SIMD

Empaquetado/Desempaqueado
Extensión de signo
Máximos y Mínimos
Operaciones horizontales

Iván Arcuschin
Organización del Computador II

10 de Septiembre de 2019

Contenidos

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. Asumimos que la longitud del array es múltiplo de 8.

Aridad de la función

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

Ejercicio completo

```
section .data:
        align 16 // Para que hago esto?
        mascara: times 8 DW 3

section .text:

; sumar(short* P, int len)
; RDI = P
; ESI = len
sumar:
        ; armo el stackframe
        push rbp
        mov rbp, rsp
```

Ejercicio completo

```
; muevo el largo al registro contador (C),
; y divido por 8
// por que estoy dividiendo por 8?
xor rcx, rcx
mov ecx, esi
shr rcx, 3

; muevo la mascara de 3s a un registro XMM
// por que muevo la mascara a un registro
// en vez de usarla directamente?
movdqa xmm2, [mascara]
// por que estoy usando MOVDQA y no MOVDQU?
```

Ejercicio completo

```
.ciclo:
    ; traemos el dato de la memoria
    movdqu xmm0, [rdi + rcx*8 + (-8)]
    // por que estoy usando MOVDQU y no MOVDQA?

    ; limpiamos el registro xmm7
    pxor xmm7, xmm7

    ; comparamos los words contra 0
    pcmpgtw xmm7, xmm0; xmm7 = [0>xmm0?, ...]

    ; hago un and entre xmm7 y los 3
    pand xmm7, xmm2; xmm7 = [0>xmm0?0:3, ...]
```

Ejercicio completo

```
        ; sumo la mascara al registro xmm0
        paddw xmm0, xmm7
        ; xmm0 queda con el valor original si era
        ; no negativo, o el valor+3 si era negativo.

        ; guardo el dato en memoria
        movdqu [rdi + rcx*8 + (-8)], xmm0

        ; itero el ciclo hasta que rcx sea 0
        loop .ciclo

    ; desarmo el stackframe
    pop rbp
    ret
```

Contenidos

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

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

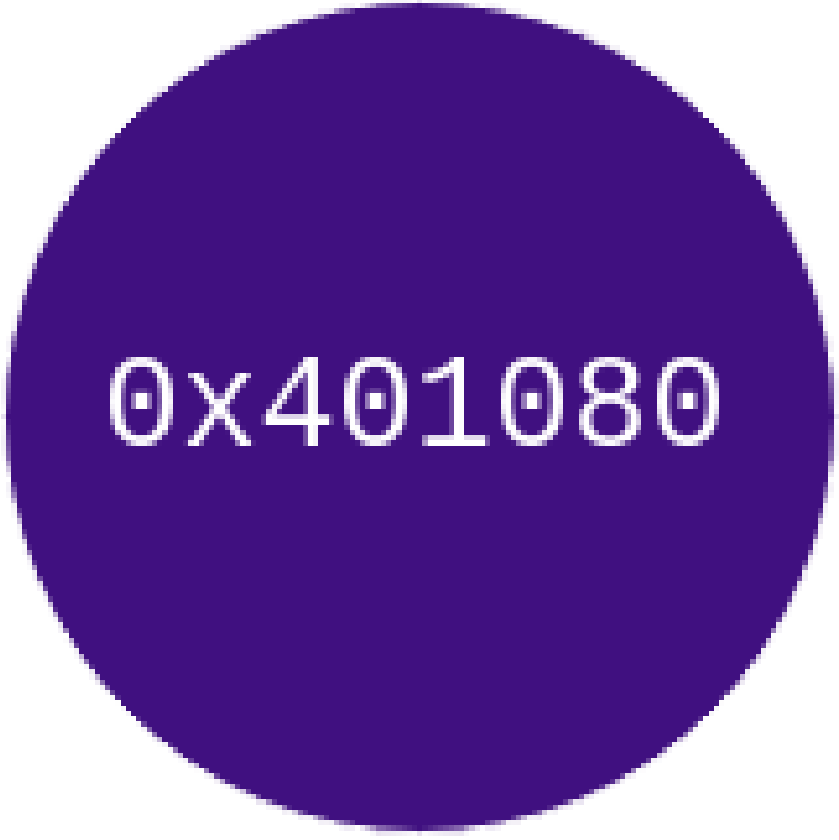
¡Cuidado!

¿Qué podría pasar con $(r + 2g + b)$?

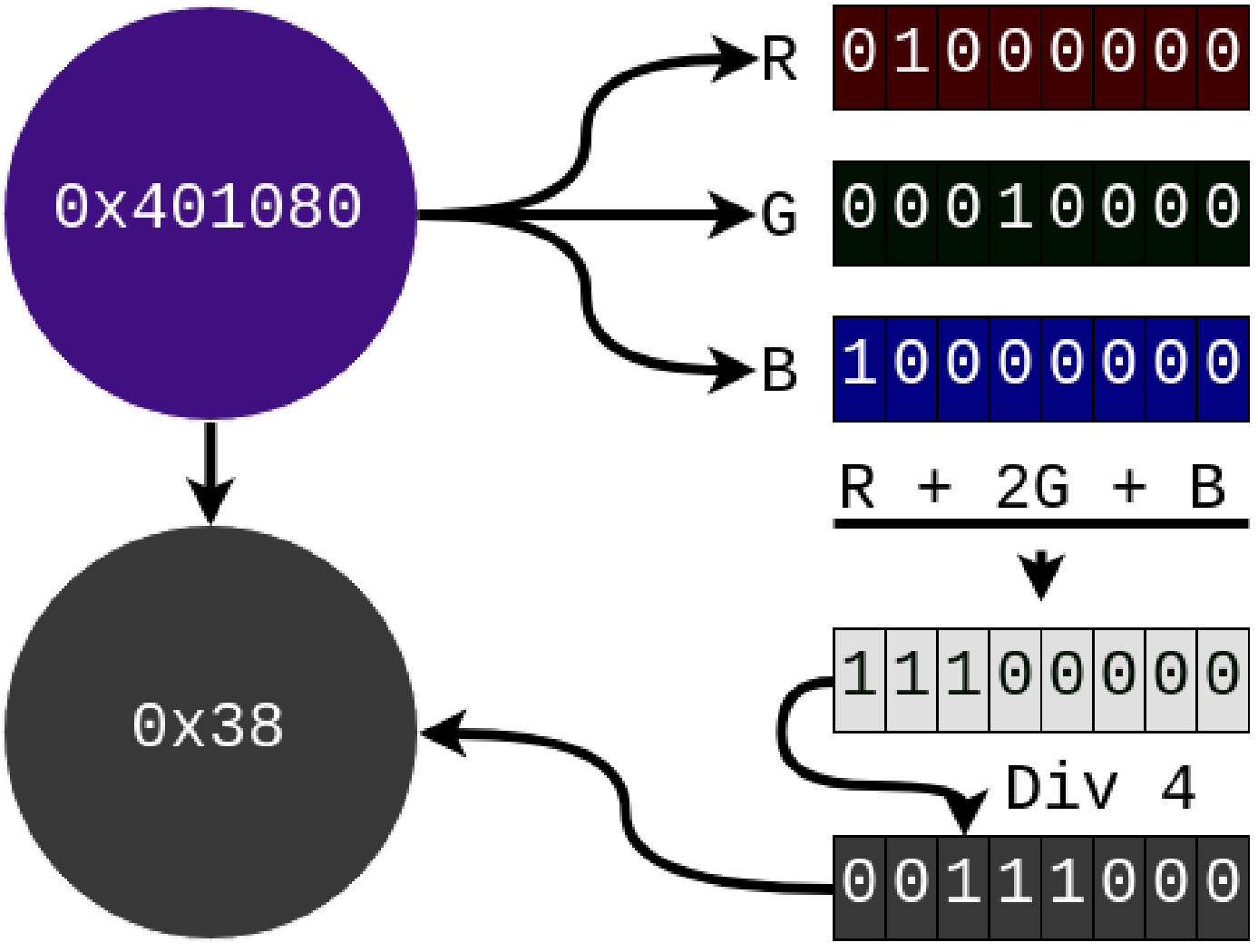
Motivación: Transformar RGB(0x401080) a grises

