|  |  |  |
| --- | --- | --- |
| Министерство образования Республики Беларусь  Учреждение образования  Белорусский Государственный Университет Информатики и Радиоэлектроники | | |
| Факультет компьютерных систем и сетей | | |
| Кафедра электронных вычислительных средств | | |
| Дисциплина: Программирование компьютеров | | |
| Отчет по лабораторной работе №4  «Технология MMX»  Вариант №8 | | |
| Выполнили:  студенты гр. 250701  Ермакович В.А.,  Зубков Н.В. |  | Проверил:  Порхун М. И. |
| Минск 2025 | | |

# Цель работы

Цели лабораторной работы:

1) ознакомиться с архитектурой и командами технологии MMX;

2) разработать ассемблерную вставку в высокоуровневый код для обработки данных при помощи команд MMX.

# Задание

## 2.1 Задание 1

В данном задании необходимо разработать ассемблерную вставку с использованием команд MMX, которая выполняет в соответствии с индивидуальным вариантом обработку массива. Элементы массива – числа типа int. Размер массива и его элементы должны вводиться с клавиатуры. Сравнить результаты работы с Си.

Найти количество нечётных элементов массива.

## 2.2 Задание 2

В данном задании необходимо разработать ассемблерную вставку с использованием команд MMX, которая выполняет в соответствии с индивидуальным вариантом обработку матриц. Количество строк матрицы и её элементы должны вводиться с клавиатуры. Количество столбцов и тип данных указаны в индивидуальном варианте задания. Сравнить результат работы ассемблерной вставки с результатом вычислений в Си.

Умножение матриц. Элементы матриц – 16-разрядные целые числа со знаком. Количество столбцов матриц – 4.

## 2.3 Дополнительные задания

1) Разработать программу для нахождения суммы элементов массива. Элементы массива – 16-разрядные целые числа со знаком (тип данных short). Обработку элементов массива выполнять параллельно (т.е. сразу по 4 16-разрядных слова);

2) Модифицировать программу из задания No2 для обработки матриц любых размеров (количество столбцов должно быть кратно размеру исходных данных);

3) Разработать ассемблерную вставку с использованием команд MMX, которая формирует негатив исходного изображения. Результат обработки сохранить в виде изображения в формате .BMP. Сравнить результат работы ассемблерной вставки с результатом вычислений в Си.

# Ход работы

В ходе выполнения лабораторной работы была составлена программа реализующие данные задания

Листинг программы приведён ниже:

#include <iostream>

#include <conio.h>

#include "img\_lib.h"

using namespace std;

void task1() {

int arr\_size;

cout << "Введите размер массива: ";

cin >> arr\_size;

int\* array = new int[arr\_size];

cout << "Введите элементы массива: ";

for (int i = 0; i < arr\_size; i++) {

cin >> array[i];

}

int odd\_count\_c = 0, odd\_count\_mmx = 0;

for (int i = 0; i < arr\_size; i++) {

if (array[i] % 2 != 0) odd\_count\_c++;

}

int one = 1;

\_\_asm {

mov esi, array

mov ecx, arr\_size

pxor mm1, mm1

check\_loop :

movd mm0, [esi]

movd eax, mm0

test eax, 1

jz skip\_inc

paddd mm1, [one]

skip\_inc :

add esi, 4

loop check\_loop

movd odd\_count\_mmx, mm1

emms

}

cout << "Кол-во нечётных элементов (C++): " << odd\_count\_c << endl;

cout << "Кол-во нечётных элементов (MMX): " << odd\_count\_mmx << endl;

delete[] array;

}

void task2() {

int row\_size = 1;

const int col\_size = 4;

cout << "Введите кол-во строк матрицы: ";

cin >> row\_size;

short\* matrix\_1 = new short[row\_size \* col\_size];

short\* matrix\_2 = new short[row\_size \* col\_size];

short\* matrix\_c = new short[row\_size \* col\_size];

short\* matrix\_MMX = new short[row\_size \* col\_size];

cout << "Введите элементы матрицы 1" << endl;

for (int i = 0; i < row\_size; i++)

for (int j = 0; j < col\_size; j++)

cin >> \*(matrix\_1 + i \* col\_size + j);

cout << "Введите элементы матрицы 2" << endl;

for (int i = 0; i < row\_size; i++)

for (int j = 0; j < col\_size; j++)

cin >> \*(matrix\_2 + i \* col\_size + j);

cout << "Матрица 1:" << endl;

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

cout << \*(matrix\_1 + i \* col\_size + j) << " ";

}

cout << endl;

}

cout << "Матрица 2:" << endl;

