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 № 118

Обучающийся в группе 6104-020302 Н.Д. Маликов

подпись, дата

Руководитель Д.С. Оплачко

подпись, дата

Самара 2024

СОДЕРЖАНИЕ

­­­­­­­

[Лабораторная работа 1 «Арифметические и логические команды в ассемблере» 4](#_Toc136031964)

[1.1 Теоретические основы лабораторной работы 4](#_Toc136031965)

[1.2 Задание 4](#_Toc136031966)

[1.3 Схема алгоритма 5](#_Toc136031967)

[1.4 Результат тестирования 6](#_Toc136031968)

[Лабораторная работа 2 «Арифметические команды и команды переходов в ассемблере» 8](#_Toc136031969)

[2.1 Теоретические основы лабораторной работы 8](#_Toc136031970)

[2.2 Задание 8](#_Toc136031971)

[2.3 Схема алгоритма 9](#_Toc136031972)

[2.4 Результаты тестирования 12](#_Toc136031973)

[Лабораторная работа 3 «Команды работы с массивами и стеком» 13](#_Toc136031974)

[3.1 Теоретические основы лабораторной работы 13](#_Toc136031975)

[3.2 Задание 13](#_Toc136031976)

[3.3 Схема Алгоритма 13](#_Toc136031977)

[3.4 Результат тестирования 16](#_Toc136031978)

[Лабораторная работа 4 «Изучение работы математического сопроцессора в среде Assembler» 17](#_Toc136031979)

[4.1 Теоретические основы лабораторной работы 17](#_Toc136031980)

[4.2 Задание 17](#_Toc136031981)

[4.3 Схема алгоритма 18](#_Toc136031982)

[4.4 Результаты тестирования 20](#_Toc136031983)

[Лабораторная работа 5 «Команд арифметического сопроцессора на языке Ассемблер. Протабулировать функцию на заданном диапазоне изменения с заданным шагом» 21](#_Toc136031984)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26](#_Toc136031989)

[Приложение А Листинг программ 27](#_Toc136031990)

[А.1 Листинг программы лабораторной работы №1 27](#_Toc136031991)

[А.2 Листинг программы лабораторной работы №2 28](#_Toc136031992)

[А.3 Листинг программы лабораторной работы №3 29](#_Toc136031993)

[А.4 Листинг программы лабораторной работы №4 30](#_Toc136031994)

Лабораторная работа 1 «Арифметические и логические команды в ассемблере»

1.1 Теоретические основы лабораторной работы

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

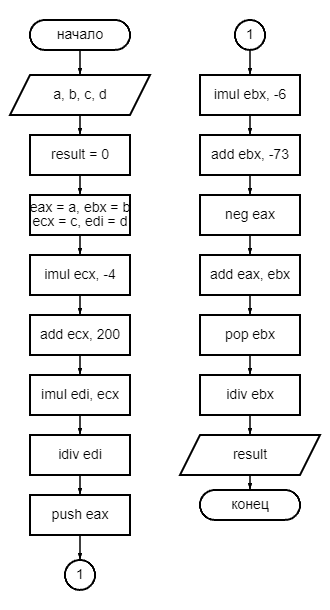
* MOV – команда копирования данных из одной переменной в другую. Команда копирует содержимое второго операнда в первый операнд. При этом содержимое второго операнда не изменяется;
* ADD – команда выполняет целочисленного сложения двух операндов и флага переноса CF. Результат сложения помещается в первый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

1.2 Задание

1. в программе необходимо реализовать функцию вычисления целочисленного выражения ( -a – 73 -6 \* b) / ( -4 \* c + 200 / d \* a ) на встроенном ассемблере MASM в среде Microsoft Visual Studio на языке C+;
2. значения переменных передаются в качестве параметров функции;
3. результат выводить в консольном приложении (проект консольное приложение Win32);
4. в программе реализовать ввод переменных из командной строки и вывод результата на экран;
5. все параметры функции 32 битные числа (знаковые и беззнаковые);
6. первые строки функции вычисления выражения заносят значения аргументов функции в соответствующие регистры;
7. где необходимо, реализовать проверки вводимых данных и вычисления отдельных операций. Например, проверка деления на 0;
8. в качестве комментария к каждой строке необходимо указать, какой промежуточный результат, в каком регистре формируется;
9. по возможности использовать команды сдвига.