Ejemplo

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



Motivación: Transformar RGB(0x401080) a grises



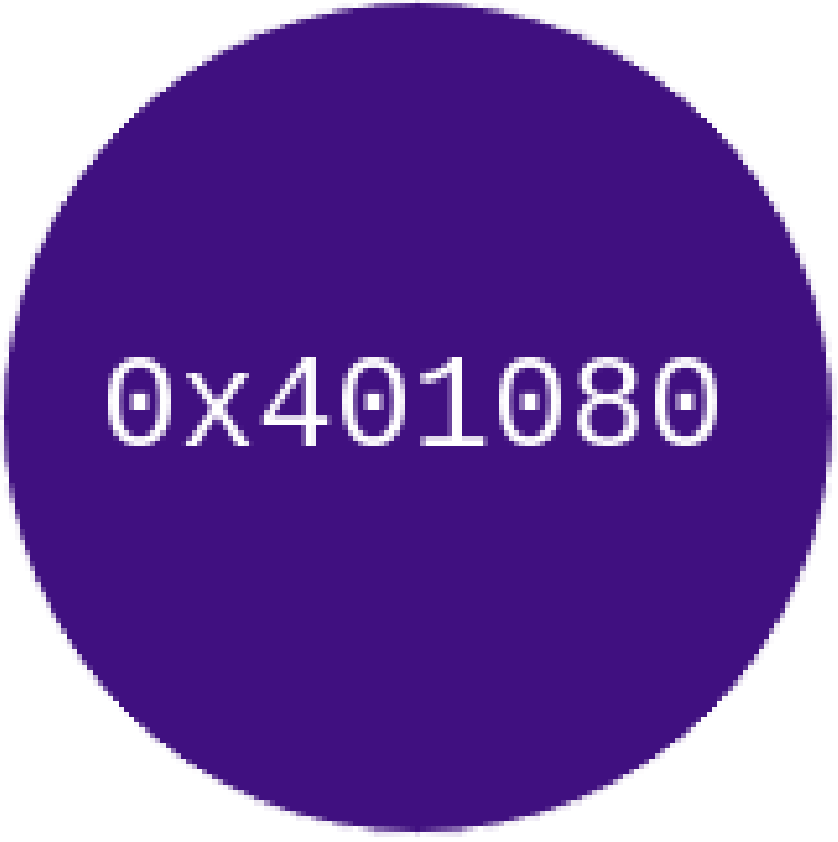
Motivación: Transformar RGB(0x40BF80) a grises

Ejemplo 2

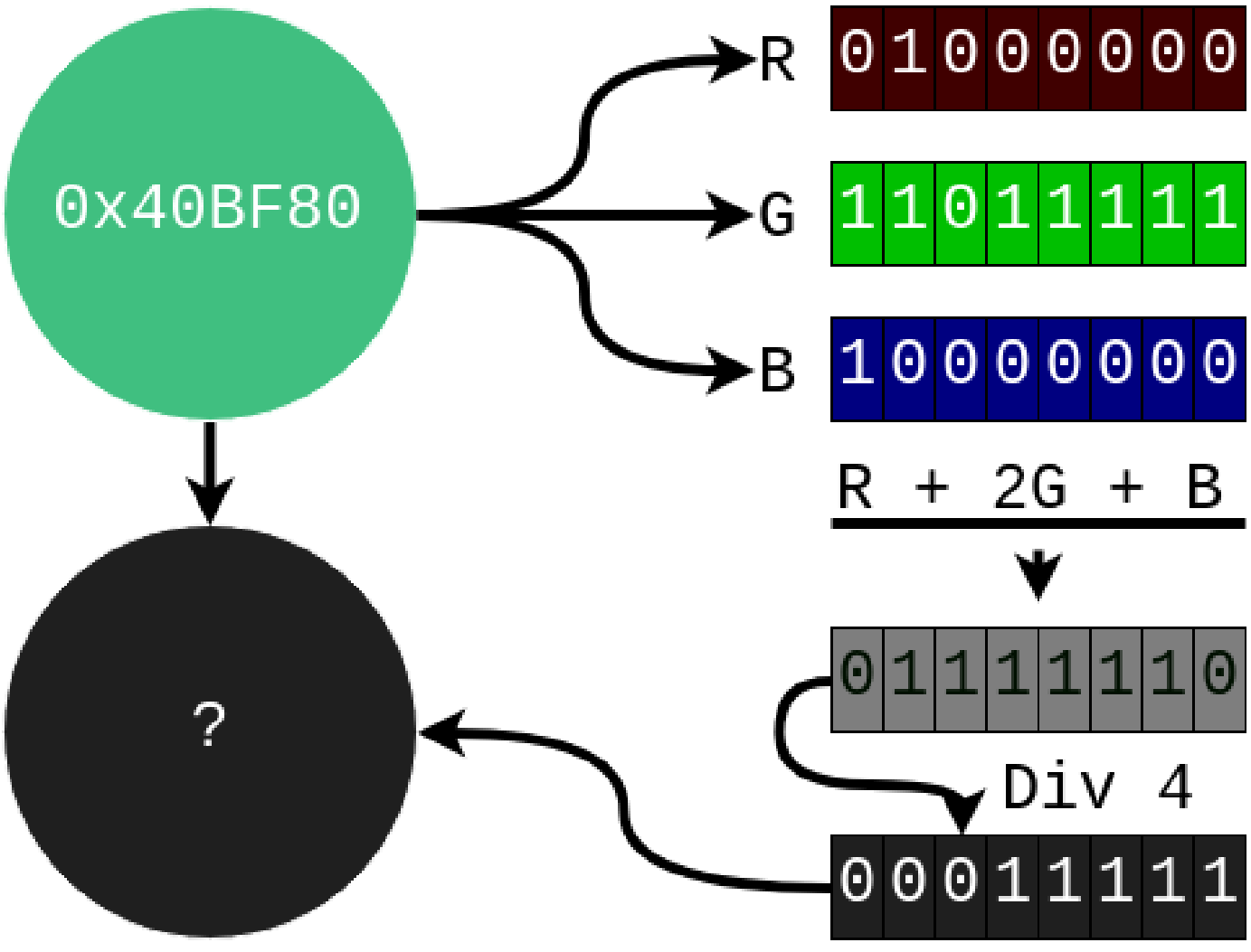
Pasar un pixel `RGB=(0x40, 0xBF, 0x80)` a escala de grises.

Motivación: Transformar RGB(0x40BF80) a grises

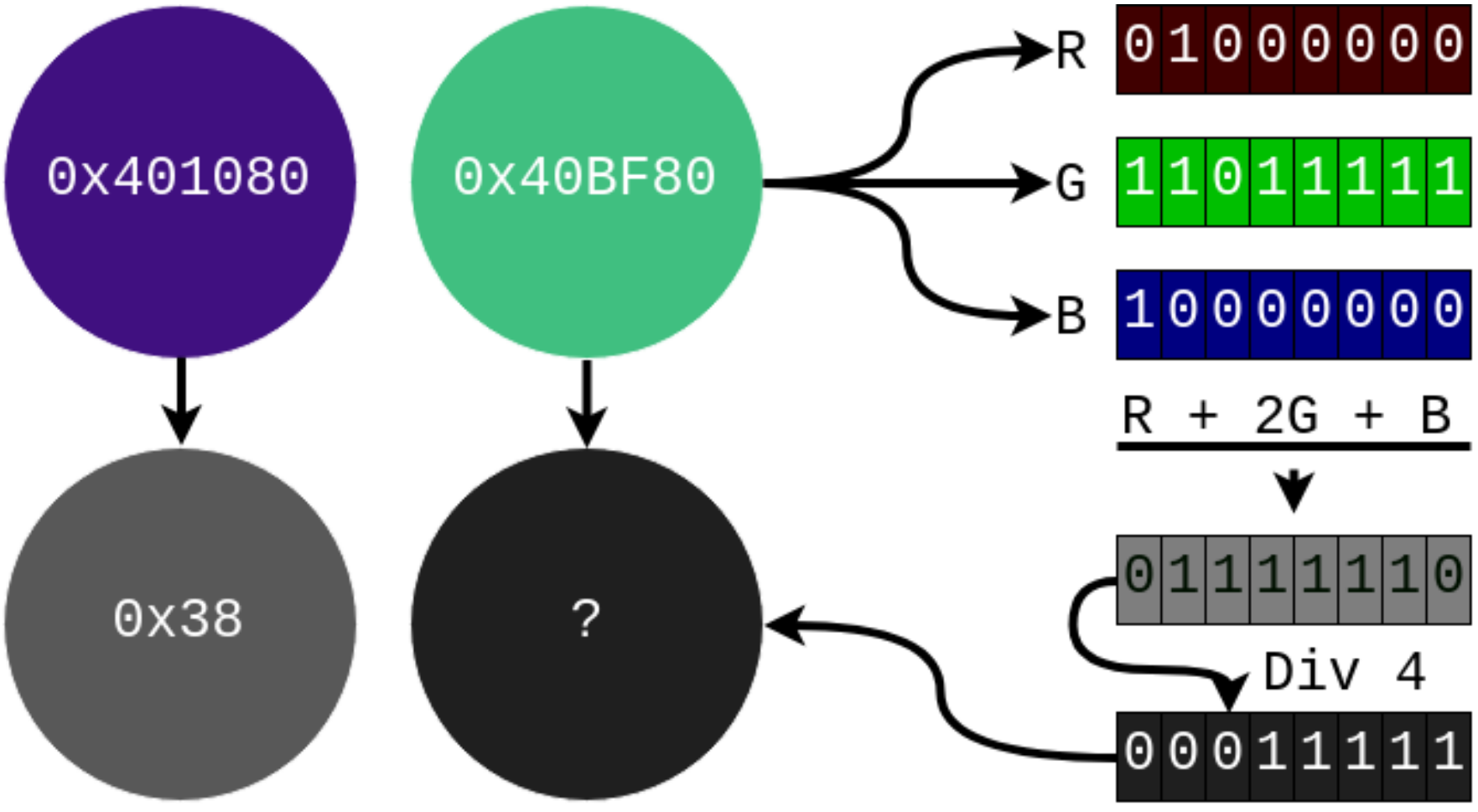
¿Cuál es más oscuro?



