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 № 42

Обучающийся в группе 6103\_020302 М.А Мананников

Руководитель Д.С Оплачко

Самара 2022

СОДЕРЖАНИЕ

[Лабораторная работа 1 «Арифметические и логические команды в ассемблере» 4](#_Toc104539750)

[1.1 Теоретические основы лабораторной работы 4](#_Toc104539751)

[1.2 Задание 5](#_Toc104539752)

[1.3 Схема алгоритма 5](#_Toc104539753)

[1.4 Результаты тестирования 7](#_Toc104539754)

[Лабораторная работа 2 «Арифметические команды и команды переходов в ассемблере» 8](#_Toc104539755)

[2.1 Теоретические основы лабораторной работы 8](#_Toc104539756)

[2.2 Задание 8](#_Toc104539757)

[2.3 Схема алгоритма 9](#_Toc104539758)

[2.4 Результаты тестирования 9](#_Toc104539759)

[Лабораторная работа 3 «Команды работы с массивами и стеком» 11](#_Toc104539760)

[3.1 Теоретические основы лабораторной работы 11](#_Toc104539761)

[3.2 Задание 11](#_Toc104539762)

[3.3 Схема алгоритма 11](#_Toc104539763)

[3.4 Результаты тестирования 12](#_Toc104539764)

[Лабораторная работа 4 «Изучение работы математического сопроцессора в среде Assembler» 13](#_Toc104539765)

[4.1 Теоретические основы лабораторной работы 13](#_Toc104539766)

[4.2 Задание 14](#_Toc104539767)

[4.3 Схема алгоритма 14](#_Toc104539768)

[4.4 Результаты тестирования 14](#_Toc104539769)

[Лабораторная работа 5 «Нахождение корня уравнения f(x) = 0 методом Ньютона» 16](#_Toc104539770)

[5.1 Теоретические основы лабораторной работы 16](#_Toc104539771)

[5.1 Задание 16](#_Toc104539772)

[5.3 Решение 16](#_Toc104539773)

[5.4 Результаты тестирования 17](#_Toc104539774)

[Лабораторная работа 6 «Определение значения элементарной функции» 19](#_Toc104539775)

[6.1 Теоретические основы лабораторной работы 19](#_Toc104539776)

[6.2 Задание 19](#_Toc104539777)

[6.3 Решение 20](#_Toc104539778)

[6.4 Результаты тестирования 20](#_Toc104539779)

[Лабораторная работа 7 «Вычисление определенного интеграла методом Симпсона» 22](#_Toc104539780)

[7.1 Теоретические основы лабораторной работы 22](#_Toc104539781)

[7.2 Задание 22](#_Toc104539782)

[7.3 Схема алгоритма 23](#_Toc104539783)

[7.4 Результаты тестирования 23](#_Toc104539784)

[Лабораторная работа 8 «Вычисление суммы ряда» 24](#_Toc104539785)

[8.1 Теоретические основы лабораторной работы 24](#_Toc104539786)

[8.2 Задание 24](#_Toc104539787)

[8.3 Решение 25](#_Toc104539788)

[8.4 Схема алгоритма 25](#_Toc104539789)

[8.5 Результаты тестирования 25](#_Toc104539790)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27](#_Toc104539791)

[ПРИЛОЖЕНИЕ А.1. Листинг программы лабораторной работы №1 29](#_Toc104539792)

[ПРИЛОЖЕНИЕ А.2. Листинг программы лабораторной работы №2 31](#_Toc104539793)

[ПРИЛОЖЕНИЕ А.3. Листинг программы лабораторной работы №3 34](#_Toc104539794)

[ПРИЛОЖЕНИЕ А.4. Листинг программы лабораторной работы №4 36](#_Toc104539795)

[ПРИЛОЖЕНИЕ А.5. Листинг программы лабораторной работы №5 39](#_Toc104539796)

[ПРИЛОЖЕНИЕ А.6. Листинг программы лабораторной работы №6 45](#_Toc104539797)

[ПРИЛОЖЕНИЕ А.7. Листинг программы лабораторной работы №7 48](#_Toc104539798)

[ПРИЛОЖЕНИЕ А.8. Листинг программы лабораторной работы №8 52](#_Toc104539799)

Лабораторная работа 1 «Арифметические и логические команды в ассемблере»

* 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.
* 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.
* ADD – команда выполняет целочисленное сложение двух операндов и флага переноса CF. Результат сложения помещается в первый операнд и выполняется соответствующая установка флагов.
* SUB – команда 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.
* IDIV – команда знакового деления. В единственном операнде указывается делитель.
* 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.
* INC – команда, которая увеличивает целочисленное значение регистра на единицу.
  1. Задание

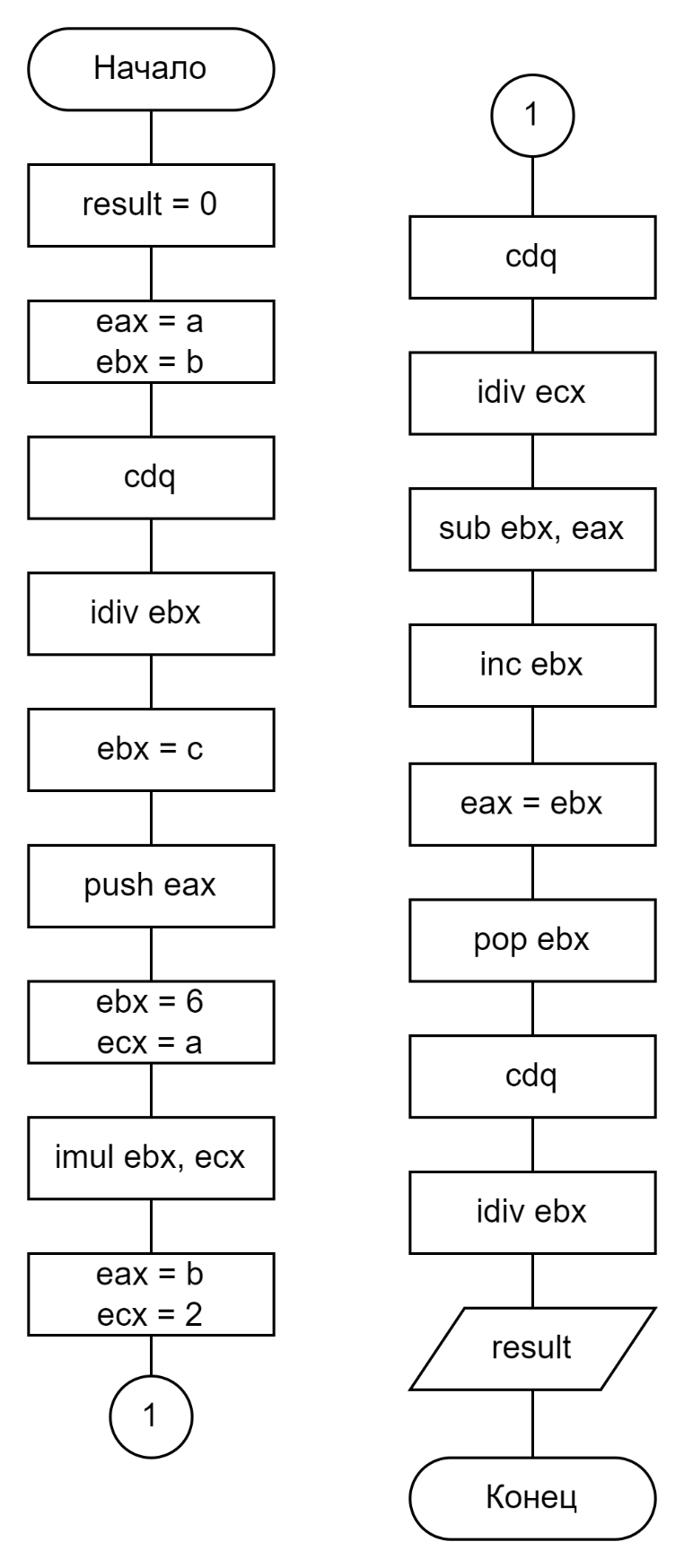
1. В программе необходимо реализовать функцию вычисления целочисленного выражения (1+6\*a – b / 2) / (c+ a \* b) на встроенном ассемблере MASM в среде Microsoft Visual Studio на языке C++.
2. Значения переменных передаются в качестве параметров функции.
3. Результат выводить в консольном приложении (проект 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.

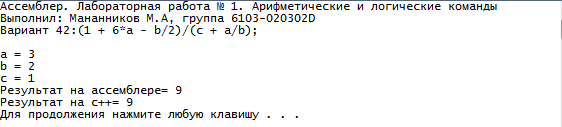
В параметры передаются значения переменных *a, b, c* целочисленного типа из главной программы. Переменной *result*, отвечающей за хранения результата вычисления исходного выражения, присваиваем значение 0. Присваиваем значения регистрам: *eax = a, ebx = b*. Выполняем преобразование *eax* в четверное слово. Делим регистры *eax = eax/eax*. Присваиваем *ebx = c.* Добавляем к регистру *eax, ebx* и сохраняем значение *eax* в стеке.

