

# SIMD

Pack/Unpack

Saturación

Comparación y Máscaras

Organización del Computador II

10 de Abril de 2018

Ezequiel Barrios

## 1 Brevísimo repaso

## 2 Instrucciones avanzadas

- Saturación
- Empaquetamiento/Desempaquetamiento
- Comparación

## 3 Ejercicio

# Brevísimo repaso I

- SSE (Streaming SIMD Extensions) es un set de instrucciones que implementa el **modelo de cómputo SIMD** (una misma instrucción para varios datos a la vez).
- Existen 16 **registros de 128 bits** (XMM0, ..., XMM15)
- Varios posibles **tipos de datos**: **enteros** (8, 16, 32, 64 bits), **flotantes de precisión simple** (32 bits) y **flotantes de precisión doble** (64 bits).
- Distintas **formas de operar**:
  - $P = \textit{Packed}$  (**E**mpaquetado / **P**aralelo).
  - $S = \textit{Scalar}$  (**E**scalar).

# Brevísimo repaso II

- Las instrucciones que **comienzan con P** sólo son para operar sobre **enteros** (ejemplo **PADDB**).
- Las instrucciones que comienzan sin *P*, y que **terminan con un determinado sufijo**, son para operar sobre flotantes (ejemplo **ADDPS**). Los sufijos son los siguientes:
  - **SS**: scalar single-precision (float)
  - **SD**: scalar double-precision (double)
  - **PS**: packed single-precision (float)
  - **PD**: packed double-precision (double)

# Saturación (motivación)

## Ejemplo

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



# Saturación

## Ejercicio

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

```
1 unsigned char IMG[N][M];  
2 unsigned char inc;
```

Podríamos hacer:

```
1 for(int i = 0; i < N; i++)  
2     for(int j = 0; j < M; j++)  
3         IMG[i][j] += inc;
```

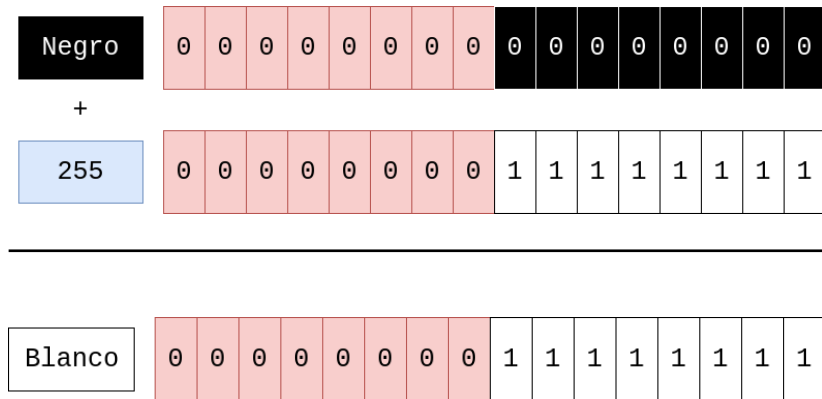
¡Cuidado!

¿Qué pasaría si `I[i][j] + b` fuera mayor a 255?

# Saturación: incremento de brillo sin saturación

## Ejemplo

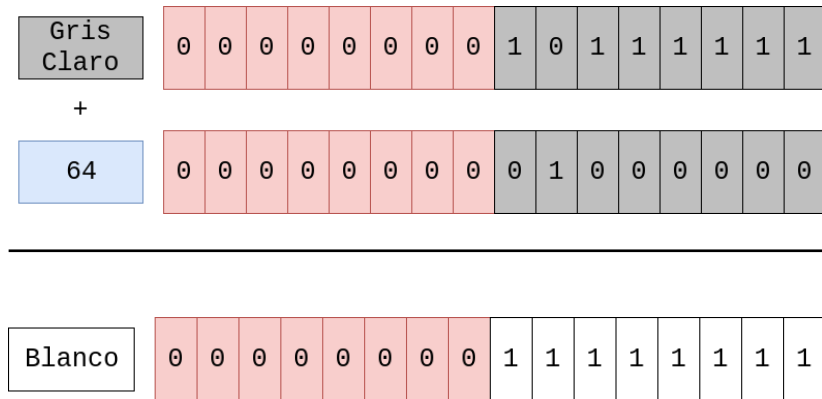
Tengo un registro con un pixel negro, y le quiero sumar 255.



