Diseño Basado en Microprocesadores
Práctica 5
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 usan estas instrucciones. 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

Crea una función en ensamblador de 64 bits que use instrucciones SSE para convertir una imagen en escala de grises de un byte por pixel en una imagen en blanco y negro de un byte por pixel. La conversión consiste en realizar la siguiente operación con cada pixel de la imagen:

```
si valor_pixel >= umbral
    valor_pixel = 255
sino
    valor_pixel = 0
finsi
```

donde `umbral` es un valor entre 0 y 255. Como vemos, si la intensidad original de un pixel es mayor o igual que `umbral` el pixel se vuelve blanco, mientras que si es menor se vuelve negro.

(a) (b)

Figura 1: Imagen original (a) y resultado de la conversión (b) para un umbral de 100.

El prototipo de la función es

```
int sse_imagen_a_blanco_y_negro(unsigned char *ptr_imagen,
                                int ancho,
                                int alto,
                                unsigned char umbral);
```

donde

`ptr_imagen` es un puntero a los datos de la imagen. Cada byte representa la intensidad de un pixel de la imagen. Debe estar alineado en una dirección divisible entre 16.

`ancho` es el ancho de la imagen en pixels.

`alto` es el alto de la imagen en pixels.

`umbral` es valor por debajo del cual los pixels de la imagen se convierten en pixels negros. Los pixels de la imagen con valores igual o mayor que `umbral` se convierten en pixels blancos.

La modificación de la imagen se realiza in situ, es decir, la imagen original resulta destruida y sustituida por la imagen de dos niveles resultante.

La función devuelve 1 si pudo trabajar correctamente y 0 si hubo algún error en los valores de los argumentos: puntero a la imagen nulo o no alineado, o alguno de los parámetros `ancho` o `alto` menor que 1.

En la página de la asignatura hay disponibles ficheros fuente de partida y una imagen de prueba.

Las instrucciones clave para realizar la conversión son PMINUB y PCMPEQB. La instrucción PMINUB realiza una comparación de 16 pares de bytes sin signo de sus dos operandos y deja en el operando destino el menor de cada par. Por ejemplo, si los registros XMM0 y XMM1 almacenan los siguientes bytes empaquetados

```
XMM0 = (100,  14,  57,  28,  90, 210,  34,   8,  23, 145,  64,  70, 198,  80,
5,  24)
XMM1 = ( 50,  50,  50,  50,  50,  50,  50,  50,  50,  50,  50,  50,  50,  50,
50,  50)
```

y se ejecuta la instrucción

```
pminub  xmm0, xmm1
```

el registro XMM0 quedará con

```
XMM0 = ( 50,  14,  50,  28,  50,  50,  34,   8,  23,  50,  50,  50,  50,  50,
5,  24)
```

La instrucción PCMPEQB compara los 16 pares de bytes empaquetados en sus dos operandos y, por cada par, deja en el destino un byte con el valor 0xFF si son iguales y 0x00 si son distintos. Por ejemplo, si los registros XMM0 y XMM1 tienen los siguientes bytes empaquetados

```
XMM0 = ( 50,  14,  50,  28,  50,  50,  34,   8,  23,  50,  50,  50,  50,  50,
5,  24)
XMM1 = ( 50,  50,  50,  50,  50,  50,  50,  50,  50,  50,  50,  50,  50,  50,
50,  50)
```

y se ejecuta la instrucción

```
pcmpeqb  xmm0, xmm1
```

el registro XMM0 quedará con

```
XMM0 = ( 255,  0, 255,   0, 255, 255,   0,   0,   0, 255, 255, 255, 255, 255,
0,   0)
```

También serán útiles otras instrucciones SSE como MOVD, MOVDQA, PXOR y PSHUFB.

3.2. Ejercicio 2

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(const 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.

Las instrucciones SSE necesarias para hacer el ejercicio son MOVAPS y ADDPS.

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(const float *vector_1,
                         const 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 mediante instrucciones SSE escalares.

Las instrucciones SSE necesarias para hacer el ejercicio son: MOVAPS, MULPS, ADDPS, HADDPS, MOVSS, MULSS y ADDSS.