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 и кибернетики\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Кафедра программных систем\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ОТЧЕТ  
  
 по лабораторному практикуму по дисциплине

«Организация ЭВМ и вычислительных систем»

Вариант № 28

Обучающийся в группе 6102\_020302 Н.С. Боряков

Руководитель Д.С. Оплачко

Самара 2022

СОДЕРЖАНИЕ

[Лабораторная работа 1 «Арифметические и логические команды в ассемблере» 4](#_Toc104398568)

[1.1 Теоретические основы лабораторной работы 4](#_Toc104398569)

[1.2 Задание 5](#_Toc104398570)

[1.3 Схема алгоритма 5](#_Toc104398571)

[1.4 Результаты тестирования 6](#_Toc104398572)

[Лабораторная работа 2 «Арифметические команды и команды переходов в ассемблере» 8](#_Toc104398573)

[2.1 Теоретические основы лабораторной работы 8](#_Toc104398574)

[2.2 Задание 8](#_Toc104398575)

[2.3 Схема алгоритма 9](#_Toc104398576)

[2.4 Результаты тестирования 11](#_Toc104398577)

[Лабораторная работа 3 «Команды работы с массивами и стеком» 13](#_Toc104398578)

[3.1 Теоретические основы лабораторной работы 13](#_Toc104398579)

[3.2 Задание 13](#_Toc104398580)

[3.3 Схема алгоритма 13](#_Toc104398581)

[3.4 Результаты тестирования 15](#_Toc104398582)

[Лабораторная работа 4 «Изучение работы математического сопроцессора в среде Assembler» 17](#_Toc104398583)

[4.1 Теоретические основы лабораторной работы 17](#_Toc104398584)

[4.2 Задание 18](#_Toc104398585)

[4.3 Схема алгоритма 18](#_Toc104398586)

[4.4 Результаты тестирования 20](#_Toc104398587)

[Лабораторная работа 5 «Нахождение корня уравнения f(x) = 0 методом Ньютона» 22](#_Toc104398588)

[5.1 Теоретические основы лабораторной работы 22](#_Toc104398589)

[5.2 Задание 22](#_Toc104398590)

[5.3 Решение 23](#_Toc104398591)

[5.4 Результаты тестирования 24](#_Toc104398592)

[Лабораторная работа 6 «Определение значения элементарной функции» 26](#_Toc104398593)

[6.1 Теоретические основы лабораторной работы 26](#_Toc104398594)

[6.2 Задание 26](#_Toc104398595)

[6.3 Решение 27](#_Toc104398596)

[6.4 Результаты тестирования 28](#_Toc104398597)

[Лабораторная работа 7 «Вычисление определенного интеграла методом Симпсона» 29](#_Toc104398598)

[7.1 Теоретические основы лабораторной работы 29](#_Toc104398599)

[7.2 Задание 29](#_Toc104398600)

[7.3 Схема алгоритма 29](#_Toc104398601)

[7.4 Результаты тестирования 31](#_Toc104398602)

[Лабораторная работа 8 «Вычисление суммы ряда» 33](#_Toc104398603)

[8.1 Теоретические основы лабораторной работы 33](#_Toc104398604)

[8.2 Задание 33](#_Toc104398605)

[8.3 Решение 34](#_Toc104398606)

[8.4 Схема алгоритма 34](#_Toc104398607)

[8.5 Результаты тестирования 35](#_Toc104398608)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 37](#_Toc104398609)

[Приложение А.1. Листинг программы лабораторной работы №1 39](#_Toc104398610)

[Приложение А.2. Листинг программы лабораторной работы №2 40](#_Toc104398611)

[Приложение А.3. Листинг программы лабораторной работы №3 41](#_Toc104398612)

[Приложение А.4. Листинг программы лабораторной работы №4 42](#_Toc104398613)

[Приложение А.5. Листинг программы лабораторной работы №5 44](#_Toc104398614)

[Приложение А.6. Листинг программы лабораторной работы №6 46](#_Toc104398615)

[Приложение А.7. Листинг программы лабораторной работы №7 48](#_Toc104398616)

[Приложение А.8. Листинг программы лабораторной работы №8 49](#_Toc104398617)

Лабораторная работа 1 «Арифметические и логические команды в ассемблере»

1.1 Теоретические основы лабораторной работы

При выполнении задания использовались арифметические и логические операторы языка Ассемблер. Рассмотрим их назначение и принцип работы [1]:

* MOV – команда копирования данных из одной переменной в другую. Команда копирует содержимое второго операнда в первый операнд. При этом содержимое второго операнда не изменяется.
* ADD – команда выполняет целочисленного сложения двух операндов. Результат сложения помещается в первый операнд и выполняется соответствующая установка флагов.
* SUB – команда, которая выполняет целочисленное вычитание по методу сложения с двоичным дополнением: для второго операнда устанавливаются обратные значения бит и прибавляется 1, а затем происходит сложение с первым операндом.
* CDQ – команда для выполнения знакового расширения операнда – источника. Результатом является операнд удвоенного размера: EDX:EAX, EAX.
* IMUL – команда знакового умножения данных выполняется. В единственном операнде указывается множитель.
* IDIV – команда знакового деления. В единственном операнде указывается делитель.
* DEC – команда, которая уменьшает целочисленное значение регистра на единицу.
* PUSH – команда добавления в вершину содержимое источника в стек. В качестве параметра «источник» может быть регистр, непосредственный операнд или переменная.
* POP – команда извлекания содержимого источника из вершины стека.

1.2 Задание

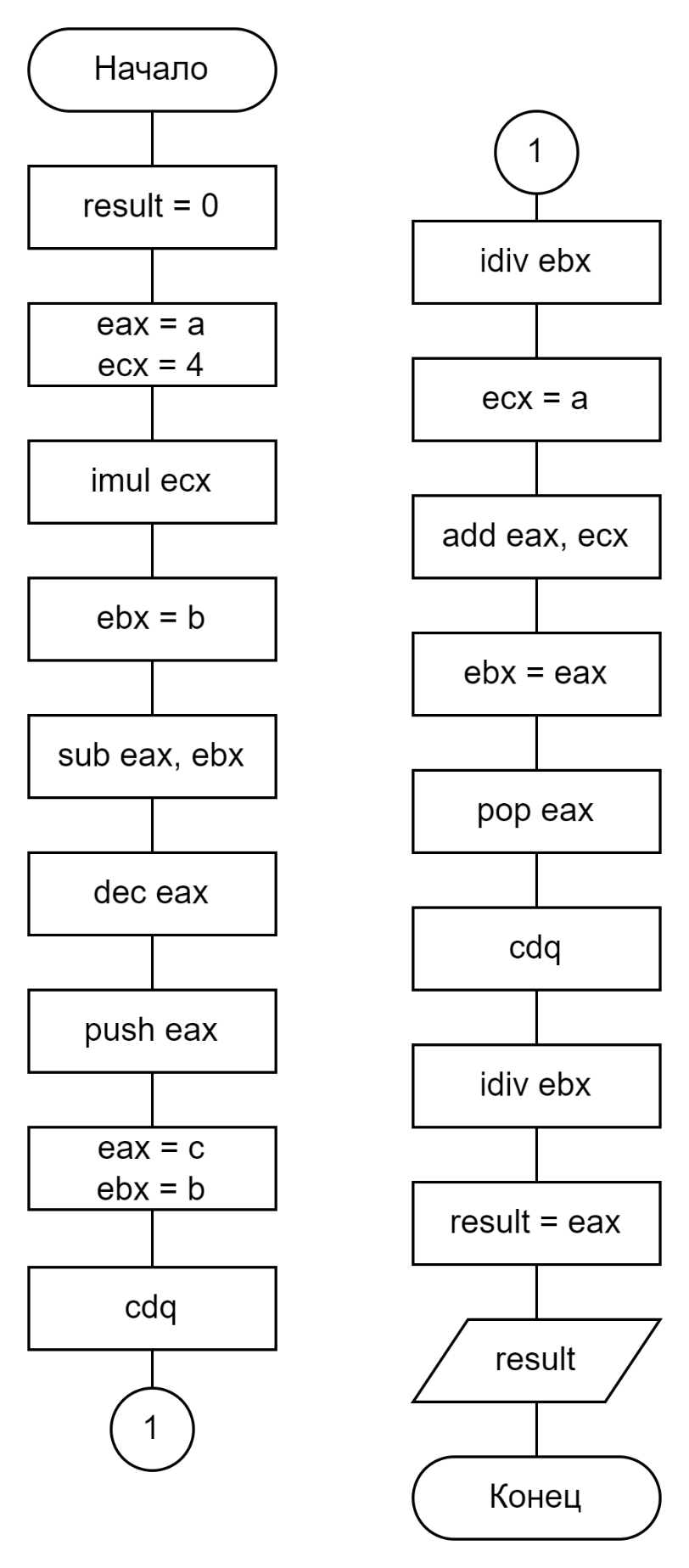
1. В программе необходимо реализовать функцию вычисления целочисленного выражения (4\*a – b -1) / (c / b + a) на встроенном ассемблере MASM в среде Microsoft Visual Studio на языке C++.
2. Значения переменных передаются в качестве параметров функции.
3. Результат выводить в консольном приложении (проект 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. Где 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 переменной y в соответствии с заданием.

В параметры передаются значения переменных *a, b, c* целочисленного типа из главной программы. Переменной *result*, отвечающей за хранения результата вычисления исходного выражения, присваиваем значение 0. Присваиваем значения регистрам: *eax = a, ecx = 4* и перемножаем их. Вычитаем из *eax, ebx* и уменьшаем *eax* на единицу. Сохраняем значение *eax* в стеке.

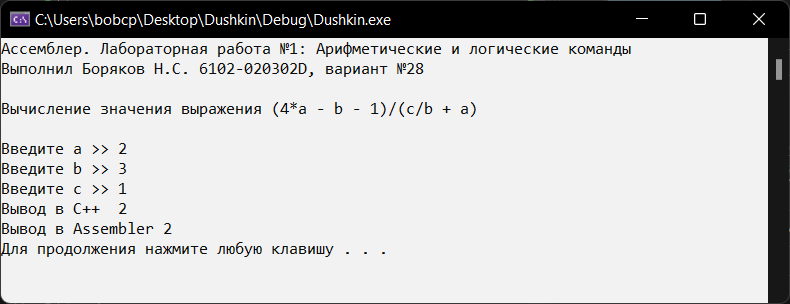
Присваиваем значение *eax = c, ebx = 4*. Выполняем преобразование *eax* в четверное слово. Получаем частное *eax = eax / ebx*. Присваиваем новое значение регистру *ecx = a,* а затем получаем сумму в регистре *eax = eax + ecx.* Присваиваем новое значение регистру *ebx = eax.* Загружаем из стека в регист *eax* значение, полученное ранее и делим его на значение регистра *eax*. Присваиваем переменной *result* значение из регистра *eax*. Возвращаем *result*.

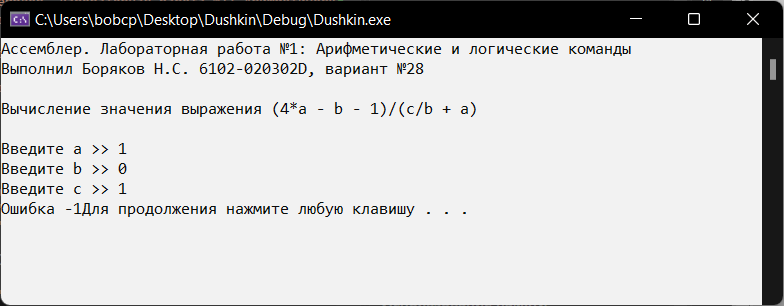
   
Рисунок 1.1 – Схема алгоритма вычисления исходного выражения

Текст программы приведен в приложении А.1.

1.4 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 1.2 – 1.3 при значениях аргумента.

   
Рисунок 1.2 – Пример работы алгоритма при значении аргумента a = 2, b = 3, c = 1

   
Рисунок 1.3 – Пример работы алгоритма при значении аргумента a = 1, b = 0, c = 1

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

2.1 Теоретические основы лабораторной работы

При выполнении задания использовались арифметические и логические операторы языка Ассемблер. Рассмотрим их назначение и принцип работы [2]:

* CMP – команда сравнения. Она устанавливает значения флагов в зависимости от полученного результата вычитания, но не изменяет содержимого операндов. В команде CMP один из операндов должен быть регистром. Другой операнд может иметь любой режим адресации.
* JG – инструкция, которая осуществляет передачу управления только в случае, если ZF = 0 или SF = OF другому флагу.
* JL – инструкция, которая осуществляет передачу управления в случае, если SF не равен OF.
* JE – инструкция, которая представляет инструкцию условного перехода, осуществляющая передачу управления только в том случае, если флаг ZF = 1.
* JMP – инструкция безусловного перехода. Эта инструкция указывает процессору, что в качестве следующей за JMP инструкцией нужно выполнить инструкцию по целевой метке.
* OR – команда объединение по «ИЛИ». Команда осуществляет логическое «ИЛИ» между всеми битами двух операндов.