# Saturación: incremento de brillo sin saturación

## Ejemplo

Tengo un registro con un pixel gris claro, y le quiero sumar 64.

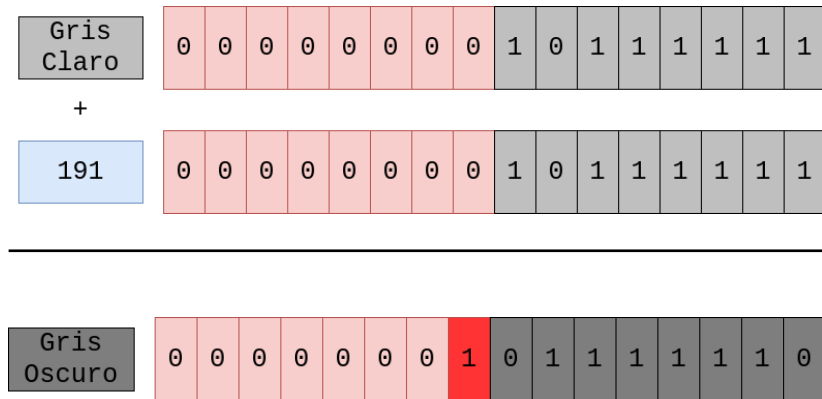




# Saturación: incremento de brillo sin saturación

## Ejemplo

Tengo el mismo registro anterior, pero le quiero sumar 191.



# Saturación: incremento de brillo sin saturación

## Más ejemplos

(cortesía de ~~nuestra vieja amiga~~ **Lena la Torre de Broadway**)



(a) Original



(b) +50 de brillo



(c) +100 de brillo



(d) +150 de brillo



(e) +200 de brillo



(f) +250 de brillo  
*(¡La original!)*

# Saturación: problemas al no saturar

El resultado de la suma podría no entrar en el registro.

- Se trunca el resultado.
- Se pasa de colores más claros (cercanos al 255) a colores más oscuros (cercanos al 0).

Una forma de evitar esto, podría ser:

## Incremento de brillo con saturación

```
1  for(int i = 0; i < N; i++) {  
2      for(int j = 0; j < M; j++) {  
3          IMG[i][j] = min( 255, IMG[i][j] + inc );  
4      }  
5  }
```

# Saturación: incremento de brillo con saturación



(g) Original



(h) +50 de brillo



(i) +100 de brillo



(j) +150 de brillo



(k) +200 de brillo



(l) +250 de brillo  
(¿Y la imagen?)

# Saturación: instrucciones comunes

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

- **PADDUSB/PSUBUSB**: Suma/Resta enteros sin signo con saturación sin signo.
- **PADDSB/PSUBSB**: Suma/Resta enteros con signo con saturación con signo.

En este caso son operaciones de a **byte**.  
También existen de a **word**.

**Importante**

Ver el manual de Intel.

# Motivación

## Ejemplo

Pasar una imagen RGB a escala de grises.



# Motivación

## Ejemplo

Pasar una imagen RGB a escala de grises.

Fórmula para pasar imágenes a escala de grises

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

¡Cuidado!

¿Qué podría pasar con  $(r + 2g + b)$ ?

# Motivación: Transformar RGB(0x401080) a grises

## Ejemplo

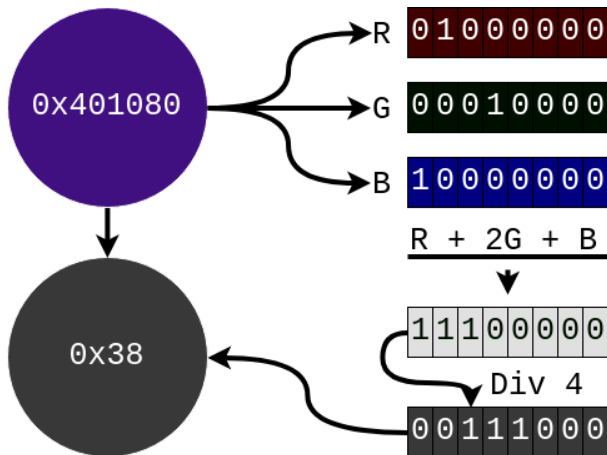
Pasar un pixel RGB=(0x40, 0x10, 0x80) a escala de grises.



