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
1.1 Теоретические основы лабораторной работы 4
1.2 Задание 5
1.3 Схема алгоритма 5
1.4 Результаты тестирования 7
Лабораторная работа 2 «Арифметические команды и команды переходов в ассемблере» 8
2.1 Теоретические основы лабораторной работы 8
2.2 Задание 8
2.3 Схема алгоритма 9
2.4 Результаты тестирования 11
Лабораторная работа 3 «Команды работы с массивами и стеком» 13
3.1 Теоретические основы лабораторной работы 13
3.2 Задание 13
3.3 Схема алгоритма 13
3.4 Результаты тестирования 15
Лабораторная работа 4 «Изучение работы математического сопроцессора в среде Assembler» 17
4.1 Теоретические основы лабораторной работы 17
4.2 Задание 18
4.3 Схема алгоритма 18
4.4 Результаты тестирования 20
Лабораторная работа 5 «Нахождение корня уравнения f(x) = 0 методом Ньютона» 22
5.1 Теоретические основы лабораторной работы 22
5.1 Задание 22
5.3 Решение 22
5.4 Результаты тестирования 23

Лабораторная работа 6 «Определение значения элементарной функции»..... 26
6.1 Теоретические основы лабораторной работы..... 26
6.2 Задание..... 26
6.3 Решение..... 27
6.4 Результаты тестирования..... 28
Лабораторная работа 7 «Вычисление определенного интеграла методом Симпсона»..... 30
7.1 Теоретические основы лабораторной работы..... 30
7.2 Задание..... 30
7.3 Схема алгоритма..... 31
7.4 Результаты тестирования..... 32
Лабораторная работа 8 «Вычисление суммы ряда»..... 34
8.1 Теоретические основы лабораторной работы..... 34
8.2 Задание..... 34
8.3 Решение..... 35
8.4 Схема алгоритма..... 35
8.5 Результаты тестирования..... 36
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ..... 38
ПРИЛОЖЕНИЕ А.1. Листинг программы лабораторной работы №1..... 40
ПРИЛОЖЕНИЕ А.2. Листинг программы лабораторной работы №2..... 42
ПРИЛОЖЕНИЕ А.3. Листинг программы лабораторной работы №3..... 44
ПРИЛОЖЕНИЕ А.4. Листинг программы лабораторной работы №4..... 46
ПРИЛОЖЕНИЕ А.5. Листинг программы лабораторной работы №5..... 49
ПРИЛОЖЕНИЕ А.6. Листинг программы лабораторной работы №6..... 53
ПРИЛОЖЕНИЕ А.7. Листинг программы лабораторной работы №7..... 55
ПРИЛОЖЕНИЕ А.8. Листинг программы лабораторной работы №8..... 59

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.

– 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.

1.2 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 = a, 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*.

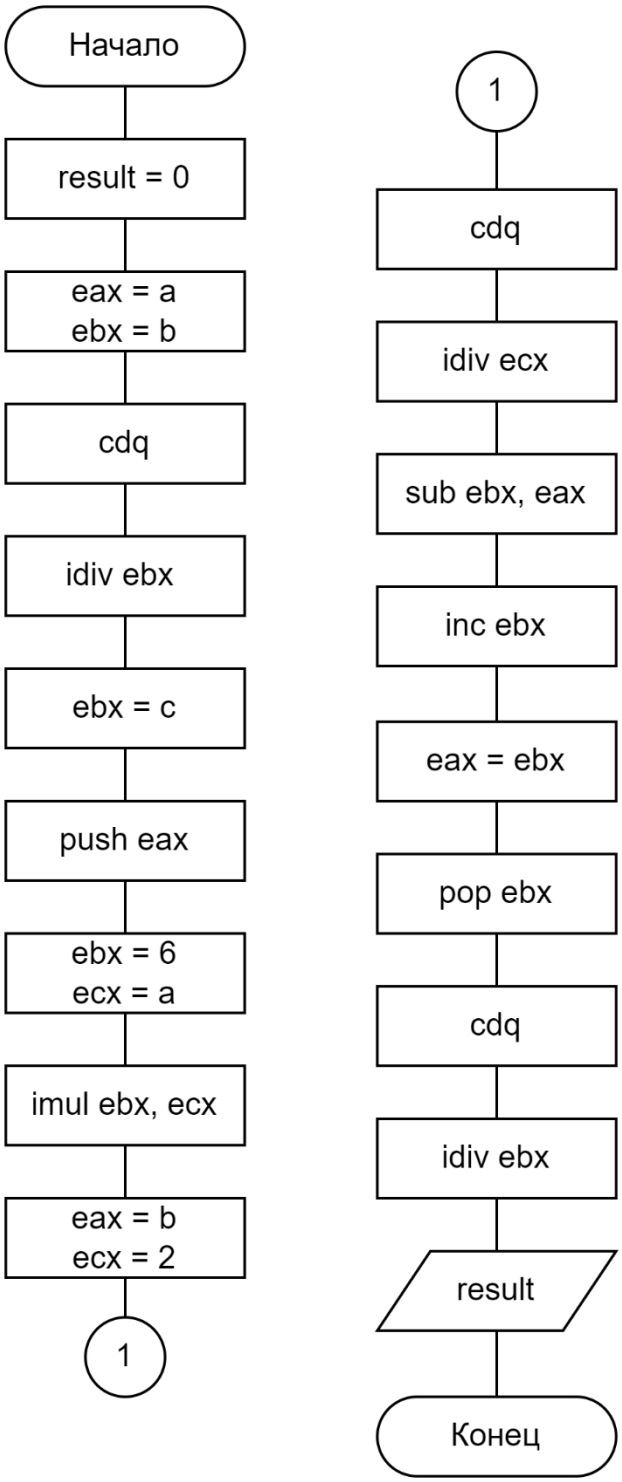


Рисунок 1.1 – Схема алгоритма вычисления исходного выражения

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.

```
Ассемблер. Лабораторная работа № 1. Арифметические и логические команды
Выполнил: Мананников М.А, группа 6103-020302D
Вариант 42:(1 + 6*a - b/2)/(c + a/b);

a = 3
b = 2
c = 1
Результат на ассемблере= 9
Результат на c++= 9
Для продолжения нажмите любую клавишу . . .
```

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

```
Ассемблер. Лабораторная работа № 1. Арифметические и логические команды
Выполнил: Мананников М.А, группа 6103-020302D
Вариант 42:(1 + 6*a - b/2)/(c + a/b);

a = 4
b = 5
c = 6
Результат на ассемблере= 3
Результат на c++= 3
Для продолжения нажмите любую клавишу . . .
```

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.

$$X = \begin{cases} a / b - a, & если \quad a > b; \\ -a, & если \quad a = b; \\ (a * b - 5) / a, & если \quad a < b; \end{cases}$$

2 Результат X – целочисленный, возвращается из функции регистре eax.

3 Значения переменных передаются в качестве параметров функции.

4 В программе реализовать вывод результата на экран.

5 Все параметры функции 32 битные числа.

