**Московский авиационный институт**

**(национальный исследовательский университет)**

Институт №3

Системы управления, информатики и электроэнергетики

Кафедра 304

Вычислительные машины, системы и сети

**Отчет по лабораторной работе**

**по учебной дисциплине**

**«Языки Ассемблера»**

**на тему:**

***«Арифметические операции языка Ассемблера»***

*Группа:* ***М30-109Б-19***

*Выполнил:*

***Кузнецов И.И.***

[*iluxandro@gmail.com*](mailto:iluxandro@gmail.com)

*Принял:*

***Ратников Максим Олегович***

***Москва 2020***

**Содержание.**

**Задание\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3**

**Конспект по теме\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 4**

**Код программы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_5**

**Тесты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7-9**

**Вывод\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10**

***Задание***

**Вариант №7**

C:\Users\Nata\Desktop\Image 1.png

***Конспект***

***Задание :*** Разобрать пример выполнения команды ADC.

Команда ADC производит сложение двух операндов одинаковой длины и значения флажка CF, установленного предыдущими операциями. Если CF= 0, то команда ADC эквивалента команде ADD.

Сложение операндов с переносом, образованным предыдущей командой записывается так:

***ADC op1, op2 ;***

***op1 := op1+op2+CF***

Пара команд ADD и ADC применяется для сложения повышенной точности.

Рассмотрим следующий пример показывающий это:

# include <iostream>

using namespace std;

int main()

{

system("color F0");

setlocale(LC\_ALL, "Russian");

short int a;

short int b;

\_asm {

mov ax, 0FFFFh // 32-разрядное число 65535

mov bx, 1 // 32-разрядное число 1

mov cx, 0 // В паре BX:CX

mov dx, 0 // В паре AX:DX

// 65535 + 1 = 65536, то есть после сложения DX:AX = 65536 (0001:0000)

add ax, bx // AX = 0000

adc dx, cx // DX = 0001

mov a, ax

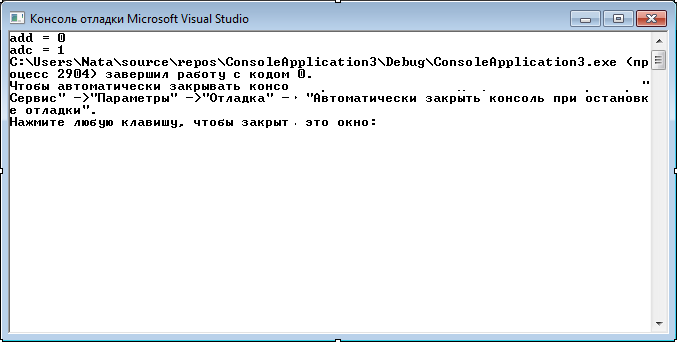
mov b, dx

}

cout << "add = " << a << endl << "adc = " << b;

return 0;

}



***Код программы***

# include <iostream>

using namespace std;

int main()

{

system("color F0");

setlocale(LC\_ALL, "Russian");

short int a = 29;

short int b = 5;

short int c = -34;

short int d = 44;

short int res;

res = ((b \* b - a) / (c + 12)) + 4 \* d;

\_asm {

//((b^2-a) Div (c+12)) + 4 \* d

mov ax, b //в регистр ax помещаем значение переменной b

mul b //умножаем b на ax сохраняя полученный результат в ax; ax = 18 \* 18 = 324

mov bx, a //в регистр bx помещаем значение переменной a

sub ax, bx //вычитаем из ax bx сохраняя полученный результат в ax; ax = 324 - 29 = 295

mov bx, c //в регистр bx помещаем значение перменной c

add bx, 12 //к регистру bx добавляем 12 сохряняя полученный результат в bx; bx = -34 + 12 = -22

xor dx, dx //очищаем мусор лежащий в dx после умножения

cwd //расширение регистра ax на два регистра ax и dx

idiv bx //знаковое деление - делим ax на bx

//в регистр dx попадет остаток от деления, а в ax делитель

//ax = -13, dx = 9

mov cx, ax //сохраним значение регистра ax в регистр cx, ибо при следующем умножении ax будет использован

mov ax, 4 //в регистр ax помещаем 4

mul d //умножаем ax на d сохраняя в ax = 4 \* 44 = 176

add ax, cx //добавляем к регистру ax значение регистра cx, сохраняя в ax 176 + (-13) = 163

mov b, ax //переносим значение из регистра ax в переменную b

}

cout << "результат asm = " << b << endl << "результат c++ = " << res << endl;

return 0;