0x401080



# Motivación: Transformar RGB(0x401080) a grises



# Motivación: Transformar RGB(0x40BF80) a grises

## Ejemplo 2

Pasar un pixel RGB=(0x40, 0xBF, 0x80) a escala de grises.



# Motivación: Transformar RGB(0x40BF80) a grises

¿Cuál es más oscuro?

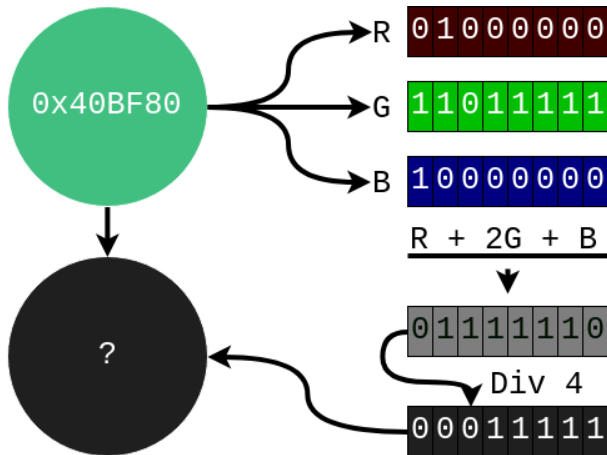


0x40BF80

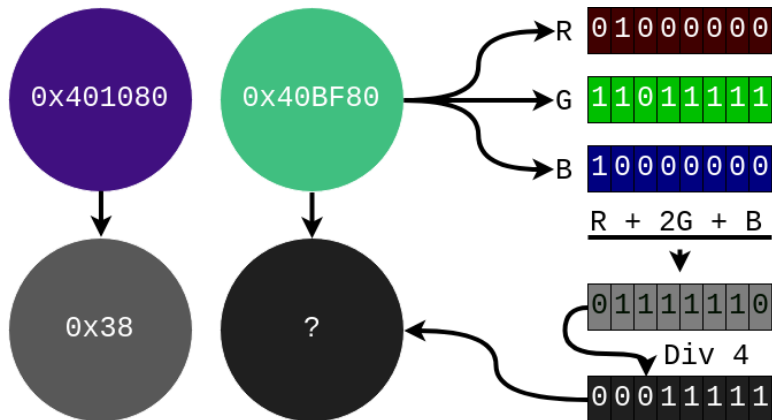


0x401080

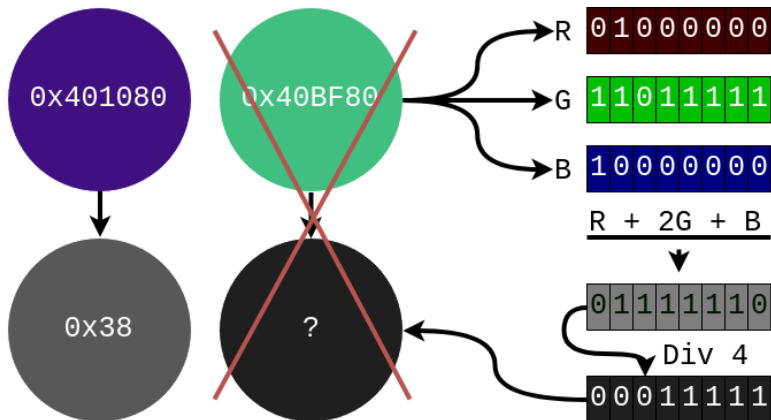
# Motivación: Transformar RGB(0x40BF80) a grises



# Motivación: Transformar RGB(0x40BF80) a grises



# Motivación: Transformar RGB(0x40BF80) a grises



**¡...la fórmula no sirve para nada!**

## ¿Cuál es el problema?

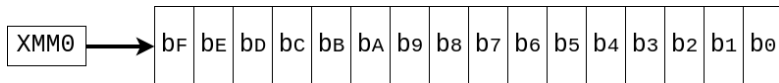
La representación utilizada (byte) no es suficiente para albergar los resultados intermedios. **Estamos perdiendo información.**

- En esta situación, es necesario manejar los resultados intermedios en un tipo de datos de mayor precisión para no perder información en los cálculos. La precisión original es de **byte**.
- Lo primero que podemos hacer es pasar los datos de **byte** a algo más grande (en este caso nos alcanza con pasar a **word**).
- ¿Cómo lo hacemos?  
**Utilizando las instrucciones de desempaqueado.**

# Empaquetamiento/Desempaquetamiento

## Extensión de la representación

Tenemos un registro XMM con bytes sin signo,  $b_0$  hasta  $b_{15}$ .

