studio на языке C++.
2. Значения переменных передаются в качестве параметров функции.
3. Результат выводить в консольном приложении (проект – консольное приложение Win32).
4. В программе реализовать ввод переменных из командной строки и вывод результата на экран.
5. Все параметры функции 32-битные числа (знаковые и беззнаковые).
6. Первые строки функции вычисления выражения заносят значения аргументов функции в соответствующие регистры.
7. Необходимо реализовать проверки вводимых данных и вычисления отдельных операций. Например, проверка деления на 0.
8. В качестве комментария к каждой строке необходимо указать, какой промежуточный результат, в каком регистре формируется.
9. По возможности использовать команды сдвига.

1.3 Схема алгоритма

На рисунке 1.1 приведена схема алгоритма вычисления целочисленного выражения (с \* b – 24 + a) / (b / 2 \* c – 1) в соответствии с заданием 1. Исходные данные (переменные a, b, c) вводятся пользователем. Происходит вычисление знаменателя (b / 2 \* c – 1), а затем, если знаменатель не равен 0, вычисляется числитель (с \* b – 24 + a), после чего программа выводит результат их деления.

  
Рисунок 1.1 – Схема алгоритма вычисления исходного выражения

1.4 Решение

#include <iostream>

using namespace std;

// функция вычисления выражения (c\*b - 24 + a)/((b/2)\*c - 1);

bool err = 0;

int calc(int a, int b, int c)

{

int result = 0;

\_\_asm {

mov ecx, 2 // ecx = 2

mov eax, b // eax = b

cdq

idiv ecx // eax = b/2

mov ebx, eax// ebx = eax

mov ecx, c // ecx = c

mov eax, a // eax = a

imul ebx, ecx //ebx = (b/2)\*c

dec ebx // ebx = (b/2)\*c - 1

je error;//проверка деления на 0

push ebx

mov ebx, b // ebx = b

imul ebx, ecx //ebx = c\*b

sub ebx, 24// ebx = c\*b - 24

add ebx, eax // ebx = c\*b - 24 + a

push ebx

pop eax // eax = c\*b - 24 + a

cdq

pop ebx // ebx = (b/2)\*c - 1

idiv ebx // eax = (c\*b - 24 + a)/((b/2)\*c - 1)

jmp exit\_1;

error:

mov err, 1;

exit\_1:

mov result, eax;//result = (c\*b - 24 + a)/((b/2)\*c - 1)

}

return result; // возвращаем результат вычисления выражения

}

int main()

{

setlocale(LC\_ALL, "Russian");

int a, b, c;

cout << "Гижевская Валерия 6113 Вариант 52" << endl;

cout << "(c\*b - 24 + a)/((b/2)\*c - 1)" << endl;

cout << "a = ";

cin >> a;

cout << "b = ";

cin >> b;

cout << "c = ";

cin >> c;

int resA = calc(a, b, c);

if (err == 1)

{

cout << "Ошибка! Деление на ноль\n";

}

else {

cout << "\nРезультат Assembler = " << resA << endl;

int resCpp = (c \* b - 24 + a) / ((b / 2) \* c - 1);

cout << "Результат C++ = " << resCpp << endl;

}

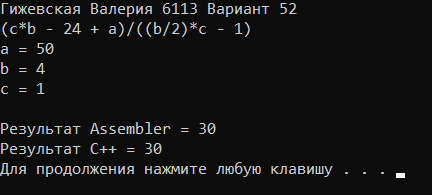
system("PAUSE");

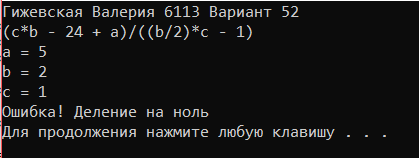
return 0;

}

1.5 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунке 1.2 и 1.3. При значениях аргумента a = 50, b = 4, c = 1 результат работы программы равен 30. При значениях аргумента b = 2, c = 1 программа выдает ошибку.

  
Рисунок 1.2 – Вычисление выражения при a = 50, b = 4, c = 1

  
Рисунок 1.3 – Вывод ошибки при делении на 0

Лабораторная работа 2 «Арифметические команды и команды переходов в ассемблере»

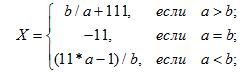
2.1 Теоретические основы лабораторной работы

В лабораторной работе были использованы команды языка ассемблера, приведенные в подразделе 1.1, а также следующие команды [2]:

* ADC – сложение с переносом (adc dst, src);
* INC – инкремент (увеличение на 1) значения (inc src);
* OR – логическое побитовое ИЛИ (or opr1, opr2);
* CMP – сравнение двух значений (регистр, область памяти, непосредственное значение) с установкой флагов (cmp opr1, opr2);
* JG – переход по указанному адресу, если первый из сравниваемых операндов больше второго (jg addr);
* JL – переход по указанному адресу, если первый из сравниваемых операндов меньше второго (jl addr);
* JO – переход по указанному адресу, если есть переполнение (jo addr).

2.2 Задание

1. В программе необходимо реализовать функцию вычисления заданного условного целочисленного выражения, используя команды сравнения, условного и безусловного переходов на встроенном ассемблере.



1. Результат – целочисленный, возвращается из функции в регистре eax.
2. Значения переменных передаются в качестве параметров функции.
3. В программе реализовать вывод результата на экран.
4. Все параметры функции 32-битные числа.
5. Проверку деления на 0 реализовать также на встроенном ассемблере.
6. В качестве комментария к каждой строке необходимо указать, какой промежуточный результат, в каком регистре формируется.
7. По возможности использовать команды сдвига.

2.3 Схема алгоритма

На рисунке 2.1 приведена схема алгоритма вычисления выражения. В начале работы все исходные данные вводятся пользователем в консоли и записываются в регистры. Далее происходит сравнение переменных между собой и проверка на ноль в зависимости от результата сравнения. После чего выполняются вычисления значения выражения.

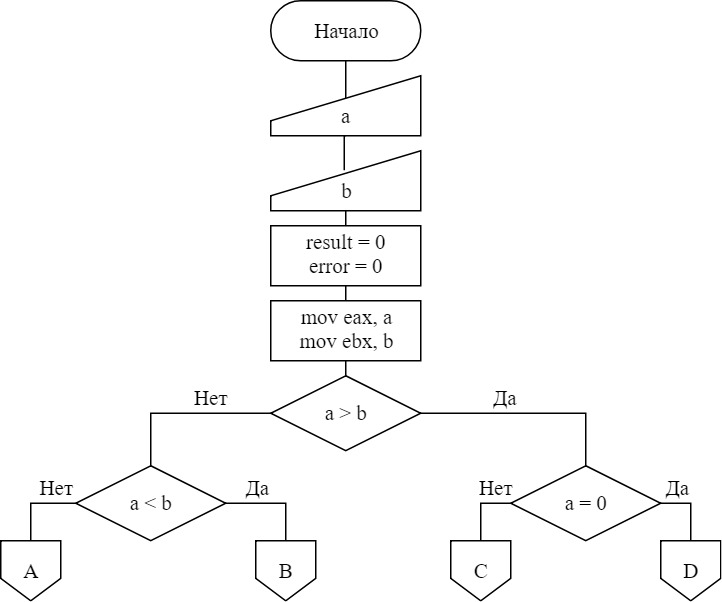


Рисунок 2.1 – Схема алгоритма вычисления исходного условного выражения (начало)

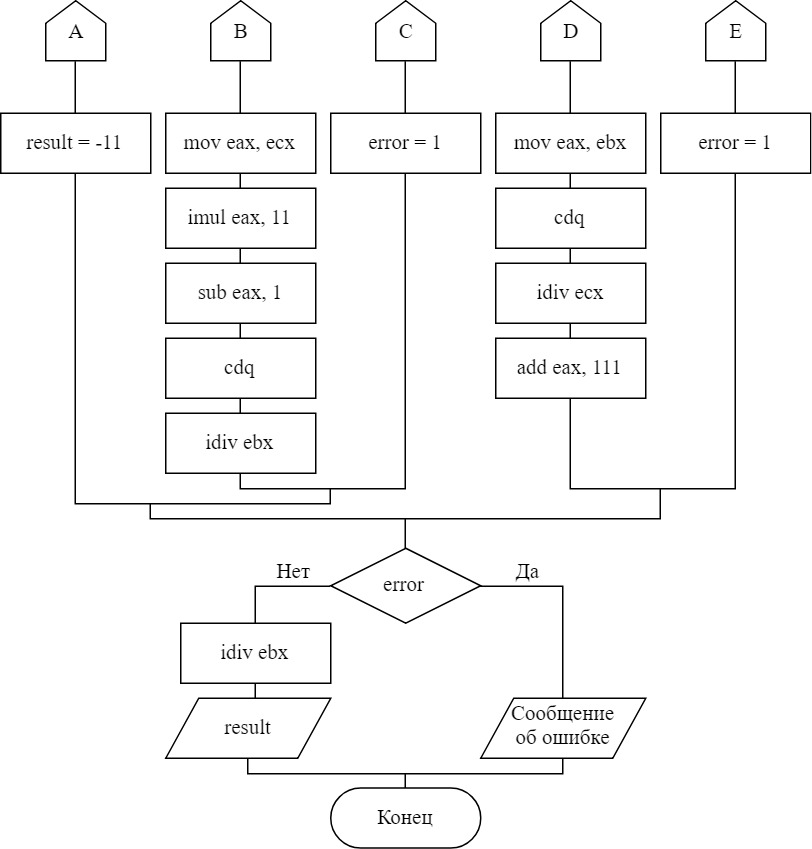


Рисунок 2.1 – Схема алгоритма вычисления исходного условного выражения (продолжение)

