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]:

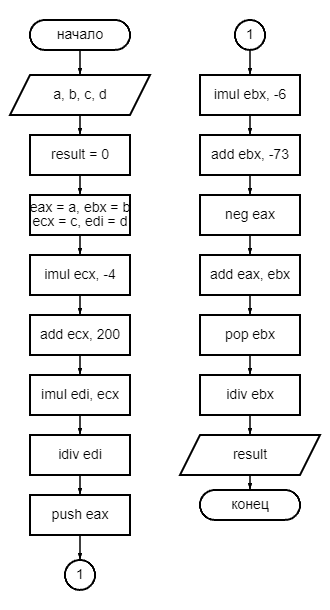
* MOV – команда копирования данных из одной переменной в другую. Команда копирует содержимое второго операнда в первый операнд. При этом содержимое второго операнда не изменяется;
* ADD – команда выполняет целочисленного сложения двух операндов и флага переноса CF. Результат сложения помещается в первый 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.

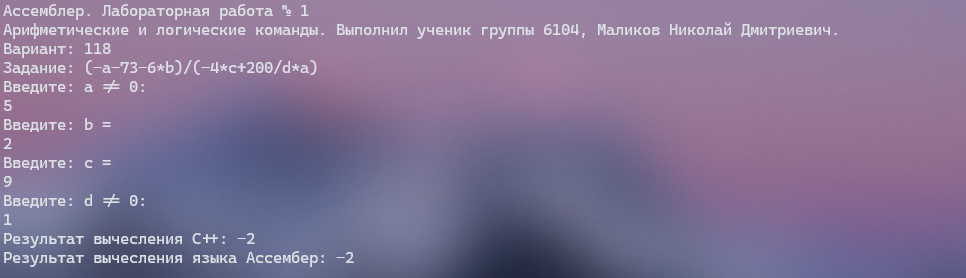


Рисунок 1.2 – Пример работы программы при значении аргументов a = 5, b= 2, c = 9, d = 1

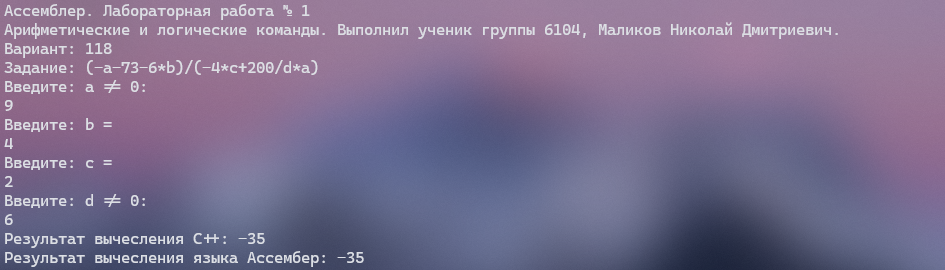


Рисунок 1.3 – Пример работы программы при значениях аргументов a = 9, b = 4, c = 2, d = 6

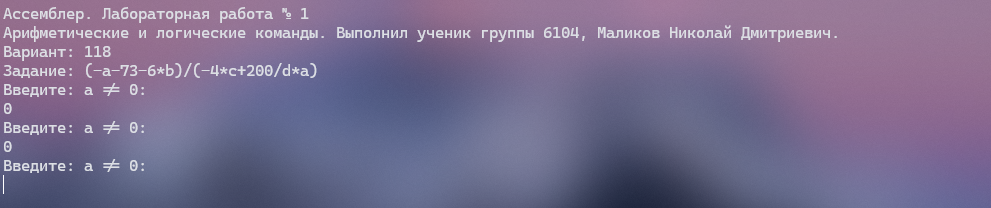


Рисунок 1.4 – Пример обработки исключительной ситуации в программе Лабораторная работа 2 «Арифметические команды и команды переходов в ассемблере»

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 (a – 9) на равенство с 0. Если (a – 9) = 0, передаем ошибку деления на нуль, иначе присваиваем переменной result заданную функцию result = b / ( a – 9 ). Если a < b, проверяем (a + b) на равенство с 0. Если (a + b) = 0, передаем ошибку деления на нуль, иначе присваиваем переменной result заданную функцию result = (6a \*b - 5)/(a + b). В случае равенства a = b, переменной result присваивается значение result = -100. Выводим значение result.