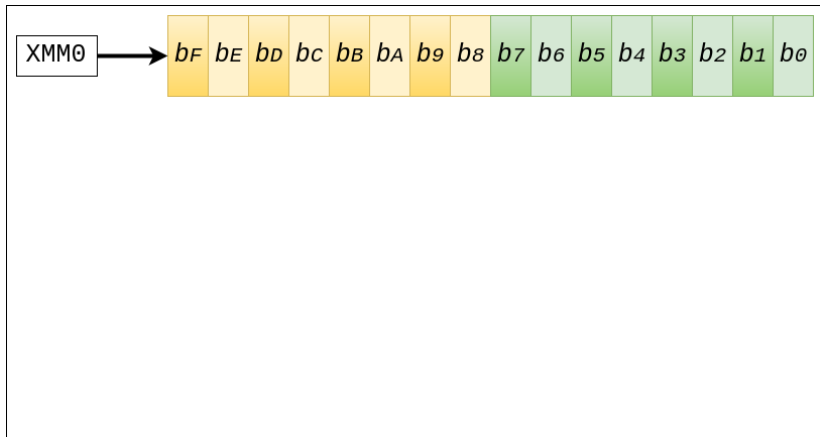


**¿Cómo duplicamos la precisión o el tamaño de los datos?**

Solamente necesitamos agregar ceros delante de cada número.