В данной лабораторной работе также использовались команды из

пункта 1.1.

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 X – целочисленный, возвращается из функции 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 функции условного целочисленного выражения.

Переменным *a, b* присваиваются значения из главной программы, переменной *result*, которая отвечает за хранение результата вычисления исходного выражения присваиваем *result = 0*. Сравниваем значения *a* и *b*. *Если a > b,* проверяем *a* на равенство *с* 0. Если *a = 0,* передаем ошибку деления на нуль, иначе присваиваем переменной *result* заданную функцию *result = b / a - 5.* Если *a < b*, проверяем *b* на равенство *с* 0. Если *a = 0*, передаем ошибку деления на нуль, иначе присваиваем переменной *result* заданную функцию *result = (3 \* a - 5) / b*. В случае равенства *a = b*, переменной result присваивается значение *result = 25*. Выводим значение *result*.

На рисунке 2.2 приведена схема алгоритма вычисления функции заданного условного целочисленного выражения на языке Ассемблера, используя команды сравнения, условного и безусловного переходов на встроенном ассемблере.

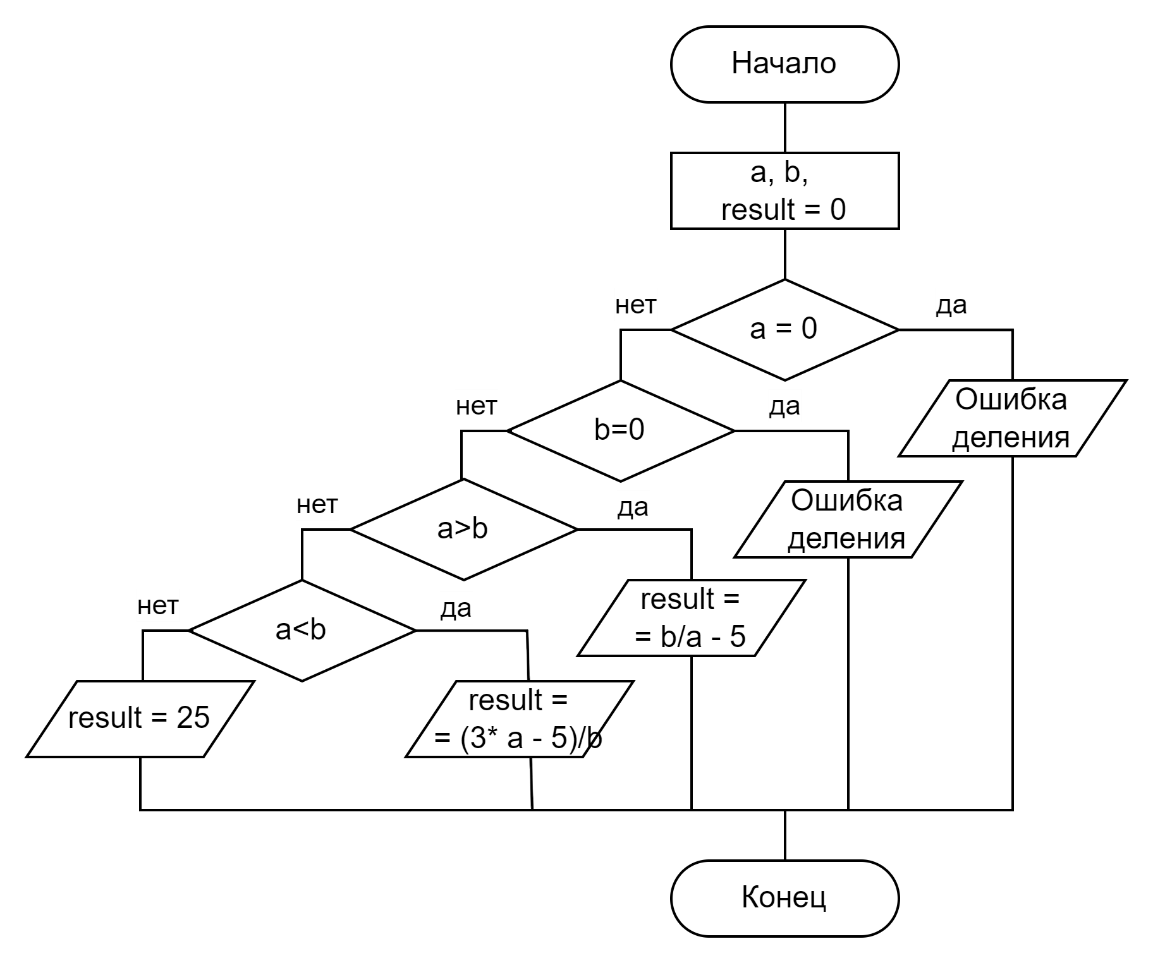
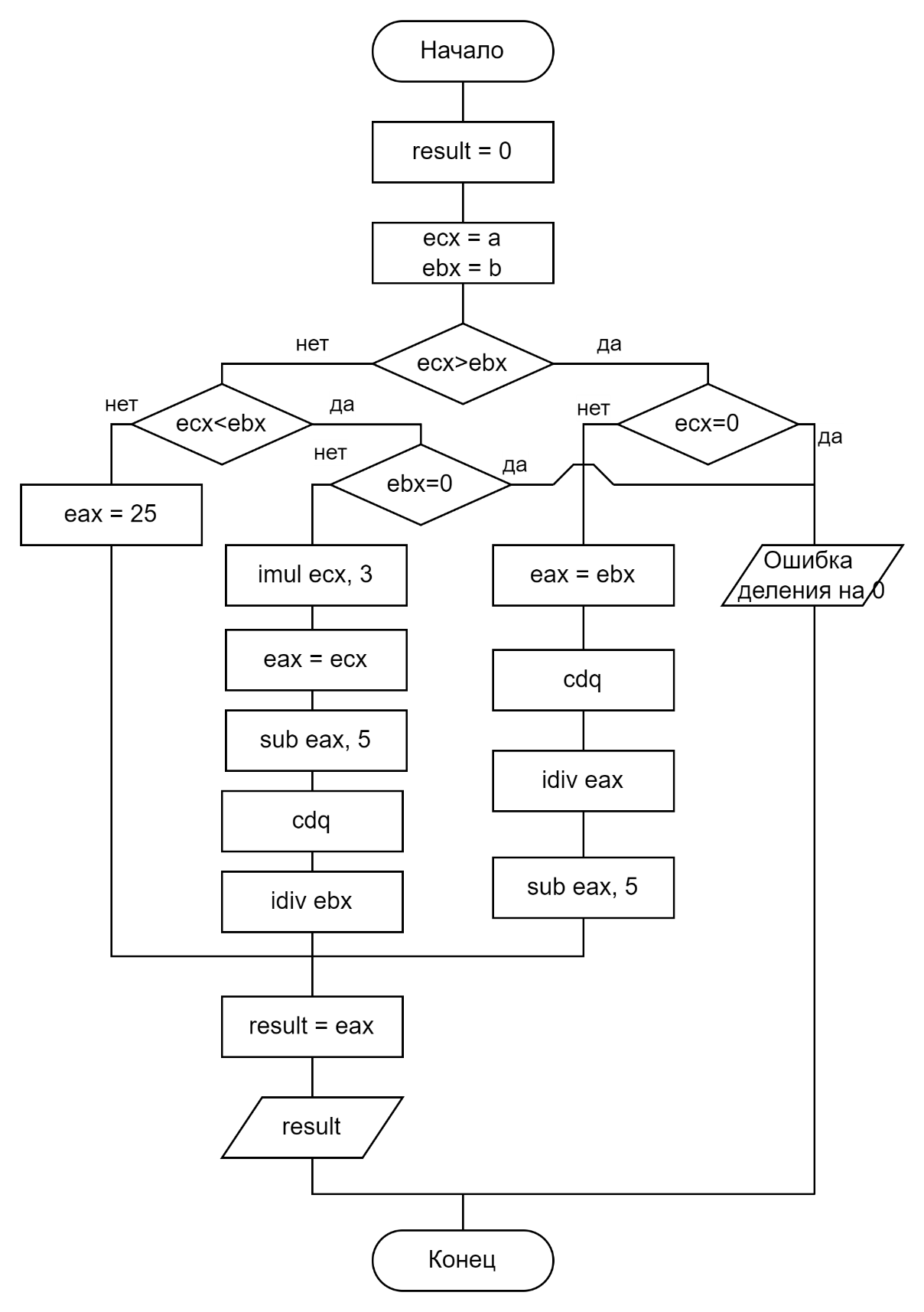
В параметры подаются значения переменных *a, b* целочисленного типа из главной программы. Переменной *result*, отвечающей за хранения результата вычисления исходного выражения, присваиваем значение 0. Присваиваем *ecx* = *a, ebx = b*. Сравниваем значения в регистрах *ecx* и *ebx*.

Если значение в *ecx* больше значения в *ebx*, то сравниваем *ecx* с 0. Если *ecx* *=0*, то выводим сообщение об ошибке, в противном случае преобразовываем eax в четверное слово, вызываем команду IDIV и получаем частное *eax=eax / ecx.* Добавляем командой ADD к полученному частному -5.

Если значение в *ecx* меньше значения в *ebx*, то сравниваем *ebx* с 0. Если *ebx=0*, то выводим сообщение об ошибке. Иначе присваиваем значение *eax = ecx*. Умножаем командой IMUL значение регистра *eax* на 3, а затем проверяем на переполнение. В *eax* получаем разность *eax = eax - 5*, а затем преобразовываем *eax* в четверное слово. Получаем частное *eax = eax / ebx*.

В случае равенства *eax = ebx*, присваиваем регистру *eax = 25.*

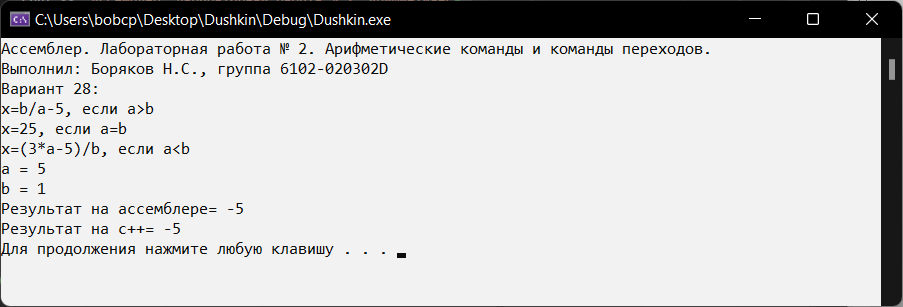
Присваиваем переменной *result* значение регистра *eax*. Проверим на правдивость значения переменной *result*. Если *result = -10000*, то передаем сообщение об ошибке, иначе выводим переменную *result*.

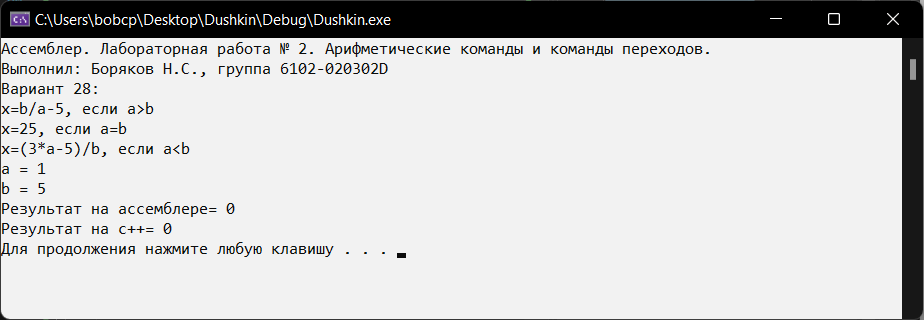
   
Рисунок 2.1 – Схема алгоритма вычисления условного выражения    
Рисунок 2.2 – Схема алгоритма на языке Ассемблера

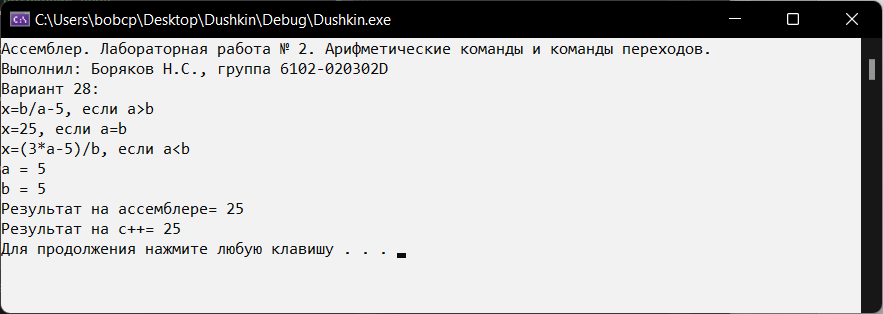
[Текст программы приведен в приложении А.2.](#Схема2)

2.4 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 2.3 – 2.5 при значениях аргумента.