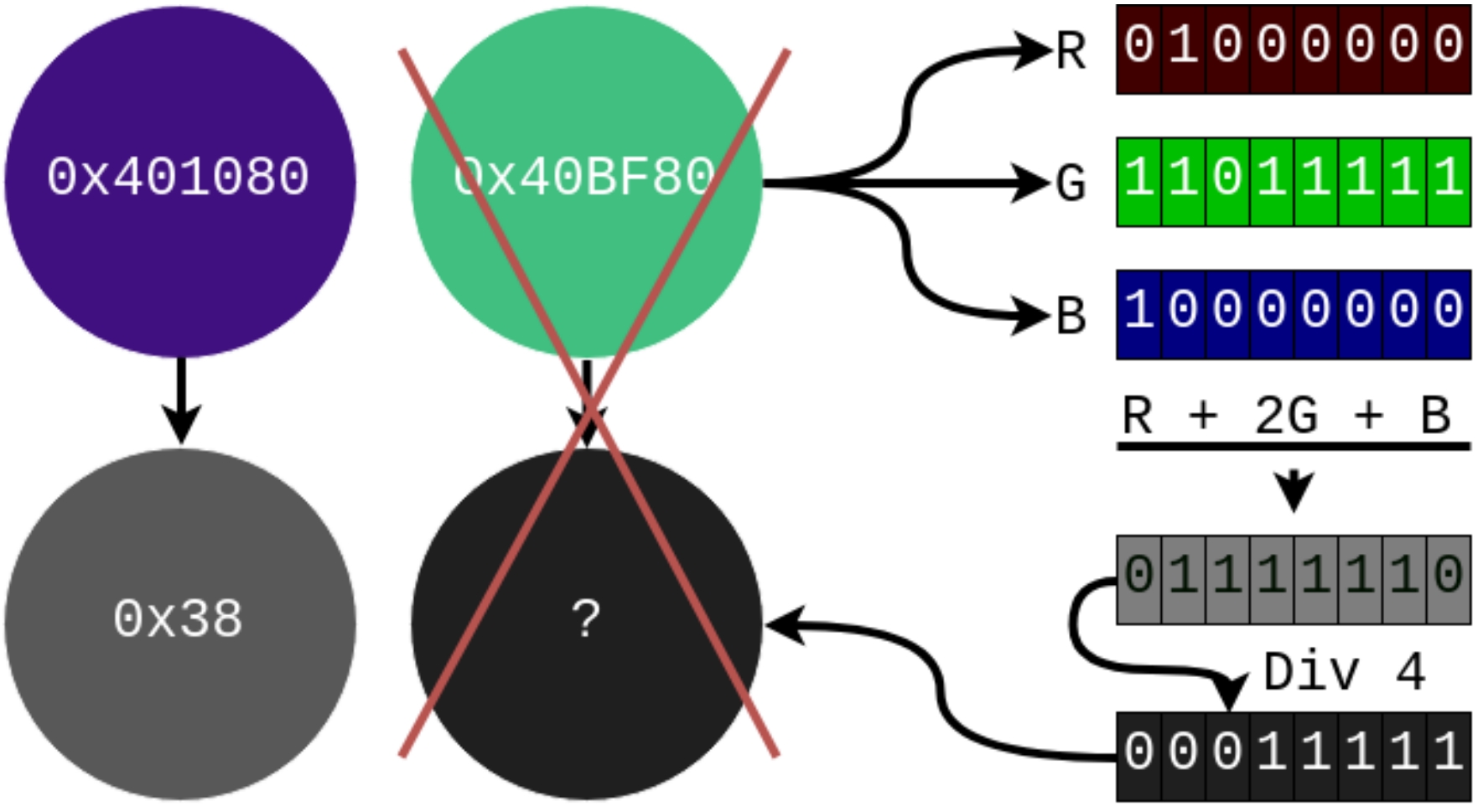
Motivación: Transformar RGB(0x40BF80) a grises



Motivación: Transformar RGB(0x40BF80) a grises



Motivación: Transformar RGB(0x40BF80) a grises



Motivación

¿Cuál es el problema?

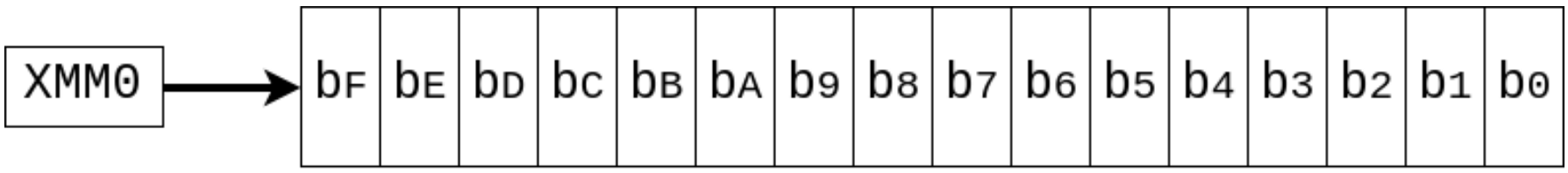
La representación utilizada (byte) no es suficiente para albergar los resultados intermedios. **Estamos perdiendo información.**

- 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 desempaquetado.