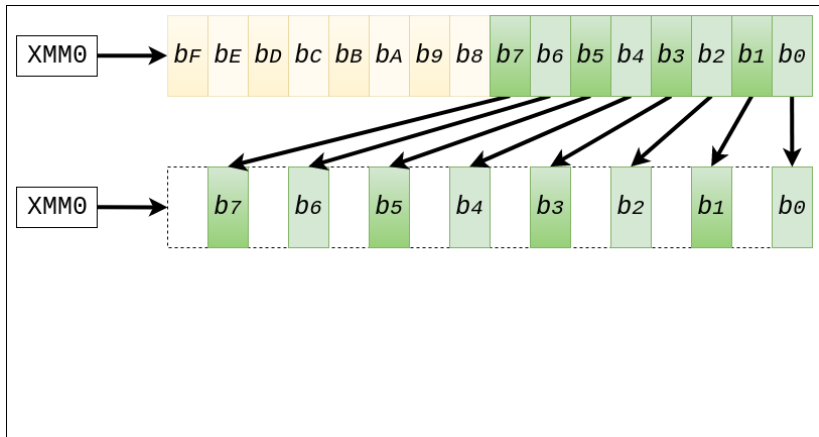


## Desempaquetar parte baja de XMM0 (idea)



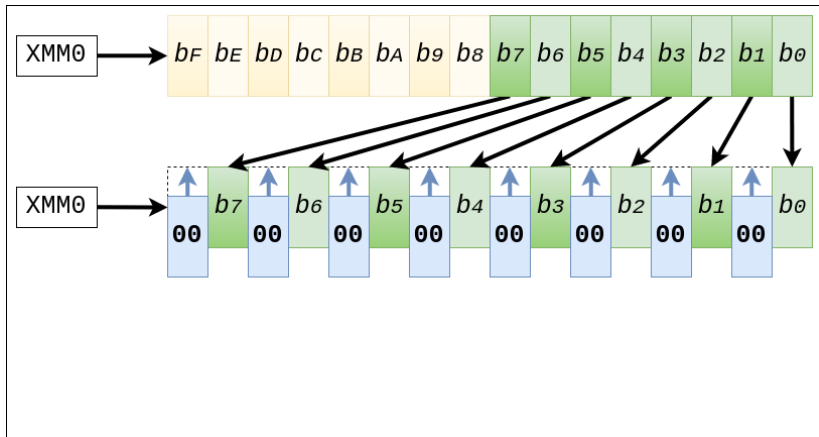
# Desempaquetamiento

## Desempaquetar parte baja de XMM0 (idea)



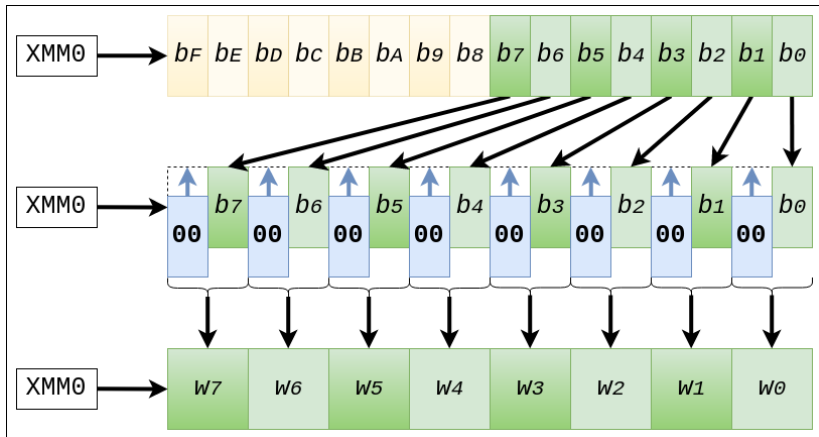
# Desempaquetamiento

## Desempaquetar parte baja de XMM0 (idea)



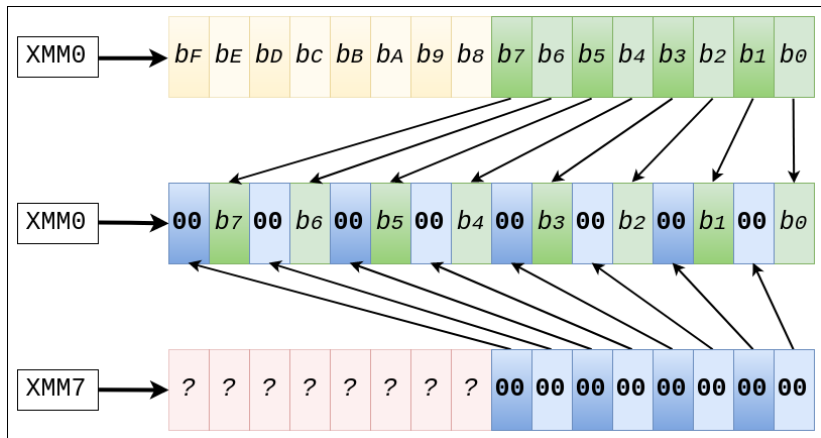
# Desempaquetamiento

## Desempaquetar parte baja de XMM0 (idea)



# Desempaquetamiento

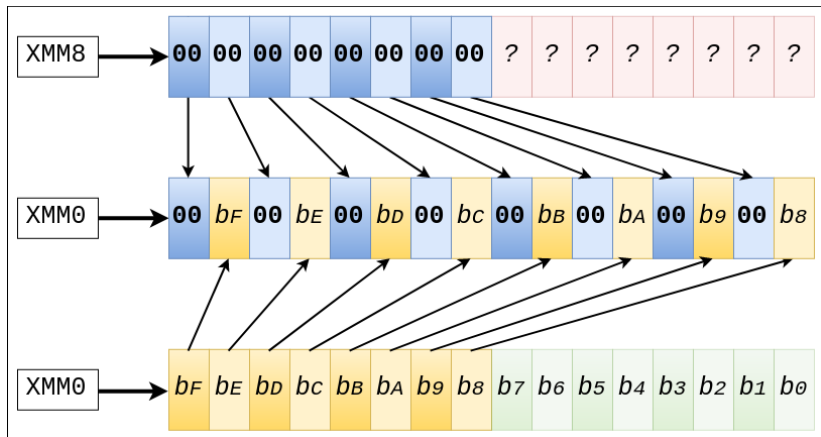
## Desempaquetar parte **baja** de XMM0



`punpcklbw xmm0, xmm7`

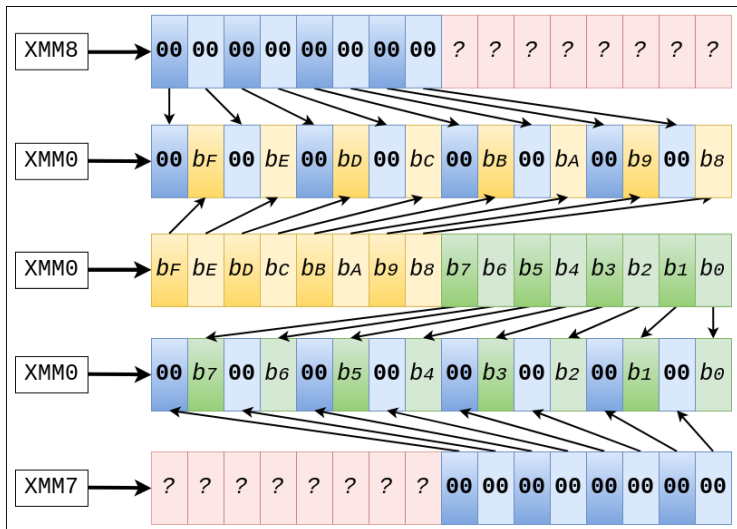
# Desempaquetamiento

## Desempaquetar parte **alta** de XMM0



`punpckhbw xmm0, xmm8`

# Desempaquetamiento de ambas partes de XMM0



¿Está bien esto?

# Desempaquetamiento de ambas partes de XMM0

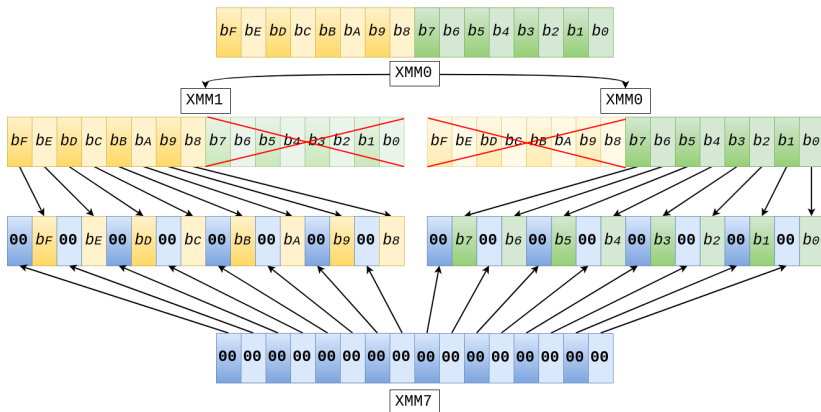
¡Importante! Para evitar errores...

- Antes, **copiamos XMM0 a otro registro** (ejemplo XMM1). ¡Sino... al desempaquetar una parte perdemos la otra!
- En uno vamos a desempaquetar **la parte baja** (con `punpcklbw`), y en el otro **la parte alta** (con `punpckhbw`).
