Министерство образования Республики Беларусь

Учреждение образования Белорусский государственный университет

информатики и радиоэлектроники

# Кафедра ЭВМ

### Отчет по лабораторной работе № 2

"Команды MMX/XMM"

Вариант 7

Выполнил:

студент группы 950502

Киреев Ю.В.

Проверил:

Одинец Д. Н.

#### Минск 2021

1. **Задание**

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 ММХ.

Вариант 6: нахождение суммы всех элементов матрицы.

1. **Теоретические сведения**

MMX (MultimediaExtensions- мультимедийное расширение) — коммерческое название дополнительного набора инструкций, выполняющих характерные для процессов кодирования/декодирования потоковых аудио/видео данных действия за одну машинную инструкцию. Впервые появился в процессорах Pentium MMX. Разработан в лаборатории Intel, в первой половине 1990-х. SIMD (англ.SingleInstruction, MultipleData) - принцип компьютерных вычислений, позволяющий обеспечить параллелизм на уровне данных. Для обработки данных и хранения промежуточных результатов в Pentium MMX используются восемь 64-разрядных регистров MM0..MM7, которые физически совмещены со стеком регистров математического сопроцессора. При выполнении любой из MMX-команд происходит установка «режима MMX» с отметкой этого в слове состояния сопроцессора (FPU TagWord). С этого момента стек регистров сопроцессора рассматривается как набор MMX-регистров; завершает работу в режиме MMX команда EMMS (EndMultiMediaState). С одной стороны, такая реализация позволила избежать проблем совместимости с механизмами переключения контекста в существующих операционных системах, поскольку число регистров процессора, и, следовательно, код, выполняющий их сохранение и восстановление, не изменились. С другой стороны, переход между режимами занимает значительное время, и совмещение, например, в одном цикле команд сопроцессора с командами MMX может не только не ускорить, а даже существенно замедлить выполнение программы. Поэтому при необходимости для достижения наилучших результатов рекомендуется группировать эти команды отдельно друг от друга.

Команды технологии MMX работают с 64-разрядными целочисленными данными, а также с данными, упакованными в группы (векторы) общей длиной 64 бита. Такие данные могут находиться в памяти или в восьми MMX-регистрах. Команды технологии MMX работают со следующими типами данных:

* упакованные байты (восемь байтов в одном 64-разрядном регистре)

(англ. packedbyte);

* упакованные слова (четыре 16-разрядных слова в 64-разрядном регистре)

(packedword);

* упакованные двойные слова (два 32-разрядных слова в 64-разрядном регистре)

(packeddoubleword);

* 64-разрядные слова (quadword).

MMX-команды имеют следующий синтаксис: instruction dest, src. Здесь instruction - имя команды, dest - выходной операнд, src - входной операнд. Большинство команд имеют суффикс, который определяет тип данных:

* B, W, D, Q указывают тип данных. Если в суффиксе есть две из этих букв, первая соответствует входному операнду, а вторая — выходному.

SSE (англ. StreamingSIMDExtensions) - это SIMD набор инструкций, разработанный Intel. Технология SSE позволяла преодолеть проблему одновременного использования сопроцессора (его регистры использовались для MMX и работы с вещественными числами). SSE включает в архитектуру процессора восемь 128-битных регистров (xmm0 до xmm7), каждый из которых трактуется как 4 последовательных значения с плавающей точкой одинарной точности. SSE включает в себя набор инструкций, который производит операции со скалярными и упакованными типами данных.

Преимущество в производительности достигается в том случае, когда необходимопроизвести одну и ту же последовательность действий над разными данными.