6 Проверку деления на 0 реализовать также на встроенном ассемблере.

7 В качестве комментария к каждой строке необходимо указать, какой промежуточный результат, в каком регистре формируется.

8 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 *c* 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 *c* 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 *a*. 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 = a* и умножаем на -1.

Присваиваем переменной *result* значение регистра *eax*. Выводим переменную *result*.

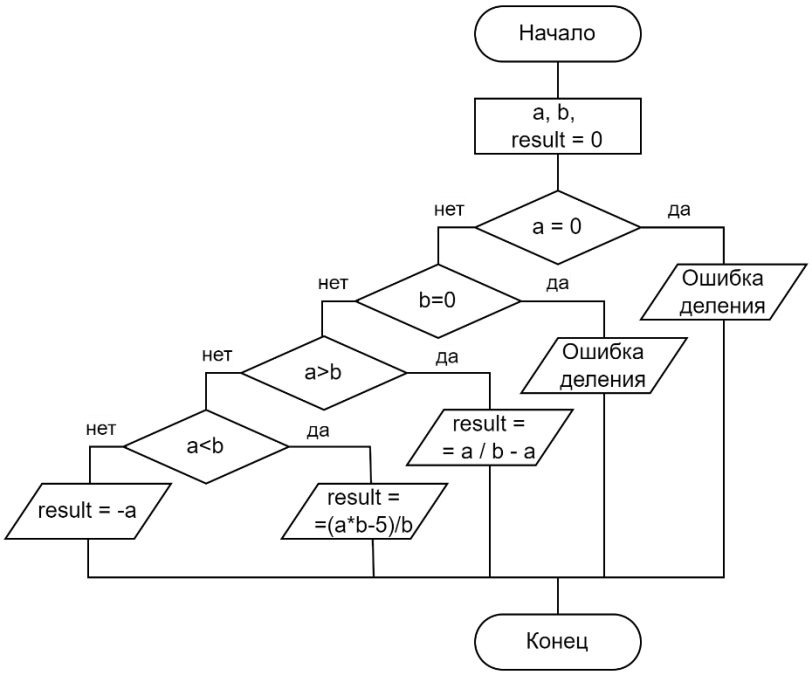


Рисунок 2.1 – Схема алгоритма вычисления условного выражения

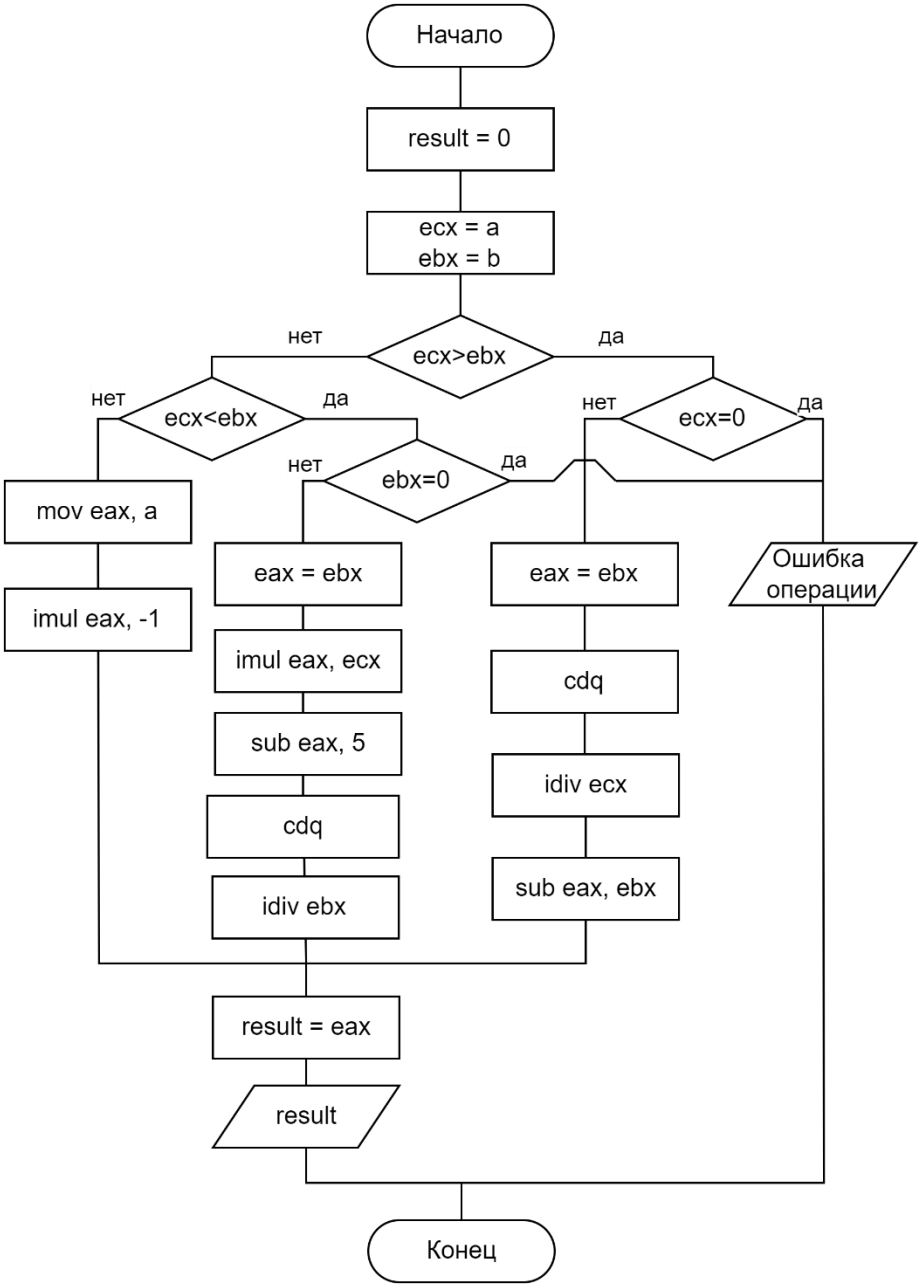


Рисунок 2.2 – Схема алгоритма на языке Ассемблера

Текст программы приведен в приложении А.2.

2.4 Результаты тестирования

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

```
Ассемблер. Лабораторная работа № 2. Арифметические команды и команды переходов.
Выполнил: Мананников М.А., группа 6103-020302D
Вариант 42:
x=a / b - a, если a>b
x=-a, если a=b
x=(a*b-5)/a, если a<b
a = 1
b = 6
Результат на ассемблере= 1
Результат на c++= 1
Для продолжения нажмите любую клавишу . . .
```

Рисунок 2.3 – Пример работы алгоритма при значениях аргументов a=1, b = 6

```
Ассемблер. Лабораторная работа № 2. Арифметические команды и команды переходов.
Выполнил: Мананников М.А., группа 6103-020302D
Вариант 42:
x=a / b - a, если a>b
x=-a, если a=b
x=(a*b-5)/a, если a<b
a = 5
b = 5
Результат на ассемблере= -5
Результат на c++= -5
Для продолжения нажмите любую клавишу . . .
```

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

```
Ассемблер. Лабораторная работа № 2. Арифметические команды и команды переходов.
Выполнил: Мананников М.А., группа 6103-020302D
Вариант 42:
x=a / b - a, если a>b
x=-a, если a=b
x=(a*b-5)/a, если a<b
a = 3
b = 7
Результат на ассемблере= 5
Результат на c++= 5
Для продолжения нажмите любую клавишу . . .
```

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 CX в реальном режиме или ECX в защищённом. Если после этого значение в CX не равно нулю, то команда LOOP выполняет переход на метку. То есть команда выполняется в два этапа. Сначала из регистра CX 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.