2.4 Решение

#include <iostream>

#include <stdio.h> // стандартный ввод/вывод

using namespace std;;

bool err = 0;

int calc(int a, int b) {

int res = 0;

\_\_asm {

mov ecx, a; ecx = a

mov ebx, b; ebx = b

cmp ecx, ebx; сравнение a и b

jg l\_bigger; переход если a > b

jl l\_smaller; переход если a < b

mov eax, -11; eax = -11

jmp exit\_1; переход на конец программы

l\_bigger :

or ecx, ecx; сравнение a и 0

je error; ошибка деление на ноль

mov eax, ebx; eax = b

cdq; подготовка деления

idiv ecx; eax = b / a

add eax, 111; eax = a / b + 111

jmp exit\_1; переход на конец программы

l\_smaller :

or ebx, ebx; сравнение b и 0

je error; ошибка деление на ноль

mov eax, ecx; eax = a

imul eax, 11; eax = a \* 11

sub eax, 1; eax = 11 \* a - 1

cdq

idiv ebx; eax = (11 \* a - 1) / b

jmp exit\_1

error : mov err, 1

exit\_1 :

mov res, eax; res = eax

}

return res;

}

void calc\_cpp(int a, int b) {

if (a > b) {

if (a == 0) cout << "\n Результат С++: ошибка!\n";

else cout << "\n Результат С++: " << b / a + 111 << endl;

}

else if (a == b) {

cout << "\n Результат С++: " << -11 << endl;

}

else {

if (b == 0) cout << "\n Результат С++: ошибка!\n";

else cout << "\n Результат С++: " << (11 \* a - 1) / b << endl;

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

int a, b;

bool m = true;

while (m) {

err = 0;

cout << "Гижевская Валерия \n Группа 6113 \n Лабораторная работа №2 \n Вариант 52\n" << endl;

cout << " | b/a + 111, a > b" << endl;

cout << " X = | -11, a = b" << endl;

cout << " | (11\*a - 1)/b, a < b" << endl;

cout << endl;

cout << " a = ";

cin >> a;

cout << " b = ";

cin >> b;

int res = calc(a, b);

if (err == 1)

cout << "\n Ошибка! Деление на ноль." << endl;

else

cout << "\n Результат: X = " << res << endl;

calc\_cpp(a, b);

cout << endl;

system("PAUSE");

system("cls");

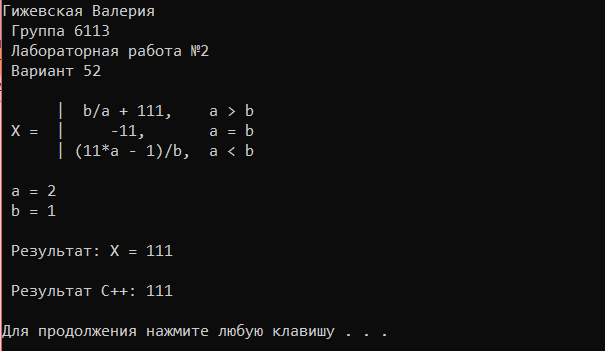
}

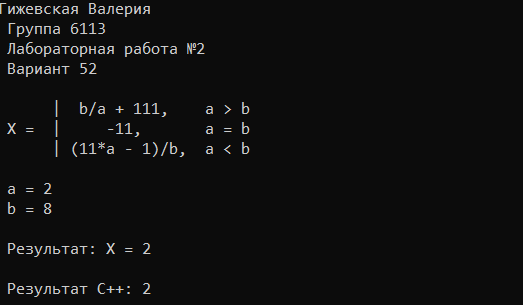
return 0;

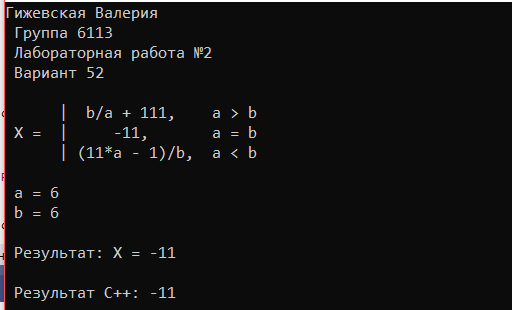
}

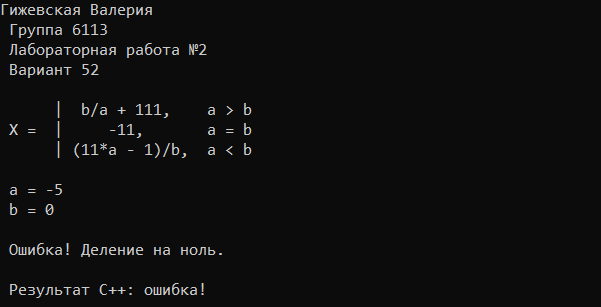
2.5 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 2.2 – 2.5.

  
Рисунок 2.2 – Вычисление выражения при a > b

  
Рисунок 2.3 – Вычисление выражения при a < b

  
Рисунок 2.4 – Вычисление выражения при a = b

  
Рисунок 2.5 – Вывод ошибки при делении на 0

Лабораторная работа 3 «Команды работы с массивами и стеком»

3.1 Теоретические основы лабораторной работы

В лабораторной работе были использованы команды языка ассемблера, приведенные в подразделе 1.1, а также следующие команды [3]:

* XOR ­– исключающее побитовое ИЛИ двух операндов (xor opr1, opr2);
* CMP – сравнение двух значений (регистр, область памяти, непосредственное значение) с установкой флагов (cmp opr1, opr2);
* JGE – переход по указанному адресу, если первый из сравниваемых операндов больше или равен второму (jge addr);
* JLE – переход по указанному адресу, если первый из сравниваемых операндов меньше или равен второму (jle addr);
* JECXZ – переход по указанному адресу, если значение в регистре ECX равно 0 (jecxz addr);
* LOOP – уменьшение счетчика (регистр ECX) на 1 и переход по указанному адресу, если значение в ECX не равно 0 (используется для организации цикла).

3.2 Задание

1. В программе необходимо реализовать функцию обработки элементов массива используя команды сравнения, переходов и циклов на встроенном ассемблере.
2. Результат – целочисленный, возвращается из функции регистре eax.
3. Массив передаётся в качестве параметра функции.
4. В программе реализовать вывод результата на экран.
5. В качестве комментария к каждой строке необходимо указать, какое действие выполняет команда относительно массива.

**Условие****:** *сформировать новый массив из одномерного массива A={a[i]} целых чисел по следующему правилу:* .

3.3 Схема алгоритма

На рисунке 3.1 приведена схема алгоритма получения нового массива из исходного по данному в задании правилу. Исходные данные вводятся пользователем с консоли. Далее находятся сумма нечётных элементов и максимальный элемент введенного массива. Затем формируется результирующий массив из исходного на основании введенных данных, который после этого выводится на консоль.



Рисунок 3.1 – Схема алгоритма получения нового массива по заданному правилу (начало)



Рисунок 3.1 – Схема алгоритма получения нового массива по заданному правилу (продолжение)



Рисунок 3.1 – Схема алгоритма получения нового массива по заданному правилу (продолжение)



Рисунок 3.1 – Схема алгоритма получения нового массива по заданному правилу (окончание)

3.4 Решение

#include <iostream>

#include <stdio.h> // стандартный ввод/вывод

#include <iostream>

using namespace std;

int\* calculator(int mas[], int n)

{

int\* result = new int[n];

bool excep = 0;

\_\_asm {

//запускаем цикл по массиву, чтобы найти сумму отрицательных и минимальный элемент

mov esi, mas; esi указывает на начало массива mas[]

mov ebx, 0; промежуточный результат суммы

mov ecx, n; счётчик цикла по всем элементам массива

jcxz success; перейти к мтеик success если длина массива ноль

mov eax, -100000; определяем первый элемент(далее - максимальный)

begin1\_loop:

mov edx, [esi]; определяем текущий элемент массива mas[]

//сумма нечетных элементов

test edx, 1

jnz nechet;

jz onward; четное, переход на метку onward

nechet :

add ebx, edx; <ebx> = <ebx> +<edx>

jmp end1\_loop;

onward:

//максимальный элемент

cmp edx, eax; сравниваем текущий элемент с max

jl end1\_loop; если текущий элемент меньше максимального, переходим к метке end1\_loop

mov eax, edx; иначе присваиваем новый максимальный элемент

end1\_loop :

add esi, 4; переходим к следующему элементу массива mas[]

loop begin1\_loop; повторяем цикл для всех элементов массива

add eax, 0; проверяем максимальный элемент на равенство нулю

jz error; переходим к метке error, если знаменатель равен нулю

push eax; загружаем mах элемент в стек

push ebx; загружаем сумму нечетных элементов в стек

//запускаем цикл по массиву, чтобы получить новый массив

mov esi, mas; esi указывает на начало массива mas[]

mov edi, result; edi указывает на начало массива result[]

mov ecx, n; счётчик цикла по всем элементам массива

pop ebx; извлекаем из вершины стека сумму нечетных элементов в <ebx>

pop n; извлекаем из вершины стека максимальный элемент в переменную n

begin3\_loop :

mov eax, [esi]; определяем текущий элемент массива mas[]

add eax, ebx; <eax> = (a[i] + сумма нечетных элементов)

cdq; подготовка к делению

idiv n; <eax> = (a[i] + сумма нечетных элементов) / максимальный элемент

mov[edi], eax; записываем полученный элемент в массив result[]

end3\_loop:

add esi, 4; переходим к следующему элементу массива mas[]

add edi, 4; переходим к следующему элементу массива result[]

loop begin3\_loop; повторяем цикл для всех элементов массива

jmp success; переход без условия

error :

mov excep, 1;

success:

