МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ

БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет автоматики и вычислительной техники

Кафедра радиоэлектронных средств

Отчет по дисциплине

«Цифровые устройства и микропроцессоры»

Лабораторная работа №3

«ИСПОЛЬЗОВАНИЕ МАТЕМАТИЧЕСКОГО СОПРОЦЕССОРА»

Вариант №11

|  |  |  |
| --- | --- | --- |
| Выполнил: студентка гр. ИКТб-3301 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Н.А. Мусихина |
| Проверил: преподаватель кафедры РЭС | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | М.А. Земцов |

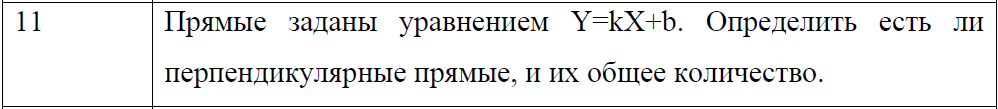
Киров 2023

**Цель работы:** изучение принципов выполнения арифметических команд с помощью математического сопроцессора FPU микропроцессоров с архитектурой x86.

**Ход работы:**

Условие варианта представлено в таблице 1.

Таблица 1 – Условие варианта



**Текст программы представлен ниже:**

.686

.model flat,stdcall

.stack 100h

.data

arr\_k dd 8.0 ; 0 элемент 4 байта два слова - короткий формат

dd -0.125 ; 1 элемент 4 байта два слова - короткий формат

dd 4.0 ; 2 элемент 4 байта два слова - короткий формат

dd -0.25 ; 3 элемент 4 байта два слова - короткий формат

dd 7.0 ; 4 элемент 4 байта два слова - короткий формат

znach1 dd 0.0 ; 0 элемент для сохранения результата 1 расчета

dd 0.0 ; 1 элемент для сохранения результата 2 расчета

dd 0.0 ; 2 элемент для сохранения результата 3 расчета

dd 0.0 ; 3 элемент для сохранения результата 4 расчета

dd 0.0 ; 4 элемент для сохранения результата 5 расчета

dd 0.0 ; 5 элемент для сохранения результата 6 расчета

dd 0.0 ; 6 элемент для сохранения результата 7 расчета

dd 0.0 ; 7 элемент для сохранения результата 8 расчета

dd 0.0 ; 8 элемент для сохранения результата 9 расчета

dd 0.0 ; 9 элемент для сохранения результата 10 расчета

CHISLO dd -1.0 ; переменная для сравнения

NULL dd 0.0 ; переменная для обнуления счетчика перпендикулярных прямых

.code

ExitProcess PROTO STDCALL :DWORD

Start:

FINIT;инициализация сопроцессора

;расчет ПЕРВОГО элемента со всеми остальными

FLD arr\_k[0\*4] ; помещаем в ST0 0 элемент массива \*4 означает, что элемент соотвествует двойному слову

FMUL arr\_k[1\*4] ; умножаем значение в ST0 на 1 элемент массива \*4 означает, что элемент соотвествует двойному слову

FST znach1[0\*4] ; помещаем результат в 0 элемент массива znach1

FFREE ST(0) ; очистка регистра стека

FLD arr\_k[0\*4] ; помещаем в ST0 0 элемент массива \*4 означает, что элемент соотвествует двойному слову

FMUL arr\_k[2\*4] ; умножаем значение в ST0 на 2 элемент массива \*4 означает, что элемент соотвествует двойному слову

FST znach1[1\*4] ; помещаем результат в 1 элемент массива znach1

FFREE ST(0) ; очистка регистра стека

FLD arr\_k[0\*4] ; помещаем в ST0 0 элемент массива \*4 означает, что элемент соотвествует двойному слову

FMUL arr\_k[3\*4] ; умножаем значение в ST0 на 3 элемент массива \*4 означает, что элемент соотвествует двойному слову

FST znach1[2\*4] ; помещаем результат в 2 элемент массива znach1

FFREE ST(0) ; очистка регистра стека

FLD arr\_k[0\*4] ; помещаем в ST0 0 элемент массива \*4 означает, что элемент соотвествует двойному слову

FMUL arr\_k[4\*4] ; умножаем значение в ST0 на 4 элемент массива \*4 означает, что элемент соотвествует двойному слову