   
Рисунок 2.3 – Пример работы алгоритма при значении аргумента a = 5, b = 1

   
Рисунок 2.4 – Пример работы алгоритма при значении аргумента a = 1, b = 5

   
Рисунок 2.5 – Пример работы алгоритма при значении аргумента a = 5, b = 5

Лабораторная работа 3 «Команды работы с массивами и стеком»

3.1 Теоретические основы лабораторной работы

Для решения лабораторной работы будем использовать «Методические указания к лабораторной работе № 3» и материалы с сайта [3, 7]. Рассмотрим основные команды для работы с массивами и стеком в ассемблере:

* LOOP – инструкция, которая уменьшает значение в регистре СХ в реальном режиме или ECX в защищённом. Если после этого значение в СХ не равно нулю, то команда LOOP выполняет переход на метку. То есть команда выполняется в два этапа. Сначала из регистра СХ вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды LOOP.
* XOR – логическая команда, которая выполняет операцию исключающего ИЛИ над битами операндов. Размерность операндов должна быть одинакова. Результат операции записывается в первый операнд.

В данной лабораторной работе также использовались команды из

пункта 2.1.

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**:** В одномерном массиве A={a[i]} целых чисел вычислить сумму квадратов всех положительных элементов массива, удовлетворяющих условию: a[i] >= b.

3.3 Схема алгоритма

На рисунке 3.1 приведена схема алгоритма вычисления функции обработки элементов массива с использованием команд сравнения, переходов и циклов на встроенном языке ассемблере в соответствии с заданием.

В параметры функции передаются значения переменных, вводимых пользователем в главной программе: *mas*, хранящая массив элементов, *size\_mas*, хранящая количество элементов массива и *b*, относительно которой сравниваются элементы массива. Переменной *result*, отвечающей за хранения результата вычисления исходного выражения, присваиваем нулевое значение. Регистрам *edi*, отвечающему за сумму нечетных элементов, подходящих под условие, и *esi*, отвечающему за работу цикла, также присваиваем 0. Присваиваем *ebx* ссылку на первый элемент массива *ebx = mas*, а регистру *ecx* присваиваем размер массива *ecx = size\_mas*.

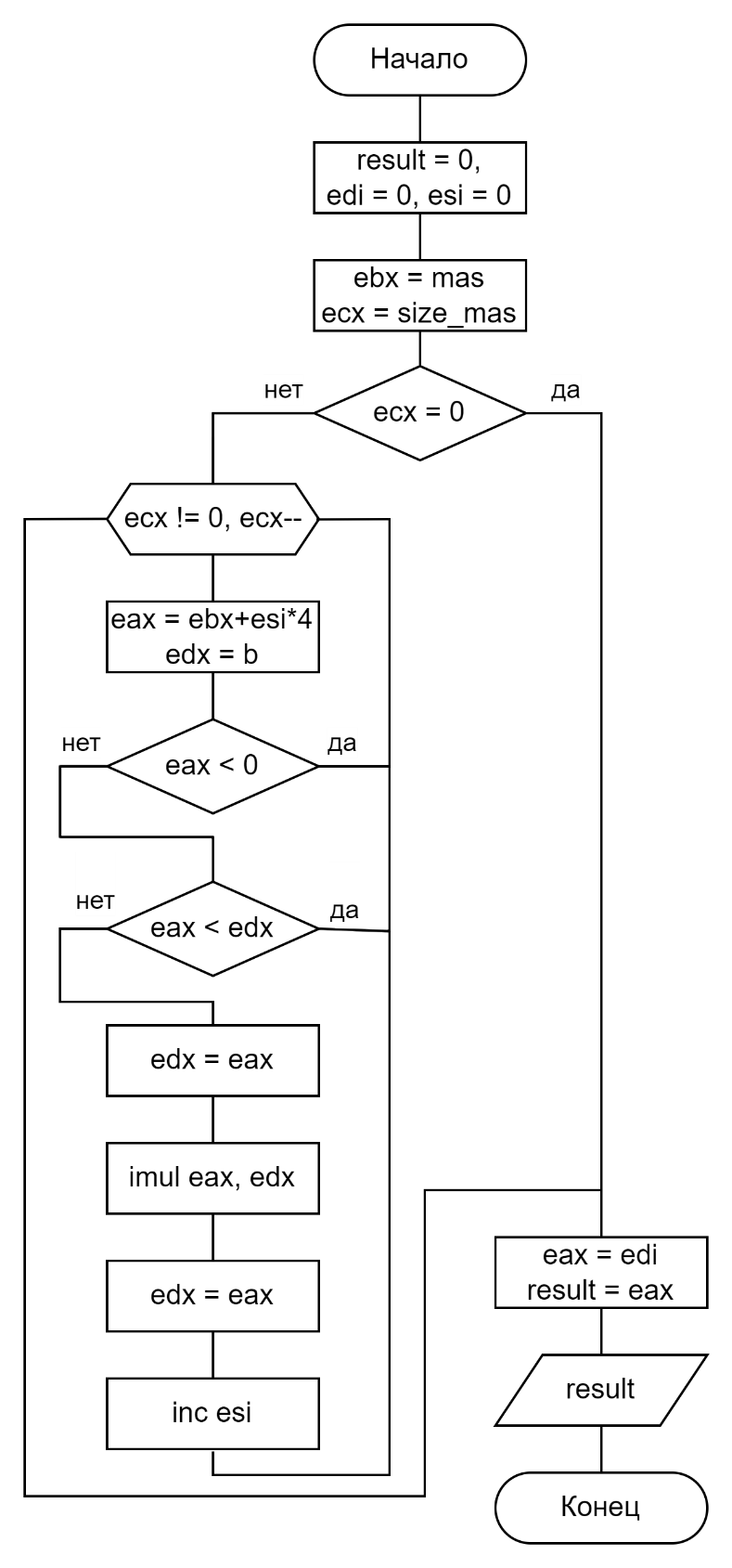
Если длина массива равна нулю, то *result* присваиваем *eax* равный нулю *result = eax* и завершаем цикл.

Если длина массива не равна нулю, то до тех пор, пока *ecx* не станет равен нулю, присваиваем регистру *eax* необходимый элемент массива. Определяем текущий элемент: *eax = [ebx + esi \* 4].* нулем.

Если *eax < 0*, то завершаем цикл. Иначе сравниваем *eax* с *edx = b*, в случае, *eax > edx* завершаем цикл.

Иначе, когда все условия выполнены, присваиваем *edx = eax* и умножаем *eax = edx \* eax*. А затем добавляем *edi = eax*, а значение регистра *esi* увеличиваем на единицу и переходим к следующему элементу массива. После выхода из цикла присваиваем *eax* значение *edi*, а переменной *result* значение регистра *eax*.

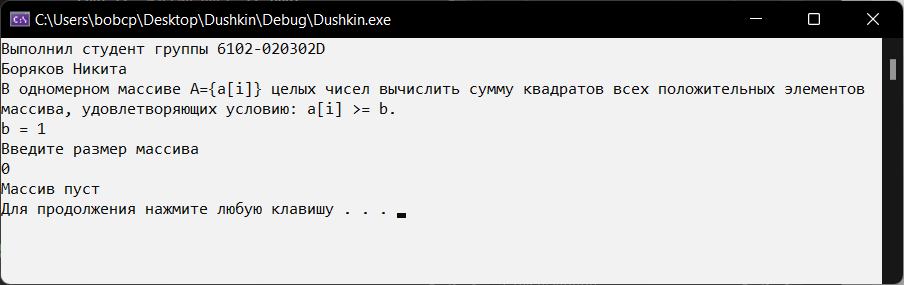
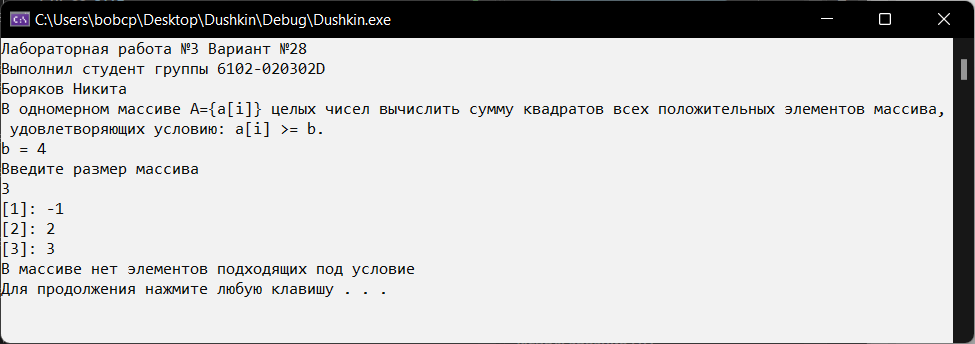
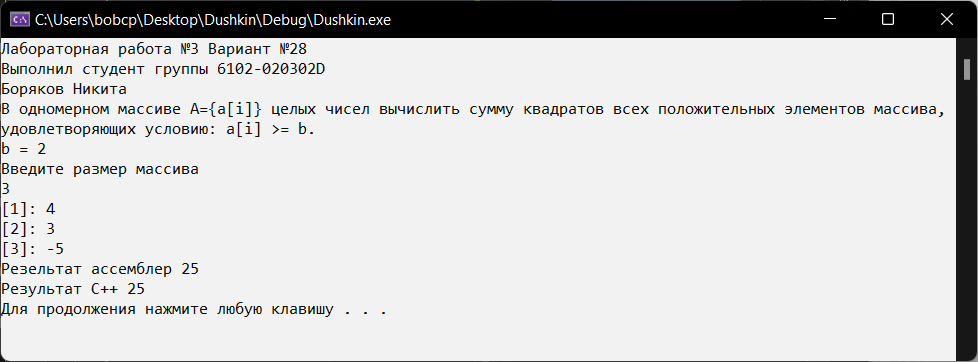
Выводим значение регистра *result*.

   
Рисунок 3.1 – Схема алгоритма вычисления исходного выражения

[Текст программы приведен в приложении А.3.](#Листинг3)

3.4 Результаты тестирования

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

  
Рисунок 3.2 – Пример работы алгоритма при значении аргументов size\_mas=0,b=1    
Рисунок 3.3 – Пример работы алгоритма при значении аргументов size\_mas=3, A={-1,2,3}, b = 4   
Рисунок 3.3 – Пример работы алгоритма при значении аргументов size\_mas=3, A={4,3,-5}, b = 2

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

4.1 Теоретические основы лабораторной работы

Для решения лабораторной работы будем использовать «Методические указания к лабораторной работе № 4» и материалы с сайта [4, 7].

Рассмотрим команды математического сопроцессора в среде Assembler используемые в лабораторной работе. Регистр ST(i) - приемник, регистр ST(0) – источник:

* FLD – команда, которая загружает из памяти в вершину стека ST(0) вещественное число.
* FILD – команда, которая загружает из памяти в вершину стека ST(0) целое число.
* FSTP – команда, которая извлекает из вершины стека ST(0) в память вещественное число. Эта команда сначала сохраняет вершину стека в памяти, а потом удаляют данные из вершины стека.
* FCOM – команда, которая выполняет вещественное сравнение.
* FSUBP – команда, которая выполняет вещественное вычитание с выталкиванием. ST(i) = ST(i) - ST(0).
* FDIVP – команда, которая выполняет вещественное деление с выталкиванием; ST(i) = ST(i) ÷ ST(0).
* FSTSW – команда, которая выполняет считывание слова состояния сопроцессора в память.
* FINIT – команда, которая выполняет инициализацию сопроцессора.

4.2 Задание

1. В программе необходимо реализовать функцию вычисления заданного условного выражения на языке ассемблера с использованием команд арифметического сопроцессора.

****

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

4.3 Схема алгоритма

Схема основного алгоритма приведена на рисунке 2.1. Схема данного алгоритма на ассемблере приведена на рисунке 4.1.