}

if (excep) throw exception("Попытка деления на ноль!");

return result;

}

void main()

{

while (true) {

system("cls");

int n;

setlocale(LC\_ALL, "");

cout << "Гижевская Лера\nГруппа 6113\nВариант 52" << endl;

cout << "Задание:" << endl;

cout << "Сформировать новый массив из одномерного массива A={a[i]} целых чисел по следующему правилу:" << endl << endl;

cout << "a[i] = (a[i] + сумма нечетных элементов)/максимальный элемент" << endl;

cout << "Введите длину массива: ";

cin >> n;

int\* mas1 = new int[n];

int\* mas2 = new int[n];

int\* mas3 = new int[n];

cout << endl;

cout << "Введите элементы массива: " << endl;

for (int i = 0; i < n; i++) { cin >> mas1[i]; mas3[i] = mas1[i]; }

cout << endl;

try {

mas2 = calculator(&\*mas1, n);

cout << "Assembler : ";

for (int i = 0; i < n; i++) cout << mas2[i] << " ";

}

catch (exception e) { cout << e.what(); }

cout << "\nРезультат С++ : ";

int sum = 0;

bool f = false;

int max;

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

if (mas3[i] % 2 == 0) {

if (!f) {

f = true;

max = mas3[i];

}

if (mas3[i] > max)

{

max = mas3[i];

}

}

else sum += mas3[i];

}

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

if (max != 0) {

mas3[i] = (mas3[i] + sum) / max;

cout << mas3[i] << " ";

}

else cout << "Деление на ноль!" << endl;

}

cout << "\nСумма нечетных элементов : " << sum << endl;

cout << "\nМаксимальный элемент : " << max << endl;

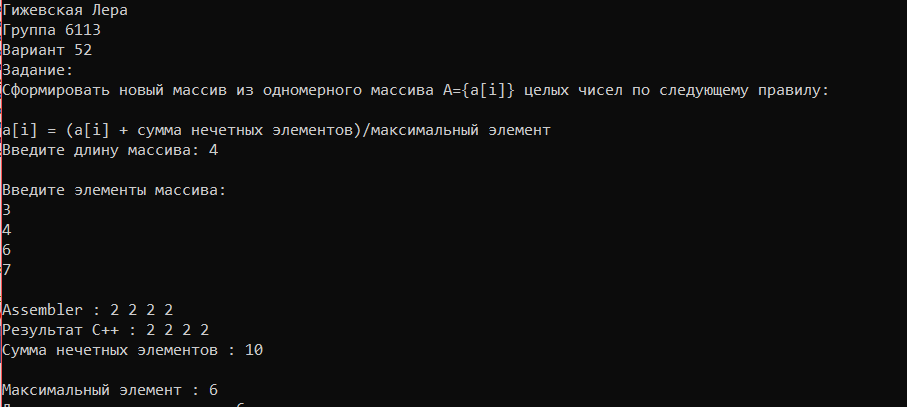
system("PAUSE");

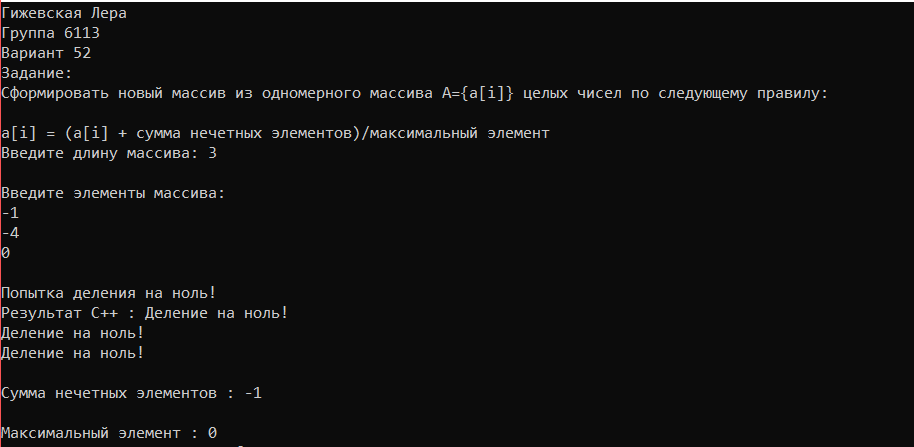
}

}

3.5 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 3.2 – 3.4.

  
Рисунок 3.2 – Выполнение программы с массивом из 4 элементов

  
Рисунок 3.3 – Вывод ошибки при делении на 0

Лабораторная работа 4 «Изучение работы математического сопроцессора в среде Assembler»

4.1 Теоретические основы лабораторной работы

4.1.1 Команда MOV [1]

MOV копирует значение из источника в приемник (mov dst, src).

4.1.2 Команды переходов:

* JA – переход по указанному адресу, если первый из сравниваемых операндов больше второго (ja addr);
* JB – переход по указанному адресу, если первый из сравниваемых операндов меньше второго (jb addr);
* JE/JZ – переход по указанному адресу, если равно или при нуле в результате (je addr/jz addr);
* JMP – безусловный переход по указанному адресу (jmp addr).

4.1.3 Команды управления сопроцессором [4]:

* FINIT – инициализация сопроцессора: установка начальных значений некоторых регистров FPU.

4.1.4 Команды передачи данных [4]:

* FLD – загрузка вещественного значения в вершину стека (fld src);
* FLD1 – загрузка в вершину стека константы 1 (fld1);
* FXCH – обмен значений между ST(0) и регистром, указанным операндом (fxch st(i)).

4.1.5 Арифметические команды [4]:

* FADDP – сложение с выталкиванием (faddp st(i), st; результат в ST(i), ST выталкивается);
* FSUBP – вычитание с выталкиванием (fsubp st(i), st; результат в ST(i), ST выталкивается);
* FSUBRP – реверсивное вычитание с выталкиванием (fsubrp st(i), st; вычитается ST(i) из ST, результат в ST(i), ST выталкивается);
* FMUL – умножение (fmul st, st(i) или fmul st(i), st; fmul без операндов – умножение ST(1) на ST(0) с выталкиванием);
* FMULP – умножение с выталкиванием (fmulp st(i), st; результат в ST(i), ST выталкивается);
* FDIVRP – реверсивное деление с выталкиванием (fdivrp st(i), st; делится ST на ST(i), результат в ST(i), ST выталкивается);
* FSQRT – вычисление квадратного корня от ST(0) (fsqrt).

4.1.6 Команды трансцендентных функций [4]:

* FCOS – вычисление косинуса от ST(0) (fcos);
* FPATAN – вычисление частичного арктангенса (fpatan; вычисляется арктангенс от ST(1)/ST(0), результат в ST(1), ST выталкивается).

4.1.7 Команды вещественного сравнения [4]:

* FCOMI – вещественное сравнение с установкой флагов основного процессора (fcomi st, st(i));
* FCOMIP – вещественное сравнение с выталкиванием и установкой флагов основного процессора (fcomi st, st(i); ST выталкивается).

4.2 Задание

1. В программе необходимо реализовать функцию вычисления заданного условного выражения на языке ассемблера с использованием команд арифметического сопроцессора.
2. Значения переменных передаются в качестве параметров функции.
3. В программе реализовать вывод результата на экран.
4. Все параметры функции имеют тип double.
5. Проверку деления на 0 реализовать также на встроенном ассемблере.
6. В качестве комментария к каждой строке необходимо указать, какой промежуточный результат, в каком регистре формируется.
7. В качестве комментария к строкам, содержащим команды сопроцессора, необходимо указать состояние регистров сопроцессора.
8. Результат можно возвращать из функции в вершине стека сопроцессора.

4.3 Схема алгоритма

На рисунке 4.1 приведена схема алгоритма вычисления условного выражения. В начале пользователь вводит с консоли значения переменных a и b, после чего они сравниваются и в зависимости от результата сравнения вычисляется соответствующее выражение. Результат вычисления возвращается из функции и выводится на консоль.

