Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра ЭВМ

Лабораторная работа № 2
« Команды MMX/XMM»
Вариант 28

Проверил:
Одинец Д.Н.
Выполнил: ст.гр. 950501
Яценко С. С.

Минск, 2021

1. Постановка задачи

Создать приложение, которое выполняет заданные вычисления (в соответствии с вариантом) тремя способами:

1) с использованием команд MMX
2) на ассемблере, без использования команд MMX
3) на языке Си

После вычислений должны быть выведены время выполнения и результат для каждого случая. Значения элементов матриц генерируются приложением (не вводятся с клавиатуры). Вычисления производятся многократно (например 1 млн раз). Размер матриц (векторов) кратен количеству элементов в регистре MMX.

2. Алгоритм решения задачи

1. Взять время до выполнения цикла
2. Генерация первой случайной матрицы 8x8
3. Генерация второй случайной матрицы 8x8
4. Вычисление минимума матриц
5. Повторить действия начиная с пункта 4 10.000.000 раз
6. Взять время после выполнения цикла
7. Вывести результат
8. Вывести разность времен после и до выполнения цикла

3 Листинг кода программы

3.1 Листинг программы на C++

```
#include <stdio.h>
#include <ctime>
#include <stdlib.h>
#include <iostream>
#include <windows.h>
#include <iomanip>

using namespace std;

#define matrixSize  8

#define actionTimes 10000000

void fillMatrix(uint8_t * *matrix, uint8_t size) {
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			matrix[i][j] = rand() % 100;
		}
	}
}

void fillMatrix(float** matrix, uint8_t size) {
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			matrix[i][j] =( rand() % 100) / 10.;
		}
	}
}

void printMatrix(uint8_t** matrix, uint8_t size) {
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			cout << std::setw(3) << (int)matrix[i][j];
		}
		cout << endl;
	}
}

void printMatrix(float** matrix, uint8_t size) {
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			cout << std::setw(4) << matrix[i][j];
		}
		cout << endl;
	}
}


uint8_t** initMatrix(uint8_t size) {
	uint8_t** matrix = (uint8_t * *)malloc(size * sizeof(uint8_t*));
	for (int i = 0; i < size; i++) matrix[i] = (uint8_t*)malloc(size *
sizeof(uint8_t));
	fillMatrix(matrix, size);
	return matrix;
}
float** initMatrixF(uint8_t size) {
	float** matrix = (float**)malloc(size * sizeof(float*));
	for (int i = 0; i < size; i++) matrix[i] = (float*)malloc(size * sizeof(float));
	fillMatrix(matrix, size);
```

```
	return matrix;
}

uint8_t** minMatrixAsm(uint8_t** m1, uint8_t** m2, uint8_t** result, uint8_t size) {

	__asm {
		mov ebx, result
		mov esi, m1
		mov edi, m2
		mov cx, matrixSize
		outerLoop :
		push cx
			push esi
			push edi
			push ebx
			mov ebx, [ebx]
			mov esi, [esi]
			mov edi, [edi]
			mov cx, matrixSize
			a :
		mov al, [esi]
			cmp al, [edi]
			jb less1
			mov al, [edi]
			less1 :
			mov[ebx], al

			add ebx, 1
			add esi, 1
			add edi, 1
			loop a
			pop ebx
			pop edi
			pop esi
			pop cx
			add ebx, 4
			add esi, 4
			add edi, 4
			loop outerLoop
	}
	return result;
}




uint8_t** minMatrixC(uint8_t** m1, uint8_t** m2, uint8_t** result, uint8_t size) {
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			result[i][j] = min(m1[i][j], m2[i][j]);
		}
	}
	return result;
}



uint8_t** cmpMatrixMmx(uint8_t** m1, uint8_t** m2, uint8_t** result, uint8_t size) {
	__asm {
		mov ebx, result
		mov esi, m1
		mov edi, m2
		mov cx, matrixSize
```

```
		outerLoop :

		push esi
			push edi
			push ebx
			mov ebx, [ebx]
			mov esi, [esi]
			mov edi, [edi]

			movq mm1, [edi]