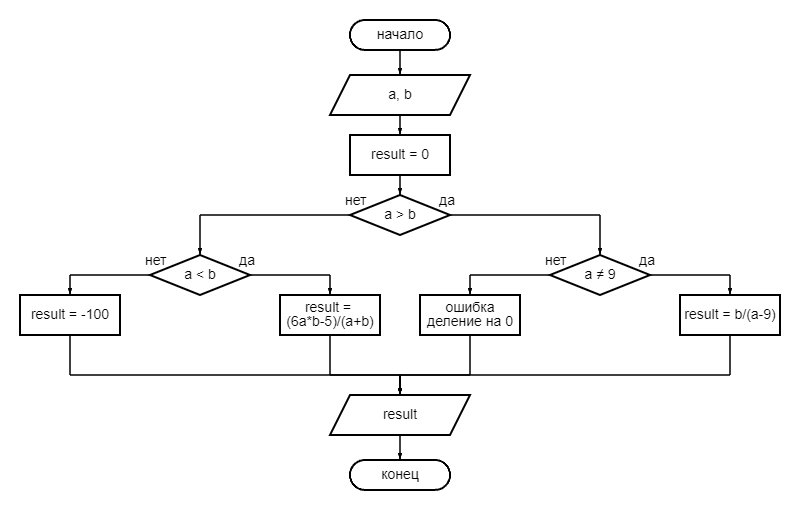


Рисунок 2.1 – Схема алгоритма вычисления условного выражения

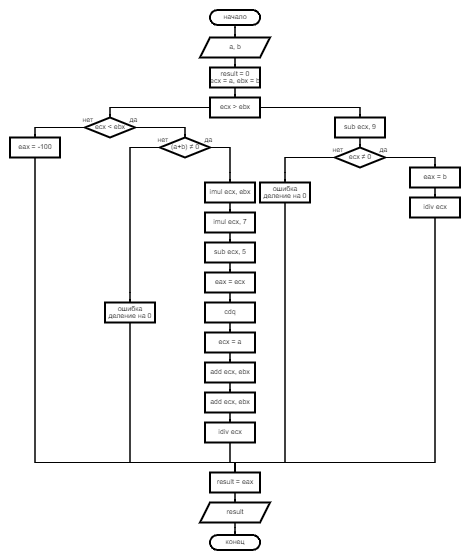


Рисунок 2.2 – Схема алгоритма на языке Ассемблера

2.4 Результаты тестирования

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

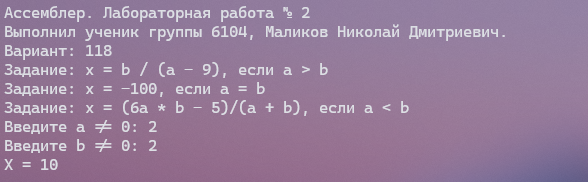


Рисунок 2.3 – Пример работы программы вычисления целочисленного значения при значениях параметров a = 2, b = 2

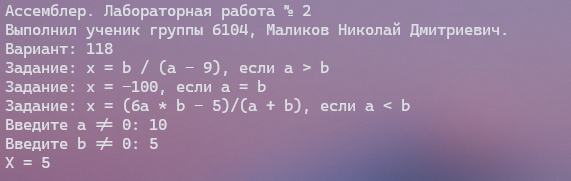


Рисунок 2.4 – Пример работы программы вычисления целочисленного значения при значениях параметров a = 10, b = 5

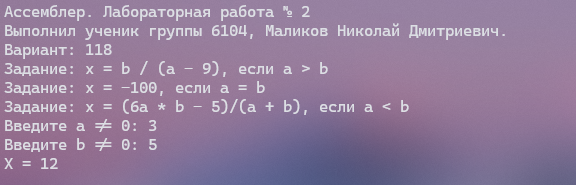


Рисунок 2.5 – Пример работы программы вычисления целочисленного значения при значениях параметров a = 3, b = 5

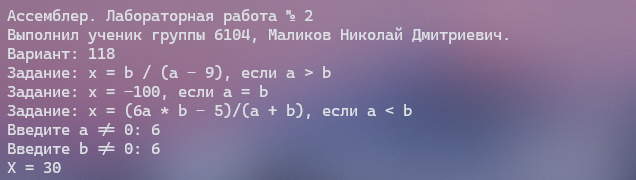


Рисунок 2.6 – Пример работы программы вычисления целочисленного значения при значениях параметров a = 6, b = 6

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**:**В одномерном массиве A={a[i]} целых чисел вычислить сумму между соседними элементам {a[i]=a[i]+a[i+1]}.

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.