Рисунок 4.1 – Схема алгоритма вычисления исходного условного выражения (начало)

4.4 Решение

#include <iostream>

#include <math.h>

using namespace std;

double func(double a, double b)

{

double res;

const int c2 = 2;

const int c5 = 5;

const int c3 = 3;

\_\_asm

{

// st0 st1 st2 st3 st4

finit // инициализация сопроцессора

fld qword ptr[a] // a

fld qword ptr[b] // b a

fcomi st, st(1) // сравнение a и b

jb a\_bigger // a > b

ja b\_bigger // a < b

fadd // a + b

fld st // a + b a + b

fmul st, st // (a + b)^2 a + b

fmul // (a + b)^3

fild c2 // 2 (a + b)^3

fsqrt // sqrt2 (a + b)^3

fmul // sqrt2\*(a + b)^3

jmp end\_

a\_bigger :

fxch st(1) // a b

fldl2e // log2e a b

fmul // a\*log2e b

fld st // a\*log2e a\*log2e b

frndint // [a\*log2e] a\*log2e b

fsub st(1), st // [a\*log2e] {a\*log2e} b

fxch st(1) // {a\*log2e} [a\*log2e] b

f2xm1 // 2^({a\*log2e})-1 [a\*log2e] b

fld1 // 1 2^({a\*log2e})-1 [a\*log2e] b

faddp st(1), st // 2^({a\*log2e}) [a\*log2e] b

fscale // e^a [a\*log2e] b

fstp st(1) // e^a b

fild c3 // 3 e^a b

fsubp st(1), st // e^a-3 b

fxch st(1) // b e^a-3

fldl2e // log2e b e^a-3

fmul // b\*log2e e^a-3

fld st // b\*log2e b\*log2e e^a-3

frndint // [b\*log2e] b\*log2e e^a-3

fsub st(1), st // [b\*log2e] {b\*log2e} e^a-3

fxch st(1) // {b\*log2e} [b\*log2e] e^a-3

f2xm1 // 2^({b\*log2e})-1 [b\*log2e] e^a-3

fld1 // 1 2^({b\*log2e})-1 [b\*log2e] e^a-3

faddp st(1), st // 2^({b\*log2e}) [b\*log2e] e^a-3

fscale // e^b [b\*log2e] e^a-3

fstp st(1) // e^b e^a-3

fimul c2 // 2\*e^b e^a-3

fild c5 // 5 2\*e^b e^a-3

faddp st(1), st // 2\*e^b+5 e^a-3

fxch st(1) // e^a-3 2\*e^b+5

fdiv // (2\*e^b+5)/(e^a-3)

jmp end\_

b\_bigger :

fxch st(1) // a b

fstp st(1) // a

fld st // a a

fldl2t // log2(10) a a

fmul // a\*log2(10) b

fld st // a\*log2(10) a\*log2(10) a

frndint // [a\*log2(10)] a\*log2(10) a

fsub st(1), st // [a\*log2(10)] {a\*log2(10)} a

fxch st(1) // {a\*log2(10)} [a\*log2(10)] a

f2xm1 // 2^({a\*log2(10)})-1 [a\*log2(10)] a

fld1 // 1 2^({a\*log2(10)})-1 [a\*log2(10)] a

faddp st(1), st // 2^({a\*log2(10)}) [a\*log2(10)] a

fscale // 10^a [a\*log2(10)] a

fstp st(1) // 10^a a

fxch st(1) // a 10^a

fild c5 // 5 a 10^a

fsub // a-5 10^a

fsqrt // sqrt(a-5) 10^a

fadd // sqrt(a-5) + 10^a

jmp end\_

end\_ :

fstp res

}

return res;

}

double Check(double a, double b)

{

double y;

if (a == b) {

y = (sqrt(2)) \* (pow(a + b, 3));

}

else if (a > b) {

y = (2 \* exp(b) + 5) / (exp(a) - 3);

}

else {

y = sqrt(a - 5) + pow(10, a);

}

return y;

}

int main()

{

setlocale(LC\_ALL, "Russian");

double a, b;

bool m = true;

while (m)

{

cout.setf(ios::fixed);

cout << " Гижевская Лера \n Группа 6113 \n Лабораторная работа №4 \n Вариант 52" << endl;

cout << " Задание:\n" << endl;

cout << " |sqrt(5a-ln(b+1)), если a > b " << endl;

cout << " x = {(2\*e^b+5)/(e^a-3),если a = b " << endl;

cout << " |sqrt(a-5) + 10^a, если a < b " << endl;

cout << endl;

cout << " a = ";

cin >> a;

cout << " b = ";

cin >> b;

double assembler = func(a, b);

double prov = Check(a, b);

cout << " Результат : x = " << assembler << endl;

cout << " Проверка C++ : x = " << prov << endl;

system("PAUSE");

system("cls");

}

return 0;

}

4.5 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 4.2 – 4.5.

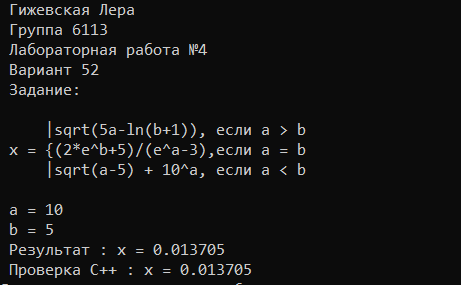


Рисунок 4.2 – Вычисление выражения при a > b

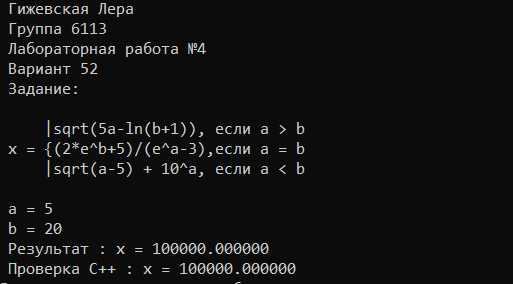


Рисунок 4.3 – Вычисление выражения при a < b

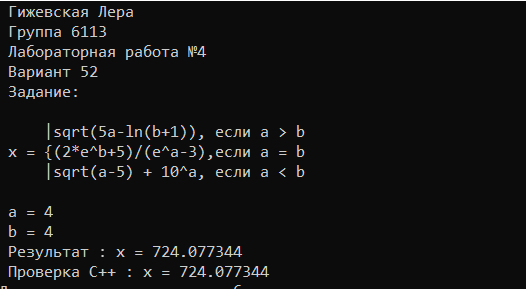


Рисунок 4.4 – Вычисление выражения при a = b

Лабораторная работа 5 «Нахождение корней уравнения методом Ньютона на языке ассемблера»

5.1 Теоретические основы лабораторной работы

5.1.1 Команды языка ассемблера

В лабораторной работе были использованы команды языка ассемблера, приведенные в подразделе 4.1, а также следующие команды [4]:

* XOR ­– исключающее побитовое ИЛИ (xor opr1, opr2);
* CMP – сравнение двух значений (регистр, область памяти, непосредственное значение) с установкой флагов (cmp opr1, opr2);
* DEC – декремент, уменьшение значения операнда на 1 (dec src);
* JL – переход по указанному адресу, если первый из сравниваемых операндов меньше второго (jl addr);
* JLE – переход по указанному адресу, если первый из сравниваемых операндов меньше или равен второму (jle addr);
* FILD – загрузка целочисленного значения в вершину стека (fild src);
* FLDZ – загрузка в вершину стека константы 0 (fldz);
* FSTP – копирование значения из ST(0) в ячейку стека или область памяти, указанную операндом, с выталкиванием ST(0) (fstp dst);
* FSUB – вычитание (fsub st, st(i) или fsub st(i), st; fsub без операндов – вычитание ST(0) из ST(1) с выталкиванием);
* FDIVP – деление с выталкиванием (fdivp st(i), st; результат в ST(i), ST выталкивается);
* FABS – абсолютная величина (модуль) ST(0) (fabs).

5.1.2 Метод Ньютона поиска корней полиномиального уравнения

Метод Ньютона или касательных заключается в том, что если – некоторое приближение к корню уравнения , то следующее приближение определяется как корень касательной к функции в точке .

Уравнение касательной к функции в точке имеет вид:

В уравнении касательной положим и .

Тогда алгоритм последовательных вычислений состоит в следующем:

5.2 Задание

1. В программе необходимо найти с заданной точностью корень уравнения методом Ньютона на языке ассемблера с использованием команд арифметического сопроцессора.
2. Значения переменных передаются в качестве параметров функции.
3. Составить таблицу расчетов корня уравнения на заданном отрезке и вывести на экран.
4. Все параметры уравнения имеют тип double.
5. Проверку деления на 0 реализовать также на встроенном ассемблере.
6. Если на заданном интервале не найден корень уравнения, то вывести соответствующее сообщение.
7. В качестве комментария к каждой строке необходимо указать, какой промежуточный результат, в каком регистре формируется.
8. В качестве комментария к строкам, содержащим команды сопроцессора, необходимо указать состояние регистров сопроцессора.
9. Результат можно возвращать из функции в вершине стека сопроцессора.

**Условие:** 𝑓(𝑥) = −5 + 30 𝑥 + 39 𝑥2 − 39 𝑥3 − 29 𝑥4 − 47 𝑥5 + 25 𝑥9 + 𝑥10

5.3 Схема алгоритма

На рисунке 5.1 приведена схема алгоритма вычисления корней уравнения методом Ньютона. В начале пользователь вводит с консоли интервал и точность. Затем программа высчитывает корни и выводит на консоль в виде таблицы.