1.3 Схема алгоритма

На рисунке 1.1 приведена схема алгоритма вычисления переменной y в соответствии с заданием. В параметры передаются значения переменных a, b, c, d целочисленного типа из главной программы. Переменной result, отвечающей за хранения результата вычисления исходного выражения, присваиваем значение 0. Присваиваем значения регистрам: eax = a, ebx = b, ecx = c, edi = d. Умножаем ecx на -4. Прибавляем к eax, получаем c \* ( - 4 ) + 200. Помещаем результат из ecx в eax. Выполняем операцию cdq, которая расширяет eax до edx:eax. Помещаем значение переменной a в ecx. Умножаем значение в регистре edi на значение в регистре ecx. Делим расширенное значение в edx:eax на значение в регистре edi: результат будет (-4 \* c + 200) / (d \* a), и помещаем его в стек. Помещаем значение переменной a в eax. Умножаем значение в регистре ebx на -6 и вычитаем 73, получаем b \* -6 – 73. Инвертируем знак значения в eax. Добавляем значение ebx к значению в eax: b \* -6 - 73 - a Получаем результат вычитания eax = eax - ebx. Переносим значение в ebx. Извлекаем значение из стека в ebx. Выполняем операцию cdq. Делим расширенное значение в edx:eax на значение в регистре ebx. Командой mov result, eax переносим содержимое регистра eax в переменную result. Возвращаем result.

  
Рисунок 1.1 – Схема алгоритма вычисления исходного выражения

Текст программы приведен в приложении А1.

1.4 Результат тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 1.2 – 1.4 при значениях аргумента.

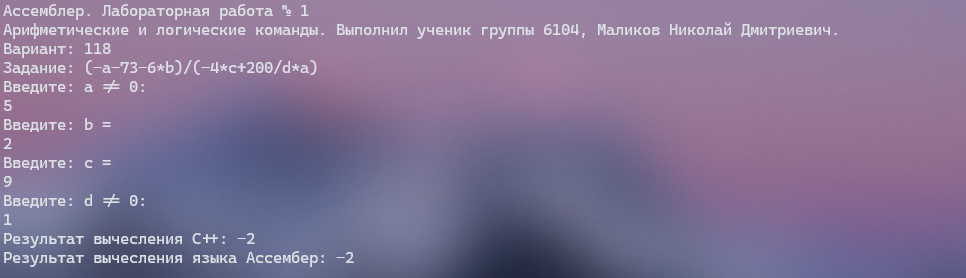


Рисунок 1.2 – Пример работы программы при значении аргументов a = 5, b= 2, c = 9, d = 1

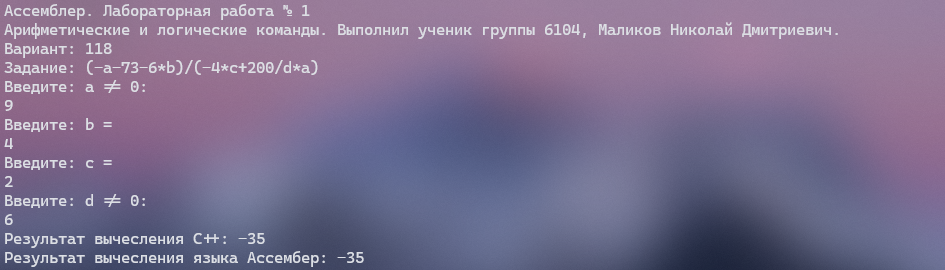


Рисунок 1.3 – Пример работы программы при значениях аргументов a = 9, b = 4, c = 2, d = 6

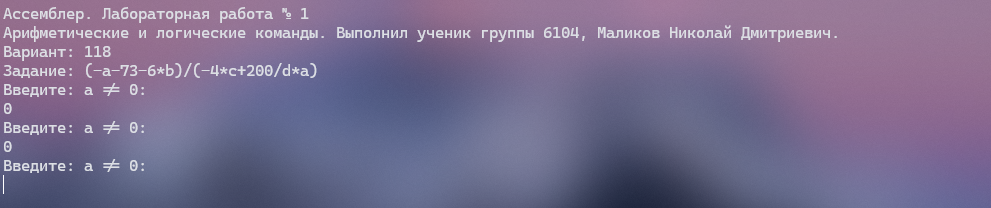


Рисунок 1.4 – Пример обработки исключительной ситуации в программе Лабораторная работа 2 «Арифметические команды и команды переходов в ассемблере»

2.1 Теоретические основы лабораторной работы

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

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

, если a > b

, если a = b

, если a < b

Результат X – целочисленный, возвращается из функции регистре eax;

1. значения переменных передаются в качестве параметров функции;
2. в программе реализовать вывод результата на экран;
3. все параметры функции 32 битные числа;
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. 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 – 7) на равенство с 0. Если (a – 7) = 0, передаем ошибку деления на нуль, иначе присваиваем переменной result заданную функцию result = (5 \* b \* b) / (a + 7). Если a < b, проверяем (b - a) на равенство с 0. Если (b - a) = 0, передаем ошибку деления на нуль, иначе присваиваем переменной result заданную функцию result = (8 \* a \* b - 5)/(b - a). В случае равенства a = b, переменной result присваивается значение result = -21 \* a. Выводим значение result.

