SIMD

PARTE 2

Sobre Saturación, Empaquetado y Comparaciones

Francisco Giordano

Organización del Computador II

1er cuatrimestre 2015

Repaso

Intel SSE

paddd 

- Single Instruction, Multiple Data
- 16 registros de 128 bits:
  - `xmm0`, `xmm1`, …, `xmm15`
- Instrucciones para distintos tipos de datos:
  - **Enteros**
    - Empiezan con p
    - Ejemplos: paddd, psubq, …
  - **Punto flotante**

Repaso

Instrucciones de punto flotante

- Dos tipos de precisión:
  - **S = Single (float)**
  - **D = Double (double)**
- Dos formas de operar:
  - **S = Scalar (escalar)**
  - **P = Packed (empaquetado/paralelo)**
- Sufijos para las instrucciones:
  - SS = scalar single-precision
  - SD = scalar double-precision
  - PS = packed single-precision
  - PD = packed double-precision
- Ejemplos: addps, `mulpd`, `minpd`, ...

¡Atención!

Algunas instrucciones tienen nombres muy parecidos.

No confundir

- paddd con addpd
- pshufd con shufpd
- etc…

Saturación

Enunciado

Escribir una función que aumente el brillo de una imagen en blanco y negro utilizando instrucciones SSE.

Saturación

Solución tentativa en C

Se tiene el brillo de cada píxel de la imagen de entrada en una matriz `I` de `N` filas por `M` columnas, y un parámetro `b` entero según el cual se debe incrementar cada píxel.

```
unsigned char I[N][M];
unsigned char b;
```

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < M; j++)
    I[i][j] += b;
```

¿Qué pasa si `I[i][j]` + `b` *es mayor que 255?*

Saturación

Ejemplo de problemas al no saturar

¿Qué pasa si `I[i][j]` + b *es mayor que 255?*



Original. Brillo (b = 50). Brillo (b = 100).

Saturación

Problemas al no saturar

El resultado de la suma no entra en 1 byte.

Entonces se trunca: se pasa de colores más claros (cercanos al 255) a colores más oscuros (cercanos al 0).

También conocido como *wrap around*.

Para evitar esto, habría que hacer

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < M; j++)
    I[i][j] = min(255, I[i][j] + b);
```

Esto se llama **saturación**.

Saturación

Ejemplo de aumento de brillo con saturación



Original. Brillo (b = 50). Brillo (b = 100).

Saturación

Instrucciones para saturación

Como esto es algo común en el procesamiento de señales, existen instrucciones específicas para operar de esta manera:

- **PADDUSB, PSUBUSB**
  Suma y resta de enteros sin signo con saturación sin signo.
- **PADDSB, PSUBSB**
  Suma y resta de enteros con signo con saturación con signo.

En este caso son operaciones de a **byte**. También existen de a **word**. Ver el manual de Intel.

Desempaquetado

Motivación

Supongamos que queremos pasar una imagen a escala de grises.

Una forma de hacerlo es mediante la fórmula:

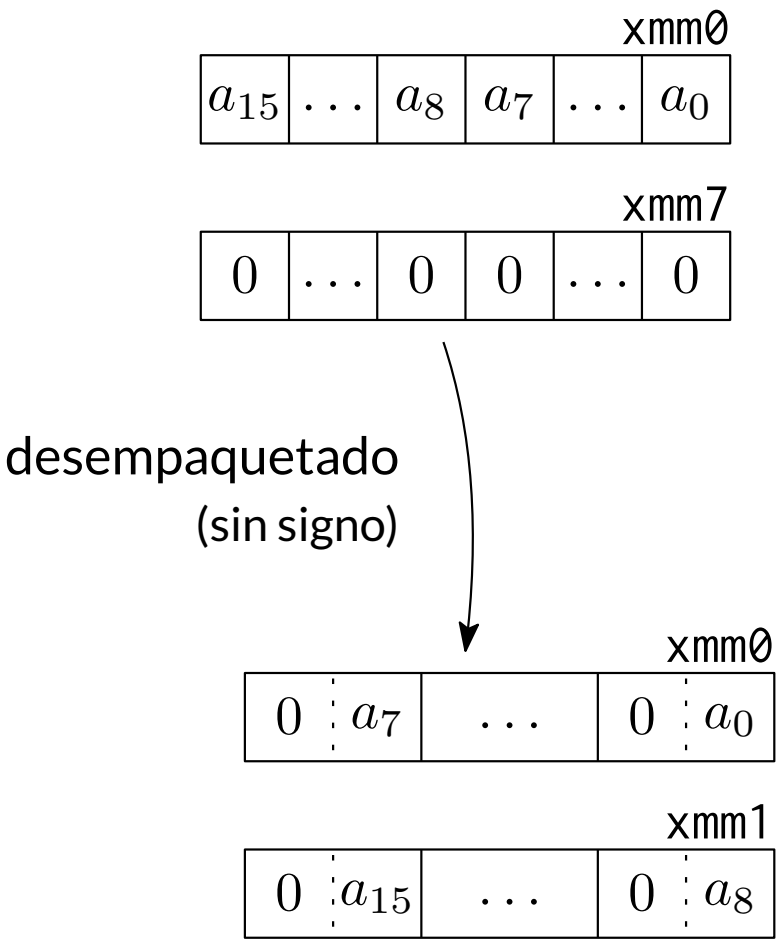
$$f(r, g, b) = \frac{1}{4} \cdot (r + 2g + b)$$

¡Atención! **Puede haber overflow en la suma.**

- Para no perder información en los cálculos, es necesario manejar los resultados intermedios en un tipo de datos con mayor rango.
  - El tipo de datos original es de **byte**.
- Deberíamos operar con datos en tamaño **word**.
- ¿Cómo hacemos esto?
  Utilizando las instrucciones de desempaquetado.