- Para los 0 de padding, **no hace falta usar dos registros distintos**. Se puede usar el mismo registro para ambas operaciones.



# Desempaquetamiento de ambas partes de XMM0

Finalmente...



XMM0 desempaquetado en XMM0 y XMM1

# Desempaquetamiento: código de ensamblador

## Versión en x86\_64 assembly

$$\begin{aligned}\text{xmm0} &= 0 \mid a_7 \mid \dots \mid 0 \mid a_0 \\ \text{xmm1} &= 0 \mid a_{15} \mid \dots \mid 0 \mid a_8\end{aligned}$$

---

```
1  ; xmm0 = [a15, a14, ..., a1, a0]
2
3  pxor xmm7, xmm7; xmm7 = [0, 0, ..., 0]
4
5  movdqu xmm1, xmm0; xmm1 = [a15, a14, ..., a0]
6
7  punpcklbw xmm0, xmm7; xmm1 = [0, a7, ..., 0, a0]
8  punpckhbw xmm1, xmm7; xmm2 = [0, a15, ..., 0, a8]
```

Ahora cada dato en XMM0 y XMM0 es de tipo **word**.

# Empaquetamiento

Después de extender los datos, realizamos las operaciones que necesitamos.

Y al final, tenemos que guardar los datos nuevamente. Y como en este caso representan píxeles de una imagen en escala de grises, deberían seguir siendo bytes, por lo que tenemos que volver a convertirlos a **byte**.

- ¿Cómo hacemos la conversión?