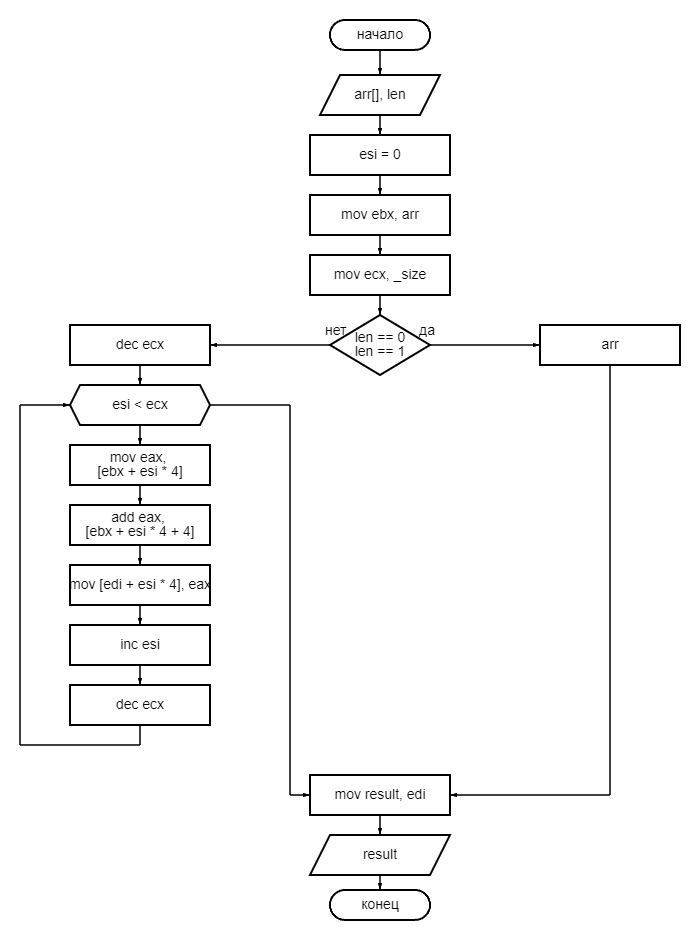
В параметры функции передаются значения переменных, вводимых пользователем в главной программе: arr, хранящая массив элементов, len, хранящая количество элементов массива. Переменной result, отвечающей за хранения результата вычисления исходного выражения. Проверяем, является ли размер массива 0 или 1, и если да, возвращает массив без изменений. Выделяем память для нового массива размером на один элемент меньше, чем входной массив. Инициализируем индекс esi, который будет использоваться для доступа к элементам массива. Сохраняем адрес первого элемента массива в регистре ebx. Сохраняем размер массива в регистре ecx. Создаем цикл, который проходит по всем парам соседних элементов массива суммируя их, сохраняем результаты в новом массиве. Уменьшаем размер массива на 1, так как мы уже обработали первую пару элементов. Возвращаем указатель на новый массив после завершения цикла. 

Рисунок 3.1 – Схема алгоритма подсчета суммы соседних элементов

3.4 Результат тестирования

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

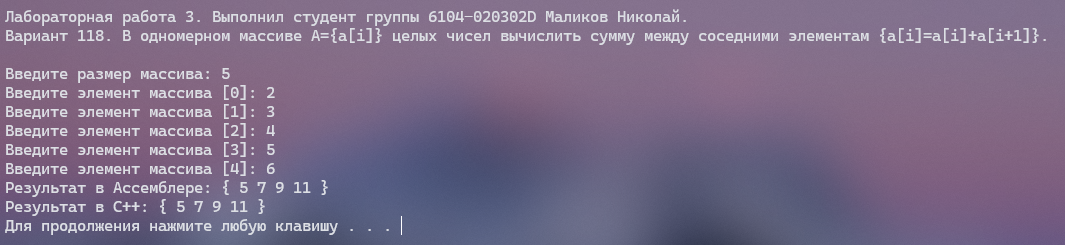


Рисунок 3.2 – Пример работы программы вычисления суммы элементов при значении массива {2; 3; 4; 5; 6;}