Empaquetamiento/Desempaquetamiento

Extensión de la representación

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

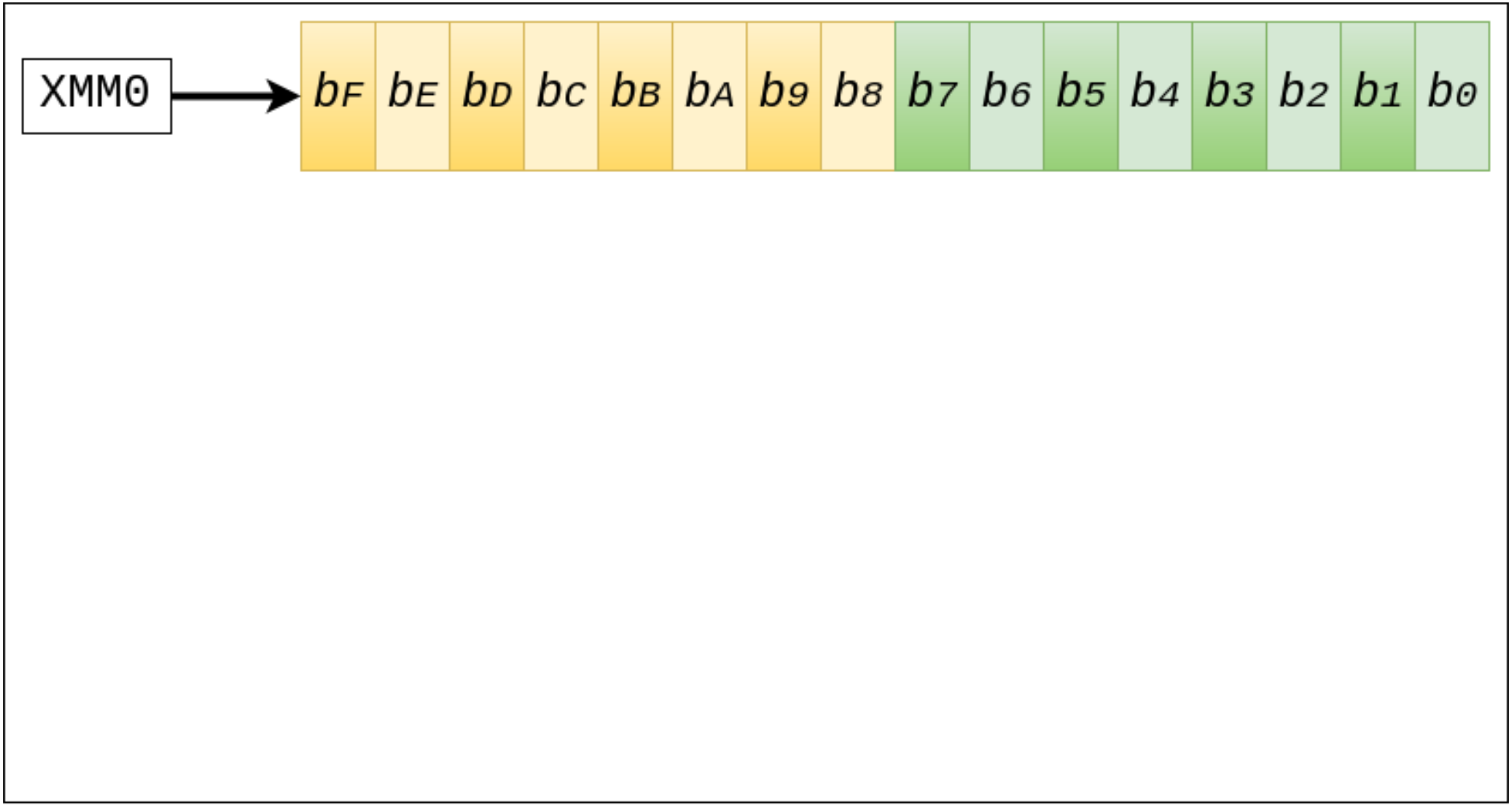


¿Cómo duplicamos la precisión o el tamaño de los datos?

Solamente necesitamos agregar ceros delante de cada número.

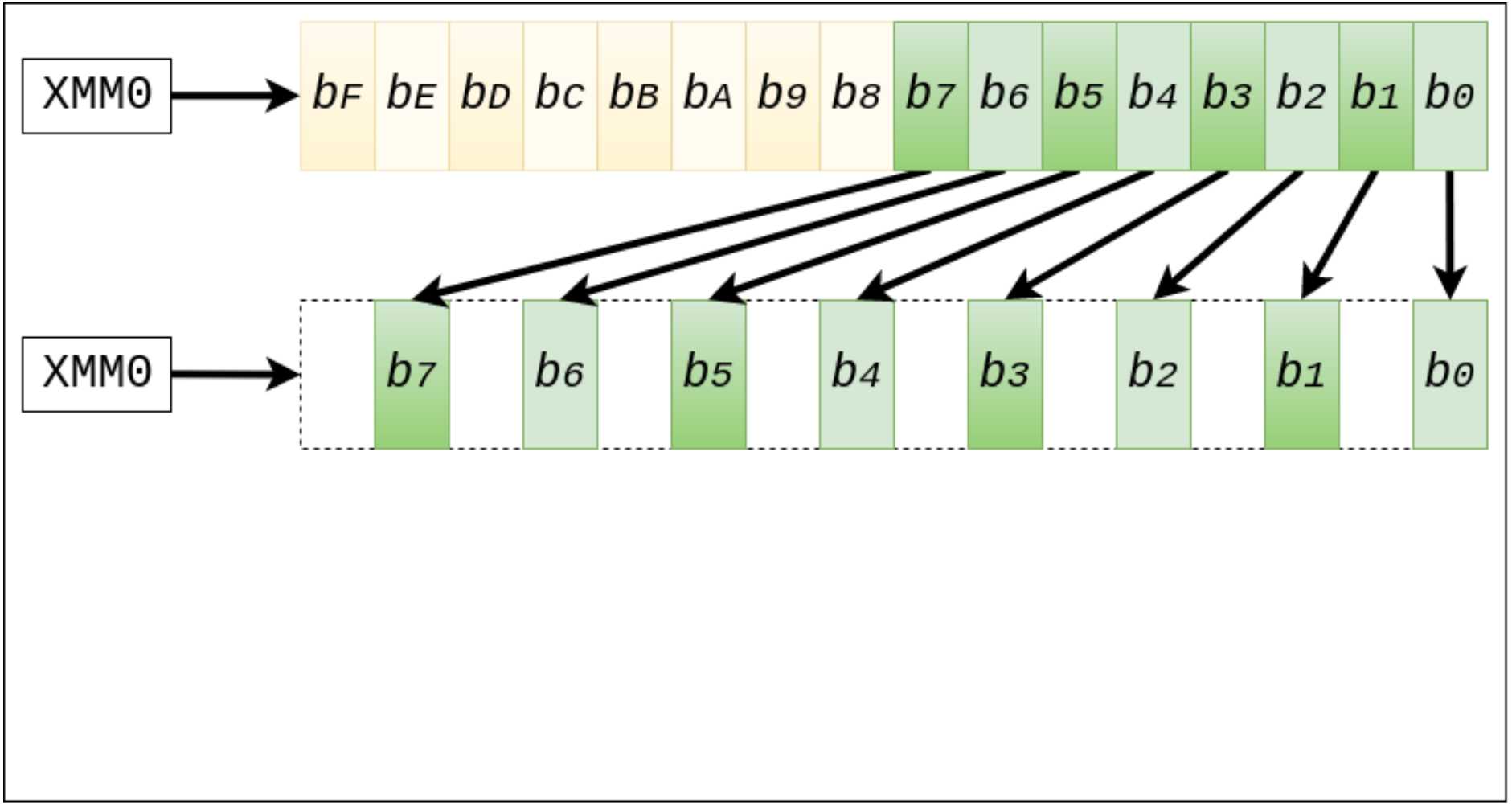
Desempaquetamiento

Desempaquetar parte baja de XMM0 (idea)