## **Empaquetando los datos.**

Las instrucciones de empaquetamiento son varias, tienen en cuenta distintos tipos de datos y si los datos tienen signo o no. Por ejemplo, en el caso de **byte** tenemos:

- **packsswb** (saturación con signo)
- **packuswb** (saturación sin signo)

# Recapitulando...

Formato de instrucciones:

- **Desempaquetado:**  $\text{punpck} + l/h + bw/wd/dq/qdq$
- **Empaquetado:**  $\text{pack} + ss/us + wb/dw$
- **Empaquetado:**  $\text{pack} + ss/us + wb/dw$

## Importante, cosas a tener en cuenta

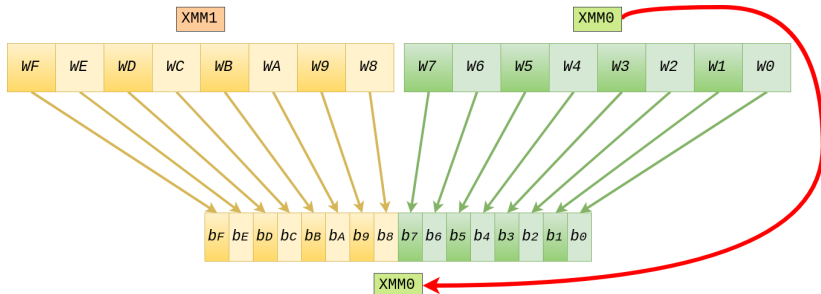
- ¿Queremos usar la parte alta o la parte baja?
- ¿De qué tipo son los datos?
- Entonces: ¿Qué tipo de saturación tengo que usar?

# Empaquetamiento/Desempaquetamiento

Volviendo al ejemplo anterior...

Ya trabajamos con los datos, ahora los queremos empaquetar nuevamente...

Usamos PACKUSWB.



Empaquetando XMM0 y XMM1 sobre XMM0

# Empaquetamiento/Desempaquetamiento

Versión en x86\_64 assembly

$$\begin{aligned}\text{xmm0} &= 0 \mid a_7 \mid \dots \mid 0 \mid a_0 \\ \text{xmm1} &= 0 \mid a_{15} \mid \dots \mid 0 \mid a_8\end{aligned}$$

Luego...

`packuswb xmm0, xmm1` ;  $\text{xmm0} = a_{15} \mid a_{14} \mid \dots \mid a_1 \mid a_0$

Ahora cada dato es de tipo **byte**

## Resumen: paso-a-paso para trabajar en estos casos

- Leer los datos a procesar.
- Extender la precisión o tamaño de los datos (*unpack*).
- Hacer las cuentas que tenemos que hacer.
- Volver a la precisión o tamaño original (*pack*).
- Guardar los datos procesados.

# Comparación

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

## Ejemplo

Tenemos 8 **words** en xmm1, y queremos saber cuáles de ellos son menores a cero.



## Ejemplo

**xmm1** = 1000 | - 456 | - 15 | 0 | 100 | 234 | - 890 | 1

pxor xmm7, xmm7 ; xmm7 = 0 | 0 | ... | 0

pcmpgtw xmm7, xmm1 ; xmm7 > xmm1 ?

El resultado de la comparación **queda en xmm7**.

**xmm7** = 0x0000 | 0xFFFF | 0xFFFF | 0x0000 | 0x0000 | 0x0000 | 0xFFFF | 0x0000

¿Qué significa este resultado?

pcmpgtw compara **word** a **word**. Si se cumple la condición, entonces setea todo en **unos** (0xFFFF), sino en **ceros**.

# Comparación: ejemplos comunes de uso

Podríamos usar el registro obtenido para:

## ■ Extender el signo:

Aprovechando que los números negativos se extienden con 1s, y los positivos con 0s. Por ejemplo:

■  $-5 = 1011$  se extiende a  $1111\ 1011$

■  $5 = 0101$  se extiende a  $0000\ 0101$

## ■ Generar máscaras:

Muy útiles para **filtrar algunos elementos del registro**. Luego, con los restantes, podemos hacer lo que necesitamos. Por ejemplo, combinar la máscara con instrucciones como PAND, POR, PXOR, etc.

# Comparación: ejemplo de extensión de signo

## Ejemplo

Tenemos 8 **words** en xmm1, y queremos extenderlos a **doubles**, pero preservando el signo.