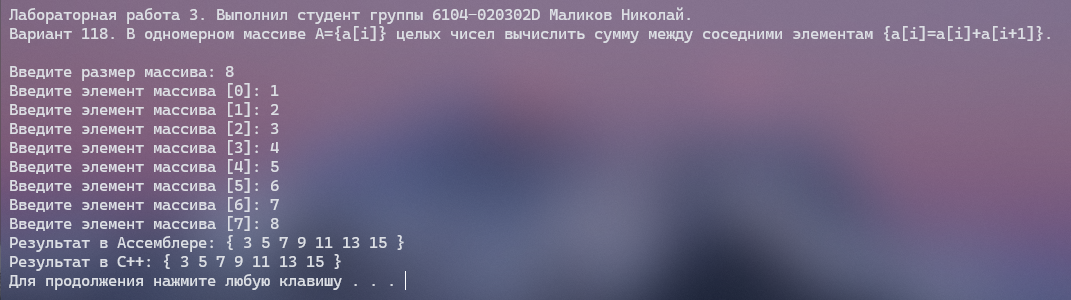


Рисунок 3.3 – Пример работы программы вычисления суммы элементов при значении массива {1; 2; 3; 4; 5; 6; 7; 8;}

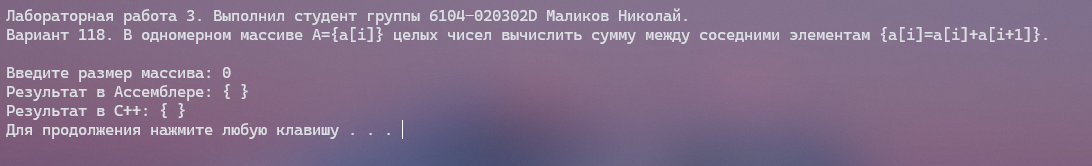


Рисунок 3.4 – Пример работы программы вычисления суммы элементов при значении массива {0}

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: X=118\*π \*sin(b)-cos(-a)-tg(b)+ctg(π \*b)- π \*(118\*b+a)/(118\*a-b)

4.3 Схема алгоритма

Схема основного алгоритма приведена на рисунке 4.1. Функция count принимает два аргумента: a и b. Она начинает с проверки, является ли b целым числом. Это делается путем сохранения b в стеке, вычисления его целой части с помощью fisttp, затем повторного сохранения b в стеке и вычитания его целой части. Если результат этой операции равен нулю, то b является целым числом, и функция переходит к обработке ошибки.

Если b не является целым числом, функция продолжает выполнение. Она сначала вычисляет 118.0 \* pi \* sin(b), затем вычитает cos(a) из этого результата. После этого она вычисляет tan(b) и вычитает его из текущего результата. Далее функция вычисляет cot(pi \* b) и добавляет его к текущему результату.

Затем функция вычисляет (118.0 \* b + a) / (118.0 \* a - b) и вычитает этот результат из текущего значения. Наконец, функция сохраняет окончательный результат в переменной result и возвращает его.

Если в процессе выполнения возникает ошибка, функция выводит сообщение об ошибке и завершает работу.

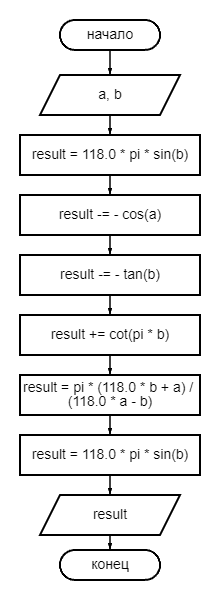


Рисунок 4.1 – Схема алгоритма подсчета вещественного выражения

4.4 Результаты тестирования

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

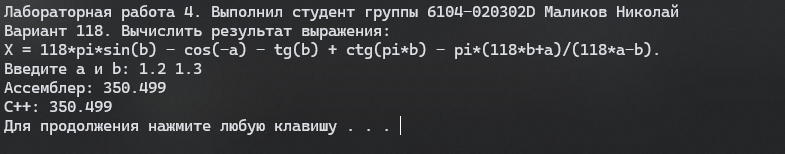


Рисунок 4.2 – Пример работы программы вычисления вещественного значения при значениях параметров a = 1.2, b = 1.3

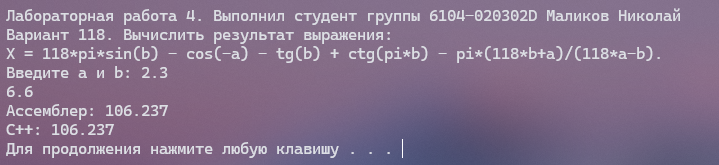


Рисунок 4.3 – Пример работы программы вычисления вещественного значения при значениях параметров a = 2.3, b = 6.6