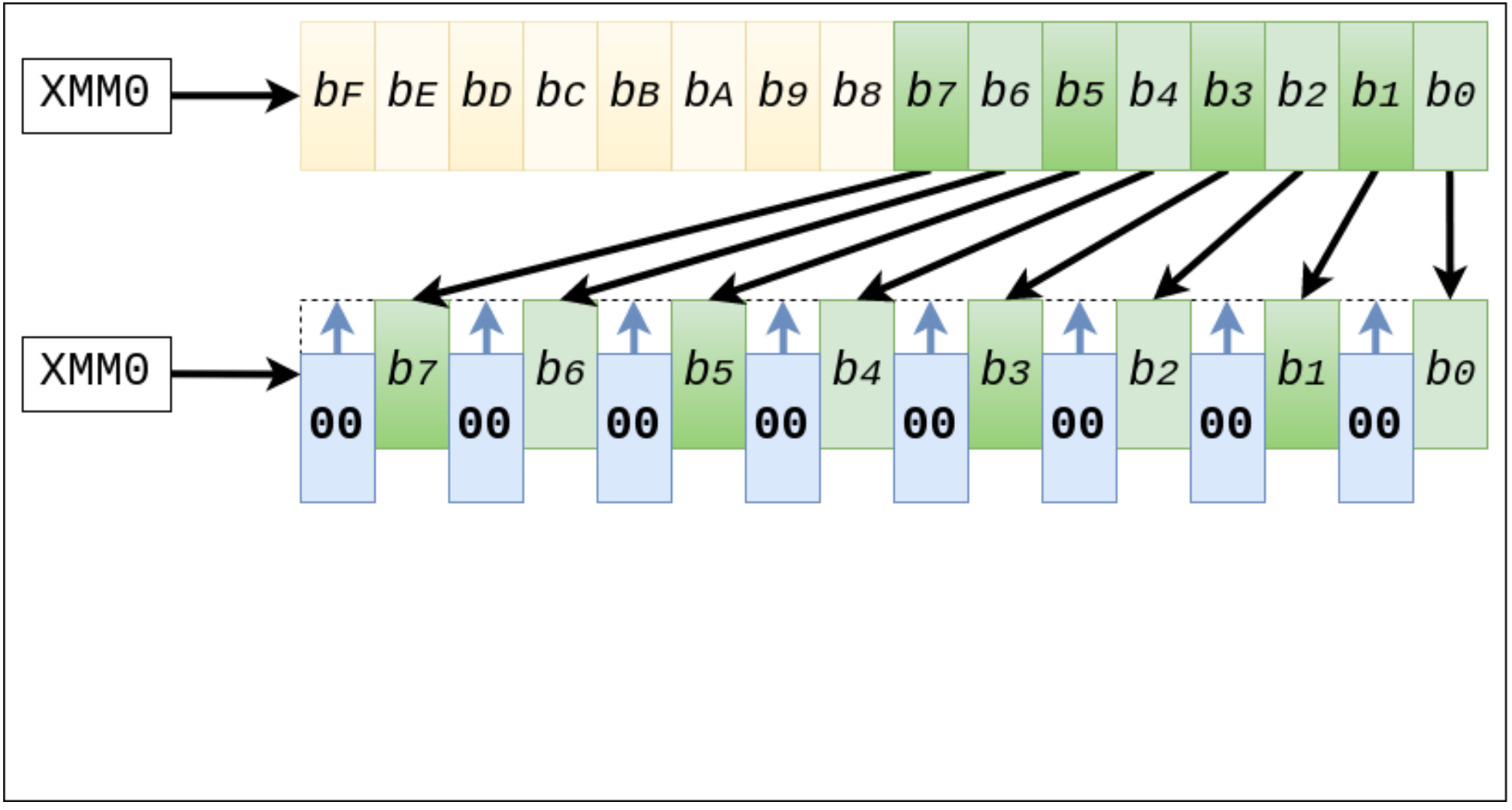
Desempaquetamiento

Desempaquetar parte baja de XMM0 (idea)



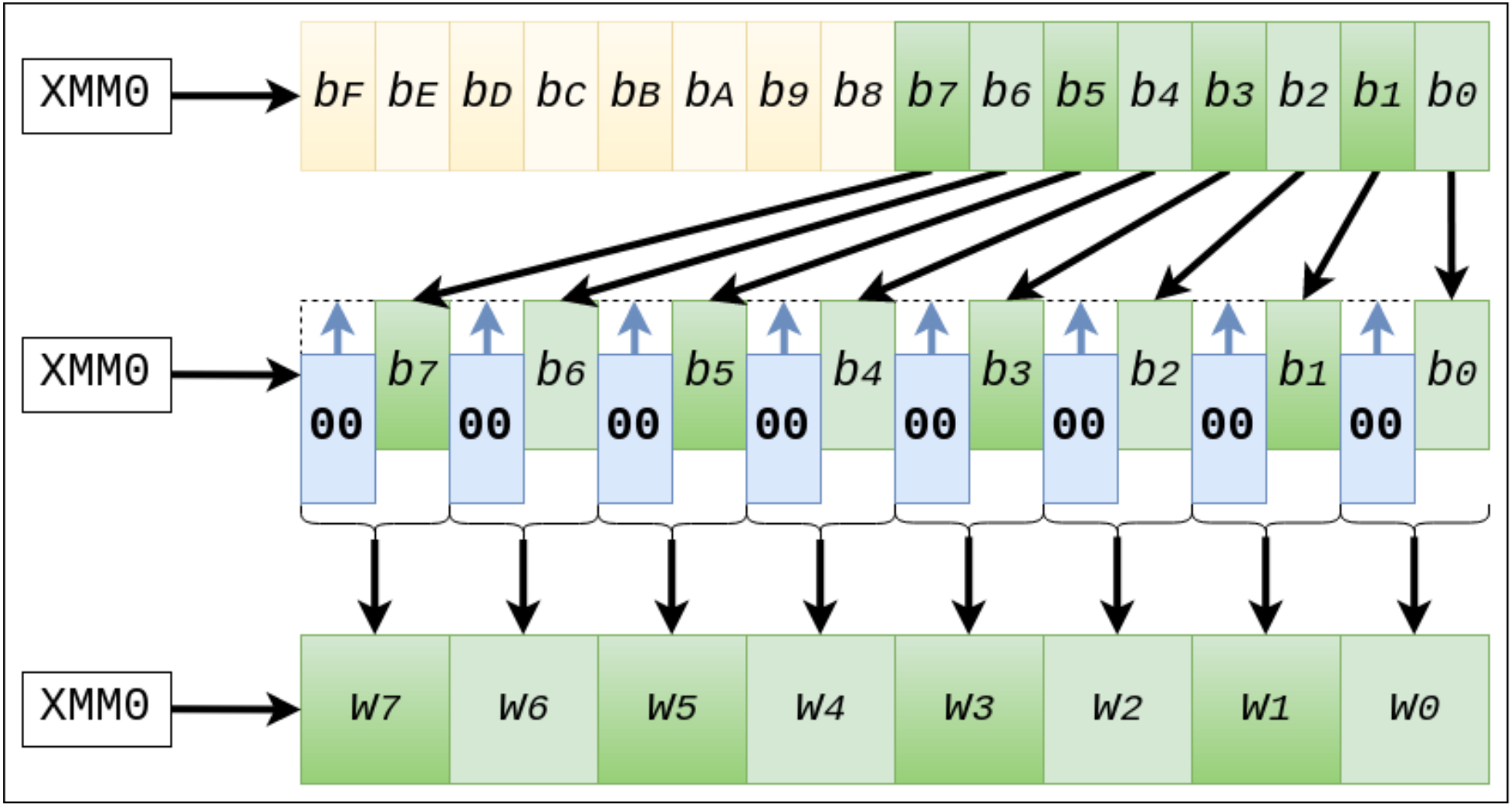
Desempaquetamiento

Desempaquetar parte baja de XMM0 (idea)



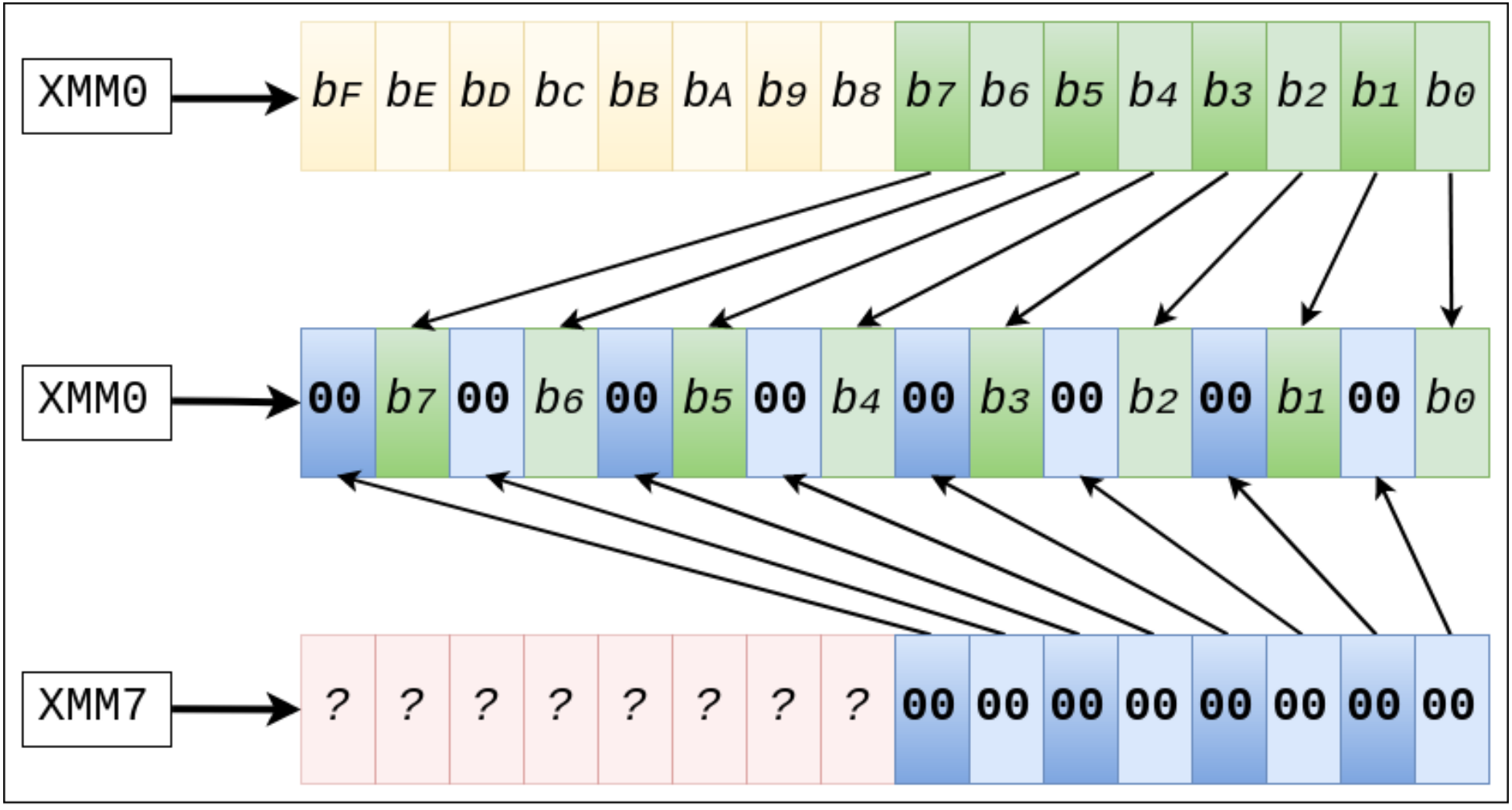
Desempaquetamiento

Desempaquetar parte baja de XMM0 (idea)