Рисунок 5.1 – Схема алгоритма нахождения корней уравнения методом Ньютона

5.4 Решение

#include <iostream>

using namespace std;

double f(double a, double b, double eps)

{

const int c5 = 5;

const int c30 = 30;

const int c39 = 39;

const int c29 = 29;

const int c47 = 47;

const int c25 = 25;

const int c78 = 78;

const int c117 = 117;

const int c116 = 116;

const int c235 = 235;

const int c225 = 225;

const int c10 = 10;

int error = 0;

\_\_asm {

// st(0) || st(1) || st(2) || st(3) || st(4)

finit; //инициализация сопроцессора

fld a; //a

begin\_loop:

//вычисление f(x)

fld st; //x || x

fmul st, st; //x ^ 2 || x

fmul st, st; //x ^ 4 || x

fmul st, st; //x ^ 8 || x

fmul st, st(1); //x ^ 9 || x

fmul st, st(1); //x ^ 10 || x

fld st(1); //x || x ^ 10 || x

fmul st, st; //x ^ 2 || x ^ 10 || x

fmul st, st; //x ^ 4 || x ^ 10 || x

fmul st, st; //x ^ 8 || x ^ 10 || x

fld st(2); //x || x ^ 8 || x ^ 10 || x

fmulp st(1), st; //x ^ 9 || x ^ 10 || x

fild c25; //25 || x ^ 9 || x ^ 10 || x

fmulp st(1), st; //25x ^ 9 || x ^ 10 || x

faddp st(1), st; //x ^ 10 + 25x ^ 9 || x

fld st(1); //x || x ^ 10 + 25x ^ 9 || x

fmul st, st; //x ^ 2 || x ^ 10 + 25x ^ 9 || x

fmul st, st; //x ^ 4 || x ^ 10 + 25x ^ 9 || x

fld st(2); //x || x ^ 4 || x ^ 10 + 25x ^ 9 || x

fmulp st(1), st; //x ^ 5 || x ^ 10 + 25x ^ 9 || x

fild c47; //47 || x ^ 5 || x ^ 10 + 25x ^ 9 || x

fmulp st(1), st; //47x ^ 5 || x ^ 10 + 25x ^ 9 || x

fsubp st(1), st; //x ^ 10 + 25x ^ 9 - 47x ^ 5 || x

fld st(1); //x || x ^ 10 + 25x ^ 9 - 47x ^ 5 || x

fmul st, st; //x ^ 2 || x ^ 10 + 25x ^ 9 - 47x ^ 5 || x

fmul st, st; //x ^ 4 || x ^ 10 + 25x ^ 9 - 47x ^ 5 || x

fild c29; //29 || x ^ 4 || x ^ 10 + 25x ^ 9 - 47x ^ 5 || x

fmulp st(1), st; //29x ^ 4 x ^ 10 + 25x ^ 9 - 47x ^ 5 || x

fsubp st(1), st; //x ^ 10 - ... - 29x ^ 4 || x

fld st(1); //x || x ^ 10 - ... - 29x ^ 4 || x

fmul st, st; //x ^ 2 || x ^ 10 - ... - 29x ^ 4 || x

fld st(2); //x || x ^ 2 || x ^ 10 - ... - 29x ^ 4 || x

fmulp st(1), st; //x ^ 3 || x ^ 10 - ... - 29x ^ 4 || x

fild c39; //39 || x ^ 3 || x ^ 10 - ... - 29x ^ 4 || x

fmulp st(1), st; //39x ^ 3 || x ^ 10 - ... - 29x ^ 4 || x

fsubp st(1), st; //x ^ 10 - ... - 39x ^ 3 || x

fld st(1); //x || x ^ 10 - ... - 39x ^ 3 || x

fmul st, st; //x ^ 2 || x ^ 10 - ... - 39x ^ 3 || x

fild c39; //39 || x ^ 2 || x ^ 10 - ... - 39x ^ 3 || x

fmulp st(1), st; //39x ^ 2 || x ^ 10 - ... - 39x ^ 3 || x

faddp st(1), st; //x ^ 10 - ... + 39x ^ 2 || x

fld st(1); //x || x ^ 10 - ... + 39x ^ 2 || x

fild c30; //30 || x || x ^ 10 - ... + 39x ^ 2 || x

fmulp st(1), st; //30x || x ^ 10 - ... + 39x ^ 2 || x

faddp st(1), st; //x ^ 10 - ... + 30x || x

fild c5; //5 || x ^ 10 - ... + 30x || x

fsubp st(1), st; //x ^ 10 - ... - 5 || x

//вычисление f'(x)

fld st(1); //x || x ^ 10 - ... - 5 || x

fmul st, st; //x ^ 2 || x ^ 10 - ... - 5 || x

fmul st, st; //x ^ 4 || x ^ 10 - ... - 5 || x

fmul st, st; //x ^ 8 || x ^ 10 - ... - 5 || x

fmul st, st(2); //x ^ 9 || x ^ 10 - ... - 5 || x

fild c10; //10 || x ^ 9 || x ^ 10 - ... - 5 || x

fmulp st(1), st; //10x ^ 9 || x ^ 10 - ... - 5 || x

fld st(2); //x || 10x ^ 9 || x ^ 10 - ... - 5 || x

fmul st, st; //x ^ 2 || 10x ^ 9 || x ^ 10 - ... - 5 || x

fmul st, st; //x ^ 4 || 10x ^ 9 || x ^ 10 - ... - 5 || x

fmul st, st; //x ^ 8 || 10x ^ 9 || x ^ 10 - ... - 5 || x

fild c225; //225 || x ^ 8 || 10x ^ 9 || x ^ 10 - ... - 5 || x

fmulp st(1), st; //225x ^ 8 || 10x ^ 9 || x ^ 10 - ... - 5 || x

faddp st(1), st; //10x ^ 9 + 225x ^ 8 || x ^ 10 - ... - 5 || x

fld st(2); //x || 10x ^ 9 + 225x ^ 8 || x ^ 10 - ... - 5 || x

fmul st, st; //x ^ 2 || 10x ^ 9 + 225x ^ 8 || x ^ 10 - ... - 5 || x

fmul st, st; //x ^ 4 || 10x ^ 9 + 225x ^ 8 || x ^ 10 - ... - 5 || x

fild c235; //235 || x ^ 4 || 10x ^ 9 + 225x ^ 8 || x ^ 10 - ... - 5 || x

fmulp st(1), st; //235x ^ 4 || 10x ^ 9 + 225x ^ 8 || x ^ 10 - ... - 5 || x

fsubp st(1), st; //10x ^ 9 - ... - 235x ^ 4 || x ^ 10 - ... - 5 || x

fld st(2); //x || 10x ^ 9 - ... - 235x ^ 4 || x ^ 10 - ... - 5 || x

fld st; //x || x || 10x ^ 9 - ... - 235x ^ 4 || x ^ 10 - ... - 5 || x

fmul st, st; //x ^ 2 || x || 10x ^ 9 - ... - 235x ^ 4 || x ^ 10 - ... - 5 || x

fmulp st(1), st; //x ^ 3 || 10x ^ 9 - ... - 235x ^ 4 || x ^ 10 - ... - 5 || x

fild c116; //116 || x ^ 3 || 10x ^ 9 - ... - 235x ^ 4 x ^ 10 - ... - 5 || x

fmulp st(1), st; //116x ^ 3 || 10x ^ 9 - ... - 235x ^ 4 x ^ 10 - ... - 5 || x

fsubp st(1), st; //10x ^ 9 - ... - 116x ^ 3 || x ^ 10 - ... - 5 || x

fld st(2); //x || 10x ^ 9 - ... - 116x ^ 3 || x ^ 10 - ... - 5 || x

fmul st, st; //x ^ 2 || 10x ^ 9 - ... - 116x ^ 3 || x ^ 10 - ... - 5 || x

fild c117; //117 || x ^ 2 || 10x ^ 9 - ... - 116x ^ 3 x ^ 10 - ... - 5 || x

fmulp st(1), st; //117x ^ 2 || 10x ^ 9 - ... - 116x ^ 3 x ^ 10 - ... - 5 || x

fsubp st(1), st; //10x ^ 9 - ... - 117x ^ 2 || x ^ 10 - ... - 5 || x

fld st(2); //x || 10x ^ 9 - ... - 117x ^ 2 || x ^ 10 - ... - 5 || x

fild c78; //78 || x || 10x ^ 9 - ... - 117x ^ 2 || x ^ 10 - ... - 5 || x

fmulp st(1), st; //78x || 10x ^ 9 - ... - 117x ^ 2 || x ^ 10 - ... - 5 || x

faddp st(1), st; //10x ^ 9 - ... + 78x || x ^ 10 - ... - 5 || x

fild c30; //30 || 110x ^ 9 - ... + 78x || x ^ 10 - ... - 5 || x

faddp st(1), st; //10x ^ 9 - ... + 30 || x ^ 10 - ... - 5 || x

//вычисление f(x) / f'(x)

fdivp st(1), st; //f(x) / f'(x) x

fsub st(1), st; //f(x) / f'(x) x - f(x) / f'(x)

fabs; //| f(x) / f'(x)| x - f(x) / f'(x)

fld eps; //eps | f(x) / f'(x)| x - f(x) / f'(x)

fcomip st, st(1);//сравниваем eps и | f(x) / f'(x)| с выталкиванием и установкой флагов

fxch st(1); //x - f(x) / f'(x) |f(x) / f'(x) |

fstp st(1); //x - f(x) / f'(x) копирование значения из st в st(1) с выталкиванием

ja exit\_p; //переход к выходу из программы, если | f(x) / f'(x)| < eps

fld b; //b || x - f(x) / f'(x)

fld a; //a || b || x - f(x) / f'(x)

main\_p:

fcomip st, st(2); //сравнение x - f(x) / f'(x) с минимальной границей

ja err; //переход к метке error, если min{ a, b } > x - f(x) / f'(x)

fcomip st, st(1);//сравнение x - f(x) / f'(x) с максимальной границей

jb err; //переход к метке error, если max{ a, b } < x - f(x) / f'(x)

jmp begin\_loop; //переход к метке begin\_loop;

err:

mov error, 1

exit\_p :