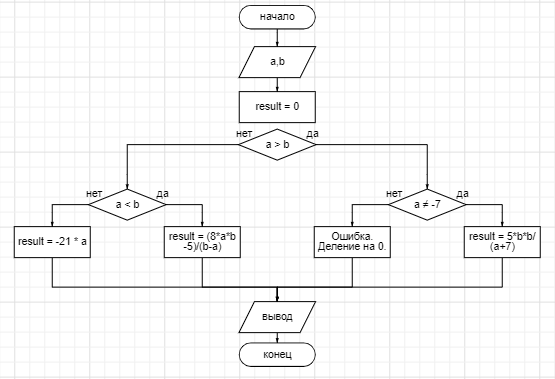


Рисунок 2.1 – Схема алгоритма вычисления условного выражения

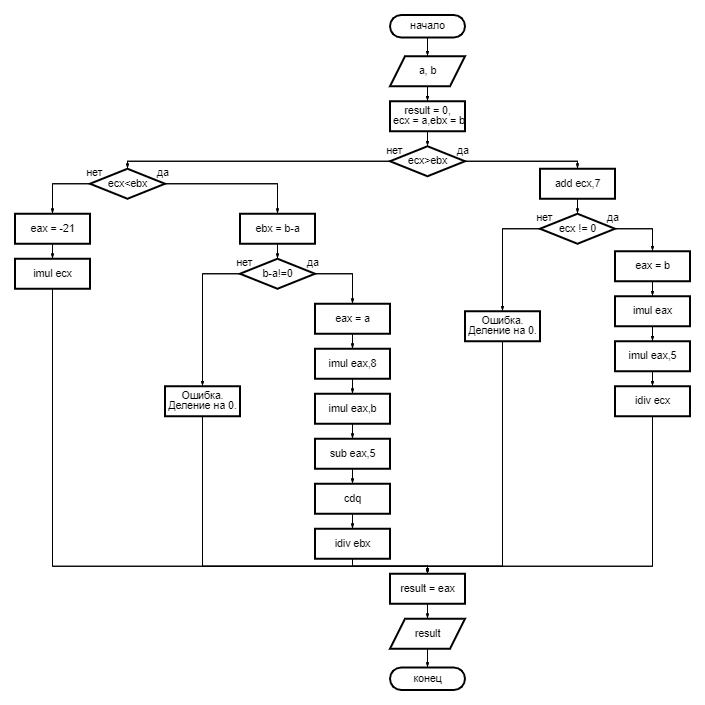


Рисунок 2.2 – Схема алгоритма на языке Ассемблера

2.4 Результаты тестирования

Для проверки работоспособности программы были выполнены тесты, результаты которых приведены на рисунках 2.3 – 2.6 при значениях аргумента.

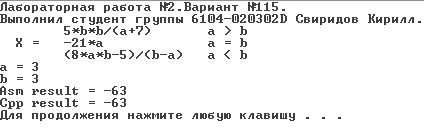


Рисунок 2.3 – Пример работы программы вычисления целочисленного значения при значениях параметров a = 3, b = 3

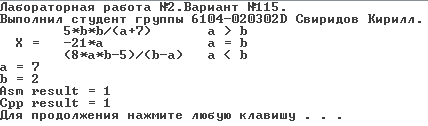


Рисунок 2.4 – Пример работы программы вычисления целочисленного значения при значениях параметров a = 7, b = 2

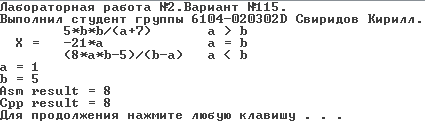


Рисунок 2.5 – Пример работы программы вычисления целочисленного значения при значениях параметров a = -7, b = -10

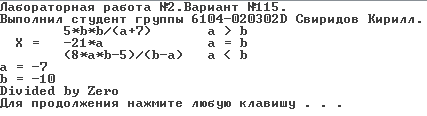


Рисунок 2.6 – Пример работы программы вычисления целочисленного значения при значениях параметров a = -7, b = -10

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». 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.

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**:**Вычислить количество элементов расположенных между первым и последним четными элементами.

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: arr, хранящая массив элементов, len, хранящая количество элементов массива. Переменной result, отвечающей за хранения результата вычисления исходного выражения, присваиваем нулевое значение Регистры esi готовим к работе посредством xor esi, esi. Присваиваем ebx ссылку на первый элемент массива ebx = arr, регистру ecx присваиваем размер массива ecx = len. Если длина массива равна нулю, то завершаем цикл. Если длина массива не равна нулю, то до тех пор, пока eax не станет равен первому четному элементу, присваиваем регистру edx номер необходимого элемента массива. Определяем текущий элемент: eax = [ebx + esi \* 4]. Проверяем eax на нечётность test eax, 1, если он чётный, то переходим ко второму циклу. Присваиваем регистру esi значение len – 1. . Значение регистра esi увеличиваем на единицу и переходим к следующему элементу массива. Начинаем второй цикл определяя текущий элемент с конца: eax = [ebx + esi \* 4]. Цикл продолжается до тех пор, пока eax не станет равен последнему четному элементу. Посчитаем количество элементов между первым и последним четными элементами. Значение регистра esi уменьшаем на единицу и переходим к следующему элементу массива. После выхода из цикла через выводим получившиеся значения в массиве.

