Diseño Basado en Microprocesadores
Práctica 5
Instrucciones SIMD SSE (II)

Índice

---

1. Objetivos

En esta práctica se usaran instrucciones SSE de enteros para realizar funciones de procesamiento básico de imágenes.

2. Ejercicios

2.1. 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
```

`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_64(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.

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 función devuelve 1 si pudo trabajar correctamente y 0 si hubo algún error en los valores de los argumentos.

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.

2.2. Ejercicio 2

Crear una función en ensamblador de 64 bits que use instrucciones SSE para seleccionar de una imagen RGB sólo las componentes de color deseadas. El prototipo de la función es

```
int sse_seleccionar_rgb_64(unsigned char *ptr_imagen,
                           int ancho,
                           int alto,
                           unsigned char seleccion_rgb);
```

donde

ptr_imagen es un puntero a los datos de la imagen. Cada pixel de la imagen está representado por tres bytes. El primero representa la intensidad de azul, el segundo la intensidad de verde y el tercero la intensidad de rojo.

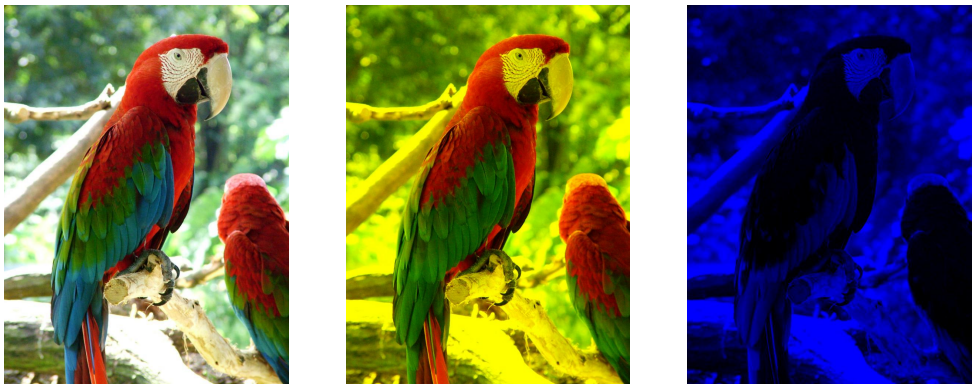
ancho es el ancho de la imagen en pixels.

alto es el alto de la imagen en pixels.

seleccion_rgb indica las componentes de color a preservar y las componentes de color a eliminar:

- Si el bit 0 de `seleccion_rgb` es 1 se preservará la componente roja de la imagen. Si el bit 0 de `seleccion_rgb` es 0 se eliminará la componente roja de la imagen.
- Si el bit 1 de `seleccion_rgb` es 1 se preservará la componente verde de la imagen. Si el bit 1 de `seleccion_rgb` es 0 se eliminará la componente verde de la imagen.
- Si el bit 2 de `seleccion_rgb` es 1 se preservará la componente azul de la imagen. Si el bit 2 de `seleccion_rgb` es 0 se eliminará la componente azul de la imagen.

La función devuelve 1 si pudo trabajar correctamente y 0 si hubo algún error en los valores de los argumentos.



(a) (b) (c)

Figura 2: Imagen original (a) y resultado de llamar a la función con `seleccion_rgb` = 3 (b) y `seleccion_rgb` = 4 (c).

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

Las instrucciones SSE más útiles para realizar el ejercicio son PXOR, POR, PAND, y MOVDQA.

Como en la imagen RGB que se va a procesar cada pixel está representado por tres bytes (uno para el color rojo, otro para el color verde y otro para el color azul, en ese orden), es mejor procesar la imagen en bloques de 48 bytes porque de esa forma un bloque con los datos de 16 pixels encaja perfectamente en tres registros XMM.

Para conservar las componentes de color seleccionadas y eliminar el resto podemos aplicar a cada bloque de 16 pixels (48 bytes) una operación AND con la máscara adecuada. A su vez, esta máscara se puede obtener combinando mediante la operación OR las máscaras que permiten seleccionar las componentes roja, verde y azul, según indique el argumento `seleccion_rgb`. Las máscaras para seleccionar las componentes roja, verde y azul pueden estar predefinidas en la sección de datos `.data`, por ejemplo:

```
                section .data
                align   16

mascara_r_1:    db      255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255
mascara_r_2:    db        0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0
mascara_r_3:    db        0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0

mascara_g_1:    db        0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0
mascara_g_2:    db      255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255
mascara_g_3:    db        0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0

mascara_b_1:    db        0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0
mascara_b_2:    db        0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0
mascara_b_3:    db      255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255,   0,   0, 255
```