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 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*.

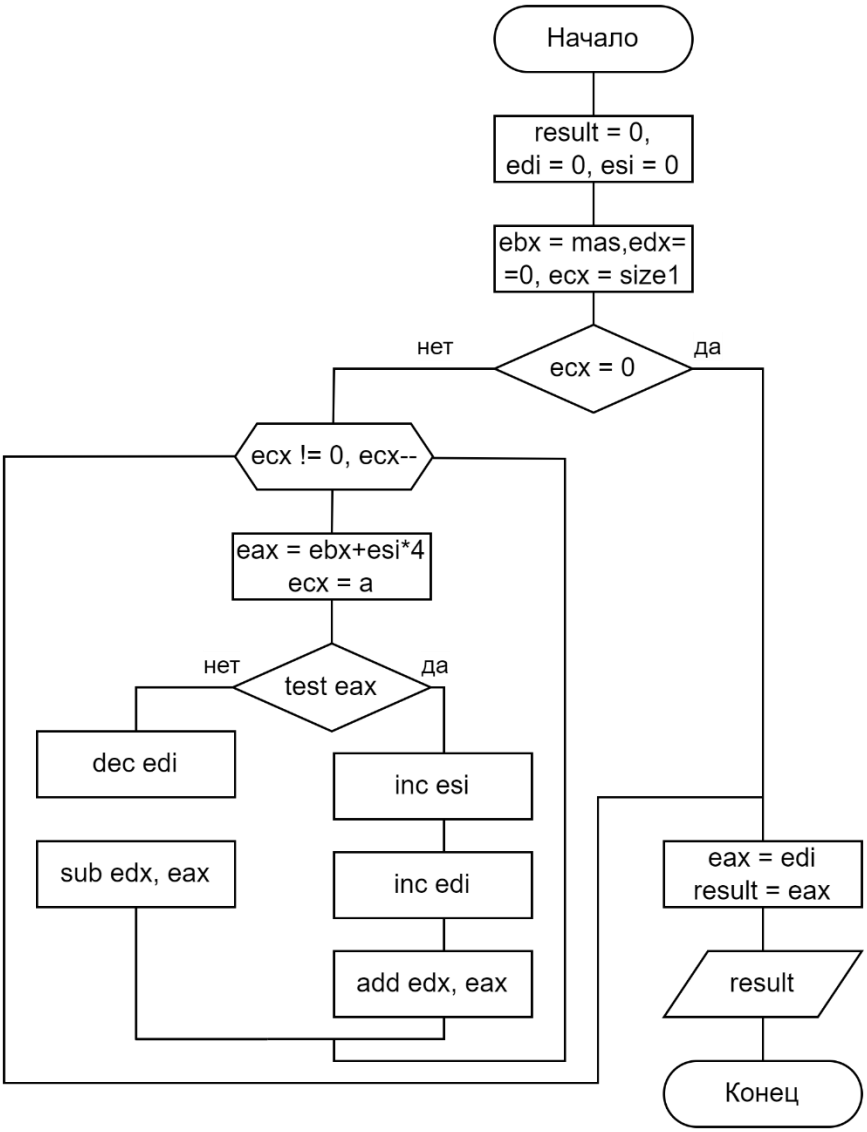


Рисунок 3.1 – Схема алгоритма вычисления исходного выражения

Текст программы приведен в приложении А.3.

3.4 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 3.2 - 3.3 при различных значениях аргумента.

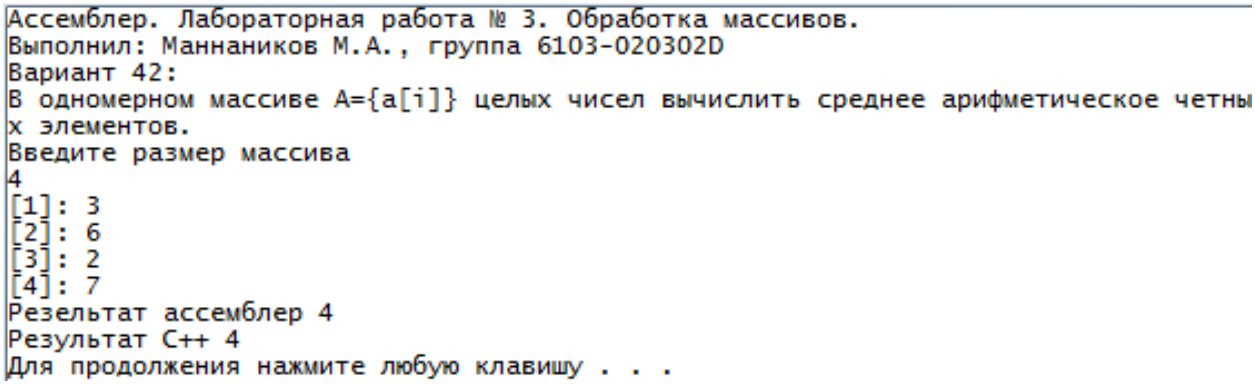


Рисунок 3.2 – Пример работы алгоритма при значениях аргументов при n = 4,

A={3, 6, 2, 7}

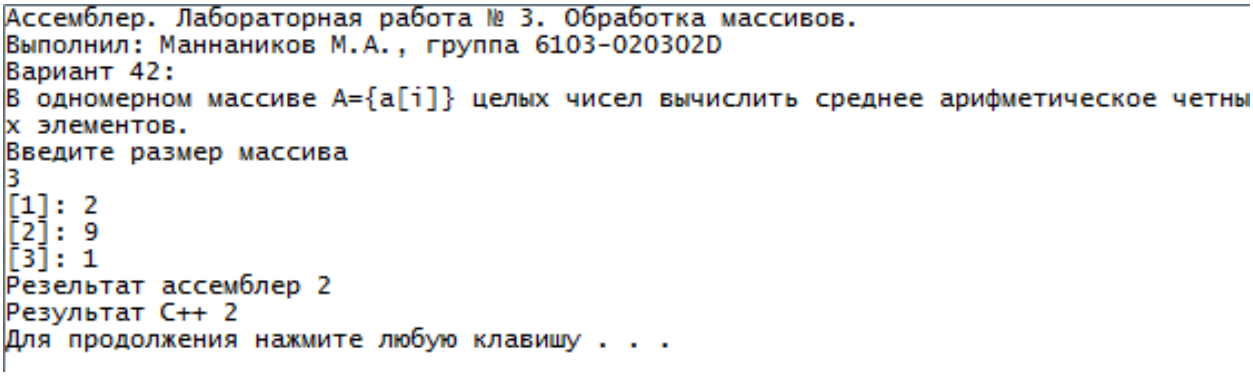


Рисунок 3.3 – Пример работы алгоритма при значениях аргументов n = 3,

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.

$$X = \begin{cases} a / b - a, & если \quad a > b; \\ -a, & если \quad a = b; \\ (a * b - 5) / a, & если \quad a < b; \end{cases}$$

2 Значения переменных передаются в качестве параметров функции.