}

if (error == 1)

{

throw exception("\nНа заданном промежутке корень не найден.");

}

}

double calc(double a, double b, double eps)

{

double f, d, res = 1;

double x = a;

for (int i = 1; abs(res) > eps; i++)

{

f = pow(x, 10) + 25 \* pow(x, 9) - 47 \* pow(x, 5) - 29 \* pow(x, 4) - 39 \* pow(x, 3) + 39 \* pow(x, 2) + 30 \* x - 5;

d = 10 \* pow(x, 9) + 225 \* pow(x, 8) - 235 \* pow(x, 4) - 116 \* pow(x, 3) - 117 \* pow(x, 2) + 78 \* x + 30;

res = f / d;

printf("%2d%12.5f%20.4f%20.4f%15.10f\n", i, x, f, d, abs(res));

x -= res;

}

return x;

}

int main()

{

setlocale(LC\_ALL, "");

cout << "Лабораторная работа №5\nВыполнила: Гижевская Валерия\nГруппа: 6113, вариант 52\n";

cout << "\nЗадание: f(x) = x^10 + 25x^9 - 47x^5 - 29x^4 - 39x^3 + 39x^2 +30x - 5" << endl;

cout << " f'(x) = 10x^9 + 225x^8 - 235x^4 - 116x^3 - 117x^2 + 78x + 30\n" << endl;

cout << "Введите границы промежутка:\n";

double a, b, eps;

cout << "a = ";

cin >> a;

cout << "b = ";

cin >> b;

cout << endl;

cout << "Введите погрешность: ";

cin >> eps;

try

{

cout << "\nРезультат на ассемблере: " << f(a, b, eps) << endl << endl;

cout << "Таблица расчётов корня на промежутке[a, b] уравнения" << endl;

printf("%2s%12s%20s%20s%15s\n", "№", "x", "f(x)", "f'(x)", "Погрешность");

calc(a, b, eps);

cout << endl;

}

catch (exception & e)

{

cout << e.what() << endl << endl;

}

system("PAUSE");

return 0;

}

5.5 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 5.2 – 5.4.

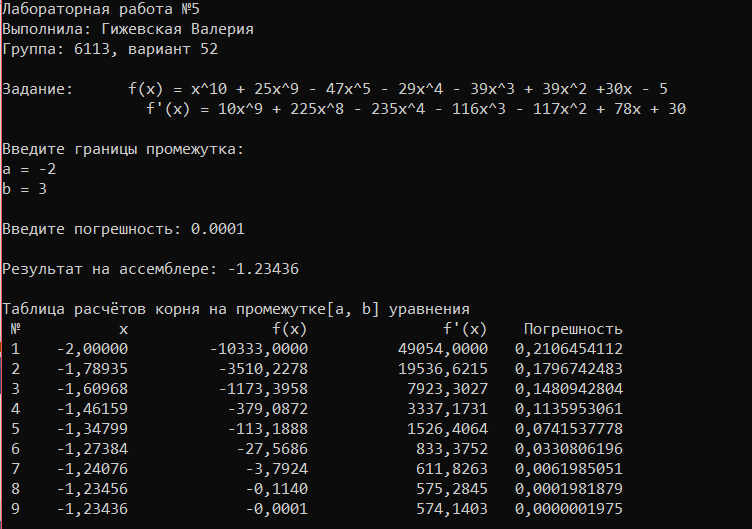


Рисунок 5.2 – Вычисление корней на промежутке [-2; 3]

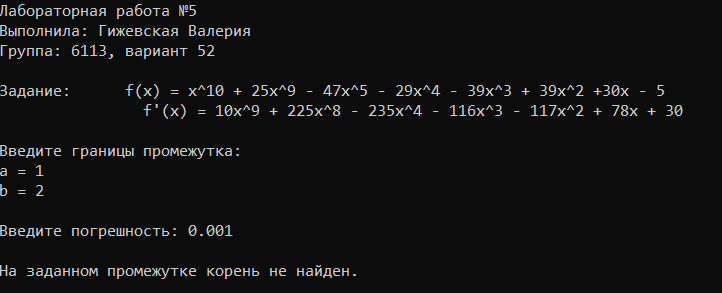


Рисунок 5.3 – Вычисление корней на промежутке [1; 2]

Лабораторная работа 6 «Вычисление определенного интеграла методом Симпсона на языке ассемблера»

6.1 Теоретические основы лабораторной работы

6.1.1 Команды языка ассемблера

В лабораторной работе были использованы команды языка ассемблера, приведенные в подразделах 2.1 и 5.1, а также следующие команды [4]:

* JNE/JNZ – переход по указанному адресу, если не равно или если результат последней операции отличен от 0 (jne addr/jnz addr);
* CALL – вызов подпрограммы по адресу, указанному операндом, при этом параметры берутся из стека (call addr);
* FST – копирование значения из ST(0) в ячейку стека или область памяти, указанную операндом;
* FIADD – целочисленное сложение значения в ST(0) со значением, указанным операндом (fiadd src);
* FIMUL – целочисленное умножение значения в ST(0) на значение, указанное операндом (fimul src).

6.1.2 Метод Симпсона приближенного вычисления определенного интеграла

Если заменить график функции на каждом отрезке , которые получены после разбиения отрезка интегрирования на равных частей дугами парабол, то получим формулу приближенного вычисления определенного интеграла .

Разобьем отрезок на равных частей (отрезков) длиной точками , причем . В точках разбиения находим значения подынтегральной функции :

.

Заменяем каждую пару соседних элементарных криволинейных трапеций с основаниями одной элементарной параболической трапецией с основанием .

Расчетная формула парабол (или Симпсона) для этого метода имеет вид:

6.2 Задание

1. В программе необходимо вычислить определённый интеграл при заданном числе интервалов N методом Симпсона на языке ассемблера с использованием команд арифметического сопроцессора.
2. Значения переменных передаются в качестве параметров функции.
3. Составить таблицу расчетов вычисления интеграла при заданном числе интервалов N и вывести на экран. Выводить пошаговый расчет интеграла по формуле Симпсона .
4. Все параметры уравнения имеют тип double.
5. Проверку деления на 0 реализовать также на встроенном ассемблере.
6. Если не найден корень интеграла, то вывести соответствующее сообщение.
7. В качестве комментария к каждой строке необходимо указать, какой промежуточный результат, в каком регистре формируется.
8. В качестве комментария к строкам, содержащим команды сопроцессора, необходимо указать состояние регистров сопроцессора.
9. Результат можно возвращать из функции в вершине стека сопроцессора.

**Условие:**

6.3 Схема алгоритма

На рисунке 6.1 приведена схема алгоритма вычисления определённого интеграла методом Симпсона при заданном числе интервалов. В начале пользователь вводит с консоли количество интервалов. Затем программа высчитывает интеграл по формуле и выводит его пошаговый расчет на консоль в виде таблицы.

Схема алгоритма оформляется в виде рисунка по ГОСТу [3].

Рисунок 6.1 – Схема алгоритма название алгоритма

6.4 Решение

#include <iostream>

using namespace std;

double integral\_asm(double a, double b, int n) {

printf("%3s%14s%20s%15s%20s\n", "i", "x", "f(x)", "Коэф.", "Сумма");

int N = 2 \* n, i, k;

const int c3 = 3;

double x, f, \_sum;

\_\_asm {

; //st(0) || st(1) || st(2) || st(3) || st(4)

finit; //инициализация сопроцессора

xor esi, esi; //esi = 0

mov edi, N; //edi = 2n

fldz; //sum = 0

fld a; //a || sum = 0

fld b; //b || a || sum = 0

fsub st, st(1); //b - a || a || sum = 0

fidiv N; //(b - a) / 2n || a || sum = 0

fxch; //x = a || (b - a) / 2n = h || sum = 0

begin\_loop:

mov i, esi; //i = esi;

fst x; //x = st(0)

fld st; //x || x || h || sum

fmul st, st; //x\*x || x || h || sum

fild c3; //3 || x\*x || x || h || sum

fmulp st(1),st; //3 \* x\*x || x || h || sum

fsub st, st(1); //3 \* x\*x - x || x || h || sum

fld1; //1 || 3 \* x\*x - x || x || h || sum

fsubp st(1), st;//3 \* x\*x - x - 1 || x || h || sum

fst f; //f = st(0)

mov k, 1; //k = 1

or esi, esi; //сравниваем номер с 0

je sum; //если равны, то k = 1, переходим к метке sum

cmp esi, edi; //сравниваем номер и 2n

je sum; //если номер = 2n, то k = 1, переходим к метке sum

test esi, 1; //проверяем номер на четность

jne odd; //если номер нечетный, переход к метке odd

mov k, 2; //k = 2

jmp sum; //переходим к метке sum

odd :

mov k, 4; //k = 4

jmp sum; //переходим к метке sum

sum :

fimul k; //k \* (3 \* x\* x - x - 1) || x || h || sum

faddp st(3), st;//x || h || sum + k \* (3 \* x \* x - x - 1)

fld st(2); //sum || x || h || sum

fstp \_sum; //\_sum = st(0)