FST znach1[3\*4] ;помещаем результат в 3 элемент массива znach1

FFREE ST(0) ; очистка регистра стека

;расчет ВТОРОГО элемента со всеми остальными

FLD arr\_k[1\*4] ; помещаем в ST0 1 элемент массива \*4 означает, что элемент соотвествует двойному слову

FMUL arr\_k[2\*4] ; умножаем значение в ST0 на 4 элемент массива \*4 означает, что элемент соотвествует двойному слову

FST znach1[4\*4] ; помещаем результат в 4 элемент массива znach1

FFREE ST(0) ; очистка регистра стека

FLD arr\_k[1\*4] ; помещаем в ST0 1 элемент массива \*4 означает, что элемент соотвествует двойному слову

FMUL arr\_k[3\*4] ; умножаем значение в ST0 на 4 элемент массива \*4 означает, что элемент соотвествует двойному слову

FST znach1[5\*4] ; помещаем результат в 5 элемент массива znach1

FFREE ST(0) ; очистка регистра стека

FLD arr\_k[1\*4] ; помещаем в ST0 1 элемент массива \*4 означает, что элемент соотвествует двойному слову

FMUL arr\_k[4\*4] ; умножаем значение в ST0 на 4 элемент массива \*4 означает, что элемент соотвествует двойному слову

FST znach1[6\*4] ; помещаем результат в 6 элемент массива znach1

FFREE ST(0) ; очистка регистра стека

;расчет ТРЕТЬЕГО элемента со всеми остальными

FLD arr\_k[2\*4] ; помещаем в ST0 2 элемент массива \*4 означает, что элемент соотвествует двойному слову

FMUL arr\_k[3\*4] ; умножаем значение в ST0 на 4 элемент массива \*4 означает, что элемент соотвествует двойному слову

FST znach1[7\*4] ; помещаем результат в 7 элемент массива znach1

FFREE ST(0) ; очистка регистра стека

FLD arr\_k[2\*4] ; помещаем в ST0 2 элемент массива \*4 означает, что элемент соотвествует двойному слову

FMUL arr\_k[4\*4] ; умножаем значение в ST0 на 4 элемент массива \*4 означает, что элемент соотвествует двойному слову

FST znach1[8\*4] ; помещаем результат в 8 элемент массива znach1

FFREE ST(0) ; очистка регистра стека

;расчет ЧЕТВЕРТОГО элемента со всеми остальными

FLD arr\_k[3\*4] ; помещаем в ST0 3 элемент массива \*4 означает, что элемент соотвествует двойному слову

FMUL arr\_k[4\*4] ; умножаем значение в ST0 на 4 элемент массива \*4 означает, что элемент соотвествует двойному слову

FST znach1[9\*4] ; помещаем результат в 9 элемент массива znach1

FFREE ST(0) ; очистка регистра стека

mov ecx, 10d ; устанавливаем количество итераций

mov ebx, 0d ; здесь регистр ebx отвечает за номера элементов массива

@cycle: ; начало цикла

FLD CHISLO

FLD znach1[ebx\*4]

FCOMI ST, ST(1) ; сравниваем два числа

je L1; если значение равные, то переходим к метке L1

jae L2; проверяем условие ST(0) >= ST(i) и идем к метке L2

jbe L2; проверяем условие L2 ST(0) <= ST(i) и идем к метке L2

L1:

FCHS ; меняем знак на противоположный

FST znach1[ebx\*4] ; записываем значение в элементы массива

jmp L3 ; переход к метке

L2:

FLD NULL

FST znach1[ebx\*4]

jmp L3

L3:

FFREE ST(0); очистка регистра стека

FFREE ST(1); очистка регистра стека

FFREE ST(2); очистка регистра стека

FFREE ST(3); очистка регистра стека

FFREE ST(4); очистка регистра стека

FFREE ST(5); очистка регистра стека

FFREE ST(6); очистка регистра стека

FFREE ST(7); очистка регистра стека

add ebx, 1d

loop @cycle ; конец цикла

FLD znach1[0\*4]

FADD znach1[1\*4]

FADD znach1[2\*4]

FADD znach1[3\*4]

FADD znach1[4\*4]

FADD znach1[5\*4]

FADD znach1[6\*4]

FADD znach1[7\*4]

FADD znach1[8\*4]

FADD znach1[9\*4]; количество перпендикулярных прямых в ST(0)

exit:

Invoke ExitProcess,1

End Start

**Расчет вручную:**

Для того чтобы выяснить, являются ли прямые перпендикулярными, необходимо перемножить их угловые коэффициенты и если произведение получилось равным -1, то прямые являются перпендикулярными. Для примера возьмем 5 угловых коэффициентов от 5 разных прямых и перемножим их между собой.

