БГУИР

Кафедра ЭВМ

Отчет по лабораторной работе №2

Тема: «Команды MMX/XMM»

Выполнил:

студент группы 750501 Новицкий А.Д.

Проверил:

к.т.н., доцент Одинец Д.Н.

Минск

2019

**СОДЕРЖАНИЕ**

1. Постановка задач………………………………………………………………..3

2. Алгоритм решения задачи…………………………………………………...4-5

3. Листинг программы…………………………………………………………..6-8

4. Тестовые примеры……………………………………………………………...9

5. Заключение…………………………………………………………………….10

**ПОСТАНОВКА ЗАДАЧ**

Создать консольное приложение, которое выполняет операцию побитового И для каждой пары соответствующих элементов 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 несколько (1 млн) раз. Размер матриц (векторов) кратен количеству элементов в регистре ММХ.

**АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ**

1. Объявить три матрицы: *A[8][8], B[8][8], C[8][8]*
2. Инициализировать значения матриц *A* и *B*
3. С помощью цикла выполнить операцию побитового И для соответствующих элементов матриц *A* и *B* на языке C и занести результат в матрицу *C*
4. Вывести результат
5. Сделать ассемблерную вставку, внутри которой:
6. Сохранить все регистры в стек
7. Установить значение счетчика *ECX* равное 10.000.000
8. Сохранить значение счетчика в стек
9. Обнулить значение регистра *ESI*
10. Установить значение счетчика *ECX* равное 8
11. Сохранить значение счетчика в стек
12. Обнулить значение регистра *EDI*
13. Установить значение счетчика *ECX* равное 8
14. Загрузить в регистр *AX* значение *A[ESI][EDI]*
15. Выполнить логическое И значения в регистре *AX* и значения *B[ESI][ESI]*
16. Занести результат в *C[ESI][EDI]*
17. Увеличить значение *EDI* на 2, так как в регистр *AX* считывается два байта
18. Если значение счетчика *ECX* не равно 0, то уменьшить его на единицу и прейти к шагу 9, в противном случае продолжить
19. Выгрузить значение регистра *ECX* из стека
20. Увеличить значение *ESI* на 2
21. Если значение счетчика *ECX* не равно 0, то уменьшить его на единицу и прейти к шагу 7, в противном случае продолжить
22. Выгрузить значение регистра *ECX* из стека
23. Если значение счетчика *ECX* не равно 0, то уменьшить его на единицу и прейти к шагу 3, в противном случае продолжить
24. Выгрузить значения всех регистров из стека
25. Вывести результат ассемблерных вычислений
26. Сделать ассемблерную вставку, внутри которой:
27. Сохранить все регистры в стек
28. Обнулить регистр MM0
29. Установить значение счетчика *ECX* равное 10.000.000
30. Сохранить значение счетчика в стек
31. Обнулить значение регистра *ESI*
32. Установить значение счетчика *ECX* равное 8
33. Сохранить значение счетчика в стек
34. Обнулить значение регистра *EDI*
35. Установить значение счетчика *ECX* равное 2
36. Загрузить в регистр *MM0* значения *A[ESI][EDI], A[ESI][EDI+2], A[ESI][EDI+4], A[ESI][EDI+6]*
37. Выполнить логическое И значений в регистре *MM0* и значений *B[ESI][EDI], B[ESI][EDI+2], B[ESI][EDI+4], B[ESI][EDI+6]*
38. Занести результат в *C[ESI][EDI], C[ESI][EDI+2], C[ESI][EDI+4], C[ESI][EDI+6]*
39. Увеличить значение *EDI* на 8, так как в регистр *MMO* считывается четыре значения по два байта
40. Если значение счетчика *ECX* не равно 0, то уменьшить его на единицу и прейти к шагу 10, в противном случае продолжить
41. Выгрузить значение регистра *ECX* из стека
42. Увеличить значение *ESI* на 2
43. Если значение счетчика *ECX* не равно 0, то уменьшить его на единицу и прейти к шагу 8, в противном случае продолжить
44. Выгрузить значение регистра *ECX* из стека
45. Если значение счетчика *ECX* не равно 0, то уменьшить его на единицу и прейти к шагу 3, в противном случае продолжить
46. Выгрузить значения всех регистров из стека
47. Вывести результат ассемблерных вычислений

**ЛИСТИНГ ПРОГРАММЫ**

//

// Created by Anjey Nov on 2/28/19.

// Copyright © 2019 Anjey Nov. All rights reserved.

