АРХИТЕКТУРА ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

Лабораторная работа №2

КОМАНДЫ ММХ/ХММ

Подготовила Остроухова С.А.

К теме:

Технология MMX. Технология SSE. Регистры MMX/XMM, типы данных и команды MMX/XMM.

ЦЕЛЬ РАБОТЫ

Изучить расширение системы команд MMX процессоров Intel.

Продолжительность работы - 4 часа.

Основы MMX-технологии

MMX (Multimedia Extensions - мультимедийное расширение) - коммерческое название дополнительного набора инструкций, выполняющих характерные для процессов кодирования и декодирования потоковых аудио/видео данных действия за одну машинную инструкцию.

Разработан в лаборатории Intel, в первой половине 1990-х.

Основы MMX-технологии

SIMD
(Single Instruction, Multiple Data)
принцип компьютерных вычислений, позволяющий обеспечить параллелизм на уровне данных.

Основная цель – достижение более высокой производительности мультимедийных приложений и систем обработки и передачи данных.

Синтаксис MMX-команд

`instruction dest, src`
`instruction` - имя команды,
`dest` - выходной операнд,
`src` - входной операнд.

+суффикс, который определяет тип данных: B, W, D, Q. Если в суффиксе есть две из этих букв, первая соответствует входному операнду, а вторая - выходному.

MMX-расширение

Численные регистры

8 (mm0..mm7) * 8 байт

mm0

MMX-расширение

mm0

Типы данных:

B - упакованные байты (*packed byte*);

W - упакованные слова (16-разрядные) (*packed word*);

D - упакованные двойные слова (*packed double word*);

Q - 64-разрядные слова (*quadword*).

MMX-регистры

Физически совмещены со стеком регистров математического сопроцессора.

При выполнении любой из MMX-команд происходит установка «режима MMX», стек регистров сопроцессора рассматривается как набор MMX-регистров.

Завершает работу в режиме MMX команда **EMMS** (End MultiMedia State).

MMX-регистры

Такая реализация позволила избежать проблем совместимости с переключением контекста, поскольку число регистров процессора, и, следовательно, код, выполняющий их сохранение и восстановление, не изменились.

MMX-регистры

С другой стороны, переход между режимами занимает значительное время.
Поэтому при необходимости работы в обоих режимах для достижения наилучших результатов рекомендуется группировать эти команды отдельно друг от друга.

SSE-команды

SSE (англ. Streaming SIMD Extensions). Для преодоления проблемы одновременного использования с сопроцессором.
SSE включает в архитектуру процессора 8 128-битных регистров (xmm0 до xmm7), каждый из которых трактуется как 4 последовательных значения с плавающей точкой одинарной точности.

Задание

Создать консольное приложение, которое выполняет вычисления (в соответствии с вариантом) :

1) на языке Си,

2) на ассемблере, без команд MMX,

3) с использованием команд MMX.

После вычислений должны быть выведены время выполнения и результат для каждого случая.

Значения элементов матриц генерируются приложением (не вводятся с клавиатуры). Вычисления производятся много (1 млн) раз. *Размер матриц (векторов) кратен количеству элементов в регистре MMX.*

Пример

Найти скалярное произведение векторов *a* и *b*

```
int i;
short a_vect[16], b_vect[16];
short cnt = 16;
int res = 0, res1 = 0;
double r;
int j, temp, sum = 0;

...//инициализация переменных
```

Пример

Найти скалярное произведение векторов *a* и *b*

Си:

```
for(j = 0; j < 16; j++)
 {
 temp = a_vect[j] * b_vect[j];
 sum+=temp;
}
```