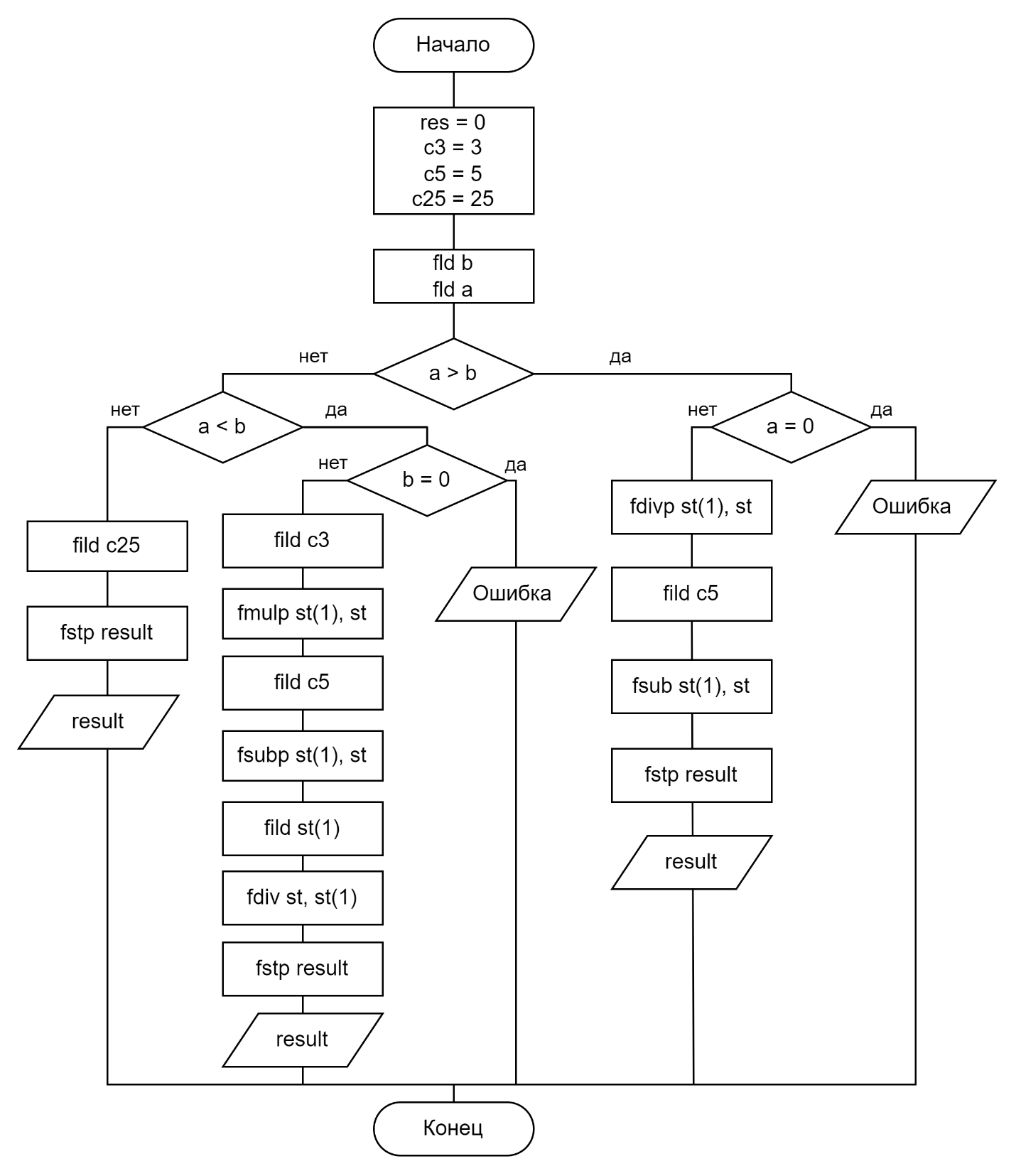
В параметры подаются значения переменных *a, b* вещественного типа из главной программы. Объявляем и инициализируем переменные *с3=3, с5=5, с25=25*. Переменной *res*, хранящей результат вычисления исходного выражения, присваиваем значение 0. Инициализируем сопроцессор, загружаем в стек *b* и *a*, сравниваем их.

Если *a > b*, то сравниваем *a* с нулем. Если *a* равно нулю, то выходимиз программы, иначе делим с выталкиванием *b* на *a*, загружаем в стек *с5* и вычитаем с выталкиванием из частного. После этого присваиваем переменной *res* получившееся выражение.

Если *a < b*, то сравниваем *b* с нулем. Если *b* равно нулю, то выходим из программы. В противном случае, загружаем в стек *с3*, выполняем умножение 3\**a* и вычитание *3\*a-5.* Затем делим *st* на *st(1)* и полученное частное присваиваем переменной *res*.

В случае равенства *a = b*, загружаем в стек *с25* и присваиваем переменной *res* это значение.

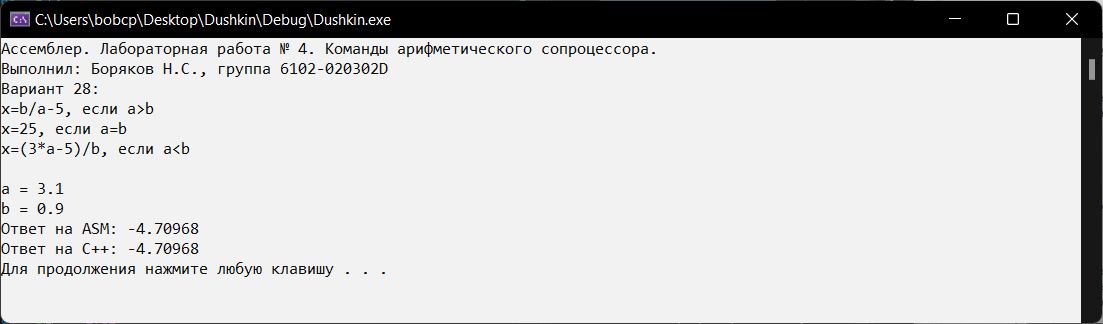
Выводим переменную *res*.

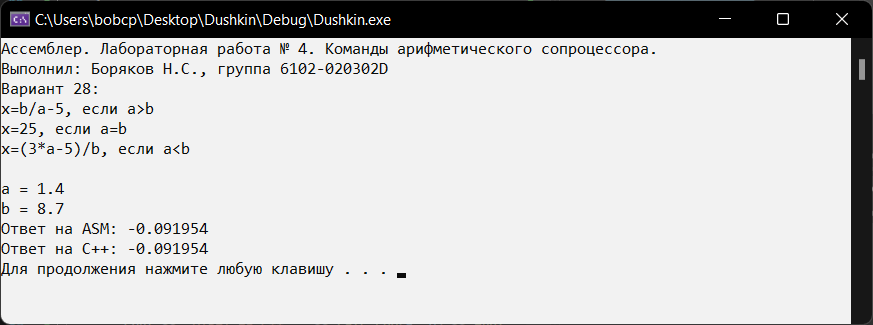
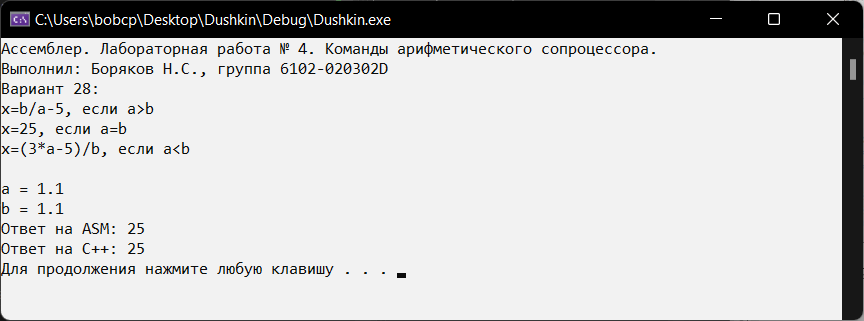
   
Рисунок 4.1 – Схема алгоритма на языке ассемблера

[Текст программы приведен в приложении А.4.](#Листинг4)

4.4 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 4.2 – 4.4 при различных значениях аргументов.

  
Рисунок 4.2 – Пример работы алгоритма при значении аргументов a=3.1, b=0.9

  
Рисунок 4.3 – Пример работы алгоритма при значении аргументов a=1.4, b=8.7   
Рисунок 4.3 – Пример работы алгоритма при значении аргументов a =1.1, b=1.1

Лабораторная работа 5 «Нахождение корня уравнения f(x) = 0 методом Ньютона»

5.1 Теоретические основы лабораторной работы

Для решения лабораторной работы будем использовать «Методические указания к лабораторной работе № 5» и материалы с сайта [7, 8].

* FMUL – команда, которая выполняет вещественное умножение. ST(i) = ST(i) \* ST(0).
* FMULP – команда, которая выполняет вещественное умножение с выталкиванием. ST(i) = ST(i) \* ST(0).
* FCOMP – команда, которая выполняет вещественное сравнение с выталкиванием.

В данной лабораторной работе также использовались команды из

пункта 4.1.

5.2 Задание

1. В программе необходимо найти с заданной точностью 𝜀 корень уравнения 𝑓(𝑥) = 0 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 [a; b] 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 [a; b] 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**:** .

5.3 Решение

Для того, чтобы решить уравнение сначала найдем производную данной функции: .

Далее воспользуемся методом Ньютона и заданными параметрами и получим рекуррентную формулу вычисления корня уравнения:

.

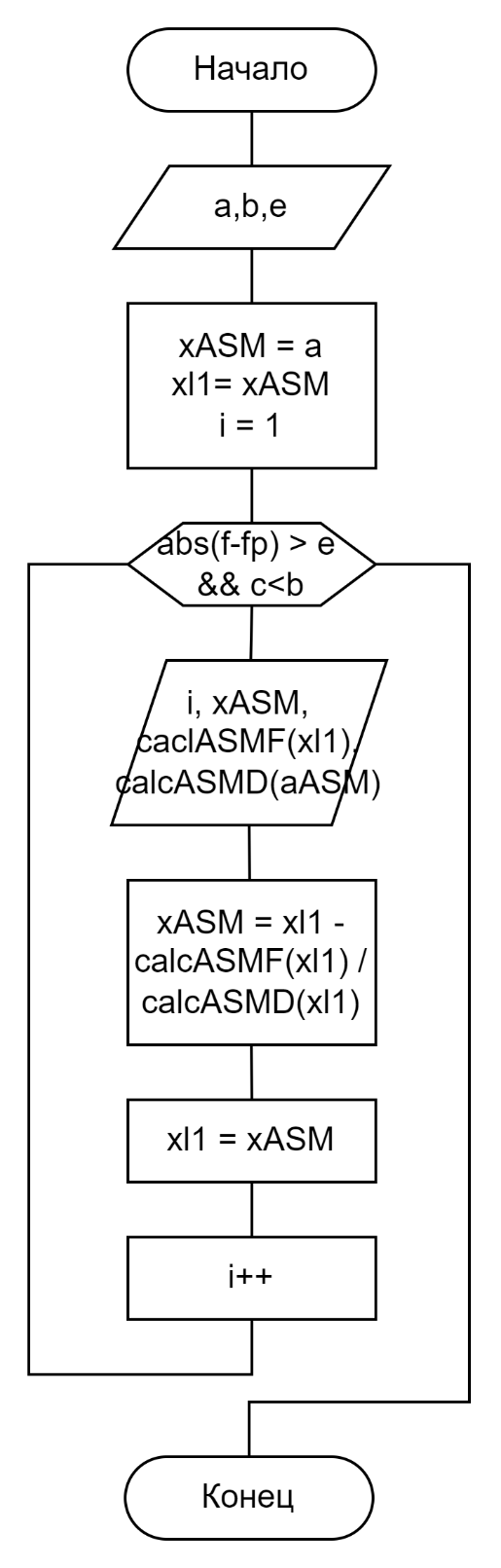
Схема алгоритма нахождения корня уравнения методом Ньютона изображена на рисунке 5.1.

Пользователь вводит вещественное значение *a*, где *a* – левая граница промежутка, вещественное значение *b*, где *b* – правая граница промежутка и вещественное значение *e*, где *e* – погрешность.

Объявляем и инициализируем переменные *xASM*, хранящую текущее значение функции, *xl1*, хранящую предыдущее значение функции и *i*, хранящую номер итерации.

Далее создаем цикл, который выполняется до того момента, пока *|f*| */ |fp| > e*. В цикле выводим таблицу, где указан номер *i*, значение *x*ASM, значение функции f(x), значение производной функции f1(x), погрешность = *.* Переменной *xASM* присваиваем значение *xASM = xl1 – calcASMF(xl1)/ calcASMD(xl1)* Потом xl1 = xASM. *calcASMF (c)* – метод подсчёта функции на Assembler, а *calcASMD (*c) – метод подсчёта производной на Assembler.

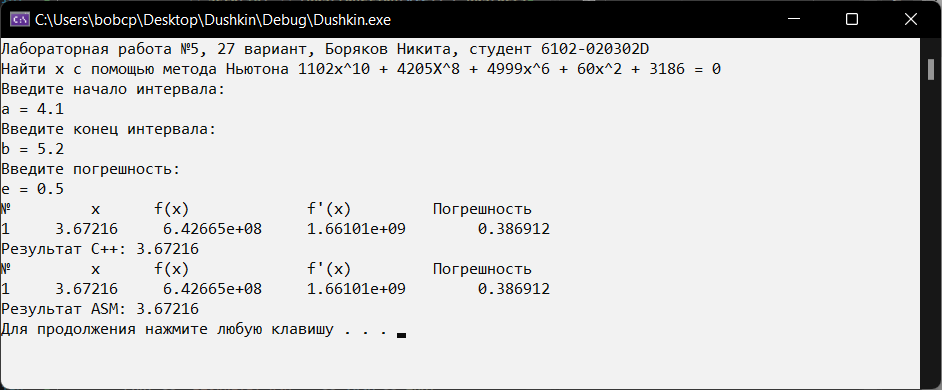
Затем увеличиваем *i* на единицу, и если *|f - fp| > e,* то цикл запускается ещё.