3 В программе реализовать вывод результата на экран.

4 Все параметры функции имеют тип double.

5 Проверку деления на 0 реализовать также на встроенном ассемблере.

6 В качестве комментария к каждой строке необходимо указать, какой промежуточный результат, в каком регистре формируется.

7 В качестве комментария к строкам, содержащим команды сопроцессора необходимо указать состояние регистров сопроцессора.

8 Результат можно возвращать из функции в вершине стека сопроцессора.

4.3 Схема алгоритма

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 *c1=-1, c5=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 выходим из программы, иначе выполняем деление *a* на *b*, затем загружаем *a* в стек, меняем знак умножение на *-1* и складываем с частным. После этого присваиваем переменной *res* получившееся выражение.

Если *a* < *b*, то сравниваем *a* с нулем. Если *a* равно нулю, то выходим из программы. В противном случае, перемножаем элементы в стеке, загружаем в него значение *c5,* вычитаем его из полученного произведения. Загружаем в стек *a* и затем делим *ST* на *ST(1)* и полученное частное присваиваем переменной *res*.

В случае равенства *a* = *b*, загружаем в стек значение c1 и выполняем умножение, затем присваиваем переменной *res* это значение.

Выводим переменную *res*.

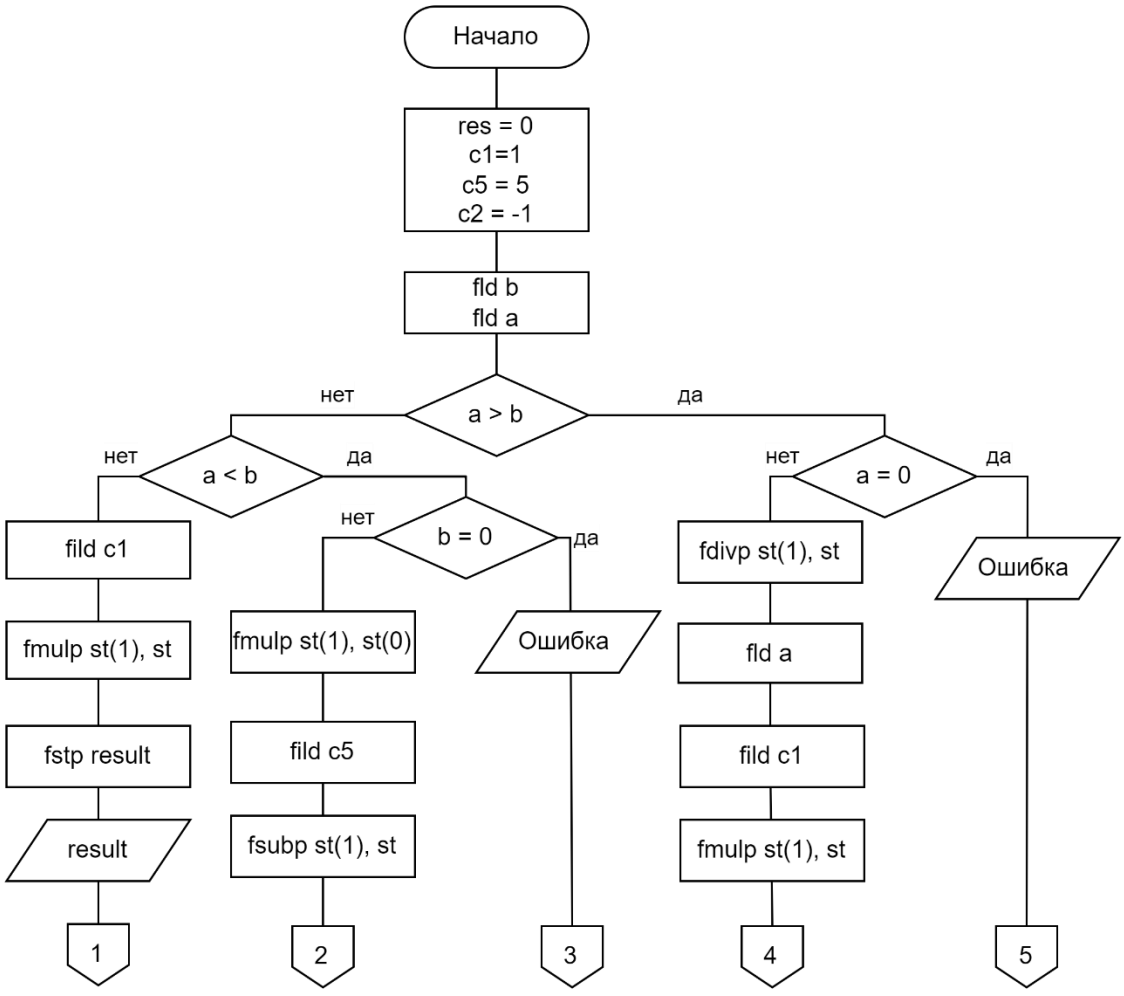


Рисунок 4.1 – Схема алгоритма на языке ассемблера (начало)

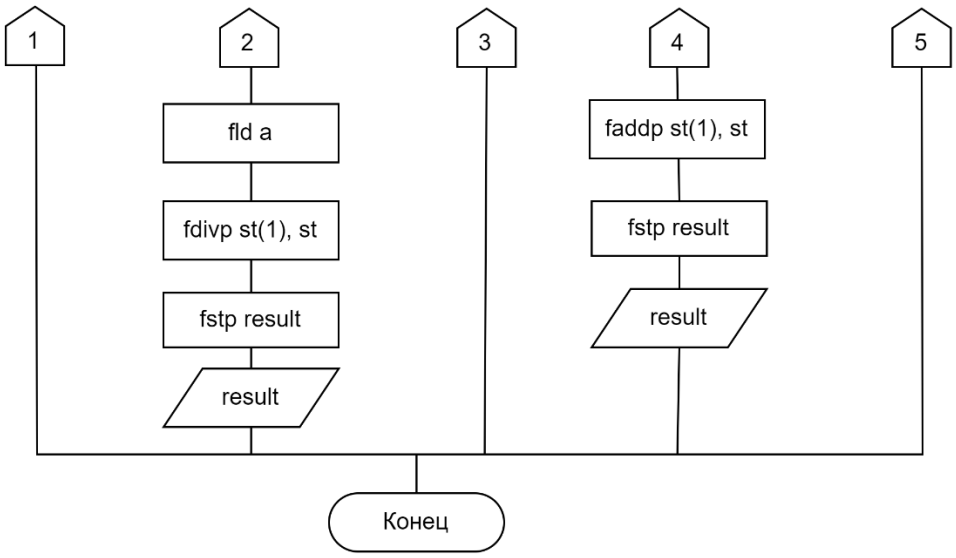


Рисунок 4.1 – Схема алгоритма на языке ассемблера (окончание)

Текст программы приведен в приложении А.4.