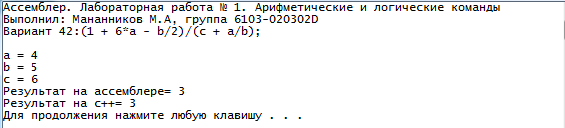
Присваиваем значение *ebx = 6, ecx = a*. Получаем произведение *eax = eax \* ebx*. Присваиваем новое значение регистрам *ecx = а, ebx = 6,* а затем получаем их произведение в регистре *ebx = ebx \* ecx.* Присваиваем новое значение регистрам *eax = b, ecx = 2* и выполняем преобразование *eax* в четверное слово, чтобы получить значение *eax = eax/ecx*. Выполняем вычитание *ebx = ebx – eax* и добавляем к этому значению единицу. Присваиваем *eax = ebx.* Достаем из стека значение в регистр *ebx*. Выполняем преобразование *eax* в четверное слово и выполняем деление. 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 различных значениях\_аргументов.   
Рисунок 1.2 – Пример работы алгоритма при значениях аргумента a = 3,

b =2, c = 1  
Рисунок 1.3 – Пример работы алгоритма при значениях аргумента a = 4,

b =5, c = 6

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]:

* 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.
* 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.
* JMP – инструкция безусловного перехода. Эта инструкция указывает процессору, что в качестве следующей за JMP инструкцией нужно выполнить инструкцию по целевой метке.
* JE – инструкция, которая представляет инструкцию условного  
  перехода, осуществляющая передачу управления только в том случае, если флаг ZF = 1.
* OR – команда объединение по «ИЛИ». Команда осуществляет логическое «ИЛИ» между всеми битами двух операндов. Один из операндов должен быть регистром. Другой операнд может иметь любой режим адресации.

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

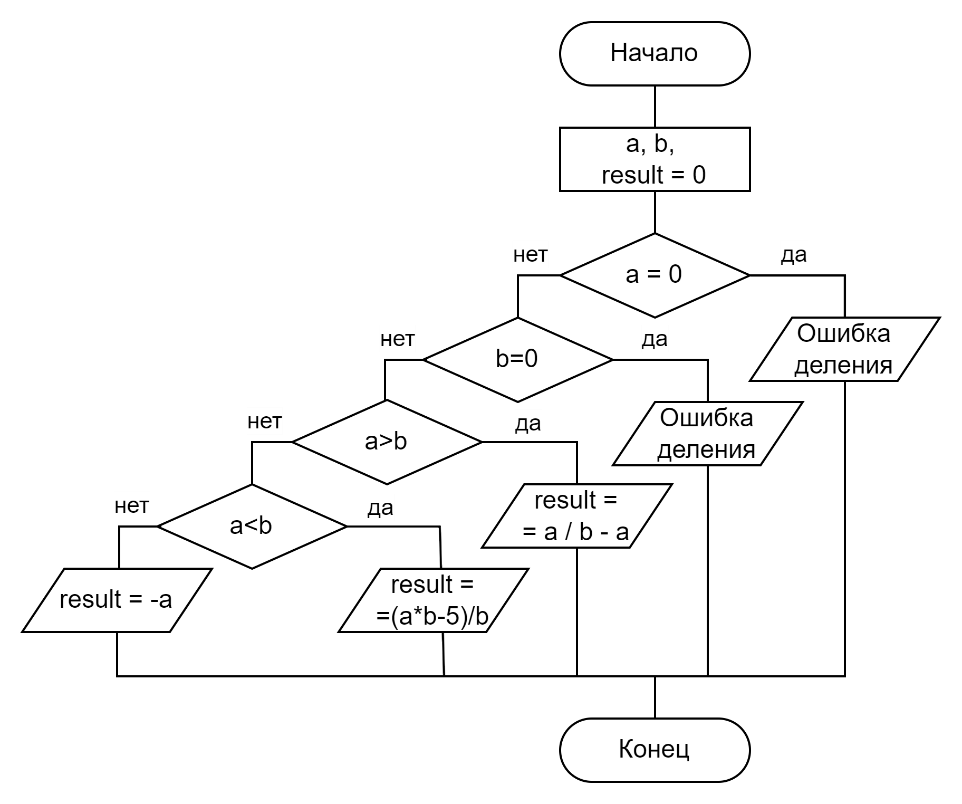
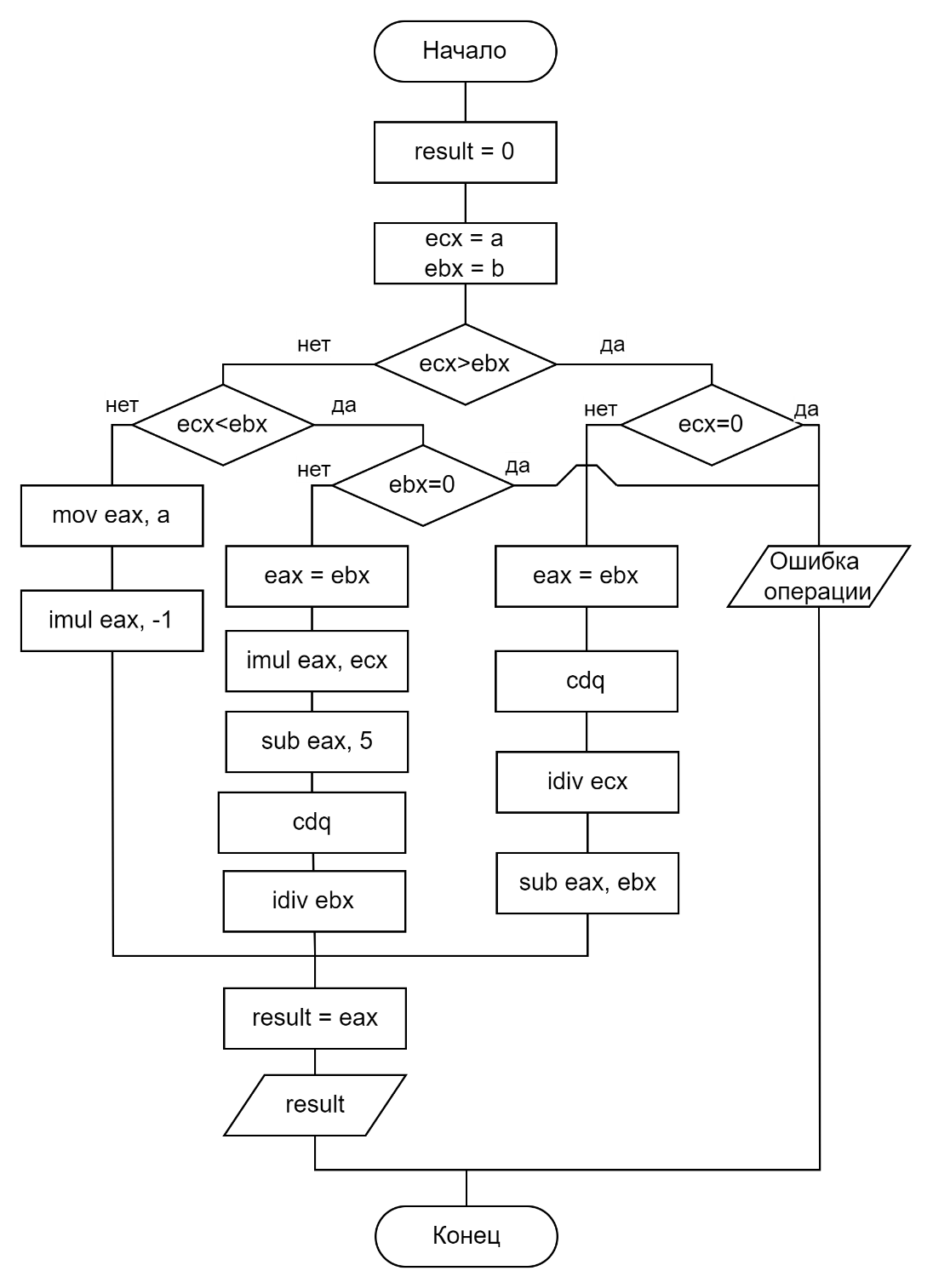
На рисунке 2.1 приведена схема алгоритма вычисления функции условного целочисленного выражения.

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 b на равенство *с* 0. Если *b = 0,* передаем ошибку деления на нуль, иначе присваиваем переменной *result* заданную функцию *result = a / b - a.* Если *a < b*, проверяем *a* на равенство *с* 0. Если *a = 0*, передаем ошибку деления на нуль, иначе присваиваем переменной *result* заданную функцию *result = (a \*b - 5) / b*. В случае равенства *a = b*, переменной result присваивается значение *result = -a*. 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.

