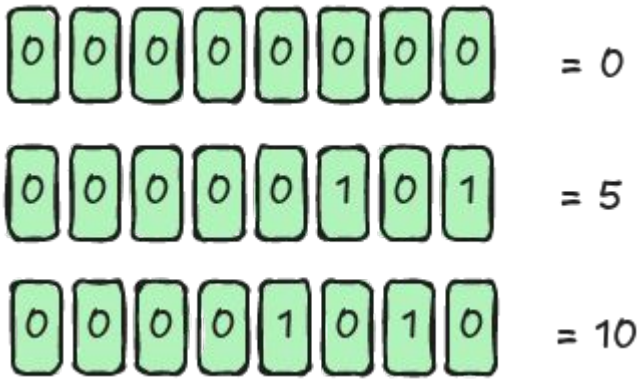
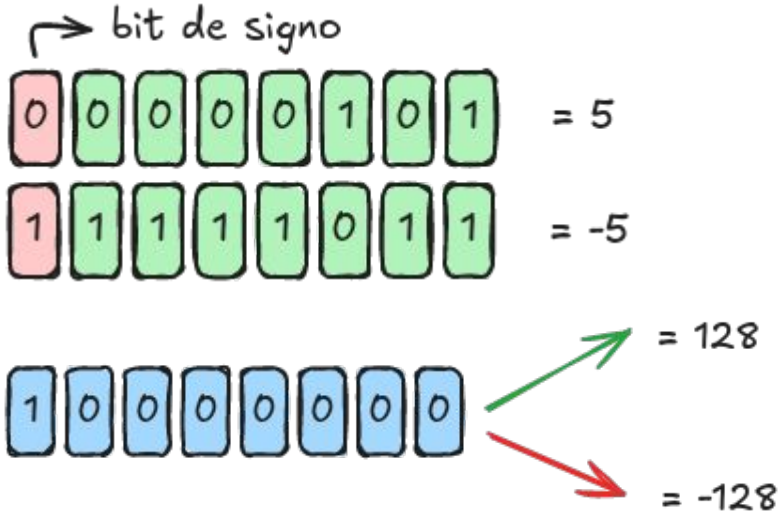
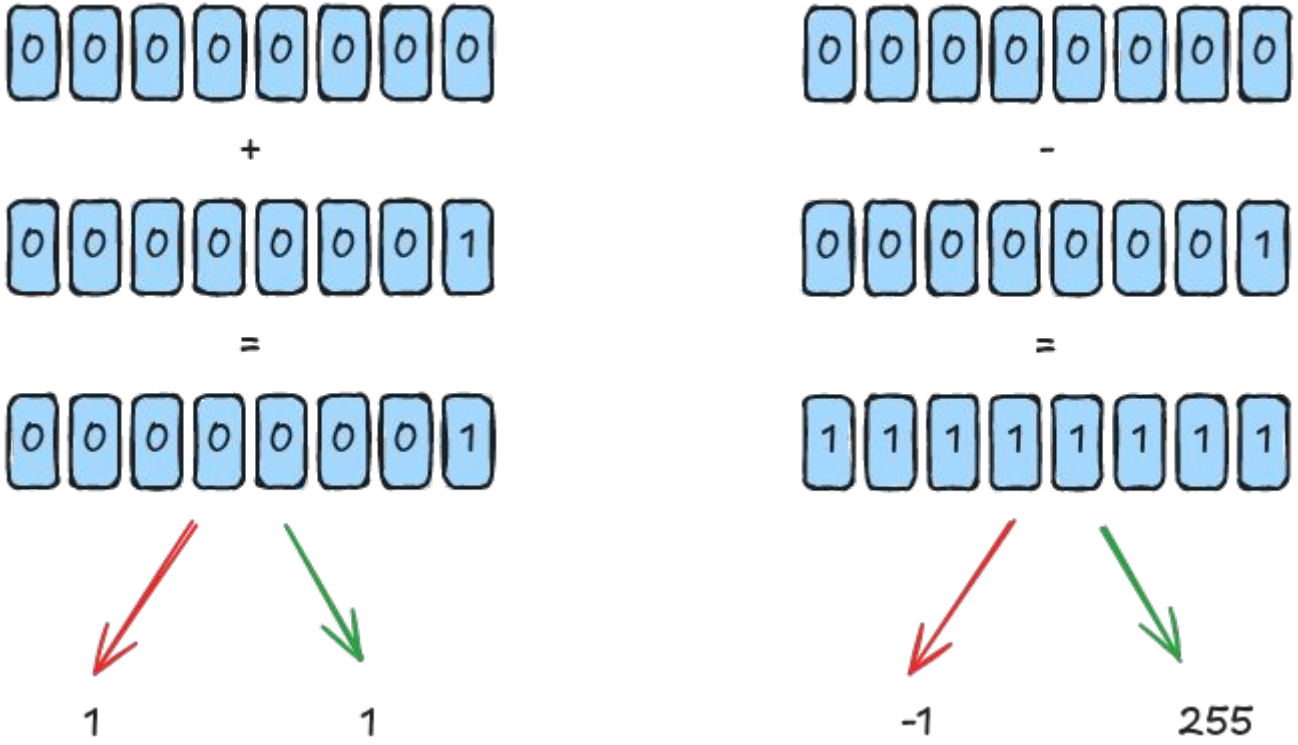
Representación de datos y paradigma SIMD