Desempaquetamiento

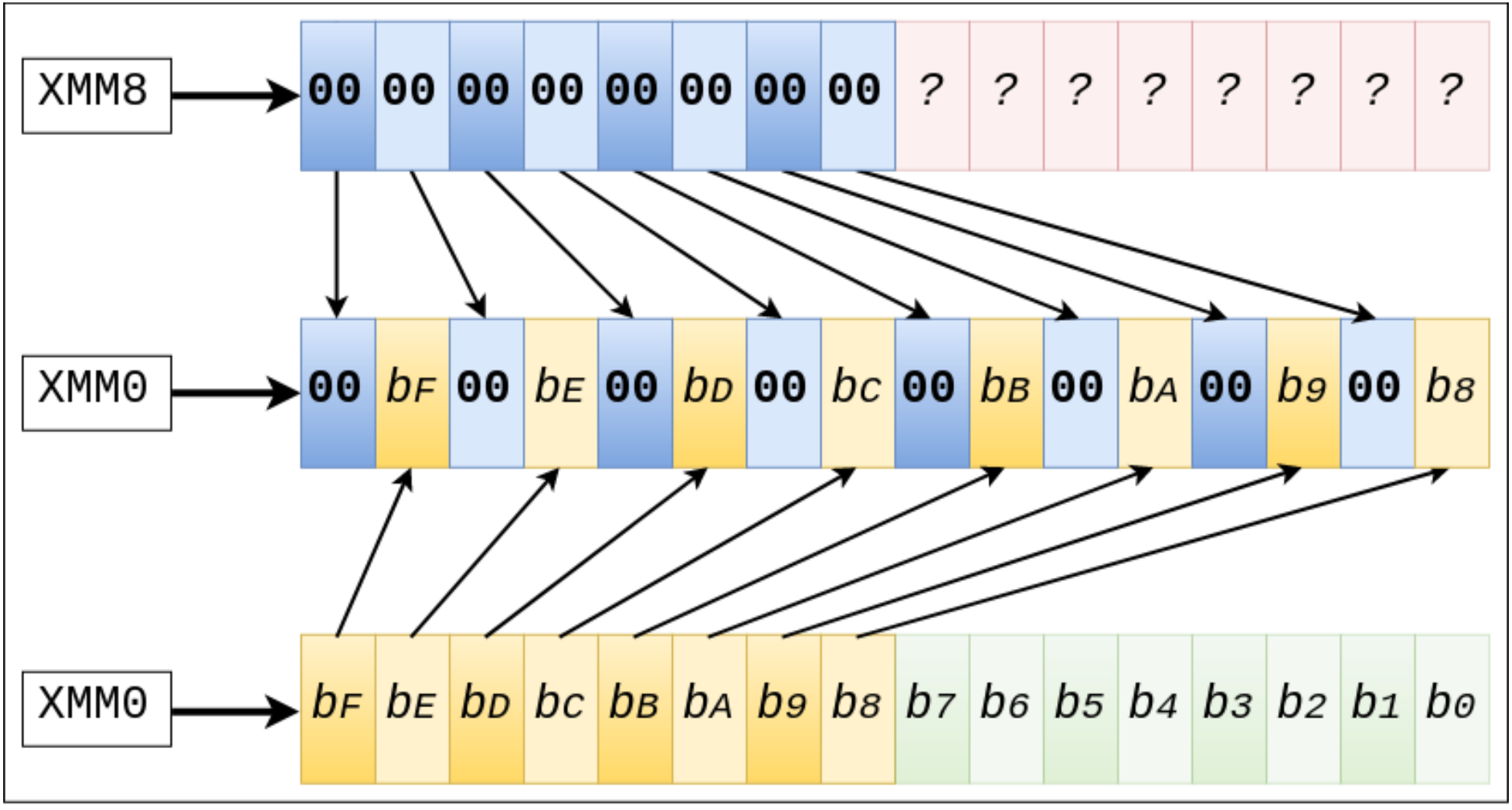
Desempaquetar parte **baja** de XMM0



`punpcklbw xmm0, xmm7`

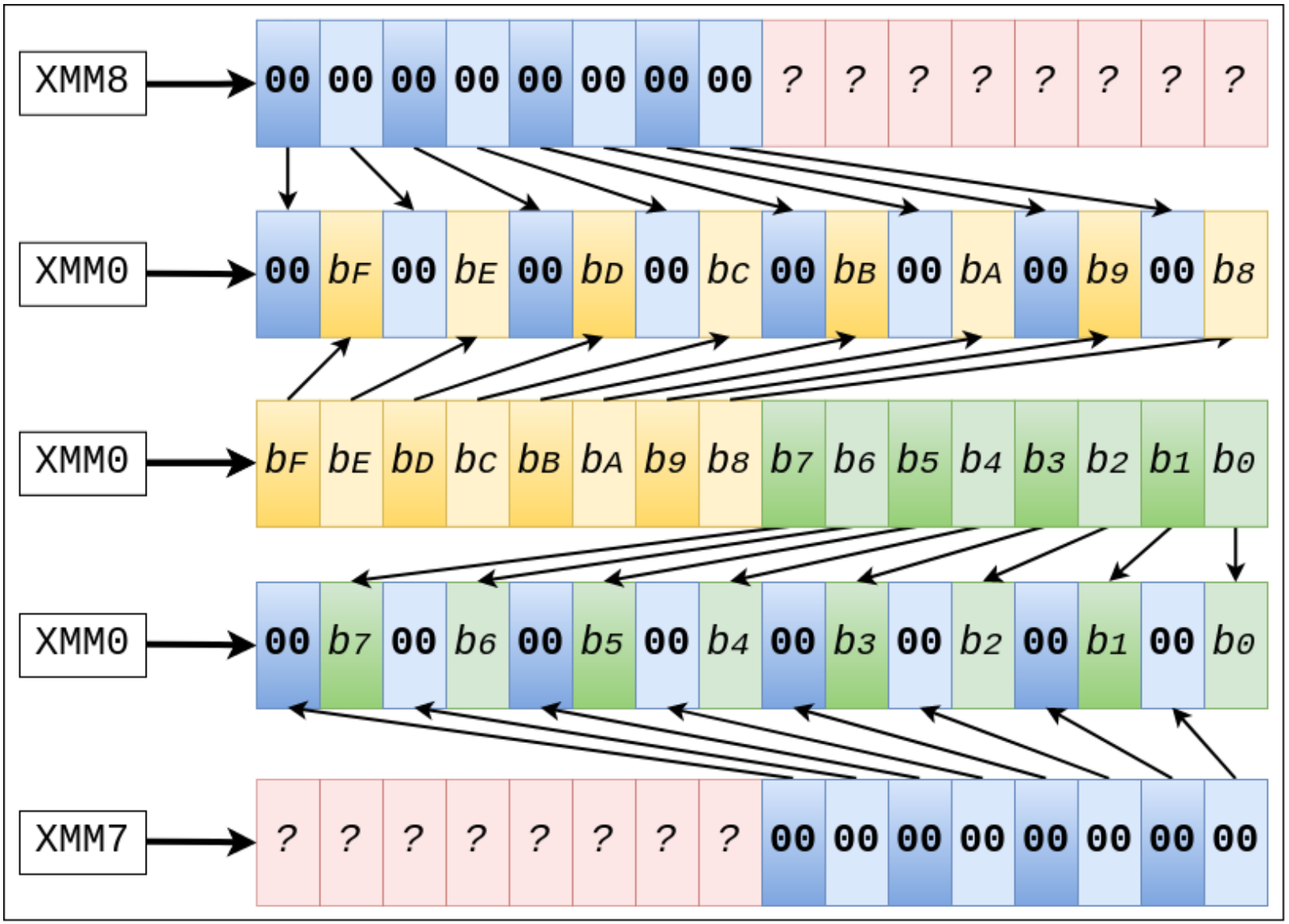
Desempaquetamiento

Desempaquetar parte **alta** de XMM0



`punpckhbw xmm0, xmm8`

Desempaquetamiento de ambas partes de XMM0



¿Está bien esto?

