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 лабораторной работе №1
на тему:
«ТЕХНОЛОГИИ MMX, SIMD»
БГУИР 6-05-0612-02 67

Выполнил студент группы 353503
КОХАН Артём Игоревич

(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 F[i]=(A[i]-B[i])*(C[i]+D[i]), i=1...8. Вариант соответствует порядковому номеру в журнале группы. Используются следующие массивы: A, B и C – 8 разрядные целые знаковые числа (_int8); D – 16 разрядные целые знаковые числа (_int16). 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).

2 ВЫПОЛНЕНИЕ РАБОТЫ

Для выполнения задания был написан код на языке C++ с ассемблерными вставками. Код использует MMX-технологию – специальный набор инструкций для параллельной обработки данных (SIMD). MMX-технология используется для параллельной обработки 4 элементов одновременно (для 16-битных значений), что ускоряет вычисления. Для корректной распаковки 8-битных чисел в 16-битные используется сравнение с нулем (pcmpgtb) для определения знака и последующая распаковка с расширением знака. Используются регистры mm0-mm7 для хранения промежуточных результатов и выполнения операций. Команда emms освобождает состояние MMX в конце вычислений, что необходимо для корректной работы FPU-инструкций.

```
#include <iostream>
#include <cstdint>

int8_t A[8] = {1, -2, 3, -4, 5, -6, 7, -8};
int8_t B[8] = {8, -7, 6, -5, 4, -3, 2, -1};
int8_t C[8] = {2, -3, 4, -5, 6, -7, 8, -9};
int16_t D[8] = {10, -20, 30, -40, 50, -60, 70, -80};

int16_t F[8] = {0};

int main()
{
    asm volatile(
        // === (A[i] - B[i]) ===

        // Load A and B to registers
        "movq %[A], %%mm0\n\t" // mm0 = A[7]...A[0]
        "movq %[B], %%mm1\n\t" // mm1 = B[7]...B[0]

        // Sign extension for A
        "pxor %%mm7, %%mm7\n\t"    // mm7 = 0
        "pcmpgtb %%mm0, %%mm7\n\t" // mm7 = sign mask for A
        "movq %%mm0, %%mm2\n\t"
        "punpcklbw %%mm7, %%mm2\n\t" // Unpack lower 4 bytes of A
        "movq %%mm0, %%mm3\n\t"
        "punpckhbw %%mm7, %%mm3\n\t" // Unpack upper 4 bytes of A

        // Sign extension for B
        "pxor %%mm7, %%mm7\n\t"
        "pcmpgtb %%mm1, %%mm7\n\t" // mm7 = sign mask for B
        "movq %%mm1, %%mm4\n\t"
        "punpcklbw %%mm7, %%mm4\n\t" // Unpack lower 4 bytes of B
        "movq %%mm1, %%mm5\n\t"
        "punpckhbw %%mm7, %%mm5\n\t" // Unpack upper 4 bytes of B
```

```
            // A - B
            "psubw %%mm4, %%mm2\n\t" // Lower halves: A[0-3] - B[0-3]
            "psubw %%mm5, %%mm3\n\t" // Upper halves: A[4-7] - B[4-7]

            // === (C[i] + D[i]) ===

            // Load C and sign extension
            "movq %[C], %%mm0\n\t"
            "pxor %%mm7, %%mm7\n\t"
            "pcmpgtb %%mm0, %%mm7\n\t" // mm7 = sign mask for C
            "movq %%mm0, %%mm4\n\t"
            "punpcklbw %%mm7, %%mm4\n\t" // Unpack lower 4 bytes of C
            "movq %%mm0, %%mm5\n\t"
            "punpckhbw %%mm7, %%mm5\n\t" // Unpack upper 4 bytes of C

            // Load D (16-bit values)
            "movq %[D], %%mm0\n\t"  // First 4 elements of D
            "movq %[D8], %%mm1\n\t" // Second 4 elements of D

            // C + D
            "paddw %%mm0, %%mm4\n\t" // Lower halves: C[0-3] + D[0-3]
            "paddw %%mm1, %%mm5\n\t" // Upper halves: C[4-7] + D[4-7]

            // === (A-B) * (C+D) ===
            "pmullw %%mm4, %%mm2\n\t" // (A-B)[0-3] * (C+D)[0-3]
            "pmullw %%mm5, %%mm3\n\t" // (A-B)[4-7] * (C+D)[4-7]

            // Save results
            "movq %%mm2, %[F]\n\t"  // First 4 result elements
            "movq %%mm3, %[F8]\n\t" // Second 4 result elements

            // Clear MMX state
            "emms\n\t"

            : [F] "=m"(F), [F8] "=m"(F[4])
            : [A] "m"(A), [B] "m"(B), [C] "m"(C), [D] "m"(D), [D8] "m"(D[4])
            :  "mm0",  "mm1",  "mm2",  "mm3",  "mm4",  "mm5",  "mm6",  "mm7",
"memory");

        std::cout << "Results:\n";
        for (int i = 0; i < 8; i++)
        {
            std::cout << "F[" << i << "] = " << F[i] << std::endl;
        }

        // Manual verification
        std::cout << "\nManual verification:\n";
        for (int i = 0; i < 8; i++)
        {
            int16_t manual = (A[i] - B[i]) * (C[i] + D[i]);
            std::cout << "F[" << i << "] = " << manual
                      << " (expected: " << manual << ", actual: " << F[i]
                      << ")" << (manual == F[i] ? " ✓" : " ✗") << std::endl;
```

```
    }

    return 0;
}
```

На основе заданных значений на рисунке 1.

```
{1, -2, 3, -4, 5, -6, 7, -8};
{8, -7, 6, -5, 4, -3, 2, -1};
{2, -3, 4, -5, 6, -7, 8, -9};
{10, -20, 30, -40, 50, -60, 70, -80};
```

Рисунок 1 – Массивы данных

Получим желаемый результат запустив код. Результат изображён на рисунке 2.

```
[koxan@victusMonjaro build]$ ./LAB1
Results:
F[0] = -84
F[1] = -115
F[2] = -102
F[3] = -45
F[4] = 56
F[5] = 201
F[6] = 390
F[7] = 623
```

Рисунок 2 – Результат работы программы

ВЫВОД

В ходе выполнения практической работы была реализована интеграция ассемблерных вставок в проект на C++ с использованием MMX-инструкций для параллельной обработки данных (SIMD). Данная реализация демонстрирует эффективное использование MMX-инструкций для векторных вычислений с целыми числами, обеспечивая параллельную обработку нескольких элементов данных за одну операцию.