Repaso de enteros...

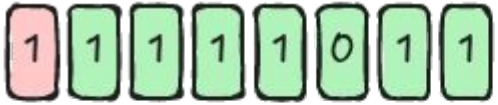




¡Ojo con como interpretan los datos!

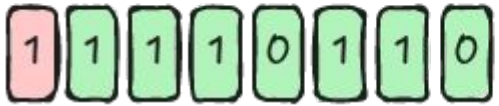


Ojo que hay números que se interpretan igual, los tests pueden dar bien para estos casos...

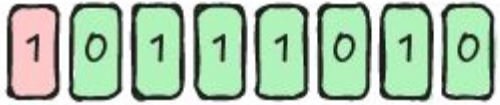


= -5

<< 1

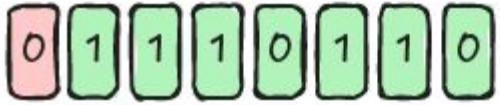


= (-5) * 2 = - 10



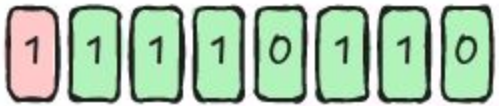
= -70

<< 1



= 118 ?

(-70) * 2 = - 140 ¡Fuera de rango!

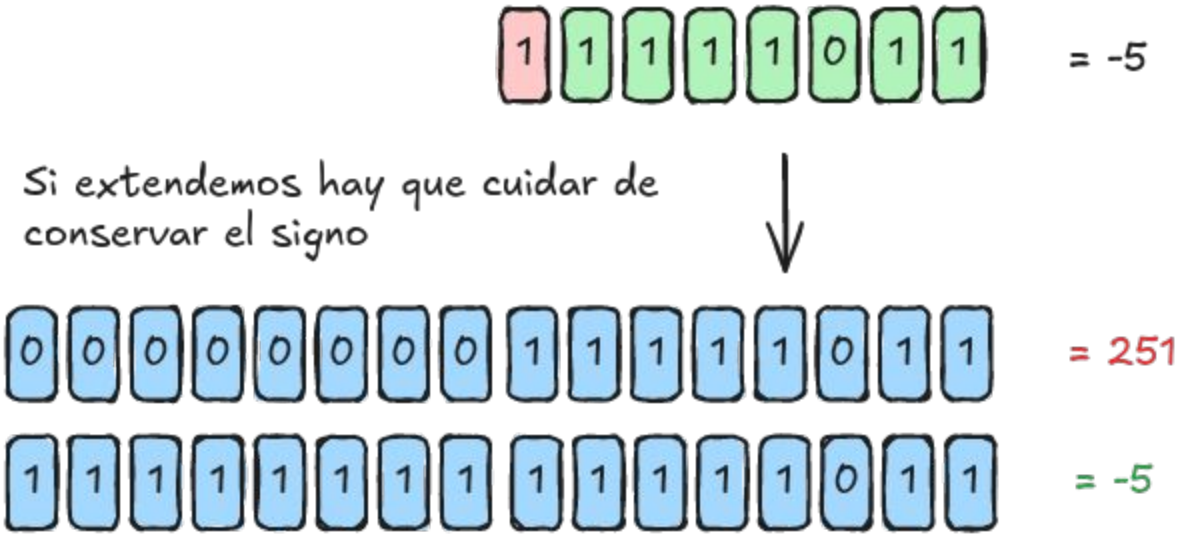


= -10

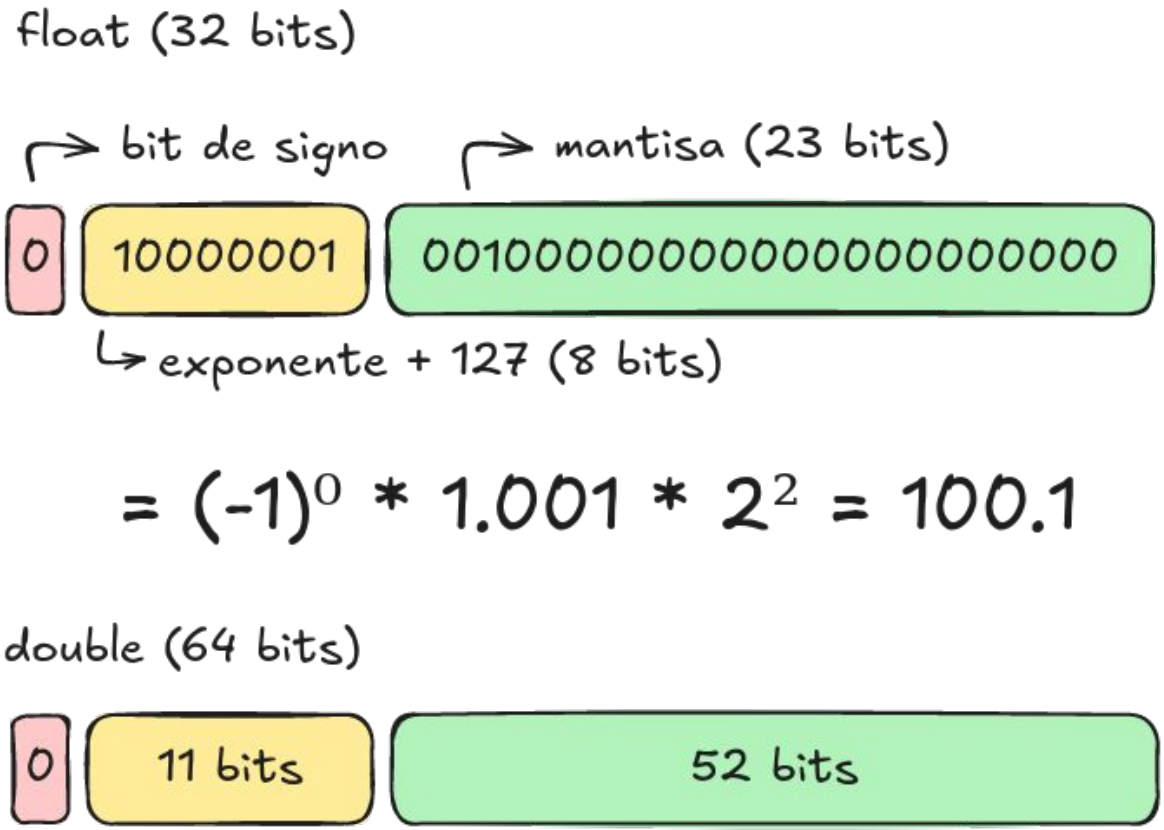
>> 1

= (-10) / 2 = 123 ?!