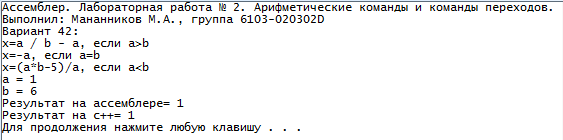
В параметры подаются значения переменных *a, b* целочисленного типа из главной программы. Переменной *result*, отвечающей за хранения результата вычисления исходного выражения, присваиваем значение 0. Присваиваем *ecx* = *b, ebx = a*. Сравниваем значения в регистрах *ecx* и *ebx*. Если значение в *ecx* больше значения в *ebx*, то сравниваем *ecx* с 0. Если *ecx* *=0*, то выводим сообщение об ошибке, в противном случае присваиваем eax = ebx и преобразовываем eax в четверное слово, вызываем команду IDIV и получаем частное *eax=eax / ecx.* Вычитаем командой SUB из полученного частного *а.* 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 = ebx*. Выполняем умножение, а затем вычитаем из значения регистра *eax* 5, а потом выполняем коррекцию целой части. Затем преобразовываем *eax* в четверное слово. Получаем частное *eax = eax / ebx*. В случае равенства *eax = ebx*, присваиваем регистру *eax = а* и умножаем на -1*.*

Присваиваем переменной *result* значение регистра *eax*. 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 различных значениях\_аргументов.   
Рисунок 2.3 – Пример работы алгоритма при значениях аргументов a=1, b = 6

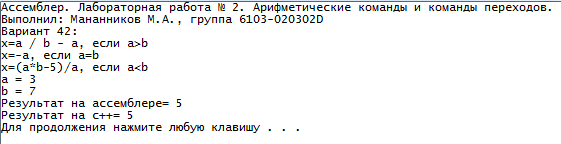
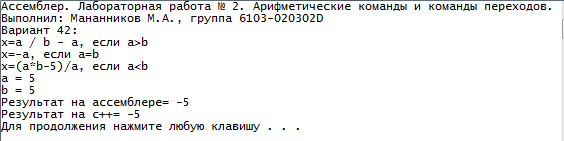
Рисунок 2.4 – Пример работы алгоритма при значениях аргументов a=5, b = 5

Рисунок 2.5 – Пример работы алгоритма при значениях аргументов a=3,b = 7

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.

В данной лабораторной работе также использовались команды из

пункта 2.1.

* 1. 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 среднее арифметическое четных элементов.

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, *size1*, хранящая количество элементов массива. Переменной *result*, отвечающей за хранения результата вычисления исходного выражения, присваиваем нулевое значение.

Регистрам *edi*, отвечающему за произведение 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 = size1*.

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

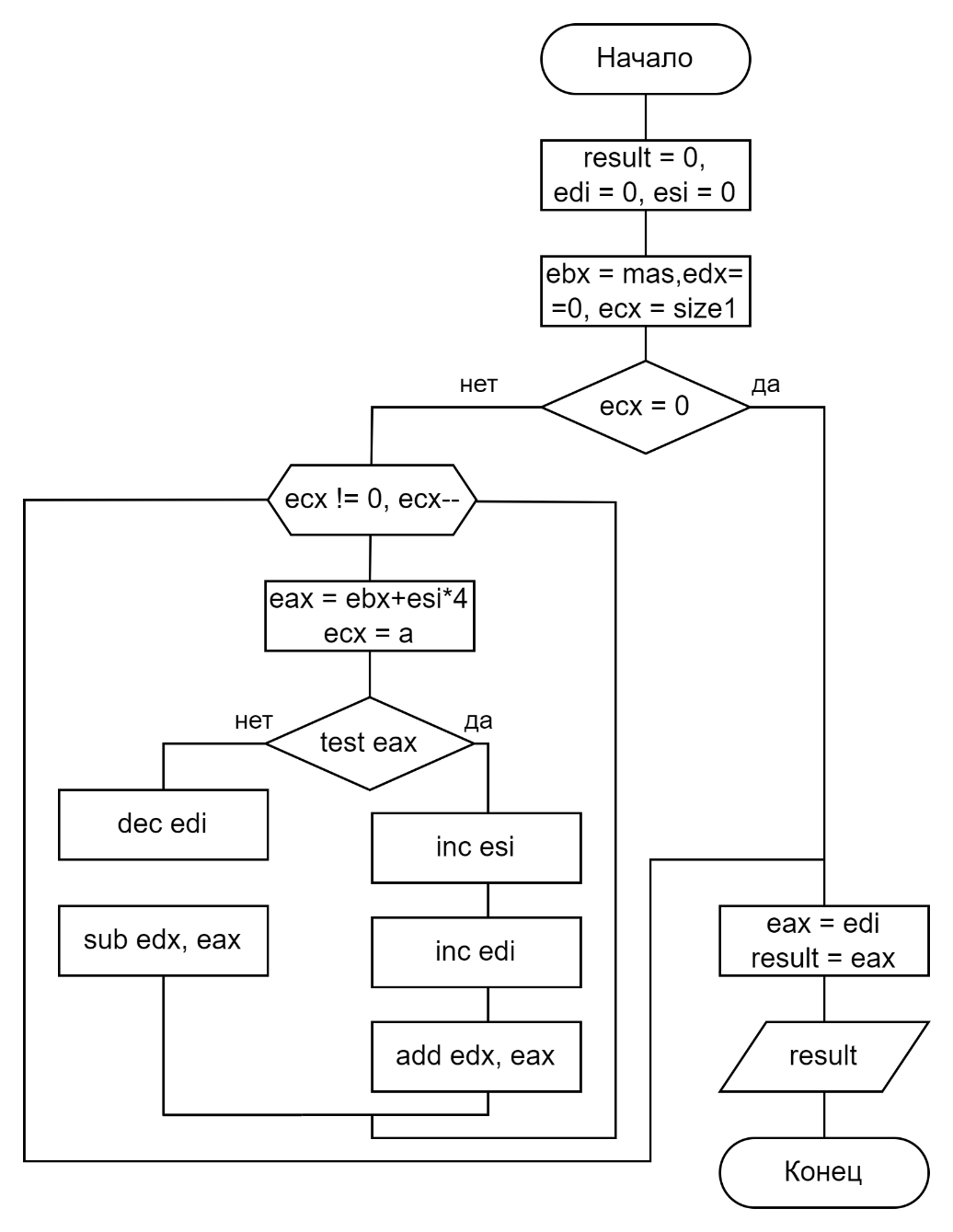
Если длина массива не равна нулю, то до тех пор, пока *ecx* не станет равен нулю, присваиваем регистру *eax* необходимый элемент массива. Определяем текущий элемент: *eax = [ebx + esi \* 4].*

Проверяем элемент на четность. В случае истины, переходим к следующему элементу, прибавляя единицу в счетчик элементов *esi = esi+1.* Записываем элемент массива в сумму и повторяем цикл.

Если число оказалось нечетным, ставим метку *zero* и выполняем уменьшение счетчика *edi = edi+1* и получаем разность *edx = edx – eax.*

После выхода из цикла присваиваем *eax* значение *edi*, а переменной *result* значение регистра *eax*.

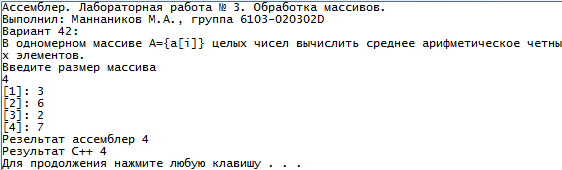
Выводим значение регистра *result*.

  
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 - 3.3 при различных значениях аргумента.

  
Рисунок 3.2 – Пример работы алгоритма при значениях аргументов при n = 4,

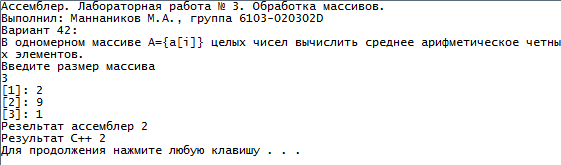
A={3, 6, 2, 7}

Рисунок 3.3 – Пример работы алгоритма при значениях аргументов n = 3, A={2, 9, 1}

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:

* FINIT – команда, которая выполняет инициализацию сопроцессора.
* FLD – команда, которая загружает из памяти в вершину стека ST(0) вещественное число.
* FSTSW – команда, которая выполняет считывание слова состояния сопроцессора в память.
* FCOMP – команда, которая выполняет вещественное сравнение с выталкиванием.
* FCOM – команда, которая выполняет вещественное.
* FMULP – команда, которая выполняет вещественное умножение с выталкиванием. ST(i) = ST(i) \* ST(0).
* FILD – команда, которая загружает из памяти в вершину стека ST(0) целое число.
* FADDP – команда, которая выполняет вещественное сложение с выталкиванием. ST(i) = ST(i) + ST(0).
* FDIVP – команда, которая выполняет вещественное деление с выталкиванием; ST(i) = ST(i) ÷ ST(0).
* FSUBP – команда, которая выполняет вещественное вычитание с выталкиванием. ST(i) = ST(i) - ST(0).
* FTST – команда, которая выполняет анализ ST(0) (сравнивает его с нулем).
* FDIVRP – команда, которая выполняет вещественное реверсивное деление с выталкиванием. ST(i) = ST(0) ÷ ST(i).
* 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.

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.