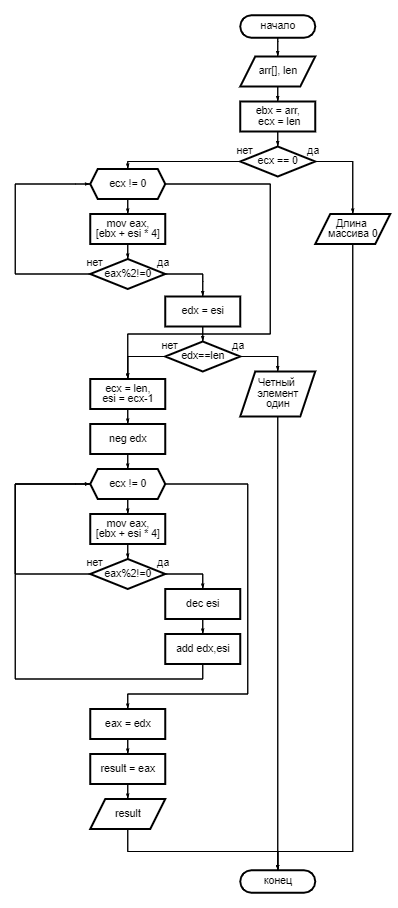


Рисунок 3.1 – Схема алгоритма подсчета количества элементов

3.4 Результат тестирования

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

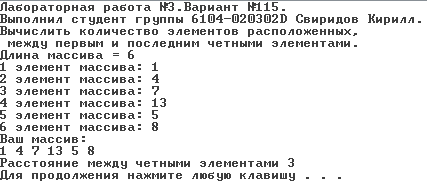


Рисунок 3.2 – Пример работы программы вычисления количества элементов при значениях параметров n = 6, A = {1;4;7;13;5;8}

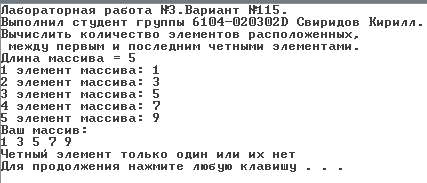


Рисунок 3.3 – Пример работы программы вычисления количества элементов при значениях параметров n = 5 , A = {1;3;5;7;9}

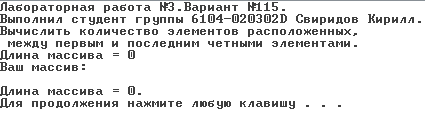


Рисунок 3.4 – Пример работы программы вычисления количества элементов при значениях параметров n = 0, A = {}

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»

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

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

4.2 Задание

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

Условие:

4.3 Схема алгоритма

Схема основного алгоритма приведена на рисунке 4.1. В параметры передаются значения переменных вещественного a, b из главной программы, объявляем и инициализируем переменну c115 = 115. Переменной result, хранящей результат вычисления исходного выражения присваиваем значение 0. Инициализируем сопроцессор, загружаем в стек число пи и c115. Меняем знак у числа в вершине стека и перемножаем два числа в верхушке стека с выталкиванием. Загружаем в стек число пи, переменную a, перемножаем с выталкиванием и заменяем значение в верхушке стека на синус от ST(0). Вычитаем из второго элемента стека ST(0) с выталкиванием. Загружаем в верхушку стека переменную пи и заменяем значение в верхушке стека на косинус от него. Вычитаем из второго элемента стека ST(0) с выталкиванием. Загружаем в стек переменную a, подсчитываем значение тангенса от полученного числа путем обратного деления с выталкивание второго элемента в стеке на первый и вычитаем его из ST(1) с выталкиванием. Загружаем в стек переменную a, меняем ее знак, подсчитываем значение котангенса от этого числа путем деления с выталкиванием второго элемента в стеке на первый. Добавляем его ко второму элементу стека с выталкиванием. Загружаем в стек два раза подряд переменную a. Перемножаем их с выталкиванием и загружаем в стек переменную b. Складываем с выталкивание второй элемент в верхушке стека с первым и загружаем в стек число пи. Производим обратное деление с выталкиванием второго элемента стека на первый. Загружаем в стек последовательно c115 и переменную a.Перемножаем два элемента в верхушке стека с выталкиванием и загружаем в стек переменную b. Вычитаем с выталкиванием первый элемент в верхушке стека из второго. Делим с выталкиваем второй элемент в верхушке стека на первый. Складываем с выталкиванием два элемента в верхушке стека. Приравниваем result к значению в верхушке стека и возвращаем result.