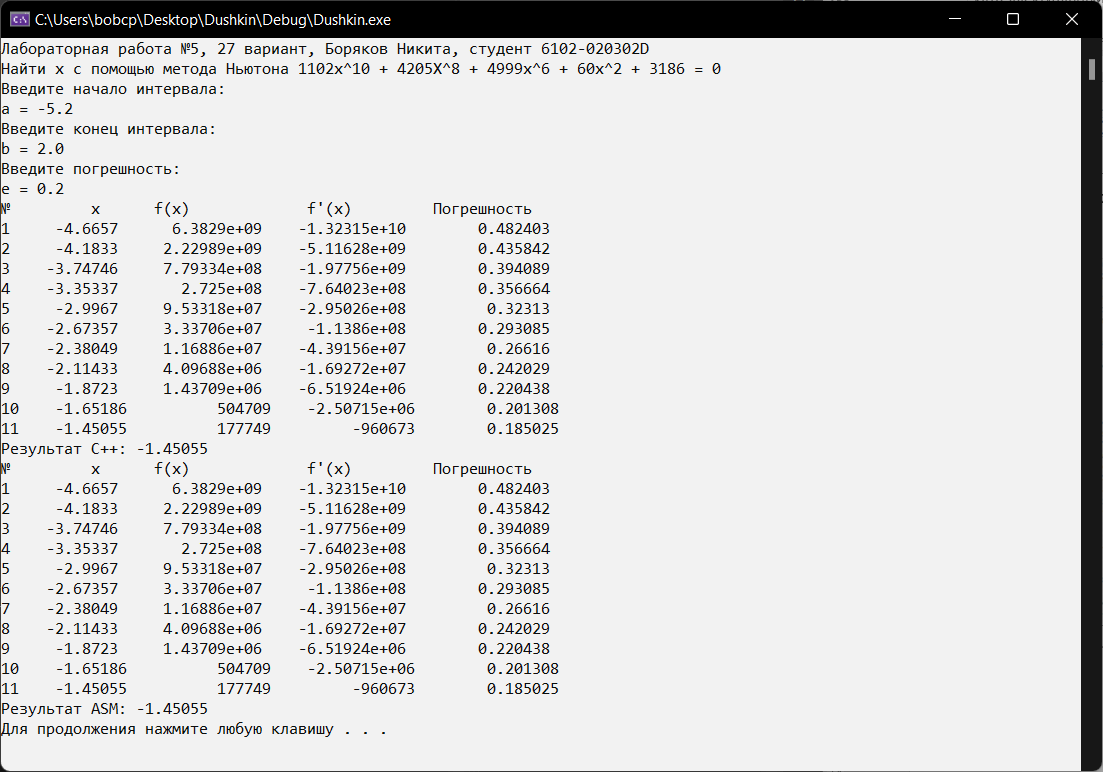
   
Рисунок 5.1 – Схема алгоритма на языке ассемблера

[Текст программы приведен в приложении А.5.](#Листинг5)

5.4 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 5.2 – 5.3 при различных значениях аргумента.

  
Рисунок 5.2 – Пример работы алгоритма при значении аргументов a = 4.1, b=5.2, e = 0.5

  
Рисунок 5.3 – Пример работы алгоритма при значении аргументов a = - 5.2, b= 2.0, e = 0.2

Лабораторная работа 6 «Определение значения элементарной функции»

6.1 Теоретические основы лабораторной работы

Для решения лабораторной работы будем использовать «Методические указания к лабораторной работе № 6» и материалы с сайта [7, 9]. Рассмотрим основные арифметические команды и команды работы со стеком в ассемблере:

* FPTAN – команда, которая вычисляет частичный тангенс ST(0), размещая в стеке такие два числа x и y, что y/x = tg(ST(0)). После выполнения команды число y располагается в ST(0), а число x включается в стек сверху (то есть записывается в ST(1)). Аргумент команды FPTAN должен находится в пределах: 0 <= ST(0) <= pi/4.
* FSIN – команда, которая вычисляет sin(x).
* FCOS – команда, которая вычисляет cos(x).
* FSQRT – команда, которая вычисляет
* FXCH – команда, которая совершает обмен содержимым верхушки стека ST(0) и численного регистра, указанного в качестве операнда команды.

В данной лабораторной работе также использовались операторы и команды из пункта 4.1.

6.2 Задание

1. В программе необходимо реализовать функцию определения значения некоторой элементарной функции y, зависящей от аргумента x на языке ассемблера с использованием команд арифметического сопроцессора.
2. Значения переменных передаются в качестве параметров функции.
3. Составить таблицу значений функции на указанном отрезке с задаваемым шагом h.
4. Номер вычисления №, значения x и f (x) вывести для контроля на экран.
5. Все параметры функции имеют тип double.
6. Проверку деления на 0 и обработку исключительных ситуаций реализовать в основной программе.
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**:** Вычислить в диапазоне [0.2; 2]

6.3 Решение

Для того, чтобы определить значение элементарной функции, последовательно найдем значение каждого слагаемого в выражении.

Напишем отдельную функцию *pow\_Asm(a,b)* для возведения в степень с входными параметрами *a –* число, возводимое в степень*, b –* степень числа.Инициализируем сопроцессор, добавляем в вершину стека *a, b* и меняем их местами в стеке. Добавляем в стек и меняем местами с *а* в стеке. Вычисляем *ln(a)* с помощью команды FYL2X. Перемножаем значения в стеке и добавляем в вершину . Перемножаем значения в стеке и получаем *b\*ln(a)\*log2(e).* Загружаем значение из ST(0) в вершину стека и округляем его с помощью FRNDINT. Командой FSUB получим разность ST(1) и ST(0) в вершине стека, а затем поменяем местами значения в ST(0) и ST(1). Вычисляем в вершине стека, используя команду F2XM1, а затем загружаем в стек единицу и складываем с полученным ранее выражением. Масштабируем значение в ST(0), выполняя команду FSCALE, и тем самым получаем значение . Возвращаем это значение.

В основной функции инициализируем и присваиваем нуль вещественной переменной *result*, которая будет возвращать вычисленное значение функции. Также инициализируем *c8 = 8* и *pow8 = pow\_Asm(8,x).* Добавляем в стек *pow8, x, c8* и перемножаем последние два значения. Вычисляем синус командой FSIN из полученного произведения, затем три раза загружаем его в стек и умножаем самого на себя, там мы получили второе необходимое слагаемое. Оставляем оба слагаемых в конце стека.

Добавляем в стек *ln(2)* командой FLDLN2, а затем *х* и *с8*, которые перемножаем. Вычисляем *ln(8x*) командой FYL2X и загружаем это значение в стек три раза, перемножая, чтобы получить третье необходимое слагаемое. Оставляем его в конце стека.

Добавляем в стек *х* и *с8,* перемножаем их и вычисляем из этого значения косинус командой FCOS. Загружаем полученное значение в стек три раза, перемножая друг на друга, чтобы получить последнее необходимое слагаемое.

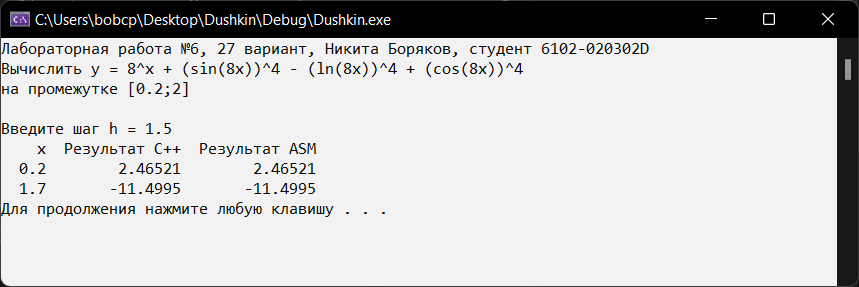
В вершину стека последовательно складываем и вычитаем полученные ранее значения, получая необходимое значение выражения в вершине стека. И извлекаем его из стека в переменную *result*.

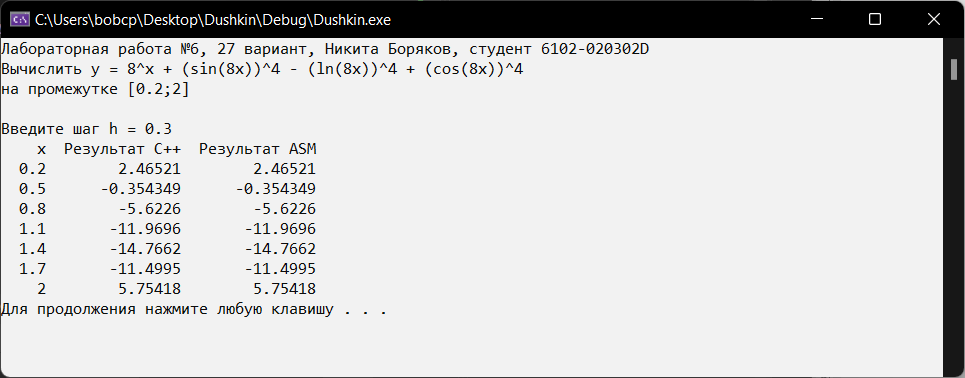
Выводим значение переменной *result*.

[Текст программы приведен в приложении А.6.](#Листинг6)

6.4 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 6.2 – 6.3 при различных значениях аргумента.

  
Рисунок 6.2 – Пример работы алгоритма при значении аргумента h=1.5

  
Рисунок 6.3 – Пример работы алгоритма при значении аргумента h=0.3

Лабораторная работа 7 «Вычисление определенного интеграла методом Симпсона»

7.1 Теоретические основы лабораторной работы

Для решения лабораторной работы будем использовать «Методические указания к лабораторной работе № 7» и материалы с сайта [7, 10].

Команды, используемые при выполнении лабораторной работы указаны в пункте 6.1.

7.2 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
3. Все параметры уравнения имеют тип double.
4. Проверку деления на 0 реализовать также на встроенном ассемблере.
5. Если не найден корень интеграла, то вывести соответствующее сообщение.
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.

Условие:.

7.3 Схема алгоритма

На рисунке 7.1 приведена схема алгоритма нахождения определенного интеграла методом Симпсона, с использованием команд арифметического сопроцессора на встроенном ассемблере. На рисунке 7.1 приведена схема основного алгоритма.

В функции *calcASM* будет вычисляться подынтегральное выражение на языке Ассемблера.

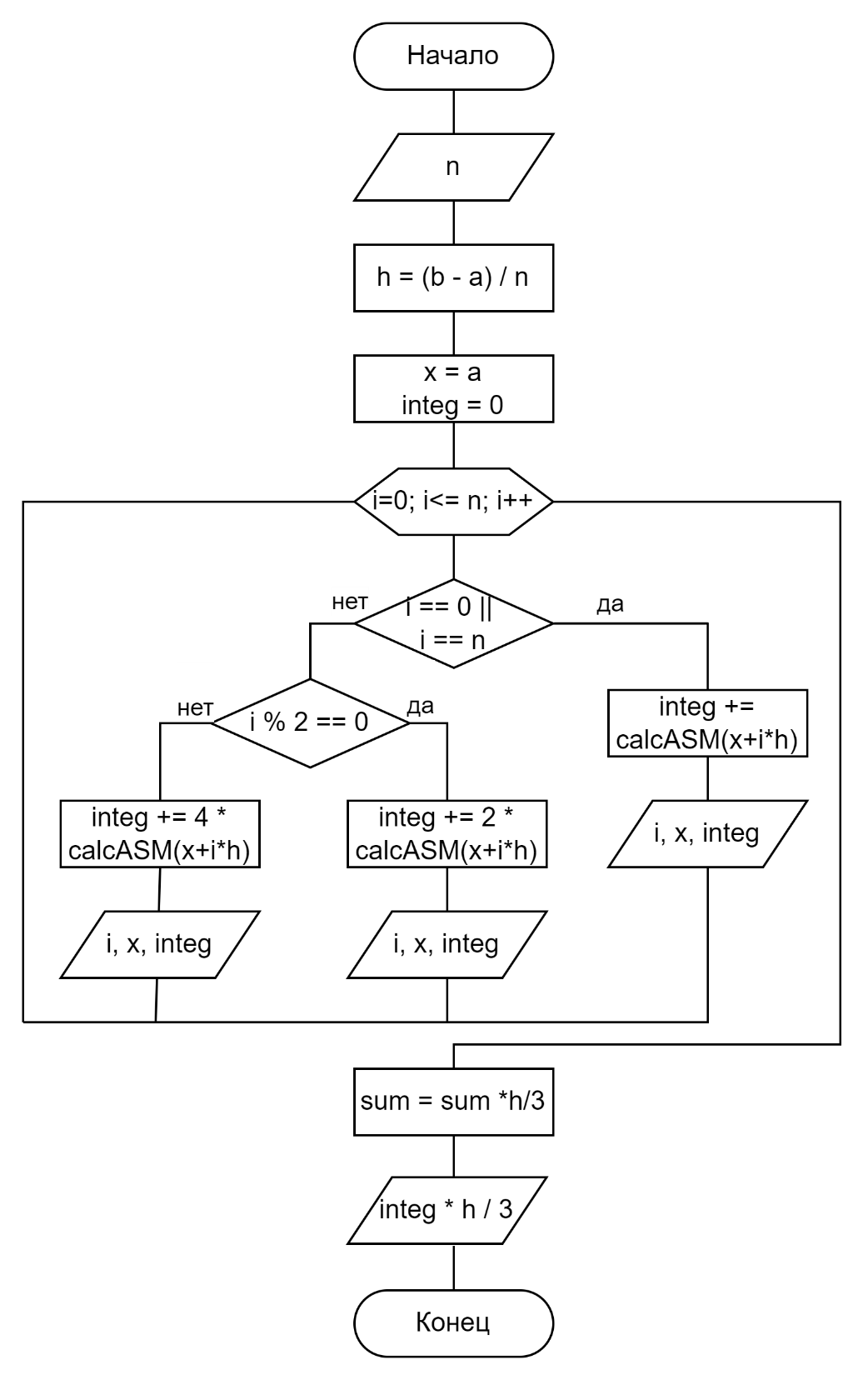
Объявляем и инициализируем переменные *f = 0, c1 = 1*. Инициализируем сопроцессор и заносим в стек значение *с1,* *lg(2), x*, а затем командой FYL2X получаем значение *lg(x)*. Дублируем это значение в вершину стека и перемножаем. Затем к полученному значению добавляем единицу. Сохраняем вещественное значение из вершины стека в переменной *result* и возвращаем переменную *result.*

В основной программе пользователь вводит четное число интервалов *n,* на которое будет делиться интеграл. Инициализируем и присваиваем значение переменной *х=a*. Вычисляем значение . Инициализируем вещественные переменные *integ = 0,* где *integ* – сумма вычисленных выражений.