4.4 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 4.2-4.4.

```
Ассемблер. Лабораторная работа № 4. Команды арифметического сопроцессора.
Выполнил: Мананников М.А., группа 6103-020302D
Вариант 42:
x=a / b - a, если a>b
x=-a, если a=b
x=(a*b-5)/a, если a<b
a = 3.1
b = 0.9
Ответ на ASM: 0.344444
Ответ на C++: 0.344444
Для продолжения нажмите любую клавишу . . .
```

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

```
Ассемблер. Лабораторная работа № 4. Команды арифметического сопроцессора.
Выполнил: Мананников М.А., группа 6103-020302D
Вариант 42:
x=a / b - a, если a>b
x=-a, если a=b
x=(a*b-5)/a, если a<b
a = 1.4
b = 8.7
Ответ на ASM: 5.12857
Ответ на C++: 5.12857
Для продолжения нажмите любую клавишу . . .
```

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

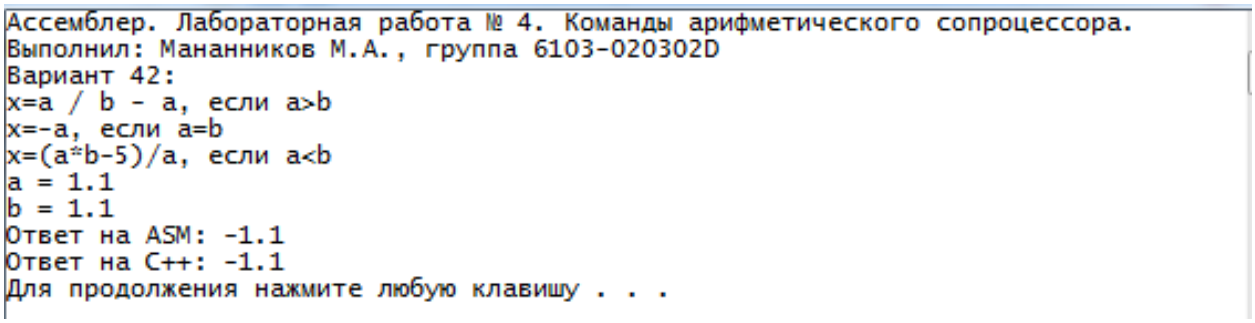


Рисунок 4.4 – Пример работы алгоритма при значениях аргумента a=1.1, b=1.1

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: $628\ x^{11} + 2164\ x^9 + 797\ x^5 + 4051\ x - 2045 = 0$

5.3 Решение

Для того, чтобы решить уравнение сначала найдем производную данной функции: $f'(x) = 6908\,x^{10} + 19476\,x^{8} + 3985\,x^{4} + 4051$.

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

$$x_{n+1} = x_n - \frac{628\,x^{11} + 2164\,x^{9} + 797\,x^{5} + 4051\,x - 2045}{6908\,x^{10} + 19476\,x^{8} + 3985\,x^{4} + 4051}.$$

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 = $|f(x_{n+1}) - f(x_n)|$. 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.

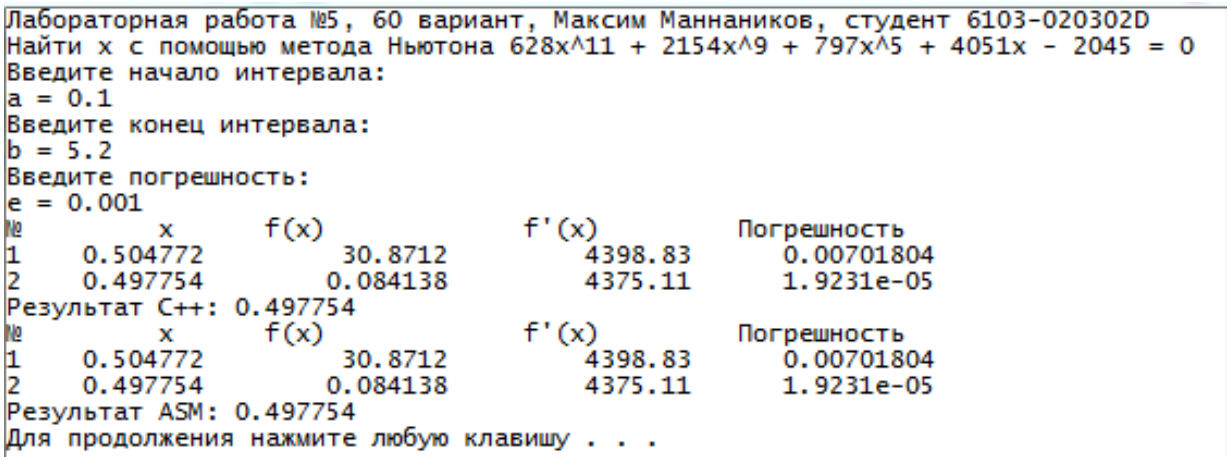
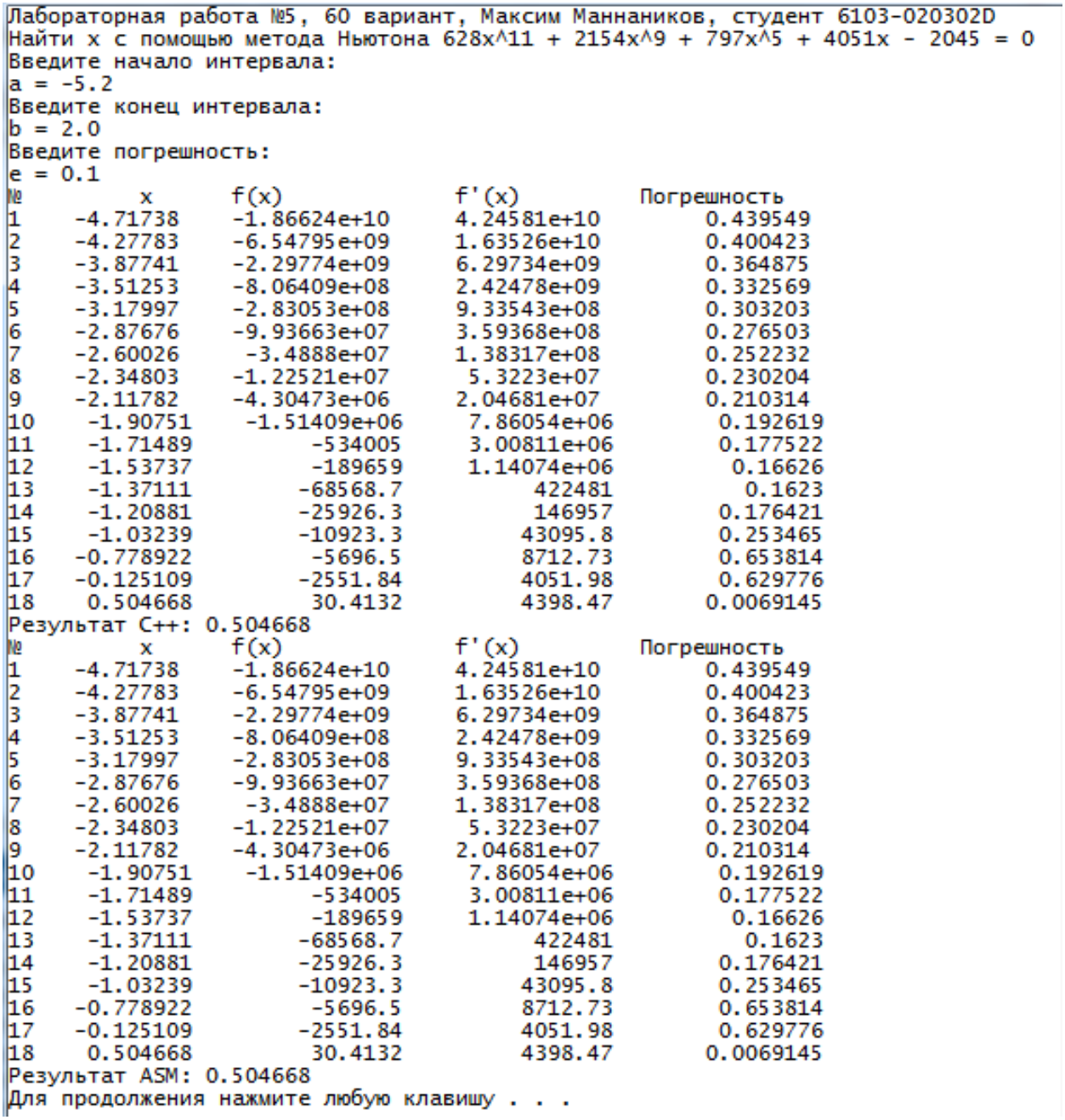