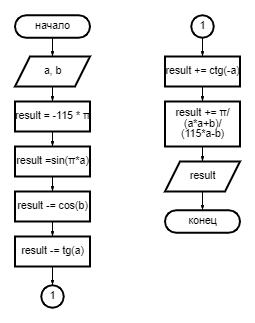


Рисунок 4.1 – Схема алгоритма подсчета вещественного выражения

4.4 Результаты тестирования

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

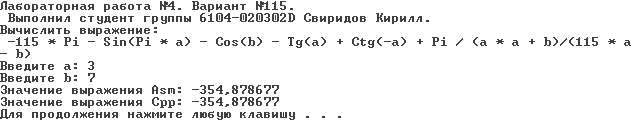


Рисунок 4.2 – Пример работы программы вычисления вещественного значения при значениях параметров a = 3, b = 7

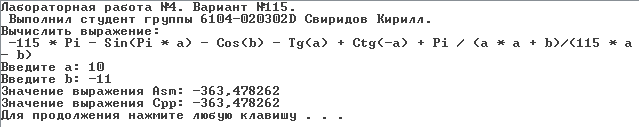


Рисунок 4.3 – Пример работы программы вычисления вещественного значения при значениях параметров a = 10, b = -11

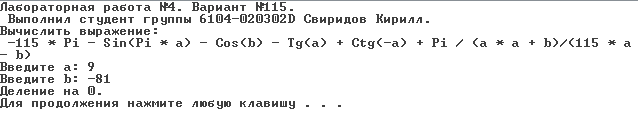


Рисунок 4.4 – Пример работы программы вычисления вещественного значения при значениях параметров a = 9, b = -81

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.

1. 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.
2. Оплачко Д.С. Методические указания к лабораторной работе №5 «Команды математического сопроцессора на языке Ассемблер. Протабулировать функцию на заданном диапазоне изменения аргумента с заданным шагом» / Д.С. Оплачко. Самара: изд-во virtual6, 2023 [Электронный ресурс]. URL: http://virtual6.ssau.ru/Moodle/mod/resource/view.php?id=9767 (дата обращения: 08.05.2023).
3. 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: 22.05.2023).
4. 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.

Приложение А  
Листинг программ

А.1 Листинг программы лабораторной работы №1

//(-a-73-6\*b)/(-4\*c+200/d\*a); var:118

#include <iostream>

#include <stdio.h>

void printLabInfo() {

std::cout << "Ассемблер. Лабораторная работа № 1" << std::endl;

std::cout << "Арифметические и логические команды. Выполнил ученик группы 6104, Маликов Николай Дмитриевич." << std::endl;

std::cout << "Вариант: 118" << std::endl;

std::cout << "Задание: (-a-73-6\*b)/(-4\*c+200/d\*a)" << std::endl;

}

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

int result = 0;

\_\_asm {

mov eax, a // <--

mov ebx, b // <--

mov ecx, c // <--

mov edi, d // <--

imul ecx, -4 // c\*-4

add ecx, 200 // c\*-4+200

mov eax, ecx // c\*-4+200 --> eax

cdq // eax = edx:eax

mov ecx, a // ecx --> a !

imul edi, ecx // d\*a

idiv edi // -4\*c+200/d\*a

push eax // --> стек (1)

mov eax, a // a --> eax

imul ebx, -6 // b\*-6

add ebx, -73 // b\*-6-73

neg eax // Смена знака с - на + и наоборот

add eax, ebx // b\*-6-73-a

mov ebx, eax // eax --> ebx

pop ebx // ebx <-- стек(1)

cdq // eax = -4\*c+200/d\*a

idiv ebx // (-a-73-6\*b)/(-4\*c+200/d\*a)

mov result, eax

}

return result;

}

int main() {

setlocale(LC\_ALL, "Russian");

printLabInfo();

int a, b, c, d;

do

{

std::cout << "Введите: a != 0: \n";

std::cin >> a;

} while (a == 0);

std::cout << "Введите: b = \n";

std::cin >> b;

std::cout << "Введите: c = \n";

std::cin >> c;

do

{

std::cout << "Введите: d != 0: \n";

std::cin >> d;

} while (d == 0);

int result = calc(a, b, c, d);

int result1 = (-a - 73 + (-6 \* b)) / ((-4 \* c + 200) / (d \* a));

printf("Результат вычесления C++: %d\n", result1);

printf("Результат вычесления языка Ассембер: %d\n", result);

return 0;

}

А.2 Листинг программы лабораторной работы №2

#include <stdafx.h>

#include <stdio.h>

#include <iostream>

#include "Windows.h"

int calc(int a, int b)

{

int result = 0;

