Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра Информатики

Дисциплина «Архитектура вычислительных систем»

**ОТЧЕТ**

к практической работе №1

на тему:

**«ТЕХНОЛОГИЯ ММХ»**

БГУИР 6-05-0612-02 005

|  |
| --- |
| Выполнила студентка группы 353504  АНТОНОВА Лидия Сергеевна |
|  |
| (дата, подпись студента) |
| Проверил ассистент каф. Информатики  КАЛИНОВСКАЯ Анастасия Александровна |
|  |
| (дата, подпись преподавателя) |

Минск 2025

# 1 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Обработать массивы из 8 элементов по следующему выражению (в зависимости от варианта). Полученный результат отобразить на форме с использованием соответствующих элементов. При распаковке знаковых чисел совместно с командами распаковки использовать команды сравнения (сравнивать с нулём перед распаковкой).

**Вариант 1.**

F[i] = (A[i] + B[i]) \* C[i] + D[i], i = 1...8;

A, B и С – 8 разрядные целые знаковые числа (\_int8);

D – 16 разрядные целые знаковые числа (\_int16).

# 2 ВЫПОЛНЕНИЕ РАБОТЫ

Для выполнения задания была разработана программа на C++ с использованием встроенного ассемблера, ориентированного на архитектуру x86. Основная задача программы — вычисление выражения вида F[i] = (A[i] + B[i]) \* C[i] + D[i], реализована с применением MMX-инструкций, что позволяет эффективно обрабатывать целочисленные массивы в параллельном режиме. Ассемблерный фрагмент осуществляет операции сложения, знаковой распаковки и умножения, используя команды paddb, punpcklbw, psraw, pmullw и другие. Это демонстрирует преимущества SIMD-подхода при работе с данными шириной 8 и 16 бит, обеспечивая прирост производительности за счёт векторизации вычислений.

#include <iostream>

#include <cstdint>

void SolveF\_MMX(const int8\_t\* A, const int8\_t\* B, const int8\_t\* C, const int16\_t\* D, int16\_t\* F) {

asm volatile (

// A + B

"movq (%0), %%mm0;"

"movq (%1), %%mm1;"

"paddb %%mm1, %%mm0;"

"pxor %%mm7, %%mm7;"

"movq %%mm0, %%mm6;"

"pcmpgtb %%mm7, %%mm6;"

"pxor %%mm2, %%mm2;"

"movq %%mm0, %%mm3;"

"punpcklbw %%mm0, %%mm2;"

"movq %%mm2, %%mm0;"

"punpckhbw %%mm3, %%mm2;"

"movq %%mm2, %%mm3;"

"psraw $8, %%mm0;"

"psraw $8, %%mm3;"

// Загрузка C и сравнение с нулём

"movq (%2), %%mm4;"

"movq %%mm4, %%mm5;"

"pxor %%mm7, %%mm7;"

"movq %%mm4, %%mm6;"

"pcmpgtb %%mm7, %%mm6;"

"pxor %%mm2, %%mm2;"

"punpcklbw %%mm4, %%mm2;"

"movq %%mm2, %%mm4;"

"punpckhbw %%mm5, %%mm2;"

"movq %%mm2, %%mm5;"

"psraw $8, %%mm4;"

"psraw $8, %%mm5;"

// Умножение: (A + B) \* C

"pmullw %%mm4, %%mm0;"

"pmullw %%mm5, %%mm3;"

// Загрузка D и прибавление

"movq (%3), %%mm6;"

"paddw %%mm6, %%mm0;"

"movq 8(%3), %%mm6;"

"paddw %%mm6, %%mm3;"

// Сохранение результата

"movq %%mm0, (%4);"

"movq %%mm3, 8(%4);"

"emms;"

:

: "r"(A), "r"(B), "r"(C), "r"(D), "r"(F)

: "mm0", "mm1", "mm2", "mm3", "mm4", "mm5", "mm6", "mm7"

);

}

int main() {

int8\_t A[8] = {12, -5, 7, 0, 23, -18, 4, 9};

int8\_t B[8] = {-3, 8, 15, -7, 2, 0, -12, 6};

int8\_t C[8] = {1, 1, 2, 3, 5, -8, 13, -21};

int16\_t D[8] = {100, 200, -150, 0, 327, -38, 42, -42};

int16\_t F[8];

SolveF\_MMX(A, B, C, D, F);

std::cout << " A(i) | B(i) | C(i) | D(i) | F(i) " << std::endl;

std::cout << "-------------------------------------------------" << std::endl;

for (int i = 0; i < 8; i++) {

std::cout << static\_cast<int>(A[i]) << " | "

<< static\_cast<int>(B[i]) << " | "

<< static\_cast<int>(C[i]) << " | "

<< D[i] << " | "

<< F[i] << "\n";

}

return 0;

}

# 4 АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

Для анализа полученных результатов запустим программный продукт с заданными заранее значениями A, B, C и D. На рисунке 4.1 представлены результаты запуска программного продукта.

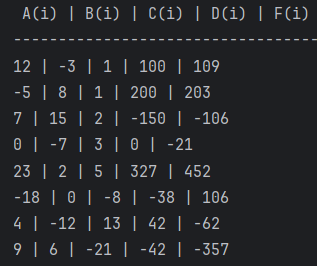


Рисунок 4.1 – Вывод программного продукта

# ВЫВОД

В ходе выполнения работы удалось реализовать эффективную обработку целочисленных массивов с использованием технологии MMX, что обеспечило заметное ускорение вычислений. Это позволило на практике продемонстрировать, как ассемблерные оптимизации могут применяться для решения задач низкоуровневой арифметики, повышая производительность и раскрывая потенциал SIMD-инструкций.