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

cout << \*(matrix\_2 + i \* col\_size + j) << " ";

}

cout << endl;

}

// Вычисление произведения матриц

for (int i = 0; i < row\_size; i++)

for (int j = 0; j < col\_size; j++)

\*(matrix\_c + i \* col\_size + j) = \*(matrix\_1 + i \* col\_size + j) \* \*(matrix\_2 + i \* col\_size + j);

cout << "Результат произведения матриц на СИ:" << endl;

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

cout << \*(matrix\_c + i \* col\_size + j) << " ";

}

cout << endl;

}

\_asm {

mov esi, matrix\_1

mov edi, matrix\_2

mov ebx, matrix\_MMX

mov ecx, row\_size

p0 :

movq mm0, [esi]

movq mm1, [edi]

pmullw mm0, mm1

movq[ebx], mm0

add esi, 8

add edi, 8

add ebx, 8

loop p0

}

cout << "Результат произведения матриц 2x2 при помощи MMX:" << endl;

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

cout << \*(matrix\_MMX + i \* col\_size + j) << " ";

}

cout << endl;

}

delete[] matrix\_1;

delete[] matrix\_2;

delete[] matrix\_c;

delete[] matrix\_MMX;

}

void dop1() {

const int ARR\_SIZE = 4 \* 5;

const int MASK = 0x000000FF;

int arr\_sum = 0;

short\* array = new short[ARR\_SIZE];

// get array from user

std::cout << " Input " << ARR\_SIZE << " elements :" << std::endl;

for (short i = 0; i < ARR\_SIZE; i++)

std::cin >> array[i];

// print entered array

std::cout << " Entered array : \n ";

for (short i = 0; i < ARR\_SIZE; i++)

std::cout << " array [" << i << "] = " << array[i] << std::endl;

\_asm{

xor esi , esi

mov esi , array // array -> esi

mov ecx , ARR\_SIZE // ecx = array size

xor ebx , ebx // tmp = 0

// calculate sum of 4 elements

lp :

movq mm0 ,[esi] // arr [ i ] -> mm 0

movq mm1 , mm0 // arr [ i ] -> mm 1

psrlq mm1 , 16 // mm 1 >> 2

paddd mm0 , mm1 // mm 0 = mm 0 + mm 1

movq mm1 , mm0 // mm 1 = mm 0

psrlq mm1 , 32 // mm 1 >> 4

pand mm0, MASK // use mask on mm 0

pand mm1, MASK // use mask on mm 1

paddd mm0, mm1 // find final sum

xor eax, eax // accumulator = 0

movd eax, mm0 // accumulator = sum of 4 elements

add arr\_sum, eax

// move to next 4 numbers

add esi, 8

sub ecx, 4

jnz lp

emms

}

std::cout << " Number of even elements is : " << arr\_sum << std::endl;

}

void dop2() {

int row\_size = 1;

int col\_size = 4;

cout << "Введите кол-во строк матрицы: ";

cin >> row\_size;

cout << "Введите кол-во строк матрицы: ";

cin >> col\_size;

int matrix\_size = row\_size \* col\_size;

short\* matrix\_1 = new short[row\_size \* col\_size];

short\* matrix\_2 = new short[row\_size \* col\_size];

short\* matrix\_c = new short[row\_size \* col\_size];

short\* matrix\_MMX = new short[row\_size \* col\_size];

cout << "Введите элементы матрицы 1" << endl;

for (int i = 0; i < row\_size; i++)

for (int j = 0; j < col\_size; j++)

cin >> \*(matrix\_1 + i \* col\_size + j);

cout << "Введите элементы матрицы 2" << endl;

for (int i = 0; i < row\_size; i++)

for (int j = 0; j < col\_size; j++)

cin >> \*(matrix\_2 + i \* col\_size + j);

cout << "Матрица 1:" << endl;

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

cout << \*(matrix\_1 + i \* col\_size + j) << " ";

}

cout << endl;

}

cout << "Матрица 2:" << endl;

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

cout << \*(matrix\_2 + i \* col\_size + j) << " ";

}

cout << endl;

}

for (int i = 0; i < row\_size; i++)

for (int j = 0; j < col\_size; j++)

\*(matrix\_c + i \* col\_size + j) = \*(matrix\_1 + i \* col\_size + j) \* \*(matrix\_2 + i \* col\_size + j);

cout << "Результат произведения матриц на СИ:" << endl;

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

cout << \*(matrix\_c + i \* col\_size + j) << " ";

}

cout << endl;