\_asm {

mov ecx, a; < ecx > = a

mov ebx, b; < ebx > = b

cmp ecx, ebx; сравнение a и b

jg l\_bigger; переход если a > b

jl l\_smaller; переход если a < b

mov eax, -21; < eax > = -21

imul ecx; < eax > = -21 \* a

jmp exit\_l; переход на конец программы

l\_bigger:

add ecx, 7; < ecx > = a + 7

or ecx, ecx; сравнение(a + 7) и 0

je error; ошибка деление на 0

mov eax, ebx; < eax > = b

imul eax; < edx:eax > = b \* b

jo error; ошибка переполнение

imul eax, 5; < eax > = 5 \* b \* b

idiv ecx; < eax > = 5\* b \* b / (a + 7)

jmp exit\_l;

l\_smaller:

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

or ebx, ebx; сравнение (b - a) и 0

je error; ошибка деление на 0

mov eax, a; < eax > = a

imul eax, 8; < eax > = 8 \* a

imul eax, b; < eax > = 8 \* a \* b

sub eax, 5; < eax > = 8 \* a \* b - 5

cdq; подготовка к делению < eax > => < edx:eax >

idiv ebx; < eax > = (8 \* a \* b - 5) / (b - a)

jmp exit\_l; переход на конец программы

error:

mov eax, -1; в случае ошибки результат программы приравнивается к -1 < eax > = -1

exit\_l:

mov result, eax; result = < eax >

}

return result;

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "RU");

printf("Лабораторная работа №2.Вариант №115.\nВыполнил студент группы 6104-020302D Свиридов Кирилл.\n");

printf("\t5\*b\*b/(a+7) a > b\n");

printf(" X = -21\*a\t\t a = b\n");

printf("\t(8\*a\*b-5)/(b-a) a < b\n");

int a, b;

printf("a = ");

scanf("%d", &a);

printf("b = ");

scanf("%d", &b);

int res = calc(a, b);

printf("Asm result = %d\n", res);

if (a > b)

{

if (a != -7)

{

res = 5 \* b \* b / (a + 7);

printf("Cpp result = %d\n", res);

}

else printf("Divided by Zero");

}

else if (a < b)

{

res = (8 \* a \* b - 5) / (b - a);

printf("Cpp result = %d\n", res);

}

else printf("Cpp result = %d\n", -21\*a);

system("PAUSE");

return 0;

}

А.3 Листинг программы лабораторной работы №3

#include <stdafx.h>

#include <stdio.h>

#include <iostream>

#include "Windows.h"

int counter(int arr[], int len)

{

int result = 0;

\_asm {

xor esi, esi; подготовка регистра индекса массива

mov ebx, arr; указывает на начало массива

mov ecx, len; счётчик цикла по элементам массива

jcxz error; Переход в error если длина массива 0

begin\_loop1:

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

test eax, 1; Побитовое сравнение И

jnp end\_loop1; Новый цикл если число нечетное

mov edx,esi; Запись номера четного элемента

cmp edx,len;Сравниваем индекс элемента и длину массива

je error; Переход если четный элемент один или их нет

jmp counter1; Переход в подготовку ко 2ому циклу

end\_loop1:

inc esi; Увеличить счетчик

loop begin\_loop1;

counter1:

mov ecx,len; Счётчик цикла по элементам массива

neg edx; Инвертируем edx

mov esi,ecx; Счетчик цикла по элементам массива с конца

dec esi; < esi > = len - 1

begin\_loop2:

mov eax, [ebx + esi \* 4]; Определяем текущий элемент с конца

test eax,1; Побитовое сравнение И

jnp end\_loop2; Новый цикл если число нечетное

dec esi; Уменьшаем индекс четного элемента на 1

add edx,esi; Вычисляем количество элементов между первым и последним четными

jmp exit\_l;Переход в конец программы

end\_loop2:

dec esi; Уменьшает номер элемента на 1

loop begin\_loop2;Повторяем цикл для всех элементов массива с конца

error:

mov edx, -1; Ошибка элемент только 1 обозначаем расстояние -1

exit\_l:

mov eax, edx; < eax > = < edx >

mov result, eax; result = < eax >

}

return result;

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "RUS");

printf("Лабораторная работа №3.Вариант №115.\nВыполнил студент группы 6104-020302D Свиридов Кирилл.\n");

printf("Вычислить количество элементов расположенных,\n между первым и последним четными элементами.\n");

printf("Длина массива = ");

int len;

scanf("%d", &len);

int\* arr = new int[len];

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

{

printf("%d элемент массива: ", i+1);

scanf("%d", &arr[i]);

}

printf("Ваш массив:\n");

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

{

printf("%d ", arr[i]);

}

printf("\n");

int count = counter(arr, len);

if (count == -1)

{

printf("Четный элемент только один или их нет\n");

}

else

{

printf("%d\n", count);

}

system("PAUSE");

return 0;

}

А.4 Листинг программы лабораторной работы №4

#include "stdafx.h"

#include <Windows.h>

#include <stdio.h>

#include <iostream>

#include <math.h>

double res(double a, double b)

{

double result = 0; const int c115 = 115;