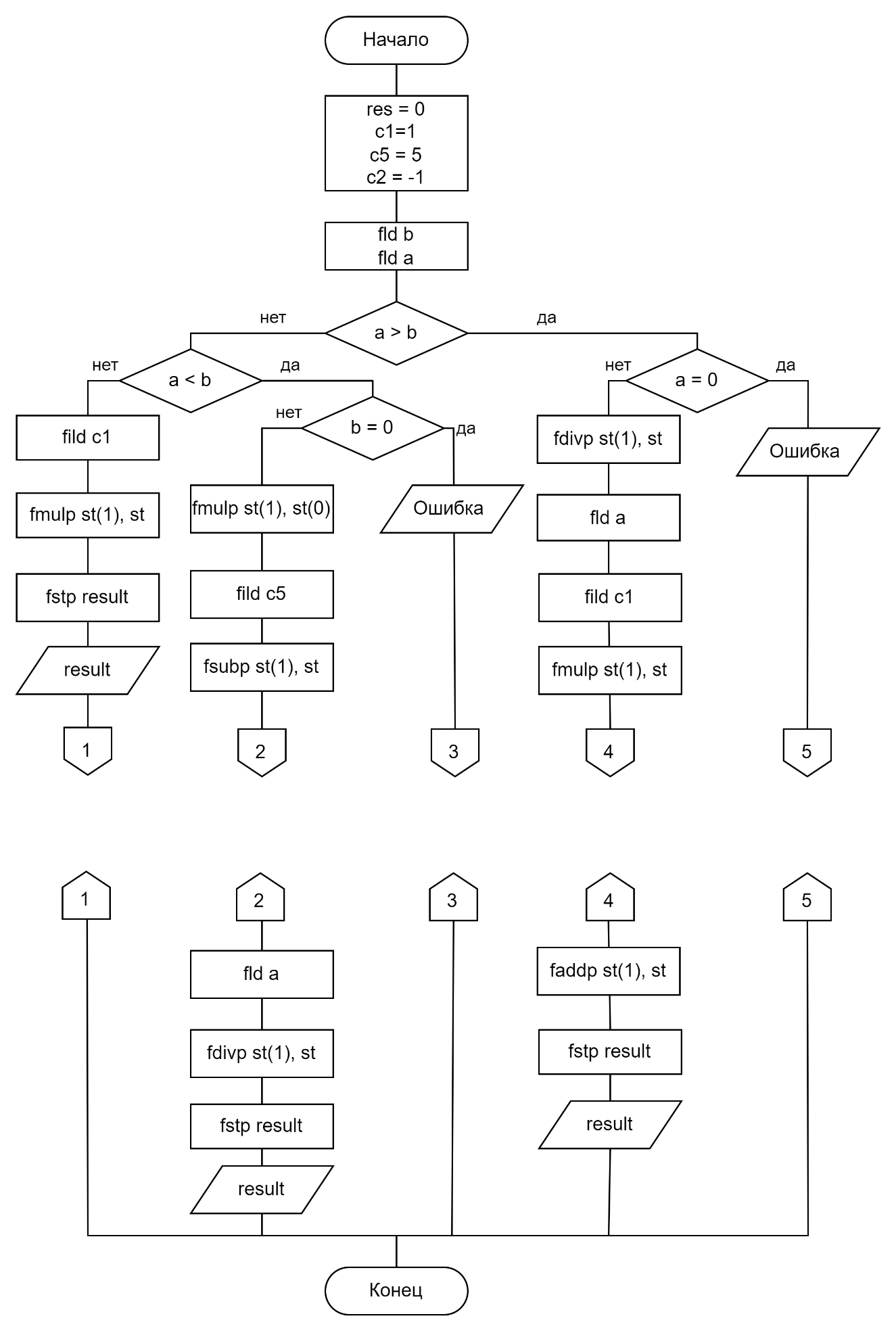
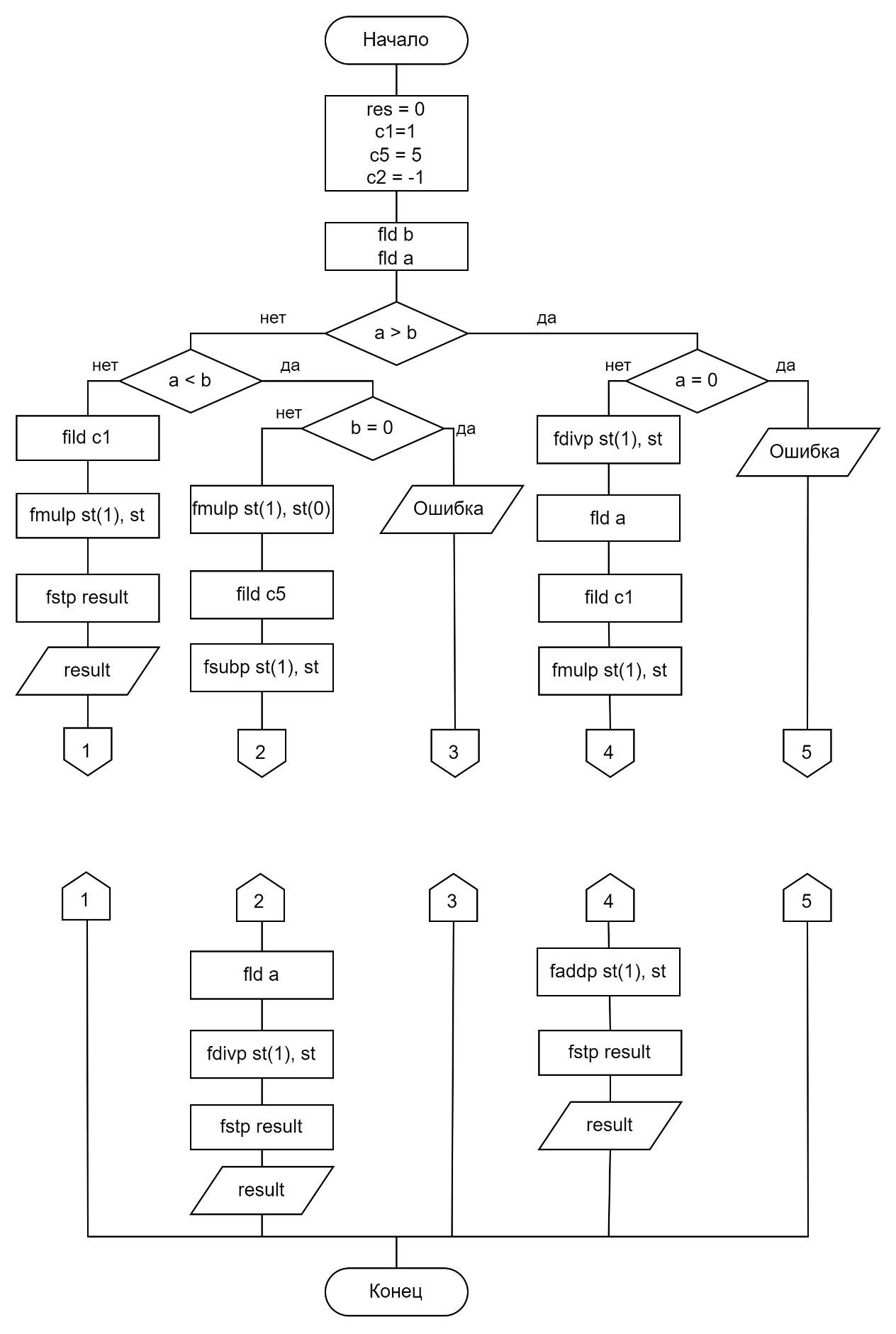
В параметры подаются значения переменных *a, b* вещественного типа из главной программы. Объявляем и инициализируем переменные *с1=-1, с5=5.* Переменной *res*, хранящей результат вычисления исходного выражения, присваиваем значение 0. Инициализируем сопроцессор, загружаем в стек *a* и *b*, сравниваем их.

Если *a > b*, то сравниваем *b* с нулем. Если *b* равно нулю, то выходимиз программы, иначе выполняем деление *а* на *b,* затем загружаем *а* в стек, меняем знак умножение на *-1* и складываем с частным. После этого присваиваем переменной *res* получившееся выражение.

Если *a < b*, то сравниваем *а* с нулем. Если *а* равно нулю, то выходим из программы. В противном случае, перемножаем элементы в стеке, загружаем в него значение *с5,* вычитаем его из полученного произведения. Загружаем в стек *а* и затем делим *ST* на *ST(1)* и полученное частное присваиваем переменной *res*.

В случае равенства *a = b*, загружаем в стек значение с1 и выполняем умножение, затем присваиваем переменной *res* это значение.

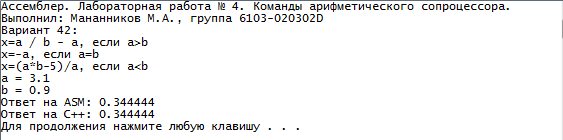
Выводим переменную *res*.

   
Рисунок 4.1 – Схема алгоритма на языке ассемблера (начало)    
Рисунок 4.1 – Схема алгоритма на языке ассемблера (окончание)

[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.

Рисунок 4.2 – Пример работы алгоритма при значениях аргументов a = 3.1,

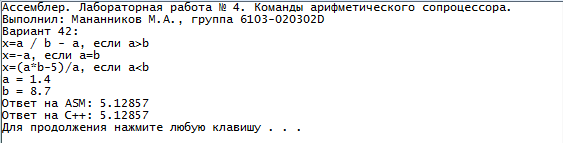
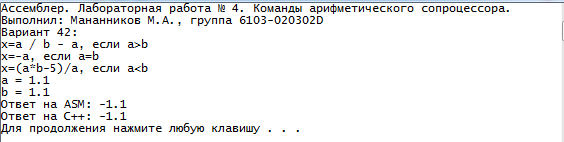
b = 0.9

Рисунок 4.3 – Пример работы алгоритма при значениях аргумента a=1.4, b=8.7

Рисунок 4.4 – Пример работы алгоритма при значениях аргумента 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)

В данной лабораторной работе также использовались команды из

пункта 4.1.

5.1 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: 628 x11 + 2164 x9 + 797 x5 + 4051 x – 2045 = 0

5.3 Решение

Для того, чтобы решить уравнение сначала найдем производную данной функции: f ′(x) = 6908 x10 + 19476 x8 + 3985 x4 + 4051.