Рисунок 5.2 – Пример работы алгоритма при значениях аргументов a = 0.1, b = 5.2, e = 0,001



```
Лабораторная работа №5, 60 вариант, Максим Маннаников, студент 6103-020302D
Найти x с помощью метода Ньютона 628x^11 + 2154x^9 + 797x^5 + 4051x - 2045 = 0
Введите начало интервала:
a = -5.2
Введите конец интервала:
b = 2.0
Введите погрешность:
e = 0.1
№          x       f(x)              f'(x)          Погрешность
1     -4.71738    -1.86624e+10     4.24581e+10         0.439549
2     -4.27783    -6.54795e+09     1.63526e+10         0.400423
3     -3.87741    -2.29774e+09     6.29734e+09         0.364875
4     -3.51253    -8.06409e+08     2.42478e+09         0.332569
5     -3.17997    -2.83053e+08     9.33543e+08         0.303203
6     -2.87676    -9.93663e+07     3.59368e+08         0.276503
7     -2.60026     -3.4888e+07     1.38317e+08         0.252232
8     -2.34803    -1.22521e+07      5.3223e+07         0.230204
9     -2.11782    -4.30473e+06     2.04681e+07         0.210314
10     -1.90751     -1.51409e+06     7.86054e+06         0.192619
11     -1.71489          -534005     3.00811e+06         0.177522
12     -1.53737           -189659     1.14074e+06          0.16626
13     -1.37111          -68568.7          422481           0.1623
14     -1.20881          -25926.3          146957         0.176421
15     -1.03239          -10923.3         43095.8         0.253465
16    -0.778922           -5696.5         8712.73         0.653814
17    -0.125109          -2551.84         4051.98         0.629776
18     0.504668           30.4132         4398.47        0.0069145
Результат C++: 0.504668
№          x       f(x)              f'(x)          Погрешность
1     -4.71738    -1.86624e+10     4.24581e+10         0.439549
2     -4.27783    -6.54795e+09     1.63526e+10         0.400423
3     -3.87741    -2.29774e+09     6.29734e+09         0.364875
4     -3.51253    -8.06409e+08     2.42478e+09         0.332569
5     -3.17997    -2.83053e+08     9.33543e+08         0.303203
6     -2.87676    -9.93663e+07     3.59368e+08         0.276503
7     -2.60026     -3.4888e+07     1.38317e+08         0.252232
8     -2.34803    -1.22521e+07      5.3223e+07         0.230204
9     -2.11782    -4.30473e+06     2.04681e+07         0.210314
10     -1.90751     -1.51409e+06     7.86054e+06         0.192619
11     -1.71489          -534005     3.00811e+06         0.177522
12     -1.53737           -189659     1.14074e+06          0.16626
13     -1.37111          -68568.7          422481           0.1623
14     -1.20881          -25926.3          146957         0.176421
15     -1.03239          -10923.3         43095.8         0.253465
16    -0.778922           -5696.5         8712.73         0.653814
17    -0.125109          -2551.84         4051.98         0.629776
18     0.504668           30.4132         4398.47        0.0069145
Результат ASM: 0.504668
Для продолжения нажмите любую клавишу . . .
```

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 $\sqrt{(\mathrm{x}-2)*\mathrm{e}^{-2\mathrm{x}}+\ln(x-2)+\ 2^{x+2}+(\mathrm{x}+8)\cos(x+8)}$

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 $\ln 2$ и меняем местами с a в стеке. Вычисляем *ln(a)* с помощью команды FYL2X. Перемножаем значения в стеке и добавляем в вершину $log2(\mathrm{e})$. 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 $2^{b*ln(a)*log2(e)}-1$ 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 a^b. 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.

6.4 Результаты тестирования

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

```
ЛабораторнаЯ работа Ь6, 60 вариант, Максим Маннаников, студент 6103-020302D
Вычислить y = (ln(x-2) + (x-2)*e^-2x + 2^(x+2) + (x-2)*sin(x-2))^0,5
на промежутке [2;8]

Введите шаг h = 1.5
    x  Результат '++  Результат ASM
    2      -nan(ind)      -nan(ind)
  3.5        6.86716        6.86716
    5        11.3808        11.3808
  6.5        18.9511        18.9511
    8        32.0018        32.0018
Для продолжения нажмите любую клавишу . . .
```