\_asm {

finit;

fldpi;

fild c115;

fchs;

fmulp st(1), st;

fld qword ptr[a];

fldpi;

fmulp st(1), st;

fsin;

fsubp st(1), st;

fld qword ptr[b];

fcos;

fsubp st(1), st;

fld qword ptr[a];

fptan;

fdivrp st(1), st;

fsubp st(1), st;

fld qword ptr[a];

fchs;

fptan;

fdivp st(1), st;

faddp st(1), st;

fld qword ptr[a];

fld st;

fmulp st(1), st;

fld qword ptr[b];

faddp st(1), st;

fldpi;

fdivrp st(1), st;

fild c115;

fld qword ptr[a];

fmulp st(1), st;

fld qword ptr[b];

fsubp st(1), st;

fdivp st(1), st;

faddp st(1), st;

fstp result;

}

return result;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "rus");

printf("Лабораторная работа №4. Вариант №115.\n Выполнил студент группы 6104-020302D Свиридов Кирилл.\n");

printf("Вычислить выражение:\n -115 \* Pi - Sin(Pi \* a) - Cos(b) - Tg(a) + Ctg(-a) + Pi / (a \* a + b)/(115 \* a - b)\n");

double a, b;

printf("Введите a: ");

scanf("%lf", &a);

printf("Введите b: ");

scanf("%lf", &b);

auto pi = acos(-1.0);

if ((b != a \* a) && (fmod(a, pi / 2) != 0.0))

{

printf("Значение выражения Asm: %lf\n", res(a, b));

double res\_cpp = -115 \* pi - sin(pi \* a) - cos(b) - tan(a) + (1 / tan(-a)) + pi / (a \* a + b) / (115 \* a - b);

printf("Значение выражения Cpp: %lf\n", res\_cpp);

}

else

{

printf("Деление на 0.\n");

}

system("pause");

return 0;

}

А.5 Листинг программы лабораторной работы №5

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include <stdio.h>

#include <iostream>

#include <math.h>

double res(double x, double y)

{

double temp = 0; double result = 0; const int c2 = 2; const int c28 = 28; const int c29 = 29; const int c3 = 3; const int c4 = 4; const int c5 = 5; const int c7 = 7; const int c8 = 8;

\_asm {

finit;

fild c28;

fild c29;

fdivp st(1), st;

fld x;

fyl2x;

fld st;

frndint;

fsub st(1), st;

fld1;

fscale;

fxch st(1);

fstp temp;

fxch st(1);

f2xm1;

fld1;

fadd;

fmulp st(1), st;

fild c8;

fild c7;

fdivp st(1), st;

fld x;

fxch st(1);

fyl2x;

f2xm1;

fld1;

fadd;

fadd;

fild c7;

fild c8;

fdivp st(1), st;

fld x;

fmulp st(1), st;

fsin;

fadd;

fild c7;

fild c8;

fdivp st(1), st;

fld x;

fmulp st(1), st;

fld st;

fmulp st(1), st;

fld1;

fxch st(1);

fyl2x;

fldl2e;

fdivp st(1), st;

fsub;

fild c7;

fild c8;

fdivp st(1), st;

fld x;

fmulp st(1), st;

fptan;

fdivp st(1), st;

fadd;

fldz;

fld y;

fcomi st,st(1);

jb A;

ja B;

fstp temp;

jmp D;

A:

fadd;

fabs;

fld y;

fild c3;

fmulp st(1),st;

fchs;

fld1;

fadd;

fld st;

fild c2;

fdivp st(1),st;

frndint;

fild c2;

fmulp st(1), st;

fcomip st, st(1);

je P;

fxch st(1);

fyl2x;

fld st;

frndint;

fsub st(1), st;

fld1;

fscale;

fxch st(1);

fstp temp;

fxch st(1);

f2xm1;

fld1;

fadd;

fmulp st(1), st;

fchs;

jmp D;

P:

fxch st(1);

fyl2x;

fld st;

frndint;

fsub st(1), st;

fld1;

fscale;

fxch st(1);

fstp temp;

fxch st(1);

f2xm1;

fld1;

fadd;

fmulp st(1), st;

jmp D;

B:

fadd;

fld1;

fld c3;

fdivp st(1),st;

fxch st(1);

fcomi st, st(1);

ja C;

jb C\_e;

fxch st(1);

fstp temp;

fld c3;

fmulp st(1), st;

fchs;

fld1;

fadd;

fld y;

fyl2x;

fld st;

frndint;

fsub st(1), st;

fld1;

fscale;

fxch st(1);

fstp temp;

fxch st(1);

f2xm1;

fld1;

fadd;

fmulp st(1), st;

jmp D;

C:

fxch st(1);

fstp temp;

fld c3;

fmulp st(1), st;

fchs;

fld1;

fadd;

fabs;

fld y;

fyl2x;

fld st;

frndint;

fsub st(1),st;

fld1;

fscale;

fxch st(1);

fstp temp;

fxch st(1);

f2xm1;

fld1;

fadd;

fmulp st(1), st;

fld1;

fdivrp st(1), st;

jmp D;