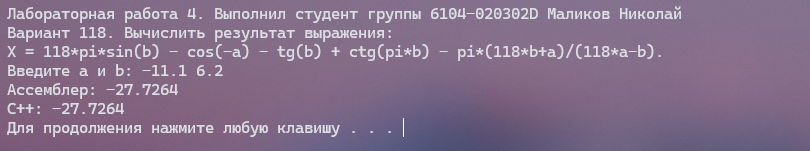


Рисунок 4.4 – Пример работы программы вычисления вещественного значения при значениях параметров a = -11.1, b = 6.2

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

#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 <iostream>

#include<cstdio>

using namespace std;

int systemOfEquations(int a, int b) {

int result = 0;

\_\_asm {

mov ecx, a // eсx <-- a

mov ebx, b // ebx <-- b

cmp ecx, ebx // a ?= b

je grp\_a // переход если a равно b \* переход если равно JE

jg grp\_b // переход если a больше b \* переход если больше JG

jl grp\_c // переход если a меньше b \* переход если меньше JL

grp\_a : // a = b

imul ebx, 5 // b \* 5

mov eax, ebx // eax <-- b \* 5

jmp go\_out // переместиться к выводу

grp\_b : // a > b

mov eax, ebx // eax <-- b

cdq // eax = edx:eax

sub ecx, 9 // a - 9

cmp ecx, 9 // a ?= 9

je error\_zero // if a = 9 --> error\_zero

idiv ecx // b / a - 9

jmp go\_out // переместиться к выводу

grp\_c : // a < b

imul ecx, ebx // a \* b

imul ecx, 7 // a \* b \* 7

sub ecx, 5 // a \* b \* 7 - 5

mov eax, ecx // eax <-- a \* b \* 7 - 5

cdq // eax = edx:eax

mov ecx, a // ecx <-- a

add ecx, ebx // a + b

cmp ecx, 0

je error\_zero

mov eax, ecx // (a \* b \* 7 - 5)/(a + b)

jmp go\_out // переместиться к выводу

error\_zero :

mov eax, -1

go\_out :

mov result, eax // result <-- eax

}

return result;

}

void printLabInfo() {

std::cout << "Ассемблер. Лабораторная работа № 2" << std::endl;

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

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

std::cout << "Задание: x = b / (a - 9), если a > b" << std::endl;

std::cout << "Задание: x = -100, если a = b" << std::endl;

std::cout << "Задание: x = (6a \* b - 5)/(a + b), если a < b" << std::endl;

}

int main() {

setlocale(LC\_ALL, "Russian");

printLabInfo();

int a, b;

do

{

cout << "Введите a != 0: ";

cin >> a;

} while (a == 0);

do

{

cout << "Введите b != 0: ";

cin >> b;

} while (b == 0);

printf("X = %d\n", systemOfEquations(a, b));

return 0;

}

А.3 Листинг программы лабораторной работы №3

#include <iostream>

int\* array\_work\_asm(int\* arr, int \_size) {

if (\_size == 0 || \_size == 1) return arr;

int\* result = new int[\_size - 1];

\_\_asm

{

xor esi, esi; // i = 0

mov ebx, arr; // кладёт адрес начала массива в ebx

mov ecx, \_size; // ecx = \_size

dec ecx; // ecx = \_size - 1

mov edi, result; // кладёт адрес начала результата в edi

loop\_start:

mov eax, [ebx + esi \* 4]; // eax = arr[i]

add eax, [ebx + esi \* 4 + 4]; // eax = arr[i] + arr[i + 1]

mov [edi + esi \* 4], eax; // result[i] = eax

inc esi; // i++

dec ecx; // ecx = ecx - 1

jnz loop\_start; // если ecx != 0, то переход в loop\_start

end:

mov result, edi;

}

return result;

}

int\* array\_work(int\* arr, int size) {

if (size == 0 || size == 1) return arr;

int\* result = new int[size - 1];

for (int i = 0; i < size - 1; ++i) {

result[i] = arr[i] + arr[i + 1];

}

return result;

}

int main()

{

setlocale(LC\_ALL, "Russian");

std::cout << "Лабораторная работа 3. Выполнил студент группы 6104-020302D Маликов Николай.\n";

std::cout << "Вариант 118. В одномерном массиве A={a[i]} целых чисел вычислить сумму между соседними элементам {a[i]=a[i]+a[i+1]}.\n\n";

int size;

std::cout << "Введите размер массива: ";

std::cin >> size;

int\* arr = new int[size];

int\* arrCopy = new int[size];

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

std::cout << "Введите элемент массива [" << i << "]: ";

std::cin >> arr[i];

arrCopy[i] = arr[i];