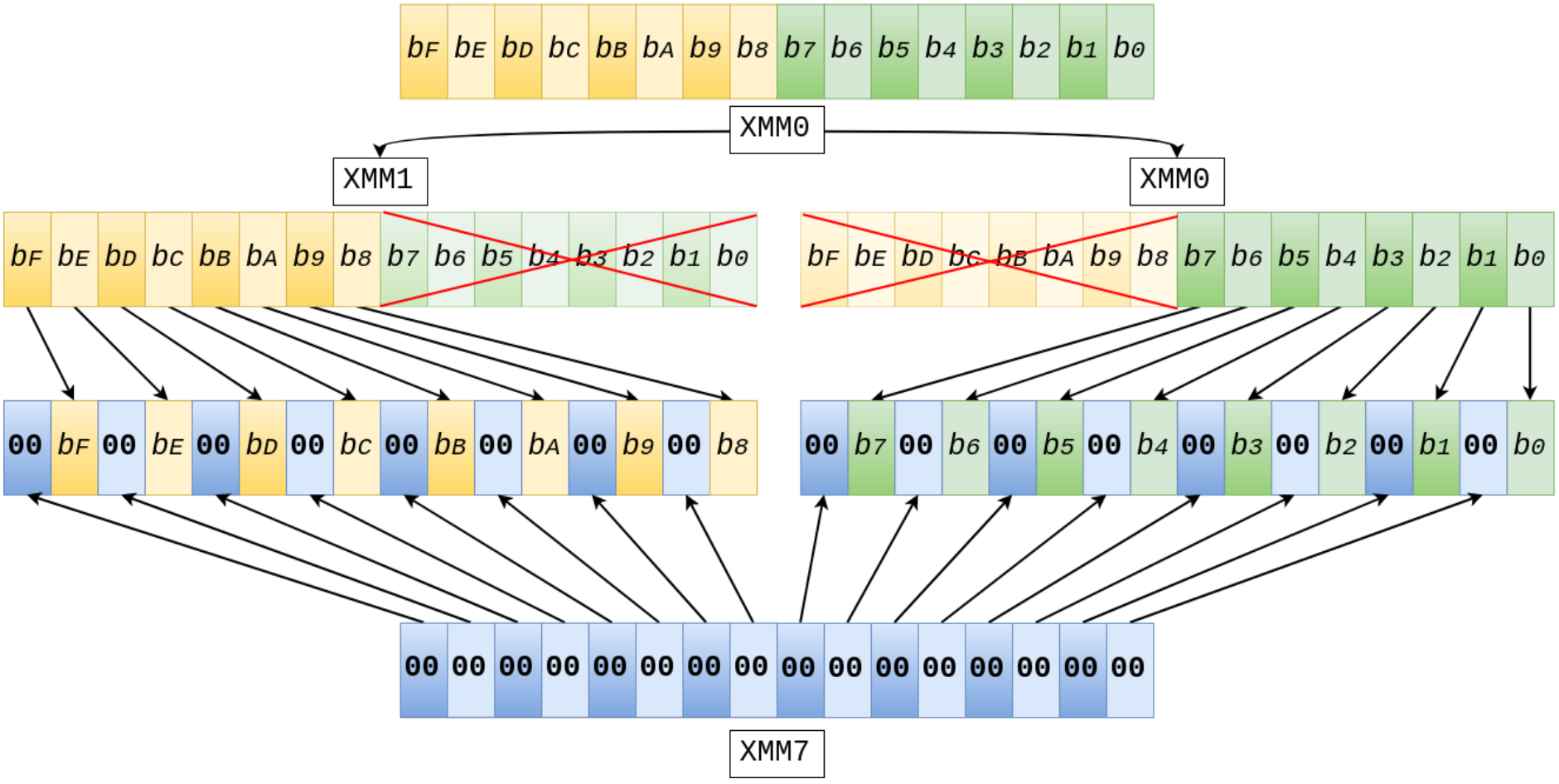
Desempaquetamiento de ambas partes de XMM0