По расчетам видно, что получилось два значения -1 и, следовательно, всего перпендикулярных = 2.

**Расчет в программе:**



Рисунок 1 – Подсчет



Рисунок 2 – Подсчет



Рисунок 3 – Подсчет



Рисунок 4 – Подсчет



Рисунок 5 – Подсчет



Рисунок 6 – Подсчет



Рисунок 7 – Подсчет



Рисунок 8 – Подсчет



Рисунок 9 – Подсчет



Рисунок 10 – Подсчет

Затем программа считает следующим образом: если число -1, то программа переходит в метку L1 и там меняет знак на противоположный и помещает число обратно в массив znach1, а если число не равно -1, то переходим в метку L2 и там происходит обнуление элемента массива и его запись обратно в массив zhach1 (все подсчитанные элементы хранятся в массиве znach1). После данных операций, программа складывает все значения массива и выводит итоговое количество перпендикулярных прямых.

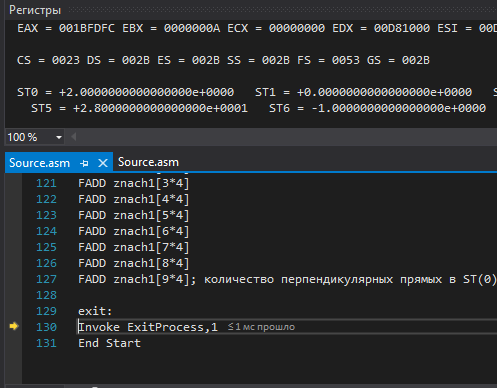


Рисунок 11 – Итоговое количество перпендикулярных прямых в ST0

**Верификация программы:**

Возьмем новые значения и произведем необходимые расчеты.

По расчетам видно, что все значения не равны -1 и, следовательно, всего перпендикулярных = 0. На рисунке 22 получилось такое же значение => программа работает корректно.



Рисунок 12 – Подсчет



Рисунок 13 – Подсчет



Рисунок 14 – Подсчет



Рисунок 15 – Подсчет



Рисунок 16 – Подсчет



Рисунок 17 – Подсчет



Рисунок 18 – Подсчет



Рисунок 19 – Подсчет



Рисунок 20 – Подсчет



Рисунок 21 – Подсчет

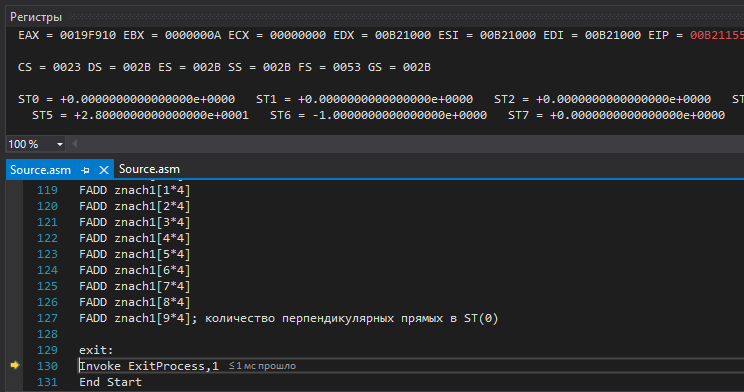
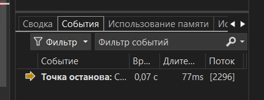


Рисунок 22 – Итоговое количество перпендикулярных прямых в ST0

**Вывод:** в ходе данной лабораторной работы познакомились с командами математического сопроцессора, служебными регистрами swr, cwr, twr, а также способом задания циклов и ветвлений в сопроцессоре, провели верификацию, и программа показала верный результат.