}

arr = array\_work\_asm(arr, size);

arrCopy = array\_work(arrCopy, size);

if (size > 1)

size -= 1;

std::cout << "Результат в Ассемблере: {";

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

std::cout << " " << arr[i];

}

std::cout << " }" << std::endl;

std::cout << "Результат в C++: {";

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

std::cout << " " << arrCopy[i];

}

std::cout << " }" << std::endl;

delete[] arr;

delete[] arrCopy;

system("pause");

}

А.4 Листинг программы лабораторной работы №4

##include <iostream>

#include <cmath>

double cot(double x) {

return 1 / tan(x);

}

double count(double a, double b) {

bool error = false;

double result;

const double c = 118.0;

int intPart;

\_\_asm

{

fld b; // put b in st(0)

fisttp intPart; // put int(b) in intPart

fld b; // put b in st(0)

fild intPart; // put intPart in st(0)

fsub; // st(0) = b - int(b) (проверка на целость числа)

fldz; // put 0 in st(0)

fucomip st(0), st(1); // compare st(0) and st(1) with EFLAGS changing

je \_error; // if st(0) == st(1) goto \_error (если b - целое число)

fstp st(0); // remove trash from st(0) after compare

fld c; // put 118.0 in st(0)

fldpi; // put pi in st(0)

fmul; // st(0) = 118.0 \* pi

fld b; // put b in st(0)

fsin; // st(0) = sin(b)

fmul; // st(0) = 118.0 \* pi \* sin(b)

fld a; // put a in st(0)

fcos; // st(0) = cos(a)

fsub; // st(0) = 118.0 \* pi \* sin(b) - cos(a)

fld b; // put b in st(0)

fptan; // st(0) = 1 | st(1) = tan(b)

fstp st(0); // remove trash 1 from st(0)

fsub; // st(0) = 118.0 \* pi \* sin(b) - cos(a) - tan(b)

fldpi; // put pi in st(0)

fld b; // put b in st(0)

fmul; // st(0) = pi \* b

fptan; // st(0) = 1 | st(1) = tan(pi \* b)

fdivr; // st(0) = 1 / tan(pi \* b) = cot(pi \* b)

fadd; // st(0) = (118.0 \* pi \* sin(b) - cos(a) - tan(b) + cot(pi \* b)) = [1]

fld c; // put 118.0 in st(0)

fld b; // put b in st(0)

fmul; // st(0) = 118.0 \* b

fld a; // put a in st(0)

fadd; // st(0) = 118.0 \* b + a

fldpi; // put pi in st(0)

fmul; // st(0) = pi \* (118.0 \* b + a)

fld c; // put 118.0 in st(0)

fld a; // put a in st(0)

fmul; // st(0) = 118.0 \* a

fld b; // put b in st(0)

fsub; // st(0) = 118.0 \* a - b

fldz; // put 0 in st(0)

fucomip st(0), st(1); // compare st(0) and st(1)

je \_error; // if st(0) == 0 goto \_error

fdiv; // st(0) = pi \* (118.0 \* b + a) / (118.0 \* a - b)

fsub; // st(0) = [1] - pi \* (118.0 \* b + a) / (118.0 \* a - b)

fst result; // result = 118.0 \* pi \* sin(b) - cos(a) - tan(b) + cot(pi \* b) - pi \* (118.0 \* b + a) / (118.0 \* a - b)

jmp end;

\_error:

mov error, 1;

end:

}

if (error) {

std::cout << "Ошибка, введено недопустимое значение!" << std::endl;

system("pause");

exit(0);

}

return result;

}

double count\_no\_asm(double a, double b) {

double pi = atan(1) \* 4;

return 118 \* pi \* sin(b) - cos(-a) - tan(b) + cot(pi \* b) - pi \* (118 \* b + a) / (118 \* a - b);

}

int main()

{

setlocale(LC\_ALL, "Russian");

std::cout << "Лабораторная работа 4. Выполнил студент группы 6104-020302D Маликов Николай" << std::endl;

std::cout << "Вариант 118. Вычислить результат выражения:\nX = 118\*pi\*sin(b) - cos(-a) - tg(b) + ctg(pi\*b) - pi\*(118\*b+a)/(118\*a-b)." << std::endl;

double a, b;

std::cout << "Введите a и b: ";

std::cin >> a >> b;

double r = count(a, b);

std::cout << "Ассемблер: " << r << std::endl;

double r2 = count\_no\_asm(a, b);

std::cout << "С++: " << r2 << std::endl;

system("pause");

return 0;

}