//

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#include <conio.h>

#include <windows.h>

short C[8][8], A[8][8], B[8][8];

void andC(void);

void andASM(void);

void andMMX(void);

void outMatrix(void);

int main(int argc, char \*argv[]) {

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

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

A[i][j] = (i + j) \* 2;

B[i][j] = (i\*j) + 2;

}

}

andC();

andASM();

andMMX();

\_getch();

return 0;

}

void andC(void)

{

double t;

LARGE\_INTEGER timerFrequency, timerStart, timerStop;

QueryPerformanceFrequency(&timerFrequency);

QueryPerformanceCounter(&timerStart);

for (double k = 0; k < 10000000; k++) {

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

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

C[i][j] = A[i][j] & B[i][j];

}

}

}

QueryPerformanceCounter(&timerStop);

t = (double)(timerStop.QuadPart - timerStart.QuadPart) / (double)timerFrequency.QuadPart;

printf("\nTime C is %lf seconds.\n", t);

outMatrix();

return;

}

void andASM(void)

{

double t;

LARGE\_INTEGER timerFrequency, timerStart, timerStop;

QueryPerformanceFrequency(&timerFrequency);

QueryPerformanceCounter(&timerStart);

\_asm {

pusha xor ECX,ECX

xor EAX, EAX

mov ECX, 10000000

loop0:

push ECX

mov ECX, 08h

xor ESI, ESI loop1:

push ECX

mov ECX, 08h

xor EDI, EDI

loop2:

mov AX, A[ESI][EDI]

and AX, B[ESI][EDI]

mov C[ESI][EDI], AX

inc EDI

inc EDI

loop loop2

inc ESI

inc ESI

pop ECX

loop loop1

pop ECX

loop loop0

popa

}

QueryPerformanceCounter(&timerStop);

t = (double)(timerStop.QuadPart - timerStart.QuadPart) / (double)timerFrequency.QuadPart;

printf("\nTime .asm is %lf seconds.\n", t);

outMatrix();

return;

}

void andMMX(void)

{

double t;

LARGE\_INTEGER timerFrequency, timerStart, timerStop;

QueryPerformanceFrequency(&timerFrequency);

QueryPerformanceCounter(&timerStart);

\_asm {

pusha

xor ECX, ECX pxor MM0, MM0

mov ECX, 10000000

loop0:

push ECX

mov ECX, 08h

xor ESI, ESI

loop1:

push ECX

mov ECX, 02h

xor EDI, EDI

loop2 :

movd MM0, A[ESI][EDI]

pand MM0, B[ESI][EDI]

movd C[ESI][EDI], MM0

add EDI, 08h

loop loop2

inc ESI inc ESI pop ECX

loop loop1

pop ECX

loop loop0

popa

EMMS

}

QueryPerformanceCounter(&timerStop);

t = (double)(timerStop.QuadPart - timerStart.QuadPart) / (double)timerFrequency.QuadPart;

printf("\nTime .asm with MMX is %lf seconds.\n", t);

outMatrix();

return;

}

void outMatrix(void)

{

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

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

printf("%2hd ", C[i][j]);

}

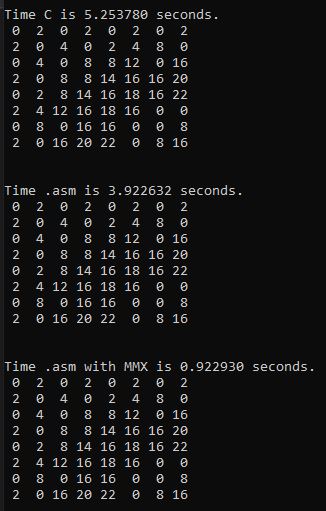
printf("\n");

}

printf("\n");

}

**ТЕСТОВЫЕ ПРИМЕРЫ**

****

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы были получены базовые навыки работы с расширение MMX.

Выяснил что расширение MMX основано на технологии SIMD (Single Instruction, Multiply Data), основная цель которой – добиться более

высокой производительности мультимедийных приложений и систем

обработки и передачи данных.

Основные направления применения SIMD технологий:

* Кодирование/декодирование и обработка сигналов
* Распознавание речи
* Обработка и захват видеосигналов
* 3D моделирование
* Промышленное проектирование (CAD)

Существует четыре типа данных с которыми работают команды MMX: упакованные байты (packed bytes) – один 64-битный операнд содержит 8 байт; упакованные слова (packed words) – один 64-битный операнд содержит 4 слова по 2 байта; упакованные двойные слова (packed double words) – один 64-битный операнд содержит 2 двойных слова по 4 байта; учетверенные слова (quad word) – один 64-битный операнд.

При завершении выполнения команд MMX необходимо использовать команду EMMS для перехода в арифметику с плавающей запятой.

Максимальное ускорение которого можно добиться – это ускорение в 8 раз из-за возможности обработки сразу 8 операндов.