Diseño Basado en Microprocesadores
Práctica 4
Instrucciones SIMD SSE

Índice

---

1. Objetivos

En esta práctica se usaran instrucciones SSE SIMD y escalares.

2. Alineamiento de datos con el compilador GCC

La mayoría de las instrucciones SSE necesitan que los datos estén alineados en la memoria en direcciones divisibles entre 16, que es el alineamiento correcto para los datos de 128 bits (16 bytes) que estas instrucciones usan. Si no tomamos esta precaución lo más probable es que provoquemos excepciones por alineación incorrecta. En cualquier caso, el alineamiento es muy conveniente ya que aumenta la velocidad a la que se accede a los datos.

Podemos obligar al compilador GCC a alinear una variable incluyendo el atributo `aligned` al declararla. Por ejemplo, si deseamos declarar un array de 4 datos de tipo `float` llamado `vector` alineado en una posición de memoria divisible entre 16 podemos declararlo de la siguiente forma

```
float vector[4] __attribute__ ((aligned (16))) = {1.0, 2.0, 3.0, 4.0};
```

3. Ejercicios

3.1. Ejercicio 1

Escribe una función en ensamblador de 64 bits que use instrucciones SSE para obtener un vector de cuatro componentes que sea la suma de las filas de una matriz 4×4 de números en punto flotante de precisión simple. Por ejemplo, si la matriz es

$$\begin{pmatrix} 1.0 & 2.0 & 3.0 & 4.0 \\ 5.0 & 6.0 & 7.0 & 8.0 \\ 9.0 & 10.0 & 11.0 & 12.0 \\ 13.0 & 14.0 & 15.0 & 16.0 \end{pmatrix}$$

el vector resultante será

$$\begin{pmatrix} 28.0 & 32.0 & 36.0 & 40.0 \end{pmatrix}$$

El prototipo de la función es

```
int sse_sumar_filas(float *ptr_matriz, float *ptr_suma);
```

donde

- `ptr_matriz` es un puntero al primer elemento de la matriz. Debe estar alineado a una dirección divisible entre 16. La matriz está almacenada en la memoria por filas.
- `ptr_suma` un puntero al lugar donde hay que almacenar el vector suma de las filas. Debe estar alineado a una dirección divisible entre 16.

Si todos los argumentos son válidos retornar 1, en caso contrario retornar 0.

3.2. Ejercicio 2

Escribe una función en ensamblador de 64 bits que use instrucciones SSE para comparar el módulo de dos vectores de cuatro componentes de tipo `double`.

El prototipo de la función es

```
int sse_comparar_modulos(double *vector_1,
                         double *vector_2,
                         int *resultado_comparacion);
```

donde

- `vector_1` es un puntero a la primera componente del primer vector. Debe estar alineado a una dirección divisible entre 16.
- `vector_2` es un puntero a la primera componente del segundo vector. Debe estar alineado a una dirección divisible entre 16.
- `resultado_comparacion` es un puntero a una variable donde se almacenará un valor –1, 0 o 1 que indicará la relación entre los módulos. Si el módulo del primer vector es menor que el del segundo vector se almacenará –1. Si los módulos son iguales se almacenará 0. Si el módulo del primer vector es mayor que el del segundo se almacenará 1.

Si todos los argumentos son válidos retornar 1, en caso contrario retornar 0.

Serán útiles las instrucciones MOVAPD, MULPD, HADDPD y COMISD. En lugar de comparar los módulos se pueden comparar sus cuadrados así que no es necesario calcular la raíz cuadrada.

3.3. Ejercicio 3

Escribe una función en ensamblador de 64 bits que use instrucciones SSE para calcular el producto escalar de dos vectores de elementos de tipo `float`. El prototipo de la función es

```
int sse_producto_escalar(float *vector_1,
                         float *vector_2,
                         unsigned int dimension,
                         float *resultado);
```

donde

- `vector_1` es un puntero a la primera componente del primer vector. Debe estar alineado a una dirección divisible entre 16.
- `vector_2` es un puntero a la primera componente del segundo vector. Debe estar alineado a una dirección divisible entre 16.
- `dimension` es la dimensión de los vectores. La dimensión no tiene que ser necesariamente múltiplo de 4.
- `resultado` es un puntero al lugar donde hay que almacenar el resultado.

Si todos los argumentos son válidos retornar 1, en caso contrario retornar 0.

Aprovecha la capacidad de las instrucciones SSE de operar en paralelo sobre varios datos aunque, si la dimensión del vector no es un múltiplo de 4, algunos elementos tendrán que ser procesados de forma secuencial.