			pminub mm1, [esi]
			movq[ebx], mm1

			emms
			pop ebx
			pop edi
			pop esi
			add ebx, 4
			add esi, 4
			add edi, 4
			loop outerLoop
	}
	return result;
}


float** addMatrixSse(float** m1, float** m2, float** result, uint8_t size) {
	__asm {
		mov ebx, result
		mov esi, m1
		mov edi, m2
		mov cx, matrixSize
		outerLoop :
		push cx
			push esi
			push edi
			push ebx
			mov ebx, [ebx]
			mov esi, [esi]
			mov edi, [edi]
			mov cx, 2
			a:
			movlps xmm0, [esi]
			movhps xmm0, [esi + 8]

			 movlps xmm1, [edi]
			 movhps xmm1, [edi + 8]

			minps xmm0, xmm1

			movlps[ebx], xmm0
			movhps[ebx + 8], xmm0


			add ebx, 16
			add esi, 16
			add edi, 16
			loop a

			pop ebx
			pop edi
			pop esi
			pop cx
			add ebx, 4
			add esi, 4
```

```
                        add edi, 4
                        loop outerLoop
        }
        return result;
}

void freeMatrix(uint8_t** matrix) {

        for (int i = 0; i < matrixSize; i++)
                free(matrix[i]);
        free(matrix);
}

int main(void)
{
        srand(time(0));

        cout << fixed<< std::setprecision(1);
        uint8_t** matrix1 = initMatrix(matrixSize);
        uint8_t** matrix2 = initMatrix(matrixSize);
        uint8_t** result = initMatrix(matrixSize);


        printMatrix(matrix1, matrixSize);
        cout << endl;
        printMatrix(matrix2, matrixSize);
        cout << endl;

        uint16_t start_time_c = clock();

        for (int i = 0; i < actionTimes; i++) {
                minMatrixC(matrix1, matrix2, result, matrixSize);
        }
        uint16_t end_time_c = clock();
        printMatrix(result, matrixSize);
        uint16_t result_time_c = end_time_c - start_time_c;
        cout << "C: " << result_time_c << " miliseconds" << endl;
        cout << "-----------------------------------------" << endl << endl;
        freeMatrix(result);
        result = initMatrix(matrixSize);

        uint16_t start_time_asm = clock();
        for (int i = 0; i < actionTimes; i++) {
                minMatrixAsm(matrix1, matrix2, result, matrixSize);
        }
        uint16_t end_time_asm = clock();
        printMatrix(result, matrixSize);
        uint16_t result_time_asm = end_time_asm - start_time_asm;
        cout << "ASM: " << result_time_asm << " miliseconds" << endl;
        cout << "-----------------------------------------" << endl << endl;


        freeMatrix(result);
        result = initMatrix(matrixSize);

        uint16_t start_time_mmx = clock();
        for (int i = 0; i < actionTimes; i++) {
                cmpMatrixMmx(matrix1, matrix2, result, matrixSize);
        }
        uint16_t end_time_mmx = clock();
        printMatrix(result, matrixSize);
        uint16_t result_time_mmx = end_time_mmx - start_time_mmx;
        cout << "MMX: " << result_time_mmx << " miliseconds" << endl;
        cout << "-----------------------------------------" << endl << endl;
```

```
	float** matrix3 = initMatrixF(matrixSize);
	float** matrix4 = initMatrixF(matrixSize);
	float** result2 = initMatrixF(matrixSize);


	printMatrix(matrix3, matrixSize);
	cout << endl;
	printMatrix(matrix4, matrixSize);
	cout << endl;




	uint16_t start_time_sse = clock();
	for (int i = 0; i < actionTimes; i++) {
		addMatrixSse(matrix3, matrix4, result2, matrixSize);
	}

	uint16_t end_time_sse = clock();
	printMatrix(result2, matrixSize);
	uint16_t result_time_sse = end_time_sse - start_time_sse;
	cout << "SSE: " << result_time_sse << " miliseconds" << endl;

}
```