Рисунок 6.3 – Пример работы алгоритма при значениях аргументов h = 1.5

```
ЛабораторнаЯ работа Ь6, 60 вариант, Максим Маннаников, студент 6103-020302D
Вычислить y = (ln(x-2) + (x-2)*e^-2x + 2^(x+2) + (x-2)*sin(x-2))^0,5
на промежутке [2;8]

Введите шаг h = 0.3
    x  Результат '++  Результат ASM
    2      -nan(ind)      -nan(ind)
  2.3        4.31115        4.31115
  2.6        4.90742        4.90742
  2.9          5.519          5.519
  3.2        6.16937        6.16937
  3.5        6.86716        6.86716
  3.8         7.6195         7.6195
  4.1        8.43497        8.43497
  4.4        9.32445        9.32445
  4.7        10.3012        10.3012
    5        11.3808        11.3808
  5.3        12.5801        12.5801
  5.6        13.9176        13.9176
  5.9        15.4122        15.4122
  6.2        17.0834        17.0834
  6.5        18.9511        18.9511
  6.8        21.0359        21.0359
  7.1        23.3593        23.3593
  7.4        25.9442        25.9442
  7.7        28.8158        28.8158
    8        32.0018        32.0018
Для продолжения нажмите любую клавишу . . .
```

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 $\int f(x)dx\ a\ b = \frac{\mathrm{b-a}}{6\mathrm{n}}\left[(y_0 - y_{2n}) + 4(y_1 + y_3 + \cdots + y_{2n-1}) + 2(y2 + y4 + \cdots + y_{2n-2}\right)$

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: $\int_1^3 \frac{(3+\mathrm{x}^{\frac{3}{4}})dx}{\sqrt{\mathrm{x}^{\frac{2}{3}}+2}}$.

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 $\frac{(3+x^{\frac{3}{4}})dx}{\sqrt{x^{\frac{2}{3}}+2}}$ 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 *c2, 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 $x = a$. Вычисляем значение $h = \frac{(b-a)}{n}$. 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).* Иначе проверяем *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.

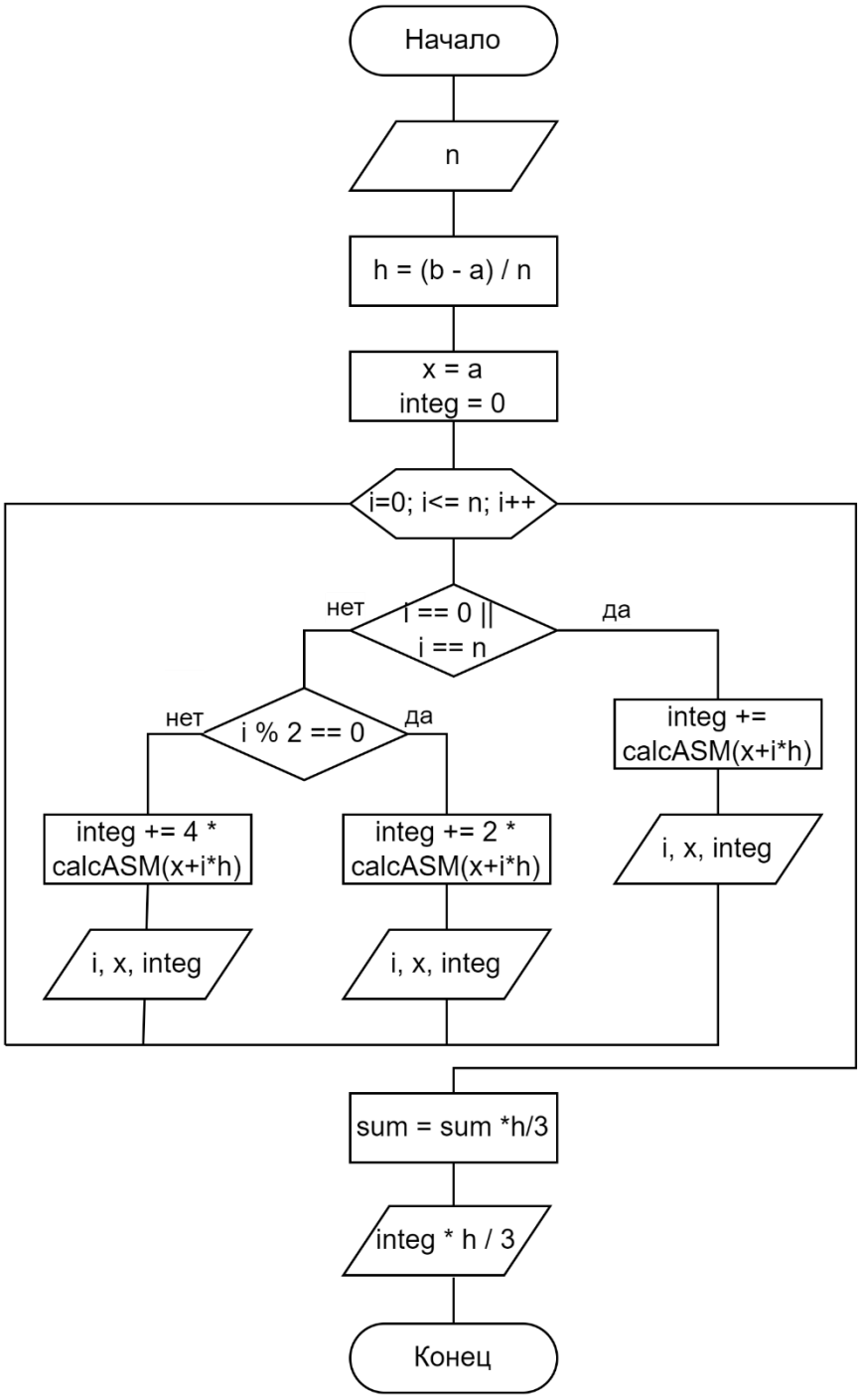


Рисунок 7.1 – Схема алгоритма вычисления интеграла

Текст программы приведен в приложении А.7.

7.4 Результаты тестирования

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

```
ЛабораторнаЯ работа Ь7, 60 вариант, Мананников Максим, студент 6103-020302D
Вычислить на промежутке [1 ; 3] Интеграл: (3+x^(3/4))/sqrt(x^(2/3)+2)
Введите n = 2
         i              x            Результат C++            Результат ASM
         1              1                   2.3094                   2.3094
         2              1                   11.547                   11.547
         3              1                  13.8564                  13.8564
Результат интегрирования asm: 4.6188
Результат интегрирования cpp: 4.6188
Для продолжения нажмите любую клавишу . . .
```

Рисунок 7.2 – Пример работы алгоритма при значениях аргументов n = 2

```
ЛабораторнаЯ работа Ь7, 60 вариант, Мананников Максим, студент 6103-020302D
Вычислить на промежутке [1 ; 3] Интеграл: (3+x^(3/4))/sqrt(x^(2/3)+2)
Введите n = 3
         i              x            Результат C++            Результат ASM
         1              1                   2.3094                   2.3094
         2              1                   11.547                   11.547
         3              1                  16.1658                  16.1658
         4              1                  25.4034                  25.4034
Результат интегрирования asm: 5.6452
Результат интегрирования cpp: 5.6452
Для продолжения нажмите любую клавишу . . .
```

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

```
ЛабораторнаЯ работа Ь7, 60 вариант, Мананников Максим, студент 6103-020302D
Вычислить на промежутке [1 ; 3] Интеграл: (3+x^(3/4))/sqrt(x^(2/3)+2)
Введите n = -1
         i              x            Результат C++            Результат ASM
Результат интегрирования asm: -0
Результат интегрирования cpp: -0
Для продолжения нажмите любую клавишу . . .
```

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 $|S_{k+1} - S_k| \leq \varepsilon$, где S_{k+1} – последующий член ряда; S_k – 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 $|S_{k+1} - S_k| \leq \varepsilon$, 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: $\sum_{n=1}^{\infty} \frac{(-1)^{n-1}(x-2)^{2n}}{2n}$.