Desempaquetado

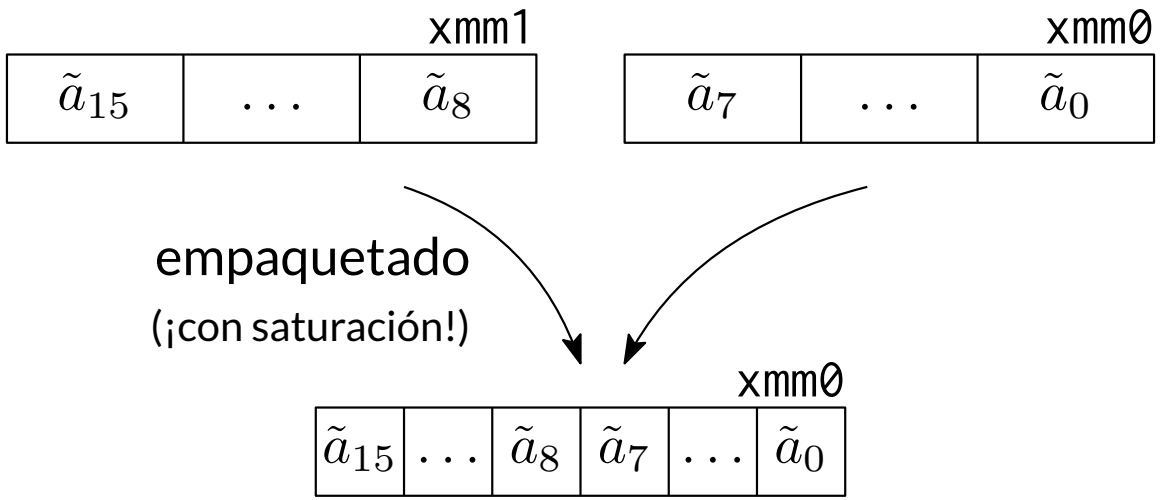
En código




```
pxor xmm7, xmm7
movdqu xmm1, xmm0
punpcklbw xmm0, xmm7
punpckhbw xmm1, xmm7
```

Ahora tenemos los valores originales en tamaño **word**.

Empaquetado



- Después de extender los datos, realizamos las operaciones que necesitamos.
- Tenemos que volver a guardar los datos, por lo que tenemos que volver a **convertirlos a byte**.
- ¿Cómo hacemos la conversión? **Empaquetando los datos.**

```
packuswb xmm0, xmm1
```

Empaquetado/Desempaquetado

Formato de instrucciones

Desempaquetado: punpck{`l`,h}{bw,wd,dq,qdq}

- `l`, h
  parte baja (**l**ow) o alta (**h**igh)
- bw, wd, dq, qdq
  de **b**yte a **w**ord, de **w**ord a **d**word, etc…

Empaquetado: pack{`ss`,us}{wb,dw}

- `ss`, us
  signed / **u**nsigned, con **s**aturación
- wb, dw
  de **w**ord a **b**yte, de **d**word a **w**ord

Empaquetado/Desempaquetado

Resumen

Resumiendo, la forma de trabajar en estos casos es

- Leer datos a procesar
- Extender precisión (unpack)
- Realizar las cuentas necesarias
- Volver a la precisión original (pack)
- Guardar datos procesados

Comparación

Introducción

En SSE también existen instrucciones de comparación, aunque se comportan un poco diferente a lo que veníamos usando.

Claramente **no se puede usar saltos condicionales** porque estamos trabajando con muchos datos a la vez.

Entonces **se usan máscaras** obtenidas a partir de comparaciones.

	-7	42	-5	57
$<$	0	0	0	0
	1	0	1	0

Comparación

Ejemplo

Supongamos que estamos trabajando con **words** y queremos saber cuáles de ellos son menores a cero.

Datos en `xmm0`

1000	-456	-15	0	100	234	-890	1

```
pxor xmm7, xmm7                 ; xmm7 = 0 | 0 | ... | 0
pcmpgtw xmm7, xmm0              ; xmm7 > xmm0 ?
```

Resultado en `xmm7`

0x0000	0xFFFF	0xFFFF	0x0000	0x0000	0x0000	0xFFFF	0x0000

Es decir, compara **word a word** y si se cumple la condición setea **unos** (0xFFFF en este caso) en el resultado, o **ceros** si no.