¡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

$$\begin{aligned} \textbf{xmm0} &= 0 \,|\, a_7 \,|\, \ldots \,|\, 0 \,|\, a_0 \\ \textbf{xmm1} &= 0 \,|\, a_{15} \,|\, \ldots \,|\, 0 \,|\, a_8 \end{aligned}$$

```
; xmm0 = [a15, a14, ..., a1, a0]

pxor xmm7, xmm7; xmm7 = [0, 0, ..., 0]

movdqu xmm1, xmm0; xmm1 = [a15, a14, ..., a0]

punpcklbw xmm0, xmm7; xmm1 = [0, a7, ..., 0, a0]
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)

Sin saturación

(a) Original

(b) +50 de brillo

(c) +100 de brillo

(d) +150 de brillo

(e) +200 de brillo

(f) +250 de brillo
(¡La original!)

Con saturación

(g) Original

(h) +50 de brillo

(i) +100 de brillo

(j) +150 de brillo

(k) +200 de brillo

(l) +250 de brillo

Recapitulando...

Formato de instrucciones:

- **Desempaquetado:** *punpck* + *l/h* + *bw/wd/dq/qdq*
- **Empaquetado:** *pack* + *ss/us* + *wb/dw*

Importante, cosas a tener en cuenta

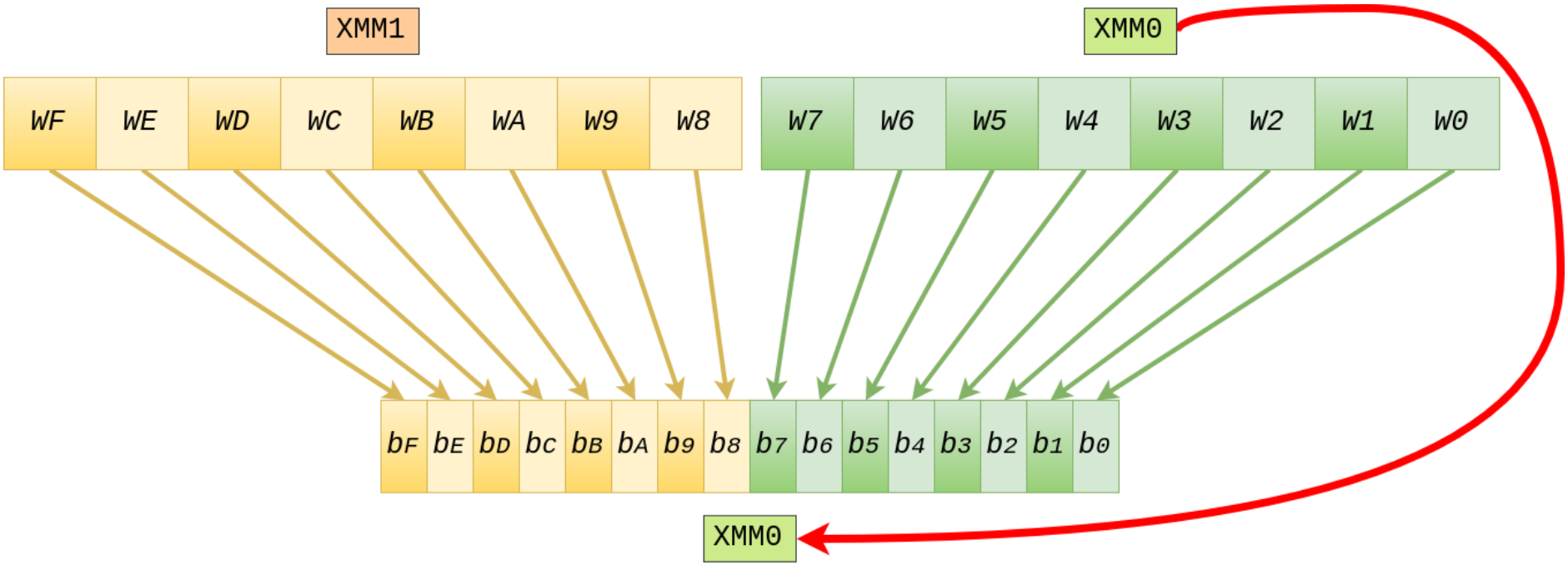
- ¿Queremos usar la parte alta o la parte baja?
- ¿De qué tipo son los datos?
- Entonces: ¿Qué tipo de saturación tengo que usar?

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

$$\begin{aligned} \textbf{xmm0} &= 0 \,|\, a_7 \,|\, \ldots \,|\, 0 \,|\, a_0 \\ \textbf{xmm1} &= 0 \,|\, a_{15} \,|\, \ldots \,|\, 0 \,|\, a_8 \end{aligned}$$

Luego...

packuswb xmm0, xmm1 ; xmm0 $= a_{15} \,|\, a_{14} \,|\, \ldots \,|\, a_1 \,|\, a_0$

Ahora cada dato es de tipo **byte**

Empaquetamiento/Desempaquetamiento

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.

Contenidos

Extensión de signo

Si necesitamos aumentar de tamaño la representación de un valor en un registro común y corriente, tenemos que moverlo del registro actual a otro más grande, teniendo cuidado de no romper el signo. Existen instrucciones específicas para esto:

- **MOVSX**: Acepta distintas variantes,
  byte a word/doubleword/quadword y
  word a doubleword/quadword.
  El valor en el registro de *src* se extiende dependiendo del bit más significativo. Si era un 1, se extiende con 1s, sino con 0s.
- **MOVSXD**: Igual a **MOVSX** pero para mover de doubleword a quadword.

Contenidos

Máximos y Mínimos

Si necesitamos encontrar el máximo o mínimo de cada entero empaquetado en dos registros `XMM`, podemos usar las instrucciones:

- **PMAXSB**/**PMAXSW**/**PMAXSD**/**PMAXSQ**: Compara los enteros empaquetados en *src* y *dst* (usando representación **con signo**) y devuelve el máximo de cada par en *dst*.
- **PMINSB**/**PMINSW**/**PMINSD**/**PMINSQ**: Idem anterior, pero devuelve el mínimo en vez del máximo.
- **PMAXUB**/**PMAXUW**/**PMAXUD**/**PMAXUQ**: Compara los enteros empaquetados en *src* y *dst* (usando representación **sin signo**) y devuelve el máximo de cada par en *dst*.
- **PMINUB**/**PMINUW**/**PMINUD**/**PMINUQ**: Idem anterior, pero devuelve el mínimo en vez del máximo.

Máximos y Mínimos

Si necesitamos encontrar el máximo o mínimo de cada float o double empaquetado en dos registros `XMM`, podemos usar las instrucciones:

- **MAXPS**/**MAXPD**: Compara los float/double empaquetados en *src* y *dst* y devuelve el máximo de cada par en *dst*.
- **MINPS**/**MINPD**: Idem anterior, pero devuelve el mínimo en vez del máximo.

Contenidos

Operaciones horizontales

Muchas veces necesitamos hacer operaciones entre los valores empaquetados de un **mismo** registro `XMM`. Hay instrucciones para eso:

- **PHADDW**/**PHADDD**: Suma los enteros empaquetados en *src* y *dst* de a pares y deja el resultado en *dst*. Del manual:

  PHADDW (with 128-bit operands)

  xmm1[15-0] = xmm1[31-16] + xmm1[15-0];
  xmm1[31-16] = xmm1[63-48] + xmm1[47-32];
  xmm1[47-32] = xmm1[95-80] + xmm1[79-64];
  xmm1[63-48] = xmm1[127-112] + xmm1[111-96];
  xmm1[79-64] = xmm2/m128[31-16] + xmm2/m128[15-0];
  xmm1[95-80] = xmm2/m128[63-48] + xmm2/m128[47-32];
  xmm1[111-96] = xmm2/m128[95-80] + xmm2/m128[79-64];
  xmm1[127-112] = xmm2/m128[127-112] + xmm2/m128[111-96];

- **PHSUBW**/**PHSUBD**: Idem anterior, pero resta en vez de sumar.

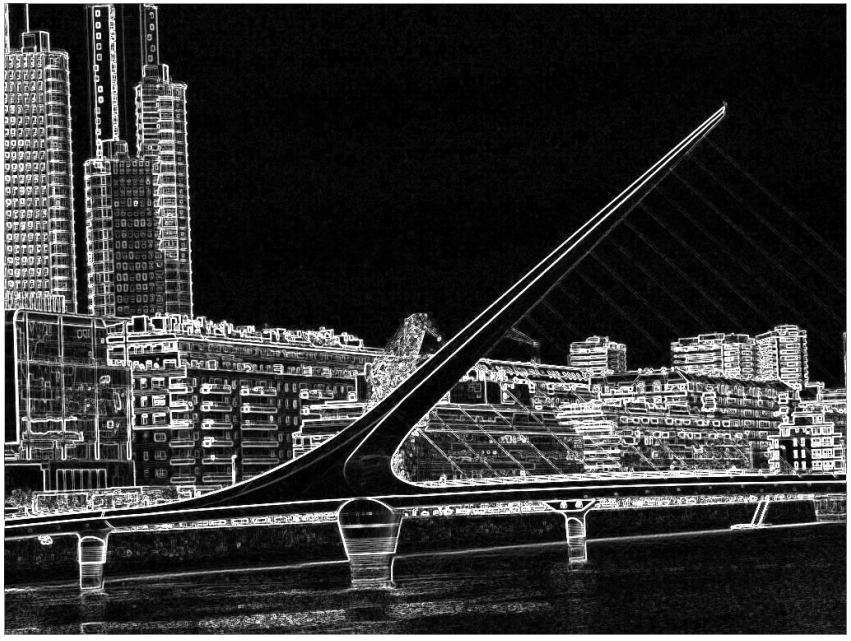
Operaciones horizontales

Otras instrucciones que pueden ser útiles:

- **PHADDSW**: Idem **PHADDW**, pero satura el resultado (en representación **con signo**).
- **PHMINPOSUW**: Determina el mínimo unsigned word en *src*. Deja su valor en el word menos significativo de *dst* y su índice en el segundo word menos significativo de *dst*.

Contenidos

Ejercicio: Edge



Ejercicio: Edge

Enunciado del filtro.

Dibuja un borde en los píxeles en donde la intensidad de la imagen cambia de forma abrupta. Vamos a usar en este caso el operador de Laplace.

$$\mathbf{M} = \begin{array}{|c|c|c|} \hline 0.5 & 1 & 0.5 \\ \hline 1 & -6 & 1 \\ \hline 0.5 & 1 & 0.5 \\ \hline \end{array}$$

$$dst(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} M(i,j) * src(x+i, y+j)$$

Prototipo de la función

```
void edge (uchar *src, uchar *dst, int width, int height)
```

Ejercicio: Edge (versión en C)

Versión en C

```c
#include <stdio.h>
#include "../tp2.h"

#define esq(x) (x>>1)
#define lat(x) (x)
#define med(x) ((-6)*x)
#define min(x,y) ((x>y)?(y):(x))
#define max(x,y) ((x>y)?(x):(y))

void edge_c (uchar *src, uchar *dst, int width, int height)
{
    // Casteo de tipo arreglo (char[]) a tipo Matriz (uchar[])[width]
    uchar (*sm)[width] = (uchar (*)[src_row_size]) src;
    uchar (*dm)[width] = (uchar (*)[dst_row_size]) dst;

    for (int i = 1; i < height - 1; i++) {
        for (int j = 1; j < width - 1; j++) {
            short px = esq(sm[i-1][j-1]) + lat(sm[i-1][ j ]) + esq(sm[i-1][j+1]) +
                       lat(sm[ i ][j-1]) + med(sm[ i ][ j ]) + lat(sm[ i ][j+1]) +
                       esq(sm[i+1][j-1]) + lat(sm[i+1][ j ]) + esq(sm[i+1][j+1]) ;
            dm[i][j] = min(max(px, 0), 255);
        }
    }
}
```

Ejercicio: Edge

Mirando la matriz dentro de la imagen

A_0	B_0	C_0	D_0	E_0	F_0	G_0
A_1	B_1	0.5	1	0.5	F_1	G_1
A_2	B_2	1	-6	1	F_2	G_2
A_3	B_3	0.5	1	0.5	F_3	G_3
A_4	B_4	C_4	D_4	E_4	F_4	G_4

Mirando la matriz dentro de la imagen (en memoria)

A_0 B_0 C_0 D_0 E_0 F_0 G_0 A_1 B_1 0.5 1 0.5 F_1 G_1 A_2 B_2 1 -6 1 F_2 G_2
A_3 B_3 0.5 1 0.5 F_3 G_3 A_4 B_4 C_4 D_4 E_4 F_4 G_4

Mirando la matriz dentro de la imagen (siguiente pixel)

A_0 B_0 C_0 D_0 E_0 F_0 G_0 A_1 B_1 C_1 0.5 1 0.5 G_1 A_2 B_2 C_2 1 -6 1 G_2
A_3 B_3 C_3 0.5 1 0.5 G_3 A_4 B_4 C_4 D_4 E_4 F_4 G_4