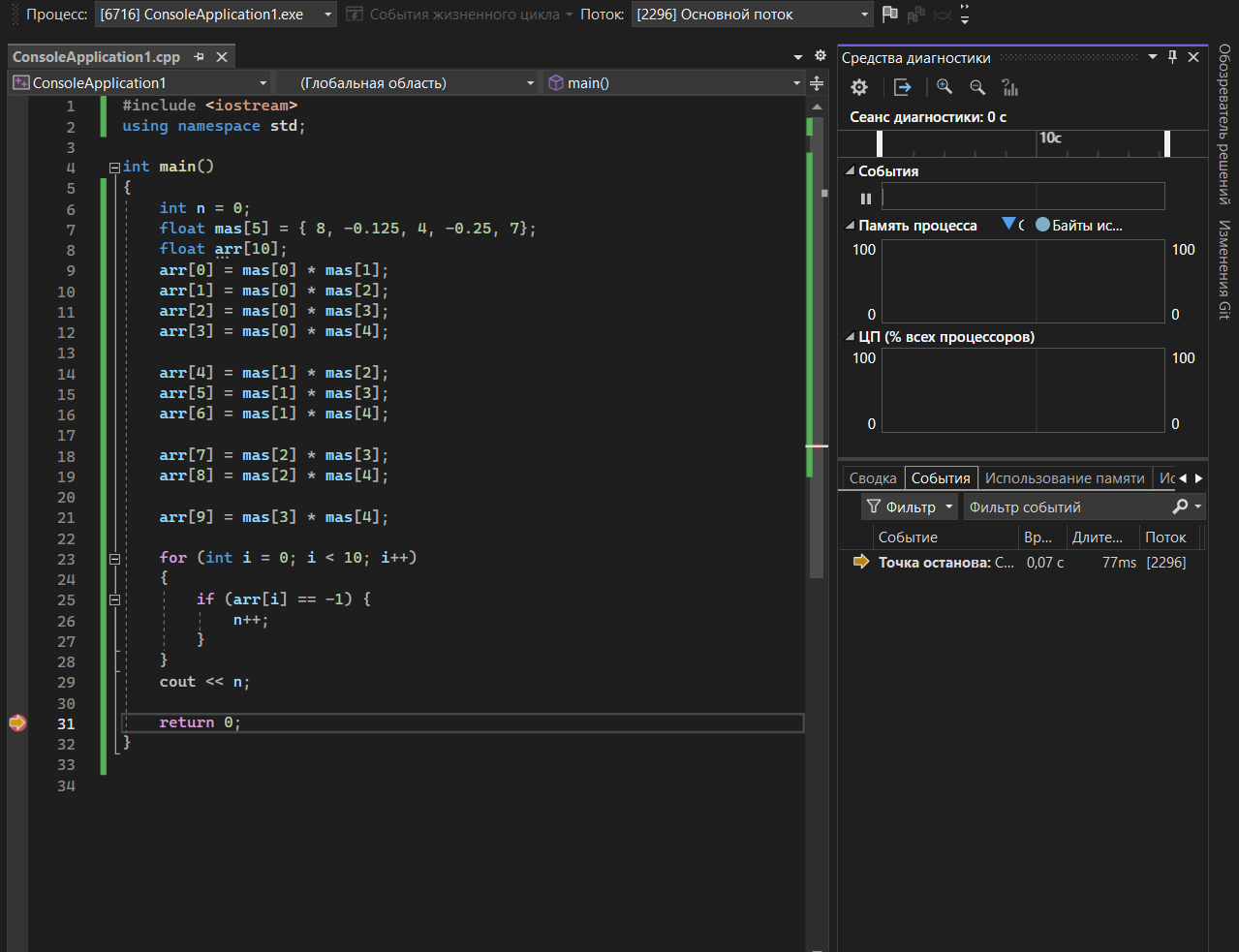
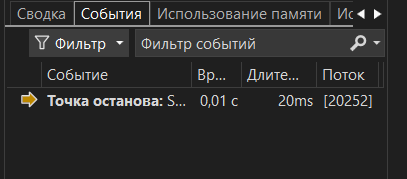