Создаем цикл от переменной *i=0*, где *i* будет номером элемента суммы. Если элемент является первым или последним, то i*nteg += calcASM(x + i \* h).* Иначепроверяем *i* на четность. Если *i* четное число, *integ += 2\*calcASM(x + i \* h),* если нечетное, то *integ += 4 \* calcASM(x + i \* h).*

Для того, чтобы наглядно видеть результат, будем выводить значения в виде таблицы, где указан номер элемента, значение x, значение функции *FAsm* и, для проверки, будем вычислять выражение на языке C++. В конце цикла выводим сумму.

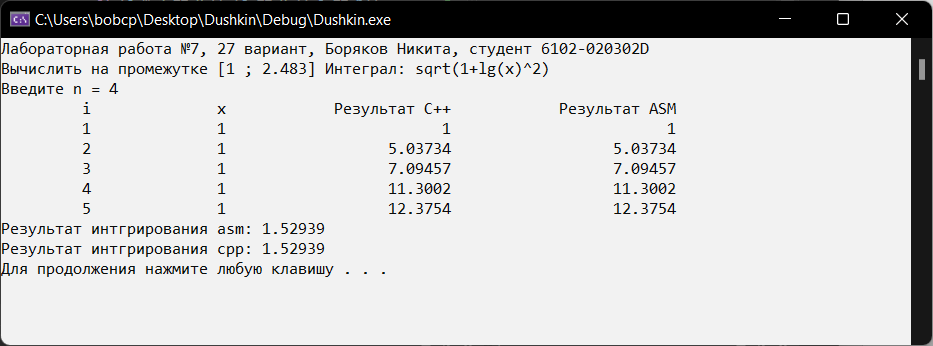
Если же *n* оказалось нечетным или отрицательным, выводим сообщение, что было введено неправильное число.

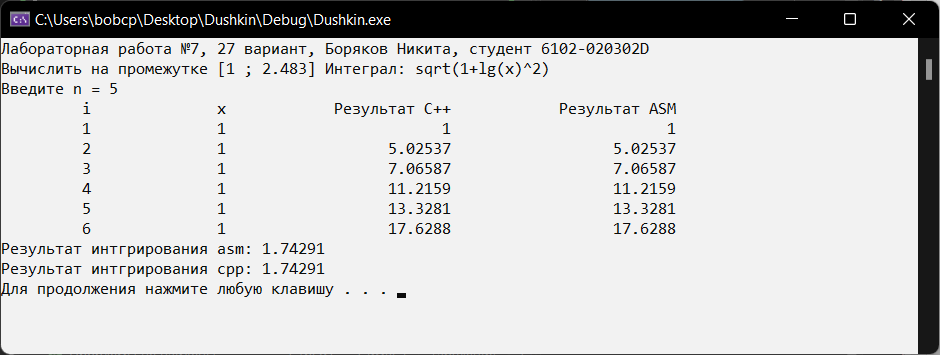
  
Рисунок 7.1 – Схема алгоритма вычисления интеграла

[Текст программы приведен в приложении А.7.](#Листинг7)

7.4 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 7.2 – 7.3 при различных значениях аргумента

   
Рисунок 7.2 – Пример работы алгоритма при значении аргумента n = 4

   
Рисунок 7.3 – Пример работы алгоритма при значении аргумента n = 5

Лабораторная работа 8 «Вычисление суммы ряда»

8.1 Теоретические основы лабораторной работы

Для решения лабораторной работы будем использовать «Методические указания к лабораторной работе № 8» и материалы с сайта [7, 11]. Команды, используемые при выполнении лабораторной работы указаны в пункте 6.1.

8.2 Задание

1. В программе необходимо реализовать функцию определения значения некоторой элементарной функции y, зависящей от аргумента x 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 N, т.е. выход из вычислительной процедуры может пройти не по условию , а по условию k > N. Значение функции и количество итераций вывести для контроля на экран.
3. Значение параметров x , ε и N передаются в качестве аргументов функции.
4. В программе необходимо также реализовать функцию вычисления значения элементарной функции на основе аналитического выражения, также с использованием команд арифметического сопроцессора. Значение функции вывести для контроля на экран.
5. Необходимо определить достигнутую погрешность, вычислив отклонение аналитического значения от значения, вычисленного с помощью ряда. Значение погрешности также вывести для контроля на экран.
6. В качестве комментария к строкам, содержащим команды сопроцессора необходимо указать состояние регистров сопроцессора.

Условие**:** .

8.3 Решение

Для того, чтобы приступить к выполнению задания, сначала найдем рекуррентную формулу подсчета суммы ряда.

8.4 Схема алгоритма

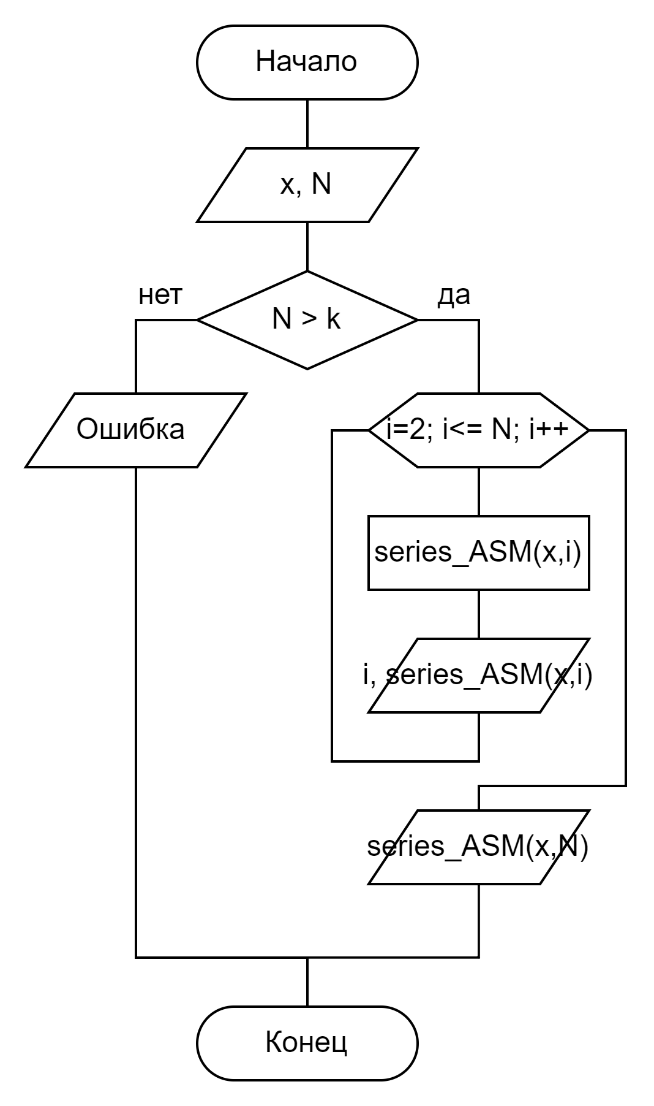
На рисунке 8.1 приведена общая схема алгоритма нахождения суммы ряда в соответствии с заданием.

В параметры функции передаются значения переменных, вводимых пользователем в главной программе: х – аргумент функции, N – количество членов ряда.

Объявляем и инициализируем переменные. Переменным *res*, отвечающей за хранения результата вычисления исходного выражения, и *counter*, считающей количество итераций, присваиваем 0 и 2. Инициализируем переменные *c05 = 1/2, c2 = 2, c0 = 0*. Инициализируем сопроцессор, заносим в стек *x* и *х*, а в регистр *ecx = n.* Перемножаем значения в стеке, добавляем в него значение переменной с2. Меняем местами первые два значения, находящиеся в стеке и выполняем деление.

Ставим метку *calc* для части алгоритма, которая будет выполняться до достижения условия выхода. Перемножаем сумму ряда и крайний элемент ряда во вторую ячейку стека, увеличиваем на единицу счетчик. Добавляем в вершину стека значение счетчика *n* для данной итерации.

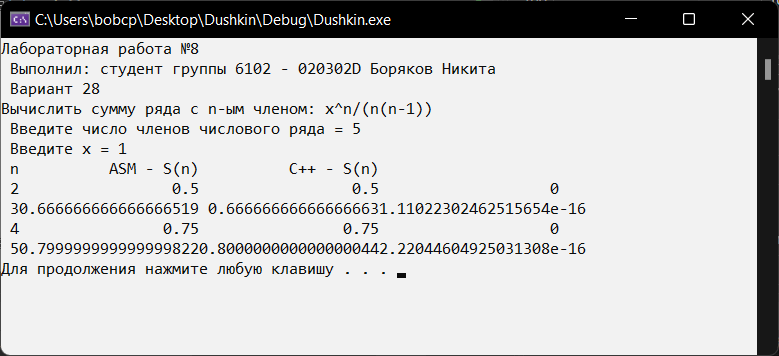
Добавляем в стек значение *с2*, вычитаем из *n,* а затем делим на него же. Перемножаем полученное выражение с предыдущим элементом. Далее сравниваем значение счетчика *n* с параметром функции *N*. Если n < N, возвращаемся к метке *calc.* Иначе сохраняем вещественное значение из вершины стека в переменной *res* и выводим переменную *res.*

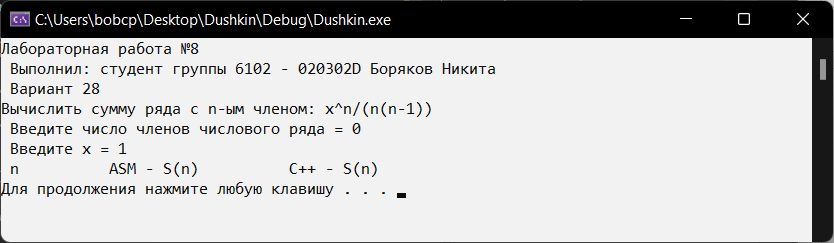
  
Рисунок 8.1 – Схема алгоритма вычисления исходного выражения

[Текст программы приведен в приложении А.8.](#Листинг8)

8.5 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 8.2 – 8.3 при различных значениях аргумента

   
Рисунок 8.2 – Пример работы алгоритма при значении аргументов x=1, N=5

   
Рисунок 8.3 – Пример работы алгоритма при значении аргументов x=1, N=0

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, 2015. 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, 2015. 29 с.
5. 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: 07.03.2021).
6. 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.
7. Система команд сопроцессора. [Электронный ресурс]. URL: http://prog-cpp.ru/asm-coprocessor-command/ (дата обращения: 05.05.2022)
8. Оплачко Д.С. Методические указания к лабораторной работе № 5 «Работа с математическим сопроцессором в среде Assembler» / Д.С. Оплачко, Л.С. Зеленко. Самара: изд-во СГАУ, 2015. 13 с
9. Оплачко Д.С. Методические указания к лабораторной работе № 6 «Работа с командами трансцендентных функций в среде Assembler» / Д.С. Оплачко, Л.С. Зеленко. Самара: изд-во СГАУ, 2015. 11 с
10. Оплачко Д.С. Методические указания к лабораторной работе № 7 «Вычисление определенного интеграла методом Симпсона в среде Assembler» / Д.С. Оплачко, Л.С. Зеленко. Самара: изд-во СГАУ, 2015. 10 с
11. Оплачко Д.С. Методические указания к лабораторной работе № 8 «Вычисление суммы ряда в среде Assembler» / Д.С. Оплачко, Л.С. Зеленко. Самара: изд-во СГАУ, 2015. 8 с