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 *f*, хранящую текущее значение функции *fp*, хранящую предыдущее значение функции *f1*, хранящую значение производной, *c*, хранящую значение x и *n*, хранящую номер итерации.

Объявляем и инициализируем переменные *xASM*, хранящую текущее значение функции, *xl1*, хранящую значение производной 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 – calculator\_asm (xl1)/ calculator\_asm (xl1)* Потом xl1 = xASM. *calculator\_asm (c)* – метод подсчёта функции на Assembler, а *calculator\_asm (*c) – метод подсчёта производной на Assembler.

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

5.4 Результаты тестирования

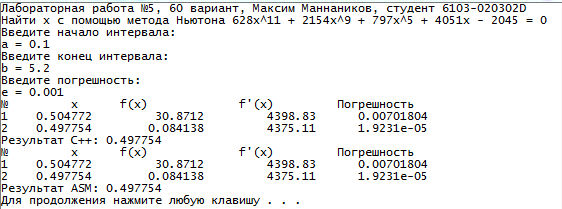
Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 5.2 – 5.3 при различных значениях аргумента.

Рисунок 5.2 – Пример работы алгоритма при значениях аргументов a = 0.1,

b = 5.2, e = 0,001

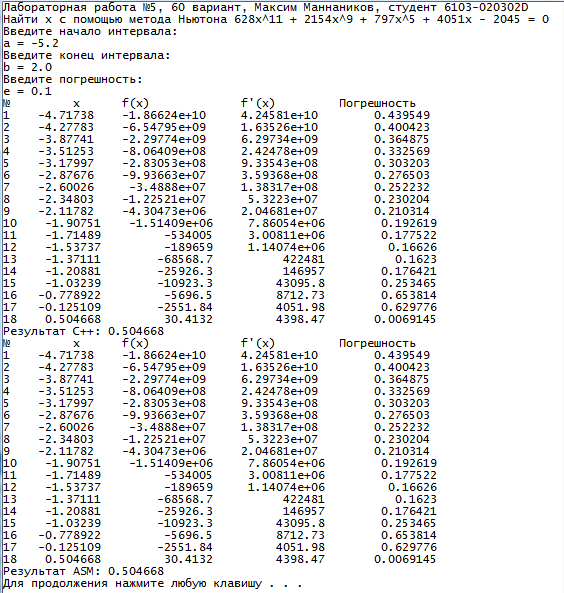


Рисунок 5.3 – Пример работы алгоритма при значениях аргументов

a = -5.2, b = 2.0, e = 0.1

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:

* FXCH – команда, которая совершает обмен содержимым верхушки стека ST(0) и численного регистра, указанного в качестве операнда команды.
* FSCALE – команда, выполняющая масштабирование: изменяет порядок значения, находящегося в вершине стека сопроцессора ST(0) на величину ST(1). Команда не имеет операндов. Величина в ST(1) рассматривается как число со знаком. Его прибавление к полю порядка 28 вещественного числа в ST(0) означает умножение на величину 2 ST(1).
* FRNDINT – команда, которая округляет значение в ST(0).
* FLD1 – команда, которая загружает в вершину стека единицу.
* F2XM1 – команда, которая вычисляет выражение вида: 𝑦 = 2 𝑥 − 1. Исходное значение x размещается в вершине стека сопроцессора ST(0) и должно лежать в диапазоне [-1; 1]. Результат замещает значение в регистре ST(0).
* FLDLN2 – команда, которая вычисляет ln(2).

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 [2;8]

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.

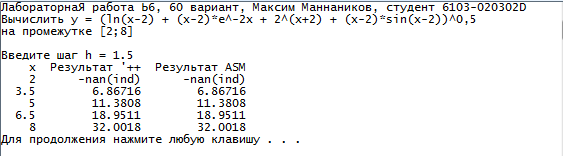
В основной функции инициализируем и присваиваем нуль вещественной переменной *result*, которая будет возвращать вычисленное значение функции. Также инициализируем *c2 = -2, pow2 = pow\_Asm(2,x+2), ePow = pow\_Asm(M\_E, -2\*x).*

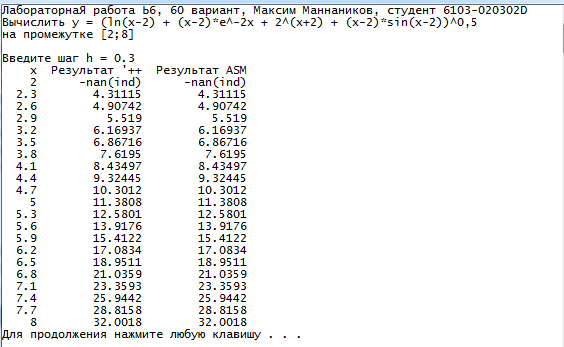
Инициализируем сопроцессор. Добавляем в стек *ln(2), x, c2* и складываем последние два значения. Вычисляем ln(4x) командой FYL2X, затем загружаем в стек *x, c2.* Складываем их и загружаем в стек *ePow*, который умножаем на сумму. Добавляем в стек *pow2, x, c2*. Складываем два последних элемента и повторяем процедуру добавления и сложения еще один раз. Вычисляем синус из значения и умножаем на сумму. Затем последовательно складываем каждое слагаемое, загруженное в стек и получаем необходимое значение подкоренного выражения. Извлекаем из стека в переменную *result* квадратный корень этого выражения.

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 6.3 – Пример работы алгоритма при значениях аргументов h = 1.5

  
Рисунок 6.2 – Пример работы алгоритма при значениях аргументов 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].

В данной лабораторной работе также использовались операторы и команды из пункта 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 ∫ 𝑓(𝑥)𝑑𝑥 𝑎 𝑏 = [(𝑦0 – 𝑦2𝑛 ) + 4(𝑦1 + 𝑦3 + ⋯ + 𝑦2𝑛−1 ) + 2(𝑦2 + 𝑦4 + ⋯ + 𝑦2𝑛−2)
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.

Напишем отдельную функцию *pow\_Asm(a,b)* для возведения в степень с входными параметрами *a –* число, возводимое в степень*, b –* степень числа.Принцип ее работы описан в пункте 6.3.

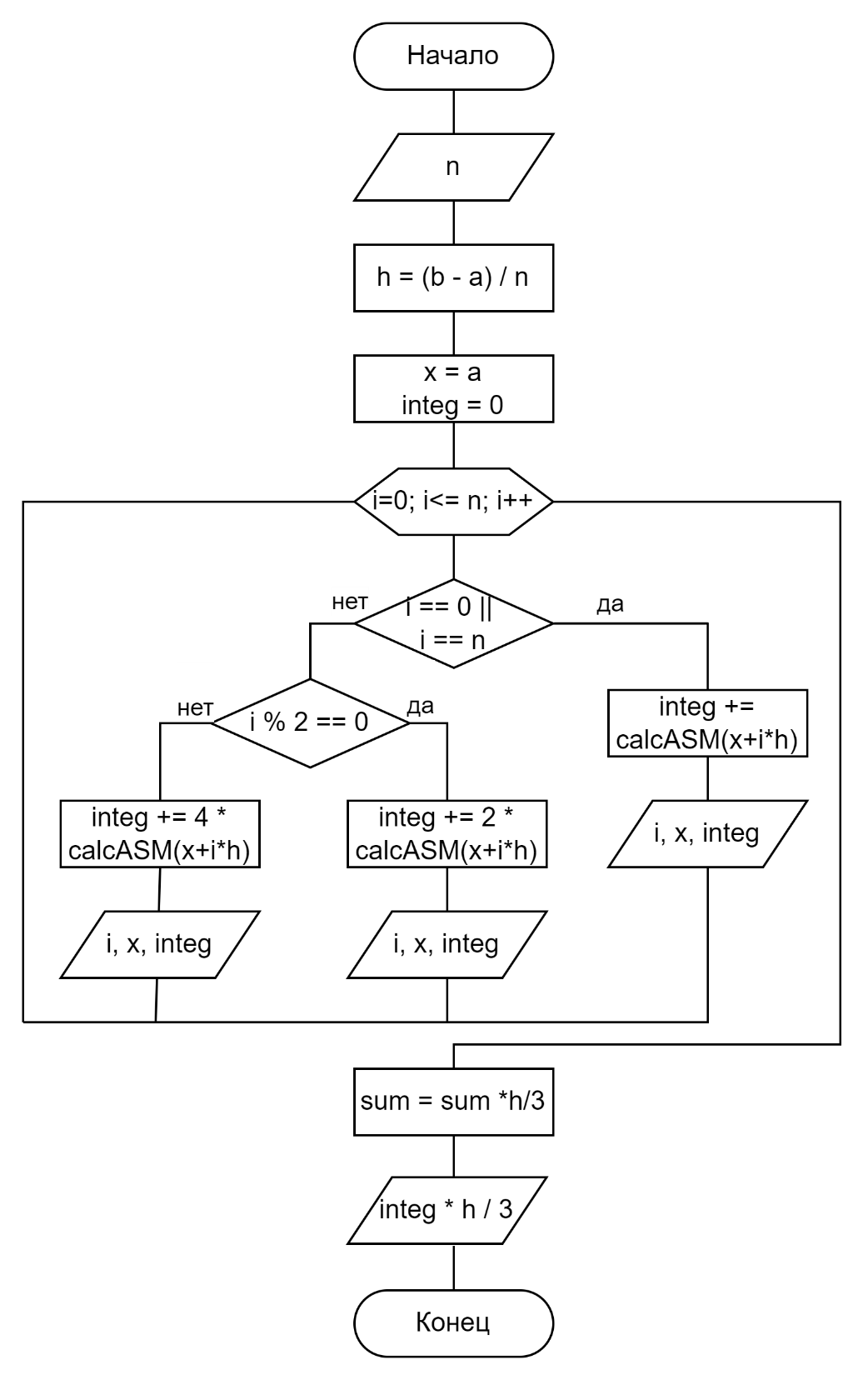
Объявляем и инициализируем переменные *result = 0, c2 = 2, c3 = 3, x34Pow = pow\_Asm(x, 3 / 4), x23Pow = pow\_Asm(x, 2 / 3)*. Инициализируем сопроцессор и заносим в стек значение *c3, x34Pow*, а затем складываем два последних элемента. Загружаем в стек *с2*, *x23Pow* и складываем их друг с другом. Вычисляем квадратный корень и выполняем деление. Сохраняем вещественное значение из вершины стека в переменной *result* и возвращаем переменную *result.*