Comparación

Usos

La comparación nos devuelve un registro con unos y ceros.

Podemos usarlo para:

- **Extender el signo** de números signados al desempaquetar. Ya que los números negativos se extienden con unos, y los positivos con ceros.
  - $-5 = 1011 \rightsquigarrow 1111\,1011$
  - $+5 = 0101 \rightsquigarrow 0000\,0101$
- **Como una máscara**, acompañado de instrucciones como: `PAND`, `POR`, etc.

Comparación

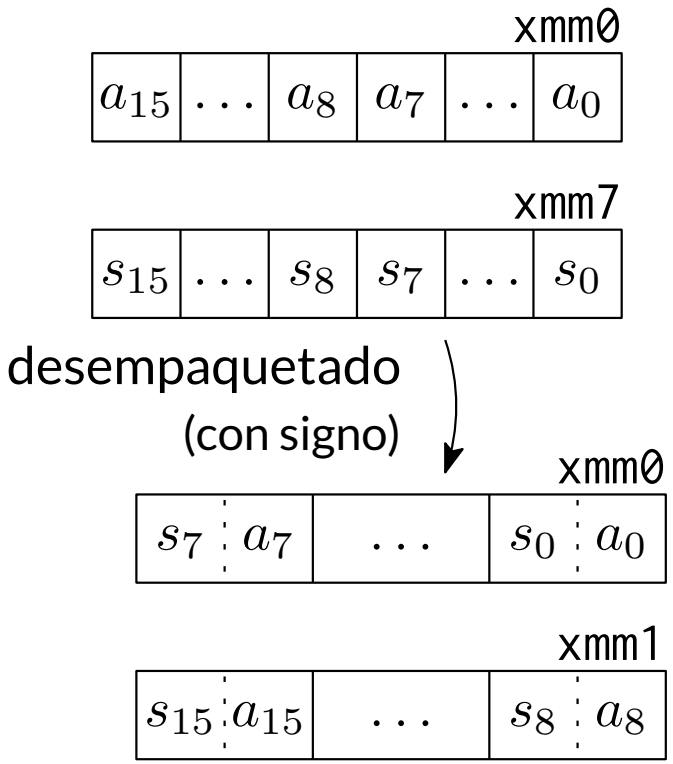
Extensión de signo en desempaquetado

Para extender el signo del registro del ejemplo anterior

xmm0							
1000	-456	-15	0	100	234	-890	1

Si tenemos el resultado de la comparación en xmm7

```
movdqu xmm1, xmm0
punpckhwd xmm0, xmm7
punpcklwd xmm1, xmm7
```



Comparación

Uso de máscaras

Supongamos ahora que se desea sumar 3 a los números menores a 0.

Tenemos los números en `xmm0`, y el resultado de la comparación en `xmm7`.

Tenemos el número 3, que queremos sumar, en un registro:

`xmm1`

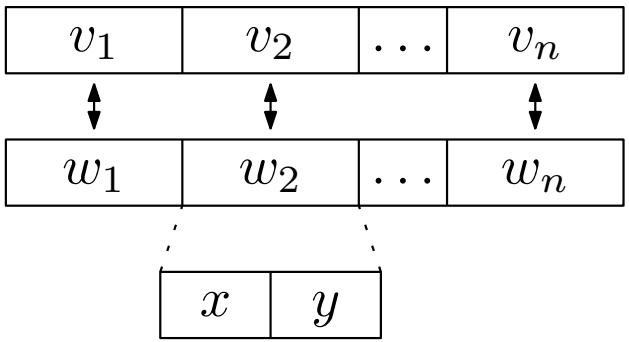
3	3	...	3

```
pand xmm7, xmm1          ; xmm7 = 0 | 3 | ... | 3 | 0
paddw xmm0, xmm7
```

Ejercicio 1

Máxima distancia

Calcular la distancia máxima entre puntos correspondientets de dos vectores.



- Los puntos (x, y) están almacenados como dos números contiguos de punto flotante de precisión simple (*floats*).
- `n` es la cantidad de puntos, y es múltiplo de 2.
- Se deben procesar dos puntos en paralelo.
- La distancia se calcula entre los puntos de *v* y los de *w*, elemento a elemento.

```
float maximaDistancia(float* v, float* w, unsigned short n);
```

Distancia

$$\text{dist}((x, y), (x', y')) = \sqrt{(x - x')^2 + (y - y')^2}$$

Para hacer

Máxima distancia

- ¿Qué pasa si ahora tenemos que comparar todos los puntos de v contra todos los de w?
- ¿Y si n no fuera múltiplo de 2?
- ¿Podemos procesar más de 2 puntos en paralelo?

Ejercicio 2

Normalizar un vector

Dado un vector de longitud n, múltiplo de 4, devolver uno normalizado: es decir que el máximo elemento sea uno, y el mínimo cero.

- Se debe procesar la máxima cantidad posible de elementos en paralelo.

```
float* normalizarVector(float* v, int n);
```

Debemos buscar el máximo y el mínimo, y aplicar a cada elemento

$$x_i \leftarrow \frac{(x_i - min)}{(max - min)}$$