Приложение А.1. Листинг программы лабораторной работы №1

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

#include <iostream> // потоковый ввод/вывод

using namespace std; // подключение библиотеки

int calc\_cpp(int a, int b, int c) { return (4 \* a - b - 1) / (c / b + a); }

// функция вычисления выражения (4\*a - b - 1)/(c/b + a);

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

int result = 0;

\_\_asm {

mov eax, a

mov ecx, 4

imul ecx // <eax> 4\*a сохранение данных в eax

mov ebx, b

sub eax, ebx // <eax> 4\*a - b

dec eax // <eax> 4\*a - b - 1

push eax // <eax> сохранение в стек

mov eax, c

mov ebx, b

cdq

idiv ebx // <eax> c/b в ячкйкм eax

mov ecx, a

add eax, ecx // <eax> c/b + a

mov ebx, eax // <ebx> c/b + a

pop eax // выталкивание из стека

cdq

idiv ebx // только eax

mov result, eax // <result> (4\*a - b - 1)/(c/b + a);

}

return result; // возвращаем результат вычисления выражения

}

int main(){

int a, b, c;

setlocale(LC\_ALL, "RUSSIAN");

cout << "Ассемблер. Лабораторная работа №1: Арифметические и логические команды\nВыполнил Боряков Н.С. 6102-020302D, вариант №28\n\nВычисление значения выражения (4\*a - b - 1)/(c/b + a)\n\n";

cout << "Введите a >> ";

cin >> a;

cout << "Введите b >> ";

cin >> b;

cout << "Введите c >> ";

cin >> c;

try{

if (b == 0) // проверка

throw - 1;

cout << "Вывод в С++ " << calc\_cpp(a, b, c) << endl; // вывод и перевод на другую строку еа консоле

cout << "Вывод в Assembler " << calc(a, b, c) << endl;}

catch (int a)

{

cout << "Ошибка " << a;

}

system("PAUSE");

return 0;

}

Приложение А.2. Листинг программы лабораторной работы №2

using namespace std;

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

#include <iostream> // потоковый ввод/вывод

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

if (a > b) {return b / a - 5;}

else if (a == b){return 25;}

else{return (3 \* a - 5) / b;}

}

pair<int, int> calc\_asm(int a, int b){

int result = 0;int err = 0;

\_\_asm {

mov ecx, a; < eсx >= 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, 25; < eax >= 25

jmp exit\_l; 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 <edx:eax> = a

idiv ecx; <eax> = b / a

add eax, -5; <eax> = b / a - 5

jmp exit\_l; переход на конец программы

l\_smaller :

or ebx, ebx; сравнение b и 0

je error; ошибка деление на ноль

imul ecx, 3; <edx:ecx> = 3 \* a

jo error; ошибка переполнение

mov eax, ecx; < eax >= a

add eax, -5; < eax >= 3 \* a - 5

adc edx, -1; коррекция старшей части

cdq; подготовка деления <edx:eax> = a

; <edx:eax> = 3 \* a – 5

idiv ebx; <eax> = (3 \* a – 5) / b

jmp exit\_l; переход на конец программы

error :

mov err, 1

exit\_l :

mov result, eax}

return pair<int, int>(result, err);}

int main(){

setlocale(LC\_ALL, "RUSSIAN");

cout << "Ассемблер. Лабораторная работа № 2. Арифметические команды и команды переходов.\n";

cout << "Выполнил: Боряков Н.С., группа 6102-020302D\n";

cout << "Вариант 28: \nx=b/a-5, если a>b \nx=25, если a=b\nx=(3\*a-5)/b, если a<b" << endl;int a, b; cout << "a = "; // потоковый вывод

cin >> a; // потоковый ввод

printf("b = "); // стандартный вывод

scanf\_s("%d", &b); // стандартный ввод

auto f = calc\_asm(a, b);

if (f.second == 1){cout << "Попытка деления на ноль\n";}

else {cout << "Результат на ассемблере= " << f.first << endl;

cout << "Результат на c++= " << calc\_cpp(a, b) << endl;}

system("PAUSE");

return 0;

}

Приложение А.3. Листинг программы лабораторной работы №3

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

#include <iostream> // потоковый ввод/вывод

using namespace std;

int calc\_cpp(int mas[], int size, int b) { double result = 0;

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

if (mas[i] > 0 && mas[i] >= b) {

result += pow(mas[i], 2); }

} return result;}

int calc\_asm(int mas[], int size\_mas, int b){int result = 0;

\_\_asm {

xor esi, esi //подготовим регистр индекса в массиве

xor edi, edi

mov ebx, mas //ebx указывает на начало массива

mov ecx, size\_mas //счётчик цикла по всем элементам массива

jcxz exit\_1 //завершить если длина массива 0

begin\_loop :

mov eax, [ebx + esi \* 4] //определяем текущий элементz

mov edx, b //подготовка сравнения с b

cmp eax, edx //сравнение a[i] и b

jl end\_loop //если меньше, то завершаем цикл

mov edx, 0 //подготовка сравнения с b

cmp eax, edx //сравнение a[i] и b

jl end\_loop //если меньше, то завершаем цикл

mov edx, eax //подготовка к умножению

imul eax, edx

add edi, eax

end\_loop :

inc esi //переходим к следующему элементу

loop begin\_loop //повторяем цикл для всех элементов

// массива

exit\_1 :

mov eax, edi //возвращаем количество элементов

mov result, eax

}

return result;}

int main(){ setlocale(LC\_ALL, "RUSSIAN");

cout << "Лабораторная работа №3 Вариант №28\nВыполнил студент группы 6102-020302D\nБоряков Никита" << endl;

cout << "В одномерном массиве A={a[i]} целых чисел вычислить сумму квадратов всех положительных элементов массива, удовлетворяющих условию: a[i] >= b." << endl;int b, size;cout << "b = ";cin >> b;

int\* mas;cout << "Введите размер массива" << endl; cin >> size;

if (size < 0) { cout << "Размерность массива не может быть отрицательной" << endl;}

else {if (size == 0) {cout << "Массив пуст" << endl; }

else {mas = new int[size];

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

cout << "[" << i + 1 << "]" << ": ";

cin >> mas[i];}

if (calc\_asm(mas, size, b) == 0) {cout << "В массиве нет элементов подходящих под условие" << endl; }

else {cout << "Резельтат ассемблер " << calc\_asm(mas, size, b) << endl << "Результат С++ " << calc\_cpp(mas, size, b) << endl;}

}

}system("PAUSE");return 0;}

Приложение А.4. Листинг программы лабораторной работы №4

using namespace std;

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

#include <iostream> // потоковый ввод/вывод

double calc\_cpp(double a, double b){

if (a > b){return b / a - 5;} else if (a == b) {return 25;}

else{return (3 \* a - 5) / b; }}

double calc\_asm(double a, double b){

double res; int status; const int c3 = 3; const int c5 = 5; const int c25 = 25;

\_\_asm {

//st0 st1 st2 st3 st4

finit; инициализация сопроцессора

fld qword ptr[b]; b

fld qword ptr[a]; a b

fcom st(1); сравниваем a и b

fstsw status; сохраняем регистр флагов сопроцессора

mov ah, byte ptr[status + 1]

sahf; записываем в регистр флагов процессора

ja a\_bigger; переход если a больше

jb b\_bigger; переход если b больше

fild c25; 25 a b; если равны

jmp endcalc

a\_bigger : ftst; сравнение a с 0

fstsw status; сохраняем регистр флагов сопроцессора

mov ah, byte ptr[status + 1]

sahf; записываем в регистр флагов процессора

je error; переход если a = 0

fdivp st(1), st; b / a

fild c5; 5 b / a

fsubp st(1), st; b / a - 5

jmp endcalc

b\_bigger : fldz; 0 a b

fcomp st(2); сравнение b с 0 a b

fstsw status; сохраняем регистр флагов сопроцессора

mov ah, byte ptr[status + 1]

sahf; записываем в регистр флагов процессора

je error; переход если b = 0

fild c3; 3 a b

fmulp st(1), st; 3 \* a b

fild c5; 5 3 \* a b

fsubp st(1), st; 3 \* a - 5 b

fld st(1); b 3 \* a - 5 b

fdivp st(1), st; (3 \* a - 5) / b

jmp endcalc

error : fldz; формируем результат ошибки

endcalc : fstp res; сохранение результата

}return res;}

int main(){

setlocale(LC\_ALL, "RUSSIAN");

cout << "Ассемблер. Лабораторная работа № 4. Команды арифметического сопроцессора.\n";

cout << "Выполнил: Боряков Н.С., группа 6102-020302D\n";

cout << "Вариант 28: \nx=b/a-5, если a>b \nx=25, если a=b\nx=(3\*a-5)/b, если a<b\n" << endl;

double a, b;cout << "a = ";cin >> a;cout << "b = ";cin >> b;

if ((a > b && a != 0) || (a < b && b != 0) || (a == b))

{cout << "Ответ на ASM: " << calc\_asm(a, b) << endl;

cout << "Ответ на C++: " << calc\_cpp(a, b) << endl;}

else{cout << "Вы ввели некоректные значения!\n";}

system("PAUSE");return 0;}

Приложение [А.5. Листинг программы лабораторной работы №5](#Схема5)

#include <stdio.h>

#include <iostream>

#include <iomanip>

using namespace std;

double calcASMF(double x){

double result;const int c1102 = 1102;const int c4205 = 4205;const int c4999 = 4999;const int c60 = 60;const int c3186 = 3186;

\_asm{

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

finit; //инициализация сопроцессора

fld x; // x

fld x; // x x

fmul st(1), st(0); // x x^2

fld st; x // x x x^2

fmulp st(1), st(0); // x^2 x^2

fmul st(1), st(0); // x^2 x^4

fmul st(1), st(0); // x^2 x^6

fmul st(1), st(0); // x^2 x^8

fmulp st(1), st(0); // x^10

fild c1102;// 1102 x^10

fmulp st(1), st(0); // 1102x^10

fld x; // x 1102x^10

fld x; // x x 1102x^10

fmul st(1), st(0); // x x^2 1102x^10

fld x; // x x x^2 1102x^10

fmulp st(1), st(0); // x^2 x^2 1102x^10

fmul st(1), st(0); // x^2 x^4 1102x^10

fmul st(1), st(0); // x^2 x^6 1102x^10

fmulp st(1), st(0); // x^8 1102x^10

fild c4205; // 4205 x^8 1102x^10

fmulp st(1), st(0); // 4205x^8 1102x^10

fld x; // x 4205x^8 1102x^10

fld x; // x x 4205x^8 1102x^10

fmul st(1), st(0); // x x^2 4205x^8 1102x^10

fld x; // x x x^2 4205x^8 1102x^10

fmulp st(1), st(0); // x^2 x^2 4205x^8 1102x^10

fmul st(1), st(0); // x^2 x^4 4205x^8 1102x^10

fmulp st(1), st(0); // x^6 4205x^8 1102x^10

fild c4999; // 4999 x^6 4205x^8 1102x^10

fmulp st(1), st(0); // 4999x^6 4205x^8 1102x^10

fld x; // x 4999x^6 4205x^8 1102x^10

fld x; // x x 4999x^6 4205x^8 1102x^10

fmulp st(1), st(0); // x^2 4999x^6 4205x^8 1102x^10

fild c60; // 60 x^2 4999x^6 4205x^8 1102x^10

fmulp st(1), st(0); // 60x^2 4999x^6 4205x^8 1102x^10

fild c3186; // 3186 60x^2 4999x^6 4205x^8 1102x^10

faddp st(1), st(0); // 3186+60x^2 4999x^6 4205x^8 1102x^10

faddp st(1), st(0); // 3186+60x^2+4999x^6 4205x^8 1102x^10

faddp st(1), st(0); // 3186+60x^2+4999x^6+4205x^8 1102x^10

faddp st(1), st(0); // 3186+60x^2+4999x^6+4205x^8+1102x^10

fstp result // сохраняем результат функции

return result;}

double calcASMD(double x){

double result;const int c11020 = 11020;const int c33640 = 33640;

const int c29994 = 29994;const int c120 = 120;