## Código ensamblador

```
    xmm1 = 1000 | - 456 | - 15 | 0 | 100 | 234 | - 890 | 1
1    ; Comparo sobre XMM7
2    pxor xmm7, xmm7; xmm7 = [0, 0, ..., 0, 0]
3    pcmpgtw xmm7, xmm1; [xmm7>xmm1?...]
4
5    ; Luego, extendiendo preservando signo
6    movdqa xmm2, xmm1; copio xmm1 a xmm2
7    punpckhwd xmm1, xmm7; xmm1 = [1000, -456, -15, 0]
8    punpcklwd xmm2, xmm7; xmm2 = [100, 234, -890, 1]
```

# Comparación: ejemplo de uso de máscaras

## Ejemplo

Tenemos 8 **words** en XMM1, y queremos sumarle 3 a todos los que sean menores a 0.

**xmm1** = [1000 | - 456 | - 15 | 0 | 100 | 234 | - 890 | 1]

**xmm3** = [3 | 3 | 3 | 3 | 3 | 3 | 3 | 3]

Generamos el registro con la comparación en XMM7.

## Código ensamblador

```
1    pxor xmm7, xmm7; xmm7 = [0, 0, ..., 0, 0]  
2    pcmpgtw xmm7, xmm1; [xmm7>xmm1?...]
```

# Comparación: ejemplo de uso de máscaras

Código ensamblador: Cargamos, y filtramos la máscara de 3s.

```
1  section .data
2      align 16
3      mascara3: times 8 DW 3 ; 8 veces 0x0003
4      ;mascara3 = [3, 3, 3, 3, 3, 3, 3, 3]
5  section .text
6      ...
7      movdqa xmm2, [mascara3]
8      pand   xmm7, xmm2; xmm7 = [0,3,3,0,0,0,3,0]
```

$\text{xmm7} = [0, 3, 3, 0, 0, 0, 3, 0]$

Ahora, la máscara tiene 3 sólo donde nos interesa.

# Comparación: ejemplo de uso de máscaras

Código ensamblador: Finalmente, sumamos.

```
1 ; xmm1 = [1000, -456, -15, 0, 100, 234, -890, 1]  
2 ; xmm3 = [0, 3, 3, 0, 0, 0, 3, 0]  
3 paddw xmm1, xmm3
```

xmm1 = [1000, -453, -12, 0, 100, 234, -887, 1]

# Ejercicio completo

## Consigna

Hacer una funcion sumar que tome un puntero a un array de shorts como primer parametro, y un entero con el largo del array como segundo parametro y que modifique el array de shorts, sumandole 3 a las posiciones en que el valor sea negativo.

## Aridad de la función

```
sumar(short* S, int len);
```

# Ejercicio completo

```
1 section .data:
2     align 16 // Para que hago esto?
3     mascara: times 8 DW 3
4
5 section .text:
6
7 ; sumar(short* P, int len)
8 ; RDI = P
9 ; ESI = len
10 sumar:
11     ; armo el stackframe
12     push rbp
13     mov rbp, rsp
```



# Ejercicio completo

```
14      ; muevo el largo al registro contador (C),
15      ; y divido por 8
16      // por que estoy dividiendo por 8?
17      xor rcx, rcx
18      mov ecx, esi
19      shr rcx, 3
20
21      ; muevo la mascara de 3s a un registro XMM
22      // por que muevo la mascara a un registro
23      // en vez de usarla directamente?
24      movdqa xmm2, [mascara]
25      // por que estoy usando MOVDQA y no MOVDQU?
```

# Ejercicio completo

```
26  .ciclo:
27      ; traemos el dato de la memoria
28      movdqu xmm0, [rdi + rcx*8 + (-8)]
29      // por que estoy usando MOVDQU y no MOVDQA?
30
31      ; limpiamos el registro xmm7
32      pxor xmm7, xmm7
33
34      ; comparamos los words contra 0
35      pcmpltw xmm7, xmm0; xmm7 = [0>xmm0?, ...]
36
37      ; hago un and entre xmm7 y los 3
38      pand xmm7, xmm2; xmm7 = [0>xmm0?0:3, ...]
39      ; el resultado que queda es 0 cuando xmm0>=0
40      ; y 1111111... cuando es negativo
```

# Ejercicio completo

```
41      ; sumo la mascara al registro xmm0
42      paddw xmm0, xmm7
43      ; xmm0 queda con el valor original si era
44      ; no negativo, o el valor+3 si era negativo.
45
46      ; guardo el dato en memoria
47      movdqu [rdi + rcx*8 + (-8)], xmm0
48
49      ; itero el ciclo hasta que rcx sea 0
50      loop .ciclo
51
52      ; desarmo el stackframe
53      pop rbp
54      ret
```