В основной программе пользователь вводит 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).*

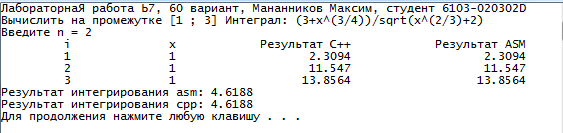
Для того, чтобы наглядно видеть результат, будем выводить значения в виде таблицы, где указан номер элемента, значение x, значение функции *FAsm* и, для проверки, будем вычислять выражение на языке C++. В конце цикла выводим сумму.

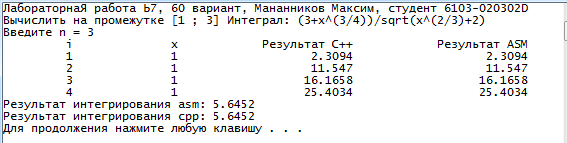
Если же *n* оказалось нечетным или отрицательным, выводим сообщение, что было введено неправильное число.

  
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 значениях аргументов n = 2

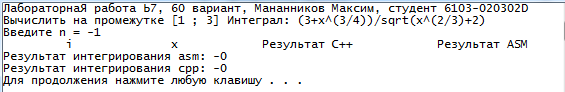
Рисунок 7.3 – Пример работы алгоритма при значениях аргументов n = 3

Рисунок 7.3 – Пример работы алгоритма при значениях аргументов n = -1

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 |𝑆𝑘+1 − 𝑆𝑘 | ≤ ε, где 𝑆𝑘+1 – последующий член ряда; 𝑆𝑘 – предыдущий член ряда. Кроме того, на случай плохой сходимости следует ограничить количество слагаемых сверху некоторым наперёд заданным N , т.е. выход их вычислительной процедуры может произойти не по условию |𝑆𝑘+1 – 𝑆𝑘 | ≤ ε, 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.

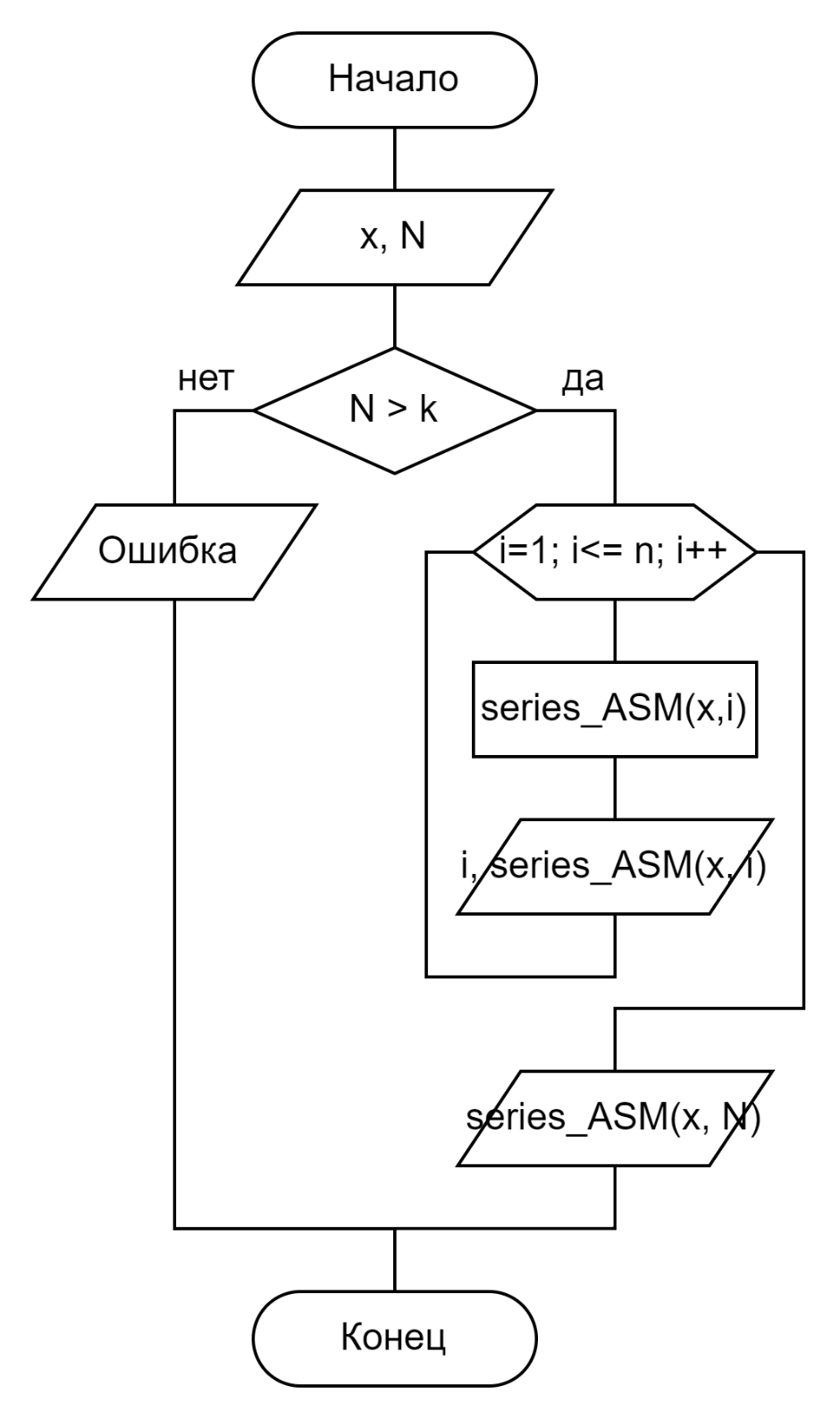
8.4 Схема алгоритма

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. Инициализируем переменные *c1 = -1, c2 = 2, c2m = -2.* Инициализируем сопроцессор, заносим в стек *x, 0* и *1*, а в регистр *ecx = n.* С помощью арифметических операций получаем в вершине стека значение первого элемента.

Ставим метку *calc* для части алгоритма, которая будет выполняться до достижения условия выхода. Последовательно загружаем в стек *х, с2м* и складываем их, дублируем это значение в стек, чтобы перемножить у получить квадрат выражения. Добавляем в стек *counter* и умножаем его значение на с1. Перемножаем два верхних значения в стеке и опять добавляем *counter,* на который делим наше выражение, затем умножаем его на предыдущий элемент ряда, добавляем *с1* и умножаем на него. 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

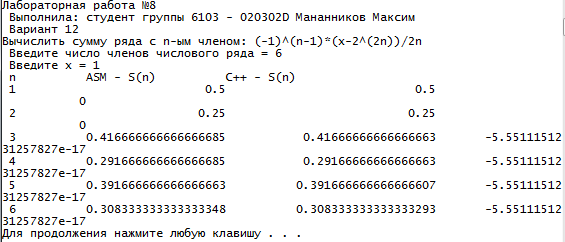
Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 8.2 – 8.4 при различных значениях аргумента.