Рисунок 23 – Время выполнения программы, написанной на С++



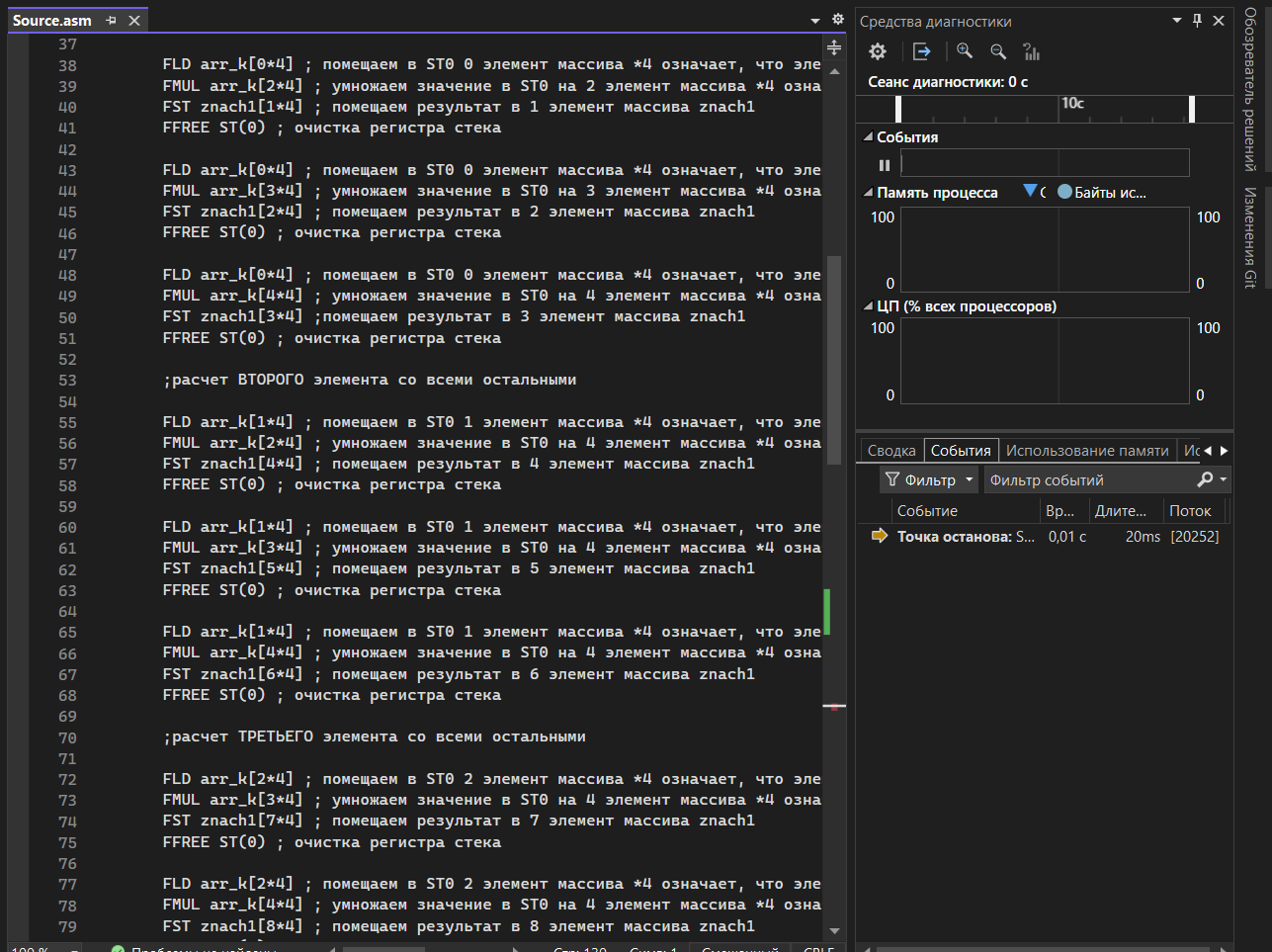


Рисунок 24 – Время выполнения программы, написанной на Ассемблере

Скорость выполнения программы на языке Ассемблере выше по сравнению с написанным на С++. Это связано с тем, что код на Ассемблере легче, чем код С++, т.к. там есть дополнительные операторы. Чем больше команд математического сопроцессора в коде, тем он более производителен.

Код C++ менее производительный в 3.85 раз, чем код, написанный на ассемблере, потому что компилятор вставляет дополнительные команды. Для повышения производительности кода на ассемблере нужно использовать как можно больше команд математического сопроцессора.