4 Примеры работы программы

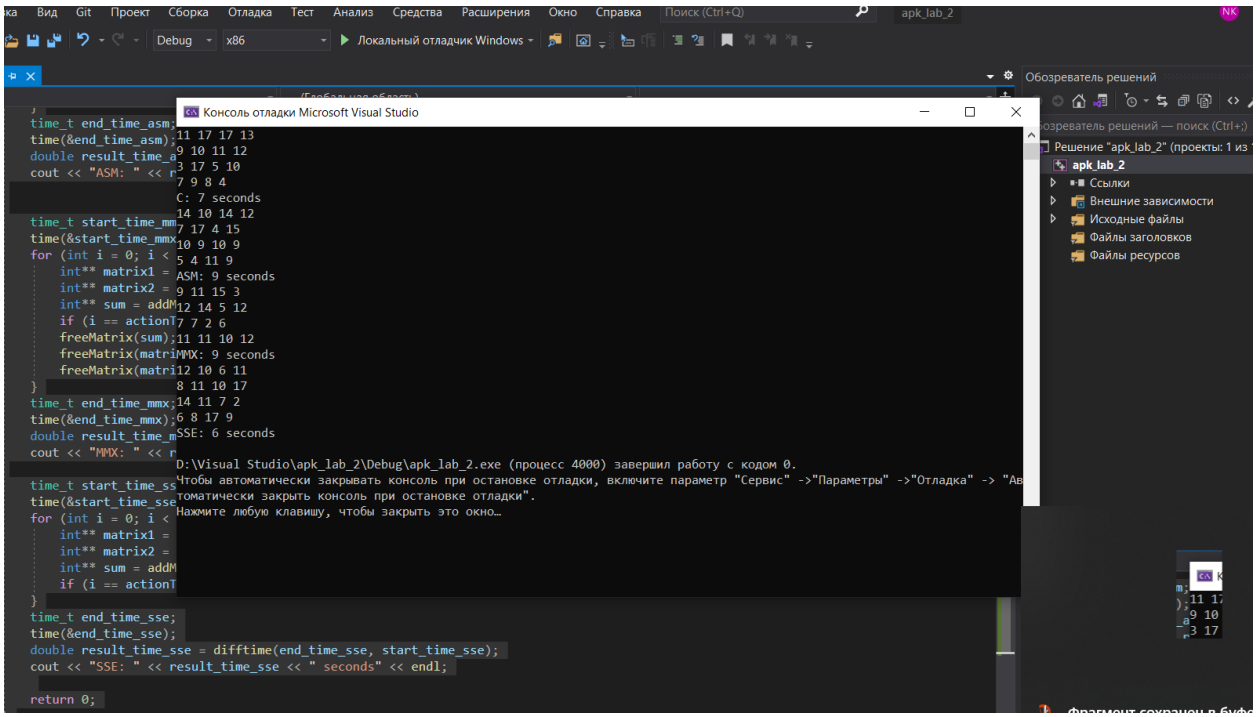

Рис.1 Программа на C++

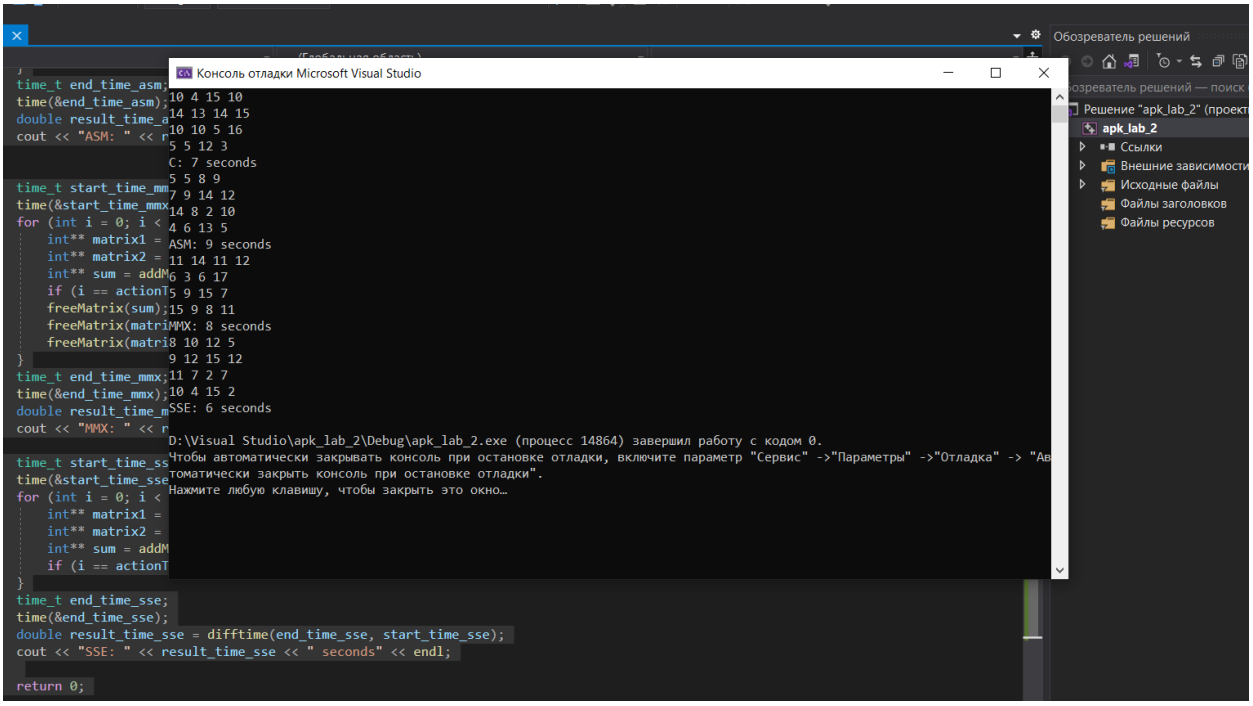

Рис.2 Программа на C++

5 Заключение

В ходе лабораторной работы мы познакомились с командами устройств MMX и SSE, а также изучили возможность языка C использовать ассемблерные вставки. В результате разработки программы были реализованы следующие задачи:

- Сложение матриц на языке C
- Сложение матриц на assembler, используя ассемблерную вставку
- Сложение матриц используя MMX и SSE

Хост ОС Windows 10 x64