Рисунок 8.2 – Пример работы алгоритма при значениях аргументов n = 6,

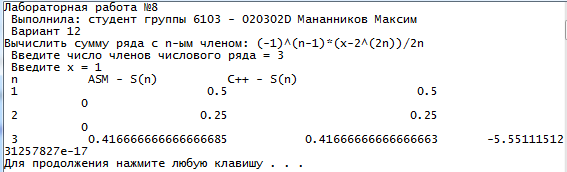
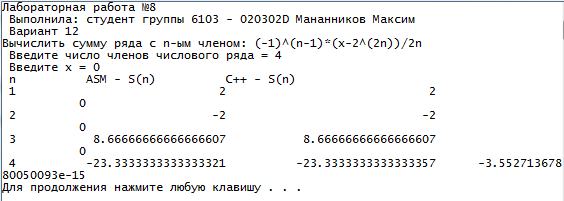
x = 1

Рисунок 8.3 – Пример работы алгоритма при значениях аргументов n = 3,

x = 1

 Рисунок 8.3 – Пример работы алгоритма при значениях аргументов n = 4,

x = 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: 10.03.2021).
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: 10.04.2022)
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

int calc\_cpp(int a, int b, int c){ return (1 + 6 \* a - b / 2) / (c + a/b);}

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

int result = 0;

\_\_asm {

mov eax, a

mov ebx, b

cdq

idiv ebx //<eax>=a/b

mov ebx, c

add eax,ebx

push eax //в стеке c+a/b

mov ebx, 6

mov ecx, a

imul ebx, ecx //<ebx>=6\*a

mov eax, b

mov ecx, 2

cdq

idiv ecx //<eax>=b/2

sub ebx, eax //<ebx>=6\*a-b/2

inc ebx //<ebx>=6\*a-b/2+1

mov eax, ebx //<eax>=6\*a-b/2+1

pop ebx //<ebx>=c+a/b

cdq

idiv ebx //<eax>=(1 + 6 \* a - b / 2) / (c + a/b)

mov result, eax

}

return result; // возвращаем результат вычисления выражения

}

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

std::cout << "Ассемблер. Лабораторная работа № 1. Арифметические и логические команды\n";

std::cout << "Выполнил: Мананников М.А, группа 6103-020302D\n";

std::cout << "Вариант 42:(1 + 6\*a - b/2)/(c + a/b);\n\n";

int a, b, c;

std::cout << "a = "; // потоковый вывод

std::cin >> a; // потоковый ввод

printf("b = ");// стандартный вывод

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

std::cout << "c = ";

std::cin >> c;

if (b == 0 || (c + a / b)==0)

{

std::cout << "Попытка деления на ноль!" << std::endl;

}

else

{

int res\_asm = calc\_asm(a, b, c);// вычисление выражения

std::cout << "Результат на ассемблере= " << res\_asm << std::endl;

int res\_cpp = calc\_cpp(a, b, c);// вычисление выражения

std::cout << "Результат на c++= " << res\_cpp << std::endl;

}

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 a / b - a;

}

else if (a == b)

{

return -a;

}

else

{

return (a\*b-5)/a;

}

}

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

{

int result = 0;

int err = 0;

\_\_asm {

mov ecx, b;// < eсx >= b

mov ebx, a;// < ebx >= a

cmp ebx, ecx;// сравнение a и b

jg l\_bigger;// переход если a > b

jl l\_smaller;// переход если a < b

mov eax, a;// < eax >= a

imul eax, -1;// < eax >= -a

jmp exit\_l;// переход на конец программы

l\_bigger :

or ecx, ecx;// сравнение b и 0

je error;// ошибка деление на ноль

mov eax, ebx;// < eax >= a

cdq;

idiv ecx;// <eax> = a/b

sub eax, ebx;// <eax> = a/b-a

jmp exit\_l;// переход на конец программы

l\_smaller :

or ebx, ebx; сравнение a и 0

je error;// ошибка деление на ноль

mov eax, ebx;// < eax >= a

imul eax, ecx;// < eax >= a\*b

jo error;// ошибка переполнение

sub eax, 5;// < eax >= a\*b-5

adc edx, -1;//коррекция старшей части

cdq;

idiv ebx;// <eax> = (a\*b-5) / a

jmp exit\_l;// переход на конец программы

error :

mov err, 1 exit\_l :

mov result, eax

}

return pair<int, int>(result, err); // возвращаем результат вычисления выражения

}

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

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

cout << "Выполнил: Мананников М.А., группа 6103-020302D\n";

cout << "Вариант 42: \nx=a / b - a, если a>b \nx=-a, если a=b\nx=(a\*b-5)/a, 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

using namespace std;

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

#include <iostream> // потоковый ввод/вывод

int calc\_cpp(int mas[], int size)

{

int count = 0;

int sum = 0;

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

{

if (mas[i] % 2 == 0)

{

count++;

sum += mas[i];

}

}

return sum/count;

}

int calc\_asm(int mas[], int size1)

{ //В одномерном массиве A={a[i]} целых чисел вычислить среднее арифметическое четных элементов.

int result = 0; //В одномерном массиве A={a[i]} целых чисел вычислить разность количества положительных и отрицательных элементов массива.

\_\_asm {

xor esi, esi;// подготовим регистр индекса в массиве

xor edi, edi;// кол-во чётных элемнтов

mov ebx, mas;// ebx указывает на начало массива

mov edx, 0;//сумма чётных элементов массива

mov ecx, size1;// счётчик цикла по всем элементам массива

jcxz exit\_1;// завершить если длина массива 0

begin\_loop:

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

test eax, 1; //Проверка числа на чётность

jz end\_loop;// четное - переход на метку Even

zero:

dec edi;

sub edx, eax;

end\_loop :

inc esi;// переходим к следующему элементу

inc edi;// прибавляем 1 в счётчик элементов

add edx, eax;// записали элемент массива в сумму

loop begin\_loop;// повторяем цикл для всех элементов массива

exit\_1:

mov eax, edx;

mov ecx, edi;

cdq;

idiv ecx; //eax=edx:eax

mov result, eax;// возвращаем разность

}

return result; // возвращаем результат вычисления выражения

}

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

cout << "Ассемблер. Лабораторная работа № 3. Обработка массивов.\n";

cout << "Выполнил: Маннаников М.А., группа 6103-020302D\n";

cout << "Вариант 42: \nВ одномерном массиве A={a[i]} целых чисел вычислить среднее арифметическое четных элементов." << endl;

int size = 0;

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];}

cout << "Резельтат ассемблер " << calc\_asm(mas, size) << endl << "Результат С++ " << calc\_cpp(mas, size) << 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 a / b - a;

}

else if (a == b)

{

return -a;

}

else

{

return (a\*b - 5) / a;

}

}

double calc\_asm(double a, double b)

{

double res;

int status;

const int c5 = 5;const int c1 = -1;

\_\_asm {

//st0 st1 st2 st3 st4

finit;// инициализация сопроцессора

fld qword ptr[a];// a

fld qword ptr[b];// b a

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

jb a\_bigger;// переход если a больше

ja b\_bigger;// переход если b больше

fild c1;

fmulp st(1), st(0);

jmp endcalc

a\_bigger : ftst;// сравнение b 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;// a / b

fld a;// a a / b

fild c1;// -1 a a/b

fmulp st(1), st;// -a a/b

faddp st(1), st;// a/b - a

jmp endcalc

b\_bigger : fldz;// 0 b a

fcomp st(2);// сравнение a с 0 b a

fstsw status;// сохраняем регистр флагов сопроцессора

mov ah, byte ptr[status + 1]

sahf;// записываем в регистр флагов процессора

je error;// переход если b = 0

fmulp st(1), st(0);// a\*b

fild c5;// 5 a\*b

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

fld a;// a a\*b-5

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

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: Мананников М.А., группа 6103-020302D\n";

cout << "Вариант 42: \nx=a / b - a, если a>b \nx=-a, если a=b\nx=(a\*b-5)/a, если a<b" << endl;

double a, b;

cout << "a = ";

cin >> a;

cout << "b = ";

cin >> b;

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

#include <stdio.h>

#include <iostream>

#include <iomanip>

using namespace std;

double calcASMF(double x)

{

double result;

const int c628 = 628;

const int c2154 = 2154;

const int c797 = 797;

const int c4051 = 4051;

const int c2045 = -2045;

\_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

fld x; // x x^10

fmulp st(1), st(0); // x^11

fild c628;// 628 x^11

fmulp st(1), st(0); // 628x^11

fld x; // x 628x^11

fld x; // x x 628x^11

fmul st(1), st(0); // x x^2 628x^11

fld x; // x x x^2 628x^11

fmulp st(1), st(0); // x^2 x^2 628x^11

fmul st(1), st(0); // x^2 x^4 628x^11

fmul st(1), st(0); // x^2 x^6 628x^11

fmulp st(1), st(0); // x^8 628x^11

fld x; // x x^8 628x^11

fmulp st(1), st(0); // x^9 628x^11

fild c2154; // 2154 x^9 628x^11

fmulp st(1), st(0); // 2154x^9 628x^11

fld x; // x 2154x^9 628x^11

fld x; // x x 2154x^9 628x^11

fmul st(1), st(0); // x x^2 2154x^9 628x^11

fld x; // x x x^2 2154x^9 628x^11

fmulp st(1), st(0); // x^2 x^2 2154x^9 628x^11

fmulp st(1), st(0); // x^4 2154x^9 628x^11

fld x; // x x^4 2154x^9 628x^11

fmulp st(1), st(0); // x^5 2154x^9 628x^11

fild c797; // 797 x^5 2154x^9 628x^11

fmulp st(1), st(0); // 797x^5 2154x^9 628x^11

fld x; // x 797x^5 2154x^9 628x^11

fild c4051; // 4051 x 797x^5 2154x^9 628x^11

fmulp st(1), st(0); // 4051x 797x^5 2154x^9 628x^11

fild c2045; // -2045 4051x 797x^5 2154x^9 628x^11

faddp st(1), st(0); // -2045+4051x 797x^5 2154x^9 628x^11

faddp st(1), st(0); // -2045+4051x+797x^5 2154x^9 628x^11

faddp st(1), st(0); // -2045+4051x+797x^5+2154x^9 628x^11

faddp st(1), st(0); // -2045+4051x+797x^5+2154x^9+628x^11

fstp result // сохраняем результат функции

}

return result;

}

double calcASMD(double x)

{

double result;

const int c6908 = 6908;

const int c19386 = 19386;

const int c3985 = 3985;

const int c4051 = 4051;