Пример

Найти скалярное произведение векторов *a* и *b*

Ассемблер без MMX:

```
cnt = 16;
_asm
{
  pusha    ; сохранить в стек все регистры
  xor     esi, esi
  xor     ecx, ecx
loop1:
  mov     ax,  a_vect[esi]  ; чтение из памяти
  mov     bx,  b_vect[esi]

          imul    ax,  bx
  add     cx,  ax

  add     esi, 2 ; short → шаг по 2 байта
  sub     cnt, 1 ; по 1 числу за итерацию
jnz     loop1

  mov     res1, ecx ; сохранить результат
  popa ; восстановить из стека сохранённые регистры
}
```

Пример

Найти скалярное произведение векторов *a* и *b*

```
loop1:
 mov     ax,  a_vect[esi]  ; чтение из памяти
 mov     bx,  b_vect[esi]

 imul    ax,  bx
 add     cx,  ax

 add     esi, 2 ; short → шаг по 2 байта
 sub     cnt, 1 ; по 1 числу за итерацию
jnz     loop1

mov     res1, ecx ; сохранить результат
```

Пример
Найти скалярное произведение векторов *a* и *b*

Ассемблер с MMX:

```
cnt = 16;
_asm
{
  pusha    ; сохранить в стек все регистры
  xor     esi, esi
  pxor    MM7, MM7
loop1:
  movq       MM0, a_vect[esi] ; чтение из памяти
  movq       MM1, b_vect[esi]

       pmaddwd MM0, MM1
  paddd       MM7, MM0
  add     esi, 8
  sub     cnt, 4
jnz     loop1

  movq    MM0, MM7
  psrlq     MM7, 32
  paddd   MM7, MM0
  movd    res, MM7
  emms
  popa
}
```

Пример

Найти скалярное произведение векторов *a* и *b*

Ассемблер с MMX:

```
pusha    ; сохранить в стек все регистры
xor      esi, esi
pxor     MM7, MM7

;MM7 – накопитель произведений координат
```

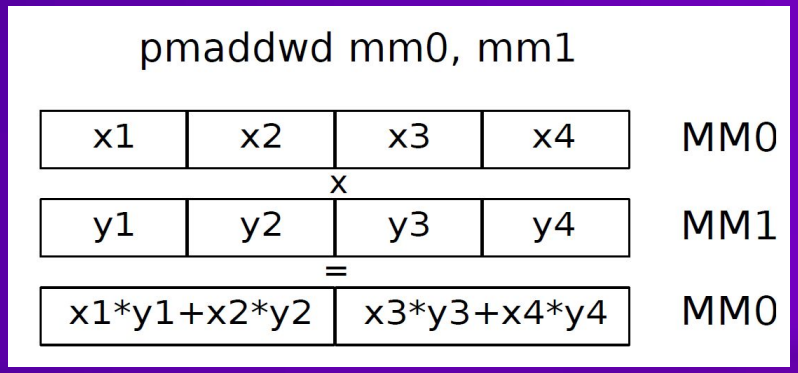
Пример

Найти скалярное произведение векторов *a* и *b*

Ассемблер с MMX:

```
loop1:
 movq  MM0, a_vect[esi] ; чтение из памяти
 movq  MM1, b_vect[esi]

 pmaddwd MM0, MM1
```



Пример

Найти скалярное произведение векторов *a* и *b*

Ассемблер с MMX:

```
loop1:
 movq  MM0, a_vect[esi] ; чтение из памяти
 movq  MM1, b_vect[esi]

 pmaddwd MM0, MM1
 paddd       MM7, MM0 ;накопление в MM7
 add     esi, 8 ; short → шаг по 2 байта * 4
 sub     cnt, 4 ; по 4 элемента за операцию
jnz     loop1
```

Пример

Найти скалярное произведение векторов *a* и *b*

Ассемблер с MMX:

!!! В MM7 сумма разбита на 2 части

```
movq    MM0, MM7    ;для сложения
psrlq   MM7, 32     ;обеих частей
paddd   MM7, MM0    ;суммы
movd    res, MM7 ; сохранить результат
emms ; вернуть режим сопроцессора
popa ; восстановить регистры
```