}

printf("%3i%14.6f%20.6f%15i%20.6f\n", i, x, f, k, \_sum);

\_\_asm {

inc esi; //esi++

fadd st, st(1); //x + h || h || sum

cmp esi, edi; //сравниваем номер и 2n

jle begin\_loop; //если меньше или равно, продолжаем цикл

fstp st; //h || sum

fmul; //sum\* h

fidiv c3; //sum\* h / 3

}

}

double integral\_cpp(double a, double b, int n) {

n = 2 \* n;

double h = (b - a) / n, x = a, f;

double result = 0;

int k;

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

if (i == 0 || i == n)

{

k = 1;

}

else if (i % 2 == 0)

{

k = 2;

}

else

{

k = 4;

}

f = 3 \* x \* x - x - 1;

result += k \* f;

x += h;

printf("%5i%14.6f%20.6f%15.1i%20.6f\n", i, x, f, k, result);

}

return (h \* result) / 3;

}

int main()

{

setlocale(LC\_ALL, "");

cout << "Лабораторная работа №6 || Выполнила: Гижевская Валерия || Группа: 6113-020302D || Вариант 30" << endl

<< "Вычисление определенного интеграла функции 3x^2 - x - 1 от x = 1 до 3" << endl << endl

<< "Введите количество интервалов: ";

int n;

cin >> n;

try {

cout << endl << "Результат (ассемблер): " << integral\_asm(1, 3, n) << endl << endl;

printf("%5s%14s%20s%15.1s%20s\n", "i", "x", "f(x)", "k", "сумма");

cout << endl << "Результат (C++): " << integral\_cpp(1, 3, n) << endl;

}

catch (exception & e) {

cout << "Ошибка: " << e.what() << endl;

}

system("PAUSE");

return 0;

}

6.5 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунке 6.2 и 6.3.

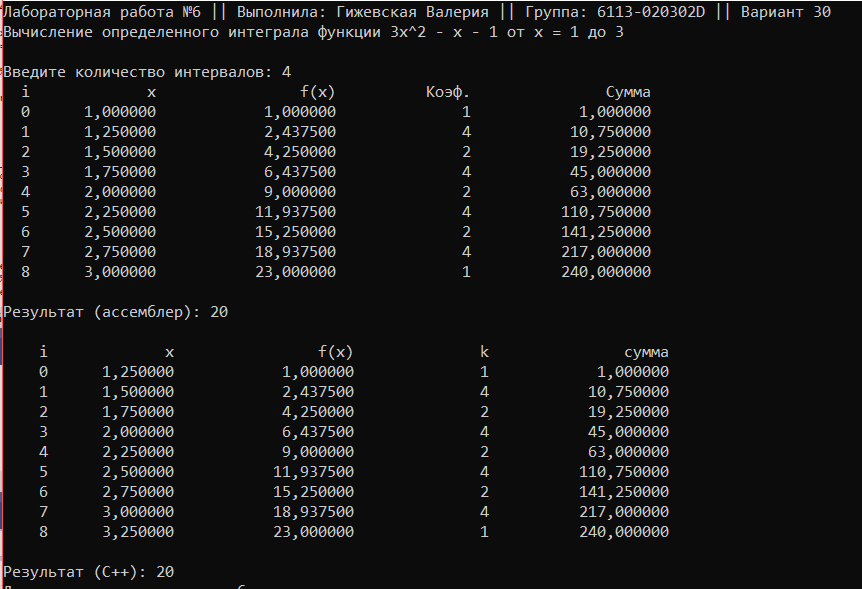
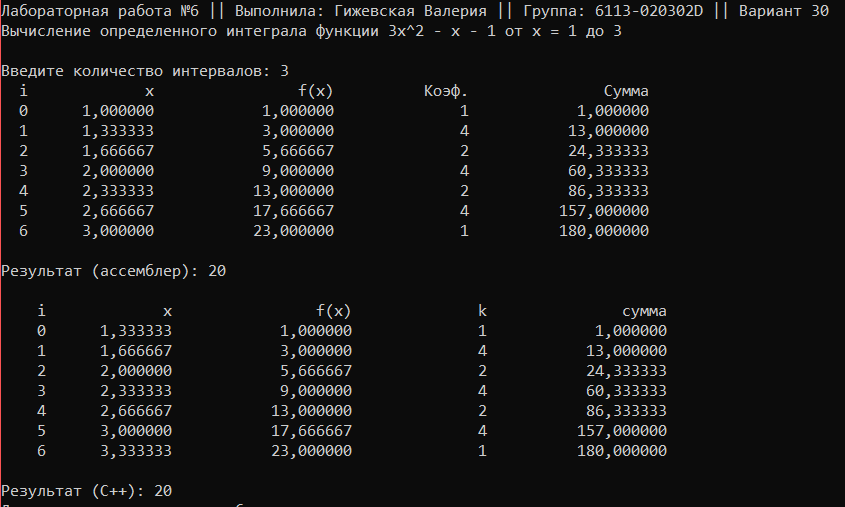
  
Рисунок 6.2 – Вычисление интеграла с четырьмя интегралами

Рисунок 6.3 – Вычисление интеграла с тремя интегралами

Лабораторная работа 7 «Вычисление суммы ряда на языке ассемблера»

7.1 Теоретические основы лабораторной работы

7.1.1 Команды языка ассемблера

В лабораторной работе были использованы команды языка ассемблера, приведенные в подразделе 6.1, а также следующие команды [4]:

* FADD – сложение (fadd st, st(i) или fadd st(i), st; fadd без операндов – сложение ST(1) и ST(0) с выталкиванием);
* FSUBR – реверсивное вычитание (fsubr st, st(i) или fsubr st(i));
* FISUB – целочисленное вычитание значения, указанного операндом, из значения ST(0) (fisub src);
* FIMUL – целочисленное деление значения в ST(0) на значение, указанное операндом (fidiv src).

7.1.2 Числовые ряды и рекуррентное вычисление их сумм

В лабораторной работе используется рекуррентный способ вычисления числового ряда , заключающийся в том, что каждый следующий член ряда вычисляется относительно предыдущего через коэффициент, зависящий от номера итерации. Формулу коэффициента можно получить делением формулы на формулу . Первый ненулевой член ряда вычисляется отдельно, последующие – через коэффициент : .

7.2 Задание

1. В программе необходимо реализовать функцию определения значения некоторой элементарной функции , зависящей от аргумента , на языке ассемблера с использованием команд арифметического сопроцессора.
2. Функция вычисляется в виде суммы ряда. Вычисления прекращаются если , где – последующий член ряда; – предыдущий член ряда. Кроме того, на случай плохой сходимости следует ограничить количество слагаемых сверху некоторым наперёд заданным , т.е. выход их вычислительной процедуры может произойти не по условию , а по условию . Значение функции и количество итераций вывести для контроля на экран.
3. Значение параметров , и передаются в качестве аргументов функции.
4. Необходимо определить достигнутую погрешность, вычислив отклонение аналитического значения от значения, вычисленного с помощью ряда. Значение погрешности также вывести для контроля на экран.
5. В качестве комментария к строкам, содержащим команды сопроцессора, необходимо указать состояние регистров сопроцессора.

|  |  |
| --- | --- |
| Условие: |  |

7.3 Схема алгоритма

На рисунке 7.1 приведена схема алгоритма вычисления суммы ряда. В начале пользователь вводит переменную x и количество членов ряда. Далее мы находим первый член ряда, затем по заданной в условии формуле программа вычисляет коэффициент, с помощью которого находится следующий член ряда. После чего предыдущий и текущий члены ряда суммируются. Данный процесс повторяется, пока не находится конечная сумма ряда. Результат выводится на консоль.

Схема алгоритма оформляется в виде рисунка по ГОСТу [3].

Рисунок 7.1 – Схема алгоритма название алгоритма

7.4 Решение

#include <math.h>

#include <iomanip>

#include <iostream>

using namespace std;

void calc\_asm(double x, int N)

{

const int c2 = 2;

const float c15 = 1.5;

int n = 2;

double sum = 0;

double curr;

bool success = false;

\_\_asm {

finit; //инициализация сопроцессора

fld sum; //sum

fld x; //x || sum

fmul st, st; //x^2 || sum

fld c15; //1.5 || x^2 || sum

fmulp st(1), st; //1.5 \* x^2 || sum

fst curr;

fadd st(1), st; //curr || sum + curr

}

do {

double prev;

double eps;

double k;