8.3 Решение

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

$$r_n = \begin{cases} \dfrac{(\mathrm{x}-2)^2}{2} & ,n = 1; \\ \dfrac{(-1)^n(x-2)^2}{2n} * r_{n-1} & ,n \geq 1. \end{cases}$$

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: x – 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 *x, c2м* и складываем их, дублируем это значение в стек, чтобы перемножить у получить квадрат выражения. Добавляем в стек *counter* и умножаем его значение на c1. Перемножаем два верхних значения в стеке и опять добавляем *counter*, на который делим наше выражение, затем умножаем его на предыдущий элемент ряда, добавляем *c1* 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*.

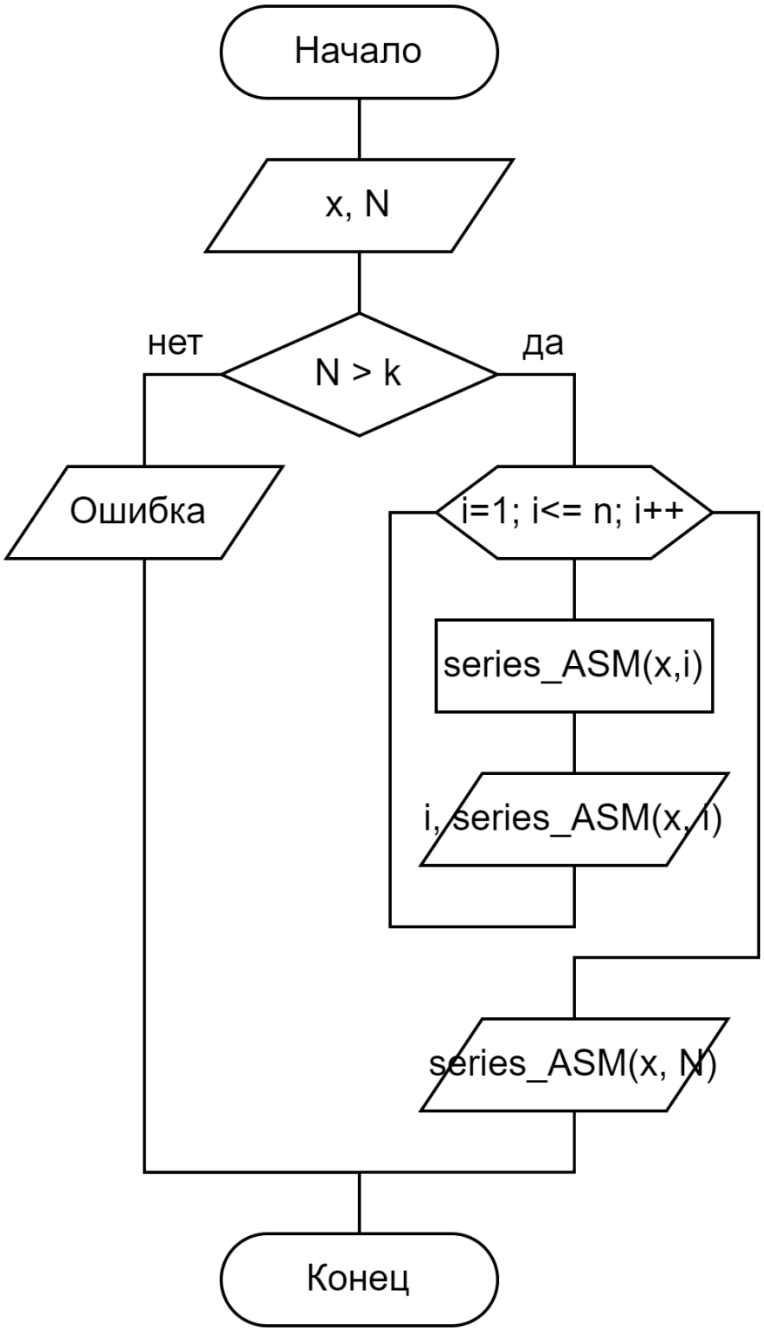


Рисунок 8.1 – Схема алгоритма вычисления исходного выражения

Текст программы приведен в приложении А.8.

8.5 Результаты тестирования

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

```
Лабораторная работа №8
 Выполнила: студент группы 6103 - 020302D Мананников Максим
 Вариант 12
Вычислить сумму ряда с n-ым членом: (-1)^(n-1)*(x-2^(2n))/2n
 Введите число членов числового ряда = 6
 Введите x = 1
 n           ASM - S(n)           C++ - S(n)
 1                          0.5                            0.5
            0
 2                         0.25                           0.25
            0
 3           0.41666666666666685           0.41666666666666663       -5.55111512
31257827e-17
 4           0.29166666666666685           0.29166666666666663       -5.55111512
31257827e-17
 5           0.39166666666666663           0.39166666666666607       -5.55111512
31257827e-17
 6           0.30833333333333348           0.30833333333333293       -5.55111512
31257827e-17
Для продолжения нажмите любую клавишу . . .
```

Рисунок 8.2 – Пример работы алгоритма при значениях аргументов n = 6, x = 1

```
Лабораторная работа №8
 Выполнила: студент группы 6103 - 020302D Мананников Максим
 Вариант 12
Вычислить сумму ряда с n-ым членом: (-1)^(n-1)*(x-2^(2n))/2n
 Введите число членов числового ряда = 3
 Введите x = 1
 n           ASM - S(n)           C++ - S(n)
 1                          0.5                            0.5
            0
 2                         0.25                           0.25
            0
 3           0.41666666666666685           0.41666666666666663       -5.55111512
31257827e-17
Для продолжения нажмите любую клавишу . . .
```

Рисунок 8.3 – Пример работы алгоритма при значениях аргументов n = 3, x = 1

```
Лабораторная работа №8
 Выполнила: студент группы 6103 - 020302D Мананников Максим
 Вариант 12
Вычислить сумму ряда с n-ым членом: (-1)^(n-1)*(x-2^(2n))/2n
 Введите число членов числового ряда = 4
 Введите x = 0
 n           ASM - S(n)           C++ - S(n)
 1                          2                              2
            0
 2                         -2                             -2
            0
 3            8.66666666666666607           8.66666666666666607
            0
 4           -23.3333333333333321          -23.3333333333333357      -3.552713678
80050093e-15
Для продолжения нажмите любую клавишу . . .
```

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;// < ecx >= 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 C++ " << 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 c 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 x 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] Єнтеграл:
(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;
}
```