1. **Листинг программы**
2. с использованием команд MMX, 2) без использования команд MMX:
3. используя язык С 4) используя команды ХММ

#include "stdio.h"

#include "windows.h"

#include "locale.h"

#define SIZE 4

int main() {

setlocale(LC\_ALL, "russian");

int ms1[SIZE][SIZE], ms2[SIZE][SIZE], ms3[SIZE][SIZE];

int time0, time1, time2, time3, time4;

long cnt = 16;

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

for (int j = 0; j < SIZE; j++) {

ms1[i][j] = i + j + 1;

}

}

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

for (int j = 0; j < SIZE; j++) {

ms2[i][j] = i + j + 2;

}

}

printf("Матрицы: \n");

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

for (int j = 0; j < SIZE; j++) {

printf("%d ", ms1[i][j]);

}

printf("\n");

}

printf("\n===============================\n");

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

for (int j = 0; j < SIZE; j++) {

printf("%d ", ms2[i][j]);

}

printf("\n");

}

time0 = GetTickCount();

printf("Time = %d ms\n", time0);

printf("\nC:\n");

for (int i = 0; i < 10000000; i++)

{

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

for (int j = 0; j < SIZE; j++) {

ms3[i][j] = ms1[i][j] \* ms2[i][j];

}

}

}

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

for (int j = 0; j < SIZE; j++) {

printf("%d ", ms3[i][j]);

}

printf("\n");

}

printf("Матрицы умножены\n");

time1 = GetTickCount() - time0;

printf("Time = %d ms\n", time1);

int res = 0;

printf("\nMMX:\n");

for (int i = 0; i < 10000000; i++)

{

cnt = 16;

\_\_asm

{

push esi

mov ecx, [cnt]

lea esi, [ms1 + ecx \* 2]

neg ecx

pcmpeqd mm2, mm2

psrlw mm2, 15; mm2 = (1, 1, 1, 1)

pxor mm0, mm0

b :

movq mm1, [esi + ecx \* 2]

pmaddwd mm1, mm2

pmullw mm0, mm1

add ecx, 4

jnz b

movq mm1, mm0

psrlq mm1, 32

pmullw mm0, mm1

movd[res], mm0

emms

pop esi

}

}

printf("Матрицы умножены\n");

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

for (int j = 0; j < SIZE; j++) {

printf("%d ", ms3[i][j]);

}

printf("\n");

}

time2 = GetTickCount() - time0 - time1;

printf("Time = %d ms\n", time2);

int ms4[SIZE][SIZE];

printf("\nAssembly:\n");

for (int i = 0; i < 10000000; i++)

{

cnt = 16;

\_\_asm

{

push eax

push ecx

push esi

xor esi, esi

xor ecx, ecx

loop2:

movsx eax, ms3[esi]

// mov ms1[esi], eax

imul ecx, eax

add esi, 2

sub cnt, 1

jnz loop2

pop esi

pop ecx

pop eax

}

}

printf("Матрицы умножены\n");

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

for (int j = 0; j < SIZE; j++) {

printf("%d ", ms3[i][j]);

}

printf("\n");

}

time3 = GetTickCount() - time0 - time1 - time2;

printf("Time = %d ms\n", time3);

printf("\nXMM:\n");

for (int i = 0; i < 10000000; i++)

{

cnt = 16;

\_\_asm

{

push esi

mov ecx, [cnt]

lea esi, [ms1 + ecx \* 2]

neg ecx

pcmpeqd xmm2, xmm2

psrlw xmm2, 15; xmm2 = (1, 1, 1, 1)

pxor xmm0, xmm0

b1 :

movups xmm1, [esi + ecx \* 2]

pmaddwd xmm1, xmm2

pmullw xmm0, xmm1

add ecx, 4

jnz b1

movq xmm1, xmm0

psrlq xmm1, 32

pmullw xmm0, xmm1

movd[res], xmm0

emms

pop esi

}

}

printf("Матрицы умножены\n");

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

for (int j = 0; j < SIZE; j++) {

printf("%d ", ms3[i][j]);

}

printf("\n");

}

time4 = GetTickCount() - time0 - time1 - time2 - time3;

printf("Time = %d ms\n", time4);

return 0;

}

Результат выполнения:  