\_\_asm

{

fst prev //сохранение вершины стека в память (prev = curr)

mov ecx, N //ecx = N

fild n; //n || curr || sum

fild n; //n || n || curr || sum

fld1; //1 || n || n || curr || sum

faddp st(1), st; //n + 1 || n || curr || sum

fxch st(1); //n || n + 1 || curr || sum

fld1; //1 || n || n + 1 || curr || sum

fsubp st(1), st; //n - 1 || n + 1 || curr || sum

fld x; //x || n - 1 || n + 1 || curr || sum

fmulp st(1), st; //x \* (n - 1) || n + 1 || curr || sum

fdivrp st(1), st; //x\*(n-1)/(n+1) || curr || sum

test n, 1; //проверяю n на четность

jnz if\_n\_nechet; //переход на метку, если n нечетный

fild n; //n || x\*(n-1)/(n+1) || curr || sum

fild n; //n || n || x\*(n-1)/(n+1) curr || sum

fld1; //1 || n || n || x\*(n-1)/(n+1) || curr || sum

faddp st(1), st; //n + 1 || n || x\*(n-1)/(n+1) curr || sum

fdivp st(1), st; //n/(n+1) || x\*(n-1)/(n+1) || curr || sum

fmulp st(1), st; //(n/(n+1))\*x\*(n-1)/(n+1)) || curr || sum

jmp if\_n\_chet;

if\_n\_nechet: //x\*(n-1)/(n+1) || curr || sum

fild n; //n || x\*(n-1)/(n+1) || curr || sum

fld1; //1 || n || x\*(n-1)/(n+1) || curr || sum

fsubp st(1), st; //n-1 || x\*(n-1)/(n+1) || curr || sum

fild n; //n || n-1 || x\*(n-1)/(n+1) || curr || sum

fld1; //1 || n || n-1 || x\*(n-1)/(n+1) || curr || sum

fld1; //1 || 1 || n || n-1 || x\*(n-1)/(n+1) || curr || sum

faddp st(1), st; //2 || n || n-1 || x\*(n-1)/(n+1) || curr || sum

faddp st(1), st; //n+2 || n-1 || x\*(n-1)/(n+1) || curr || sum

fdivrp st(1), st; //n+2/(n-1) || x\*(n-1)/(n+1) || curr || sum

fmulp st(1), st; //(n+2/(n-1))\*x\*(n-1)/(n+1)) || curr || sum

if\_n\_chet:

fst k;

fmulp st(1), st; //curr \* k || sum

fst curr;

fadd st(1), st; //curr || sum + curr

fxch st(1); //sum || curr

fst sum; //sum || curr

fxch st(1); //curr || sum

fld curr; //curr || curr || sum

fld prev; //prev || curr || curr || sum

fsubp st(1), st; //curr - prev || curr || sum

fabs; //|curr - prev| || curr || sum

fstp eps;

inc n; //n ++;

mov edx, n;

cmp ecx, edx;

jg exit\_1;

jmp exit\_2;

exit\_2:

mov success, 1

exit\_1 :

}

cout << setw(3) << (n - 1) << " " << setw(3) << (n - 1) << " " << setw(12) << setprecision(6) << prev << " " << setw(12) << setprecision(6) << curr << " " << setw(5) << setprecision(6) << k << " " << setw(7) << setprecision(6) << eps << " " << fixed << setw(12) << setprecision(6) << sum << " " << endl;

} while (!success);

cout << endl << "Ответ: " << setprecision(6) << sum << endl;

}

void check(double x, int N) {

double eps;

double curr;

curr = 1.5 \* pow(x,2);

double sum = curr;

double prev;

double k;

double n = 2;

do {

prev = curr;

k = (x \* ((n - 1) / (n + 1))) \* ((n + 1 + pow(-1, n + 1)) / (n + pow(-1, n)));

curr = prev \* k;

eps = abs(curr - prev);

sum = sum + curr;

n++;

cout << setw(3) << (int)(n - 1) << " " << setw(3) << (int)(n - 1) << " " << setw(12) << setprecision(6) << prev << " " << setw(12)

<< setprecision(6) << curr << " " << setw(5) << setprecision(6) << k << " " << setw(7) << setprecision(6) << eps << " "

<< fixed << setw(12) << setprecision(6) << sum << " " << endl;

} while ((n + 1) <= N);

cout << endl << "Ответ: " << setprecision(6) << sum << endl;

}

int main()

{

setlocale(LC\_ALL, "");

cout << "Лабораторная работа №7 || Выполнила: Гижевская Валерия || Группа: 6113-020302D || Вариант 30 " << endl << endl;

cout << "Задание: вычислить сумму ряда ((n + (-1)^n)/(n \* (n - 1))) \* x ^ n по n от 2 до бесконечности" << endl << endl;

double x;

int N;

cout << "Введите x: ";

cin >> x;

do

{

cout << "Введите N: ";

cin >> N;

if (N < 0)

{

cout << "N не может быть отрицательным. Повторите ввод." << endl;

}

} while (N < 0);

N++;

cout << endl << "Результат(ассемблер)" << endl;

cout << setw(3) << "№" << " " << setw(3) << "n" << " " << setw(12) << setprecision(6) << "Sn" << " "

<< setw(12) << setprecision(6) << "Sn+1" << " " << setw(8) << setprecision(6) << "k" << " " << setw(8) << setprecision(6) << "eps" << " "

<< fixed << setw(12) << setprecision(6) << "Сумма" << " " << endl;

calc\_asm(x, N);

cout << endl << "Проверка(С++" << endl;

cout << setw(3) << "№" << " " << setw(3) << "n" << " " << setw(12) << setprecision(6) << "Sn" << " "

<< setw(12) << setprecision(6) << "Sn+1" << " " << setw(8) << setprecision(6) << "k" << " " << setw(8) << setprecision(6) << "eps" << " "

<< fixed << setw(12) << setprecision(6) << "Сумма" << " " << endl;

check(x, N);

cout << endl;

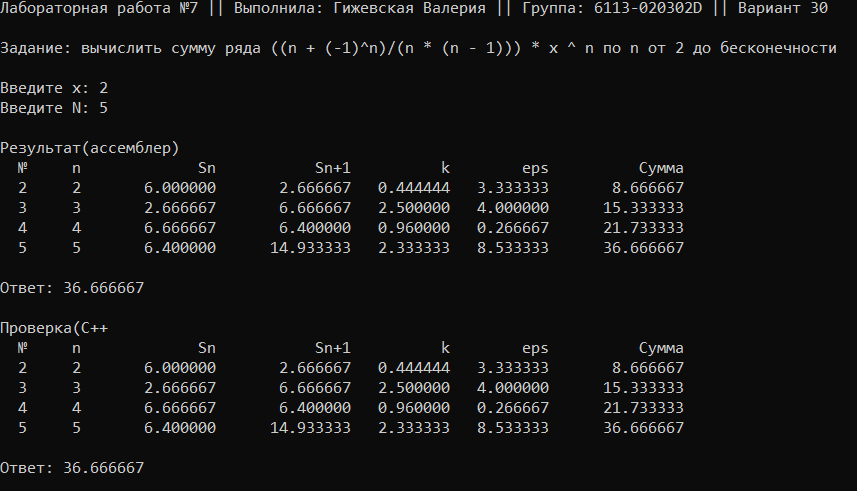
system("PAUSE");

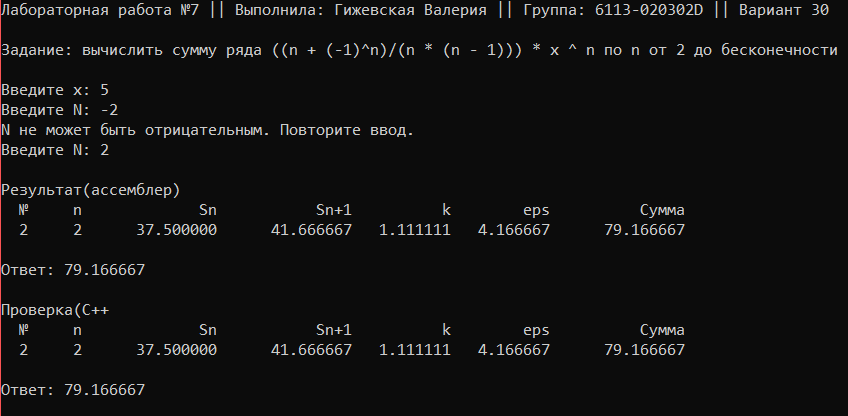
return 0;

}

7.5 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунке 7.2 и 7.3.

  
Рисунок 7.2 – Вычисление суммы ряда при х = 1 и N = 5

  
Рисунок 7.3 – Вывод ошибки при неправильном вводе N

Список использованных источников

1. Зеленко Л.С. Методические указания к лабораторной работе № 1 «Арифметические и логические команды в ассемблере»/ Л.С. Зеленко, Д.С. Оплачко. Самара: изд-во СГАУ, 2015. 24 с.
2. Зеленко Л.С. Методические указания к лабораторной работе № 2 «Арифметические команды и операторы условного перехода» /Л.С. Зеленко, Д.С. Оплачко. Самара: изд-во СГАУ, 2015. 24 с.
3. Оплачко Д.С. Методические указания к лабораторной работе № 3 «Работа с массивами и стеком на языке Assembler»/ Д.С. Оплачко, Л.С. Зеленко. Самара: изд-во СГАУ, 2012. 19 с.
4. Оплачко Д.С. Методические указания к лабораторной работе № 4 «Работа с математическим сопроцессором в среде Assembler» / Д.С. Оплачко, Л.С. Зеленко. Самара: изд-во СГАУ, 2012. 19 с.
5. ГОСТ 19.701-90 (ИСО 5807-85). ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. Введ. 1990-01-01. М.: Изд-во стандартов, 1991. 26 с.
6. СТО 02068410-004-2018. Общие требования к учебным текстовым документам: методические указания [Электронный ресурс]. URL: https://ssau.ru/docs/sveden/localdocs/STO\_SGAU\_02068410-004-2018.pdf (дата обращения: 20.04.2019).