C\_e:

fstp temp;

fstp temp;

fld1;

D:

fadd;

fild c5;

fld y;

fmulp st(1), st;

fsin;

fild c5;

fmulp st(1), st;

fadd;

fild c4;

fld y;

fmulp st(1), st;

fcos;

fsubp st(1), st;

fld y;

fchs;

fld1;

fadd;

fsqrt;

fld1;

fxch st(1);

fyl2x;

fldl2e;

fdivp st(1), st;

fild c4;

fmulp st(1), st;

fsubp st(1), st;

fild c5;

fldz;

fld y;

fcomi st,st(1);

jb E\_1;

je E\_e;

fxch st(1);

fstp temp;

fxch st(1);

fyl2x;

fld st;

frndint;

fsub st(1), st;

fld1;

fscale;

fxch st(1);

fstp temp;

fxch st(1);

f2xm1;

fld1;

fadd;

fmulp st(1), st;

jmp F;

E\_1:

fadd;

fabs;

fxch st(1);

fyl2x;

fld st;

frndint;

fsub st(1), st;

fld1;

fscale;

fxch st(1);

fstp temp;

fxch st(1);

f2xm1;

fld1;

fadd;

fmulp st(1), st;

fld1;

fdivrp st(1), st;

jmp F;

E\_e:

fadd;

fadd;

fstp temp;

fld1;

F:

fadd;

fld1;

fld y;

fcos;

fdivp st(1), st;

fadd;

fst result;

}

return result;

}

double res\_c(double a, double b)

{

double result = 0;

double temp = 28.0 / 29.0;

result = pow(a, temp);

temp = -a;

result += pow((7.0 / 8.0), temp);

result += sin((7.0 \* a / 8.0));

temp = pow(7.0 \* a / 8.0, 2);

result -= log(temp);

temp = sin((7.0 \* a / 8.0));

temp /= cos((7.0 \* a / 8.0));

result += temp;

result += pow(b, (1 - 3 \* b));

result += 5 \* sin((5 \* b));

result -= cos((4 \* b));

temp = sqrt(1 - b);

result -= 4 \* log(temp);

result += pow(5, b);

temp = 1 / cos(b);

result += temp;

return result;

}

int main()

{

setlocale(LC\_ALL, "rus");

printf("Лабораторная работа №5. Вариант №28.\n Выполнил студент группы 6104-020302D Свиридов Кирилл.\n");

printf("Вычислить выражение:\n x^(28/29)+(7/8)^(-x)+sin(7\*x/8)-ln(7\*x/8)^2+ctg^(-1)(7x/8)+y^(1-3\*y)+5\*sin(5\*y)-cos(4\*y)-4\*ln((1-y)^0.5)+5^y+sec(y)\n");

double x1, x2, y1, y2, hx, hy;

bool key = true;

while (key)

{

printf("Введите границы x через пробел(x > 0):");

scanf("%lf %lf", &x1, &x2);

if (x1 > x2)

{

printf("Некорректно введены границы x. Повторите снова.\n");

}

else

{

printf("Введите границы y через пробел(0<y<1, или y <= 0, но принадлежит целым):");

scanf("%lf %lf", &y1, &y2);

if (y1 > y2)

{

printf("Некорректно введены границы y. Повторите снова.\n");

}

else

{

printf("Введите шаг для x, находящегося в пределах от %lf до %lf: ", x1, x2);

scanf("%lf", &hx);

if (hx<0 || hx>(x2 - x1))

{

printf("Некорректно введен шаг для x. Повторите снова.\n");

}

else

{

printf("Введите шаг для y, находящегося в пределах от %lf до %lf: ", y1, y2);

scanf("%lf", &hy);

if (hy<0 || hy>(y2 - y1))

{

printf("Некорректно введен шаг для y. Повторите снова.\n");

}

else { key = false; }

}

}

}

}

int k = 1;

auto pi = acos(-1.0);

printf("№\t x\t y\t\tf(x,y) ассемблер\t\tf(x,y) c++\n");

for (double i = x1; i <= x2; i += hx)

{

for (double j = y1; j <= y2; j += hy)

{

if ((i > 0) && (cos(7 \* i / 8) != 0) && (sqrt(1 - j) > 0) && ((1 - j) >= 0) && ((j >= 0) || (fmod(1-3\*j, 1) == 0)) && (cos(j) != 0))

{

printf("%2d\t%5.2lf\t%5.2lf\t%20.2lf\t\t%20.2lf\n", k, i, j, res(i, j), res\_c(i, j));

k++;

}

else

{

printf("%2d\t%5.2lf\t%5.2lf\t\t\tВычисление невозможно.\n", k, i, j);

k++;

}

}

}

system("pause");

return 0;

}