\_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

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

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

fild c6908; // 6908 x^10

fmulp st(1), st(0); // 6908x^10

fld x; // x 6908x^10

fld x; // x x 6908x^10

fmul st(1), st(0); // x x^2 6908x^10

fld st; x // x x x^2 6908x^10

fmulp st(1), st(0); // x^2 x^2 6908x^10

fmul st(1), st(0); // x^2 x^4 6908x^10

fmul st(1), st(0); // x^2 x^6 6908x^10

fmulp st(1), st(0); // x^8 6908x^10

fild c19386; // 19386 x^8 6908x^10

fmulp st(1), st(0); // 19386x^8 6908x^10

fld x; // x 19386x^8 6908x^10

fld x; // x x 19386x^8 6908x^10

fmul st(1), st(0); // x x^2 19386x^8 6908x^10

fmul st(1), st(0); // x x^3 19386x^8 6908x^10

fmulp st(1), st(0); // x^4 19386x^8 6908x^10

fild c3985; // 3985 x^4 19386x^8 6908x^10

fmulp st(1), st(0); // 3985x^4 19386x^8 6908x^10

fild c4051; // 4051 3985x^4 19386x^8 6908x^10

faddp st(1), st(0); // 4051+3985x^4 19386x^8 6908x^10

faddp st(1), st(0); // 4051+3985x^4+19386x^8 6908x^10

faddp st(1), st(0); // 4051+3985x^4+19386x^8+6908x^10

fstp result // сохраняем результат функции

}

return result;

}

double derivative(double x)

{

return 6908 \* pow(x, 10) + 19386 \* pow(x, 8) + 3985 \* pow(x, 4) + 4051;

}

double function(double x)

{

return 628 \* pow(x, 11) + 2154 \* pow(x, 9) + 797 \* pow(x, 5) + 4051 \* x - 2045;

}

int main()

{

try

{

setlocale(LC\_ALL, "RUSSIAN");

cout << "Лабораторная работа №5, 60 вариант, Максим Маннаников, студент 6103-020302D\n";

cout << "Найти х с помощью метода Ньютона 628x^11 + 2154x^9 + 797x^5 + 4051x - 2045 = 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

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <stdio.h>

#include <cmath>

#include <iomanip>

using namespace std;

double calcC(double x)

{

return sqrt(log(x-2) + (x - 2)\*pow(M\_E,-2 \* x ) + pow(2, (x+2)) + (x-2)\*sin(x-2));

}

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 c2 = -2;

double pow2 = pow\_Asm(2,x+2);

double ePow = pow\_Asm(M\_E, -2 \* x);

\_asm

{

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

finit; //инициализациЯ сопроцессора

fldln2;// ln(2)

fld x;// x ln(2)

fild c2;// -2 x ln(2)

faddp st(1), st(0);// x-2 ln(2)

fyl2x;// ln(x-2)

fld x;// x ln(x-2)

fild c2;// -2 x ln(x-2)

faddp st(1), st(0);// x-2 ln(x-2)

fld ePow;// e^-2x x-2 ln(x-2)

fmulp st(1), st(0);// e^-2x\*(x-2) ln(x-2)

fld pow2;// 2^(x+2) e^-2x\*(x-2) ln(x-2)

fld x;// x 2^(x+2) e^-2x\*(x-2) ln(x-2)

fild c2;// -2 x 2^(x+2) e^-2x\*(x-2) ln(x-2)

faddp st(1), st(0);// x-2 2^(x+2) e^-2x\*(x-2) ln(x-2)

fld x;// x x-2 2^(x+2) e^-2x\*(x-2) ln(x-2)

fild c2;// -2 x x-2 2^(x+2) e^-2x\*(x-2) ln(x-2)

faddp st(1), st(0);// x-2 x-2 2^(x+2) e^-2x\*(x-2) ln(x-2)

fsin;// sin(x-2) x-2 2^(x+2) e^-2x\*(x-2) ln(x-2)

fmulp st(1), st(0);// sin(x-2)\*(x-2) 2^(x+2) e^-2x\*(x-2) ln(x-2)

faddp st(1), st(0);// sin(x-2)\*(x-2)+2^(x+2) e^-2x\*(x-2) ln(x-2)

faddp st(1), st(0);// sin(x-2)\*(x-2)+2^(x+2)+e^-2x\*(x-2) ln(x-2)

faddp st(1), st(0);// sin(x-2)\*(x-2) + 2^(x+2) + e^-2x\*(x-2) + ln(x-2)

fstp result; // сохранЯем результат функции

}

return sqrt(result);

}

int main()

{

try

{

setlocale(LC\_ALL, "RUSSIAN");

cout << "‹абораторнаЯ работа Ь6, 60 вариант, Њаксим Њаннаников, студент 6103-020302D\n";

cout << "‚ычислить y = (ln(x-2) + (x-2)\*e^-2x + 2^(x+2) + (x-2)\*sin(x-2))^0,5\n";

cout << "на промежутке [2;8]\n\n";

cout << "‚ведите шаг h = ";

double h;

cin >> h;

double x = 2;

cout << setw(5) << "x" << setw(15) << "ђезультат ‘++" << setw(15) << "ђезультат ASM" << endl;

while (x <= 8)

{

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

#define \_USE\_MATH\_DEFINES

#include <cmath>

#include <math.h>

#include <iomanip>

#include <stdio.h>

#include <iostream>

using namespace std;

double calcC(double x)

{

return (3 + pow(x, 3 / 4)) / sqrt(2 + pow(x, 2 / 3));

}

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 c2 = 2;

const int c3 = 3;

double x34Pow = pow\_Asm(x, 3 / 4);

double x23Pow = pow\_Asm(x, 2 / 3);

\_asm

{

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

finit; //инициализациЯ сопроцессора

fild c3;// 3

fld x34Pow;// x^(3/4) 3

faddp st(1), st(0);// x^(3/4)+3

fild c2;// 2 x^(3/4)+3

fld x23Pow;// x^(2/3) 2 x^(3/4)+3

faddp st(1), st(0);// x^(2/3)+2 x^(3/4)+3

fsqrt;// sqrt(x^(2/3)+2) x^(3/4)+3

fdiv;// (x^(3/4)+3)/sqrt(x^(2/3)+2)

fstp result // сохранЯем результат функции

}

return result;

}

int main()

{

try

{

setlocale(LC\_ALL, "RUSSIAN");

double n, x, a = 1, b = 3, integ = 0, h, xc, integc = 0, count = 1;

cout << "‹абораторнаЯ работа Ь7, 60 вариант, Њананников Њаксим, студент 6103-020302D\n";

cout << "‚ычислить на промежутке [1 ; 3] €нтеграл: (3+x^(3/4))/sqrt(x^(2/3)+2)\n";

cout << "‚ведите n = ";

cin >> n;

x = a;

xc = a;

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

cout << setw(10) << "i" << setw(15) << "x" << setw(25) << "ђезультат ‘++" << setw(25) << "ђезультат ASM" << endl;

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

{

// ASM

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);

}

}

// c++

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 << "ђезультат интгрированиЯ asm: " << integ \* h / 3 << endl;

cout << "ђезультат 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

#include <cmath>

#include <iostream>

#include <iomanip>

using namespace std;

double series\_ASM(double x, int n)

{

int status;

const int c1 = -1;

const int c2m = -2;

const int c2 = 2;

int counter = 1;

double result;

\_\_asm {

xor eax, eax

xor ebx, ebx

xor edx, edx

xor ecx, ecx

mov ecx, n // ecx = n

finit //

fld x // x

fild c2m // -2 x

faddp st(1), st(0)// x-2

fld st // x-2 x-2

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

fild c2 // 2 (x-2)^2

fdiv //(x-2)^2/2

fldz // 0 (x-2)^2/2

fld x // x 0 (x-2)^2/2

fxch st(2) //s=(x-2)^2/2 sum=0 x

calc :

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

inc counter

fild c2m // -2 s sum +s x

fadd st(0), st(3) // x-2 s sum +s x

fld st // x-2 x-2 s sum +s x

fmulp st(1), st(0) // (x-2)^2 s sum +s x

fild counter // k (x-2)^2 s sum +s x

fild c1 // -1 k (x-2)^2 s sum +s x

faddp st(1), st(0) // k-1 (x-2)^2 s sum +s x

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

fild counter // k (k-1)\*(x-2)^2 s sum +s x

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

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

fild c1 // -1 (k-1)\*(x-2)^2\*s/k sum +s x

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

cmp ecx, counter;

jge calc

jl endcalc

endcalc :

fstp result //сброс с вершины стека текущего члена s

fstp result

}

n = counter;

return result;

}

double series\_CPP(double x, int n)

{

double result = 0;

for (int i = 1; i <= n; i++)

{

result += pow(-1, i - 1)\*pow(x - 2, 2 \* i) / (2 \* i);

}

return result;

}

int main()

{

setlocale(LC\_ALL, "RUSSIAN");

double x;

int n;

cout << "Лабораторная работа №8 " << endl << " Выполнила: студент группы 6103 - 020302D Мананников Максим"

<< endl << " Вариант 12 " << endl <<

"Вычислить сумму ряда с n-ым членом: (-1)^(n-1)\*(x-2^(2n))/2n" << 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 = 1; i <= n; i++)

{

cout << setw(2) << i << setw(30) << series\_ASM(x, i) << setw(30) << series\_CPP(x, i) << setw(30) << series\_CPP(x, i) - series\_ASM(x, i) << endl;

}

system("PAUSE");

return 0;

}