}

\_asm{

mov esi, matrix\_1

mov edi, matrix\_2

mov ebx, matrix\_MMX

mov ecx, matrix\_size

lp :

movq mm0 ,[esi]

movq mm1 ,[edi]

pmullw mm0 , mm1

movq[ebx] , mm0

add esi, 8

add edi, 8

add ebx, 8

sub ecx, 4

jnz lp

}

cout << "Результат произведения матриц 2x2 при помощи MMX:" << endl;

for (int i = 0; i < row\_size; i++) {

for (int j = 0; j < col\_size; j++) {

cout << \*(matrix\_MMX + i \* col\_size + j) << " ";

}

cout << endl;

}

delete[] matrix\_1;

delete[] matrix\_2;

delete[] matrix\_c;

delete[] matrix\_MMX;

}

void dop3() {

char src\_file[] = "lena.bmp";

char proc\_file[] = "lena\_C.bmp";

char proc\_file\_mmx[] = "lena\_MMX.bmp";

const unsigned int MASK = 0xFFFFFFFF;

unsigned \_\_int8 \* src\_img;

unsigned \_\_int8 color\_map[1024];

BmpFileHeader im\_header;

BmpInfoHeader im\_info;

src\_img = imread(src\_file, im\_info, im\_header, color\_map);

const unsigned int IMG\_SIZE = im\_info.width \* im\_info.height;

\_asm{

mov esi , src\_img

mov ecx , IMG\_SIZE

lp :

movd mm0 , MASK

movd mm1 ,[esi]

psubd mm0 , mm1

movd[esi] , mm0

add esi , 4

sub ecx , 4

jnz lp

}

imwrite(proc\_file, src\_img, im\_info, im\_header, color\_map);

cout << " C ++ processing is done ." << endl;

cout << " width : " << im\_info.width << "\n";

cout << " height : " << im\_info.height << "\n";

delete[] src\_img;

}

int main() {

setlocale(LC\_ALL, "rus");

dop3();

}

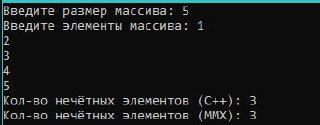


Рисунок 1 – Результат 1 задания

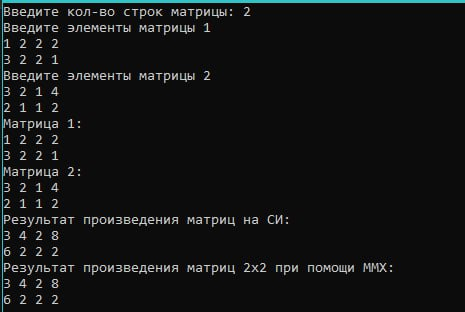


Рисунок 2 – Результат 2 задания

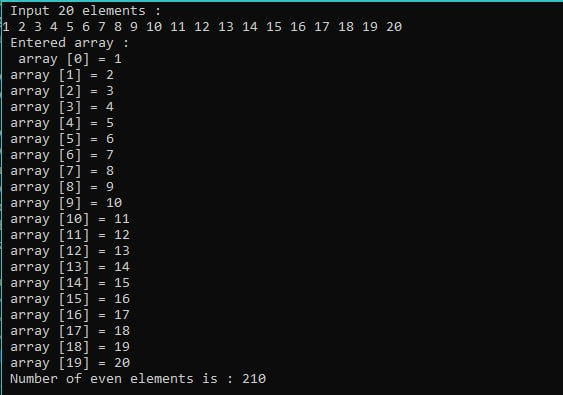


Рисунок 3 – Результат выполнения 1 дополнительного задания

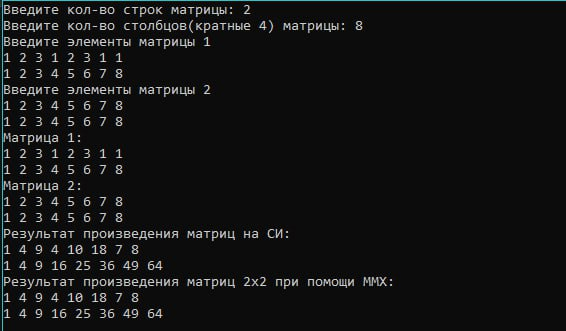


Рисунок 4 – Результат выполнения 2 дополнительного задания



Рисунок 5 – Исходное изображение в 3 долнительном задании



Рисунок 6 – Результат выполнения 3 дополнительного задания

# Вывод

В ходе выполнения данной лабораторной работы мы ознакомились с архитектурой и командами технологии MMX, а также разработали ассемблерные вставки в высокоуровневый код для обработки данных при помощи команд MMX.