\_asm{

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

finit; //инициализация сопроцессора

fld x; // x

fld x; // x x

fmul st(1), st(0); // x x^2

fld x; // x x x^2

fmulp st(1), st(0); // x^2 x^2

fmul st(1), st(0); // x^2 x^4

fmul st(1), st(0); // x^2 x^6

fmulp st(1), st(0); // x^8

fld x; // x x^8

fmulp st(1), st(0); // x^9

fild c11020; // 11020 x^9

fmulp st(1), st(0); // 11020x^9

fld x; // x 11020x^9

fld x; // x x 11020x^9

fmul st(1), st(0); // x x^2 11020x^9

fld st; x // x x x^2 11020x^9

fmulp st(1), st(0); // x^2 x^2 11020x^9

fmul st(1), st(0); // x^2 x^4 11020x^9

fmulp st(1), st(0); // x^6 11020x^9

fld x; // x x^6 11020x^9

fmulp st(1), st(0); // x^7 11020x^9

fild c33640; // 33640 x^7 11020x^9

fmulp st(1), st(0); // 33640x^7 11020x^9

fld x; // x 33640x^7 11020x^9

fld x; // x x 33640x^7 11020x^9

fmul st(1), st(0); // x x^2 33640x^7 11020x^9

fmul st(1), st(0); // x x^3 33640x^7 11020x^9

fmul st(1), st(0); // x x^4 33640x^7 11020x^9

fmulp st(1), st(0); // x^5 33640x^7 11020x^9

fild c29994; // 29994 x^5 33640x^7 11020x^9

fmulp st(1), st(0); // 29994x^5 33640x^7 11020x^9

fld x; // x 29994x^5 33640x^7 11020x^9

fild c120; // 120 x 29994x^5 33640x^7 11020x^9

fmulp st(1), st(0); // 120x 29994x^5 33640x^7 11020x^9

faddp st(1), st(0); // 120x+2994x^5 33640x^7 11020x^9

faddp st(1), st(0); // 120x+2994x^5+33640x^7 11020x^9

faddp st(1), st(0); // 120x+2994x^5+33640x^7+11020x^9

fstp result // сохраняем результат функции

}return result;}

double derivative(double x){return 11020 \* pow(x, 9) + 33640 \* pow(x, 7) + 29994 \* pow(x, 5) + 120 \* x;}

double function(double x){return 1102 \* pow(x, 10) + 4205 \* pow(x, 8) + 4999 \* pow(x, 6) + 60 \* pow(x, 2) + 3186;}

int main(){

try{setlocale(LC\_ALL, "RUSSIAN");

cout << "Лабораторная работа №5, 27 вариант, Боряков Никита, студент 6102-020302D\n";cout << "Найти х с помощью метода Ньютона 1102x^10 + 4205X^8 + 4999x^6 + 60x^2 + 3186 = 0\n";

cout << "Введите начало интервала: \n"; double a, b, e;

cout << "a = "; cin >> a; cout << "Введите конец интервала: \n";

cout << "b = "; cin >> b; cout << "Введите погрешность: \n";

cout << "e = "; cin >> e; double xC = a, xl2 = xC; int i = 1;

cout << "№" << setw(10) << "x"<< setw(10) << "f(x)" << setw(18) << "f'(x)" << setw(20) << "Погрешность"<< endl;

do{xC = xl2 - function(xl2) / derivative(xl2); xl2 = xC;

cout << i << setw(12) << xC<< setw(16) << function(xl2) << setw(16) << derivative(xl2) << setw(16) << (abs(function(xl2)) / abs(derivative(xC))) << endl; i++;}

while (function(xC) != 0 && (abs(function(xl2)) / abs(derivative(xC))) > e && xC <= b);cout << "Результат C++: " << xC << endl;

double xASM = a, xl1 = xASM;i = 1;

cout << "№" << setw(10) << "x" << setw(10) << "f(x)" << setw(18) << "f'(x)" << setw(20) << "Погрешность" << endl;

do{

xASM = xl1 - calcASMF(xl1) / calcASMD(xl1);

xl1 = xASM; cout << i << setw(12) << xASM

<< setw(16) << calcASMF(xl1)<< setw(16) << calcASMD(xASM)

<< setw(16) << (abs(calcASMF(xl1)) / abs(calcASMD(xASM)))

<< endl; i++;

} while (calcASMF(xASM) != 0 && (abs(calcASMF(xl1)) / abs(calcASMD(xASM))) > e && xASM <= b);

cout << "Результат ASM: " << xASM << endl;

system("PAUSE");

return 0;

}

catch (invalid\_argument& e)

{

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

}

}

Приложение [А.6. Листинг программы лабораторной работы №6](#Схема6)

#include <iostream>

#include <stdio.h>

#include <cmath>

#include <iomanip>

using namespace std;

double calcC(double x){

return pow(8, x) + pow(sin(8 \* x), 4) - pow(log(8 \* x), 4) + pow(cos(8 \* x), 4);}

double pow\_Asm(double a, double b) {

double res = 0;const int c1 = 1;

\_\_asm {

finit// st0 st1 st2 st3 st4

fld a// a

fld b// b a

fxch st(1)// a b

fldln2// ln(2) a b

fxch st(1)// a ln(2) b

fyl2x// ln(a) b

fmulp st(1), st(0)// bln(a)

fldl2e// log2(e) bln(a)

fmul// bln(a)log2(e)

fld st// bln(a)log2(e) bln(a)log2(e)

frndint// [bln(a)log2(e)] bln(a)log2(e)

fsub st(1), st// [bln(a)log2(e)] {bln(a)log2(e)}

fxch st(1)// {bln(a)log2(e)} [bln(a)log2(e)]

f2xm1// 2^{bln(a)log2(e)}-1 [bln(a)log2(e)]

fld1// 1 2^{bln(a)log2(e)}-1 [bln(a)log2(e)]

fadd// 2^{bln(a)log2(e)} [bln(a)log2(e)]

fscale// a^b

fstp st(1)//

fstp res//

}

return res;

}

double calcASM(double x) {

double result;const int c8 = 8;double pow8 = pow\_Asm(8, x);

\_asm {

// st(0) st(1) st(2) st(3) st(4) st(5) st(6) st(7)

finit; //инициализация сопроцессора

fld pow8;// 8^x

fld x;// x 8^x

fild c8;// 8 x 8^x

fmulp st(1), st(0);// 8x 8^x

fsin;// sin(8x) 8^x

fld st;// sin(8x) sin(8x) 8^x

fmul st(1), st(0);// sin(8x) sin(8x)^2 8^x

fmul st(1), st(0);// sin(8x) sin(8x)^3 8^x

fmulp st(1), st(0);// sin(8x)^4 8^x

fldln2;// ln(2) sin(8x)^4 8^x

fld x;// x ln(2) sin(8x)^4 8^x

fild c8;// 8 x ln(2) sin(8x)^4 8^x

fmulp st(1), st(0);// 8x ln(2) sin(8x)^4 8^x

fyl2x;// ln(8x) sin(8x)^4 8^x

fld st;// ln(8x) ln(8x)) sin(8x)^4 8^x

fmul st(1), st(0);// ln(8x) ln(8x)^2 sin(8x)^4 8^x

fmul st(1), st(0);// ln(8x) ln(8x)^3 sin(8x)^4 8^x

fmulp st(1), st(0);// ln(8x)^4 sin(8x)^4 8^x

fld x;// x ln(8x)^4 sin(8x)^4 8^x

fild c8;// 8 x ln(8x)^4 sin(8x)^4 8^x

fmulp st(1), st(0);// 8x ln(8x)^4 sin(8x)^4 8^x

fcos;// cos(8x) ln(8x)^4 sin(8x)^4 8^x

fld st;// cos(8x) cos(8x) ln(8x)^4 sin(8x)^4 8^x

fmul st(1), st(0);// cos(8x) cos(8x)^2 ln(8x)^4 sin(8x)^4 8^x

fmul st(1), st(0);// cos(8x) cos(8x)^3 ln(8x)^4 sin(8x)^4 8^x

fmulp st(1), st(0);// cos(8x)^4 ln(8x)^4 sin(8x)^4 8^x

fsubp st(1), st(0);// -cos(8x)^4+ln(8x)^4 sin(8x)^4 8^x

fsubp st(1), st(0);// cos(8x)^4-ln(8x)^4+sin(8x)^4 8^x

faddp st(1), st(0);// cos(8x)^4-ln(8x)^4+sin(8x)^4+8^x

fstp result; }

return result;}

int main(){

try{

setlocale(LC\_ALL, "RUSSIAN");

cout << "Лабораторная работа №6, 27 вариант, Никита Боряков, студент 6102-020302D\n";

cout << "Вычислить y = 8^x + (sin(8x))^4 - (ln(8x))^4 + (cos(8x))^4\n";

cout << "на промежутке [0.2;2]\n\n";

cout << "Введите шаг h = ";

double h;cin >> h;

double x = 0.2;

cout << setw(5) << "x" << setw(15) << "Результат С++" << setw(15) << "Результат ASM" << endl;

while (x <= 2){cout << setw(5) << x << setw(15) << calcC(x) << setw(15) << calcASM(x) << endl;x += h;}

system("PAUSE"); return 0; }

catch (invalid\_argument& e){cout << e.what() << endl; }

}

Приложение [А.7. Листинг программы лабораторной работы №7](#Схема7)

#define \_USE\_MATH\_DEFINES

#include <cmath>

#include <math.h>

#include <iomanip>

#include <stdio.h>

#include <iostream>

using namespace std;double

calcC(double x){return sqrt(1 + pow(log10(x), 2));}

double calcASM(double x){double result;const int c1 = 1;

\_asm {

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

finit; //инициализация сопроцессора

fild c1;// 1

fldlg2;// lg(2) 1

fld x;// x lg(2) 1

fyl2x;// lg(x) 1

fld st;// lg(x) lg(x) 1

fmulp st(1), st(0);// lg(x)^2 1

faddp st(1), st(0);// lg(x)^2+1

fsqrt;// sqrt(lg(x)^2+1)

fstp result // сохраняем результат функции

} return result;}

int main(){

try {

setlocale(LC\_ALL, "RUSSIAN");

double n, x, a = 1, b = 2.483, integ = 0, h, xc, integc = 0, count = 1;

cout << "Лабораторная работа №7, 27 вариант, Боряков Никита, студент 6102-020302D\n";

cout << "Вычислить на промежутке [1 ; 2.483] Интеграл: sqrt(1+lg(x)^2)\n";

cout << "Введите n = ";

cin >> n;

x = a;

xc = a;

h = ((b - a) / (n));

cout << setw(10) << "i" << setw(15) << "x" << setw(25) << "Результат С++" << setw(25) << "Результат ASM" << endl;

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

if (i % 2 == 0 && i != 0 && i != n)

{

integ += 2 \* calcASM(x + i \* h);

}

else

{

if (i % 2 == 0 && (i == 0 || i == n))

{

integ += calcASM(x + i \* h);

}

else

{

integ += 4 \* calcASM(x + i \* h);

}

}

if (i % 2 == 0 && i != 0 && i != n) {integc += 2 \* calcC(xc + i \* h);}

else {if (i % 2 == 0 && (i == 0 || i == n))

{ integc += calcC(xc + i \* h); }

else { integc += 4 \* calcC(xc + i \* h); }

}

cout << setw(10) << count << setw(15) << x << setw(25) << integc << setw(25) << integ << endl;

count++;}

cout << "Результат интгрирования asm: " << integ \* h / 3 << endl;

cout << "Результат интгрирования cpp: " << integc \* h / 3 << endl;

system("PAUSE"); return 0;}

catch (invalid\_argument& e){ cout << e.what() << endl;}

}

Приложение [А.8. Листинг программы лабораторной работы №8](#Схема8)

#define \_USE\_MATH\_DEFINES

#include <cmath>

#include <iomanip>

#include <stdio.h>

#include <iostream>

using namespace std;

double series\_ASM(double x, int n){

int status;

const int c2 = 2;

const int c05 = 1 / 2;

const int c0 = 0;

int counter = 2;//x^n/(n(n-1))

double result; // x(n-2)/n

\_\_asm {

xor eax, eax

xor ebx, ebx

xor edx, edx

xor ecx, ecx

mov ecx, n // ecx = n

finit //

fld x // x

fld st// x x

fmul st, st(1)// x^2 x

fild c2// 2 x^2 x

fxch st(1)//x^2 2 x

fdiv st, st(1)// x^2/2 2 x

calc :

fadd st(1), st(0) // s sum +s x

inc counter

fmul st(0), st(2) // s\*x sum + s x

fild counter // k s\*x sum + s x

fild c2 // 2 k s\*x sum + s x

fsubp st(1), st(0)//k-2 s\*x sum + s x

fild counter//k k-2 s\*x sum + s x

fdiv//k-2/k s\*x sum + s x

fmulp st(1), st(0)// s\*x\*(k-2)/k sum + s x

cmp ecx, counter;

jge calc

jl endcalc

endcalc :

fstp result //сброс с вершины стека текущего члена s

fstp result

} n = counter; return result - 2;}

double series\_CPP(double x, int n){

double result = 0;

for (int i = 2; i <= n; i++){ result += pow(x, i) / (i \* (i - 1)); }

return result;}

int main(){ setlocale(LC\_ALL, "RUSSIAN");double x; int n;

cout << "Лабораторная работа №8 " << endl << " Выполнил: студент группы 6102 - 020302D Боряков Никита"

<< endl << " Вариант 28 " << endl <<"Вычислить сумму ряда с n-ым членом: x^n/(n(n-1))" << endl;cout << " Введите число членов числового ряда = ";

cin >> n; cout << " Введите x = ";cin >> x;cout.precision(18);

cout << setw(2) << "n" << setw(20) << "ASM - S(n)" << setw(20) << "C++ - S(n)" << endl;

for (int i = 2; i <= n; i++) { cout << setw(2) << i << setw(20) << series\_ASM(x, i) << setw(20) << series\_CPP(x, i) << setw(20) << series\_CPP(x, i) - series\_ASM(x, i) << endl;

} system("PAUSE"); return 0;}