}

***Тесты***

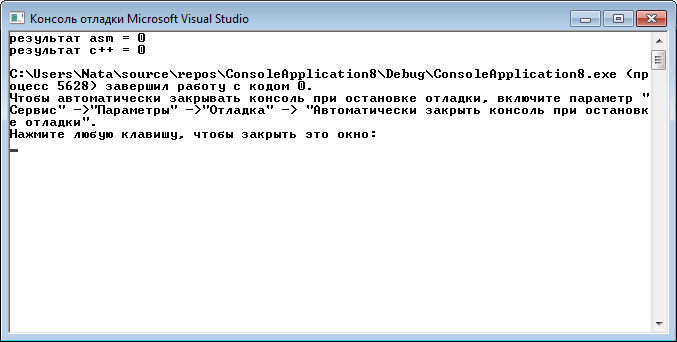
**Тест 1**

***Цель:*** проверить работу программы при всех нулях:

a = 0; b = 0; c = 0; d = 0;

***Ожидаемый результат****:*  0

***Полученный результат:***



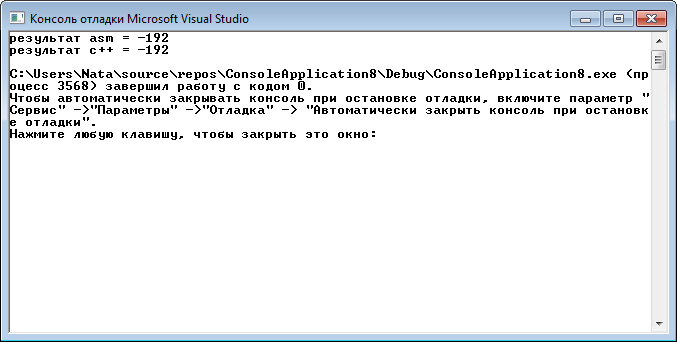
**Тест 2**

***Цель:*** проверить работу программы при всех отрицательных значениях:

a = -29; b = -18; c = -34; d = -44;

***Ожидаемый результат:*** -192

***Полученный результат:***



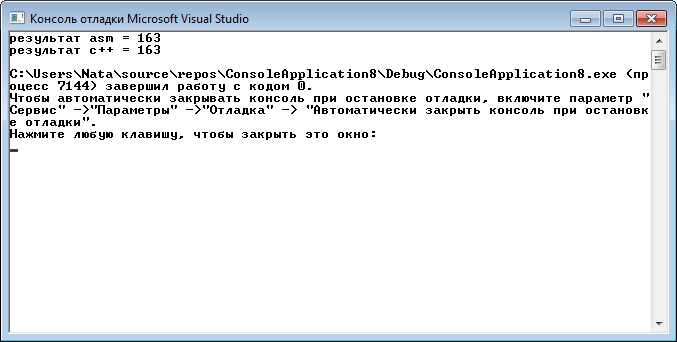
**Тест 3**

***Цель:*** проверить работу программы при заданных вариантом значениях:

a = 29; b = 18; c = -34; d = 44;

***Ожидаемый результат:*** 163

***Полученный результат:***

****

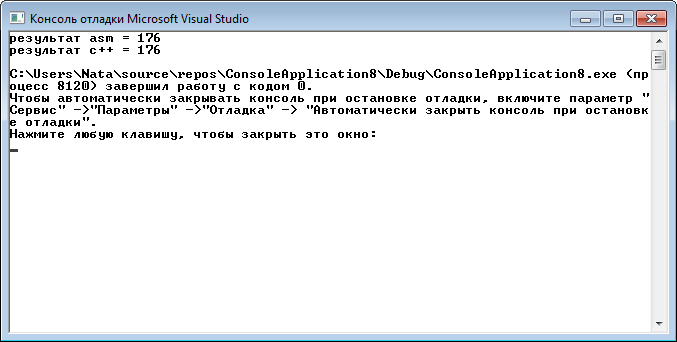
**Тест 4**

***Цель:*** проверить работу программы при (b\*b – a) < 0:

a = 29; b = 5; c = -34; d = 44;

***Ожидаемый результат:*** 176

***Полученный результат:***



**Вывод:**

таким образом, мы научились работать с базовыми арифметическими операциями в ассемблере, протестировали программу, выявили некоторые проблемы и исправили их. Также изучили команду ADC, как команда, работающая в совокупности с командой ADD.