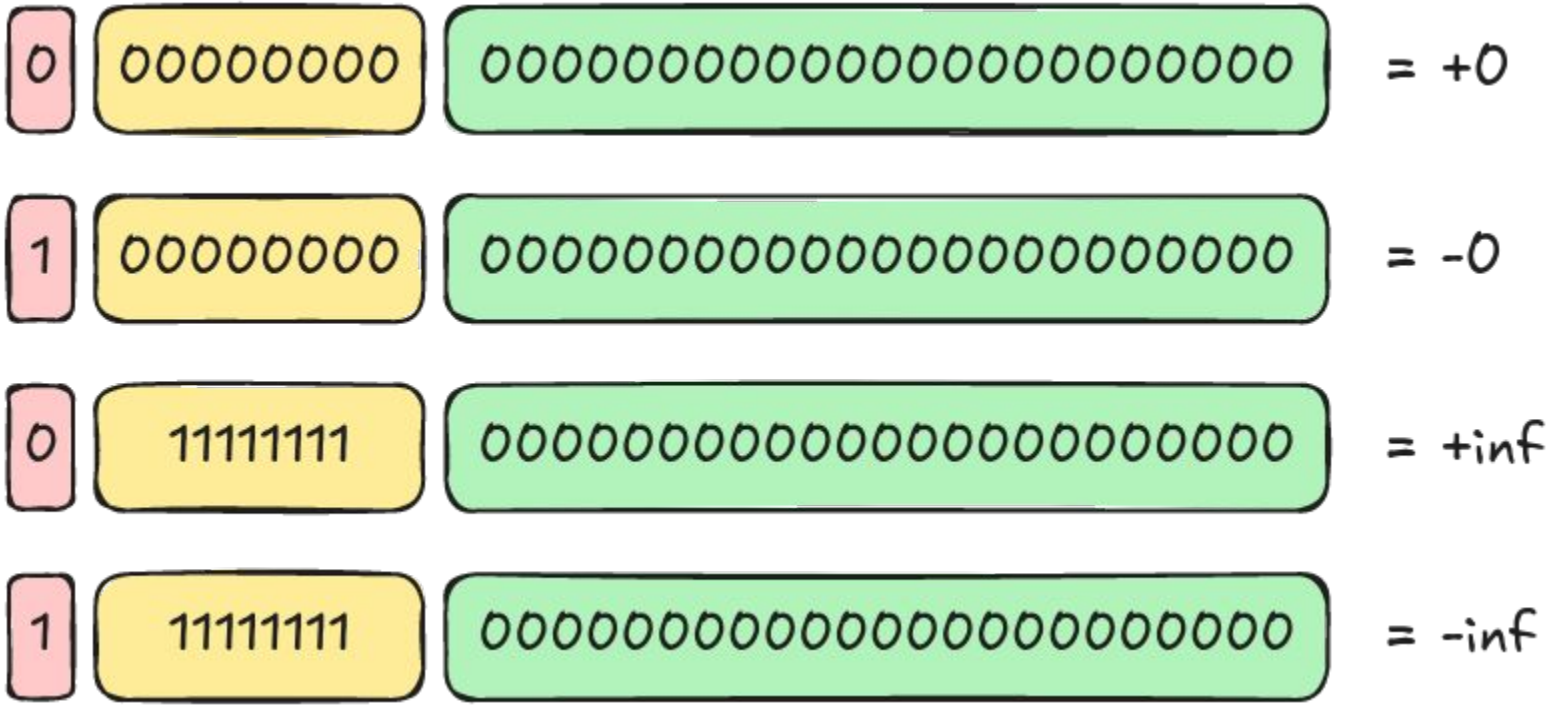
¡Ojo con la diferencia entre shift lógico y aritmético cuando es a derecha!


1 1 1 1 1 0 1 1
= -5
Si extendemos hay que cuidar de conservar el signo
0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1
= 251
1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
= -5

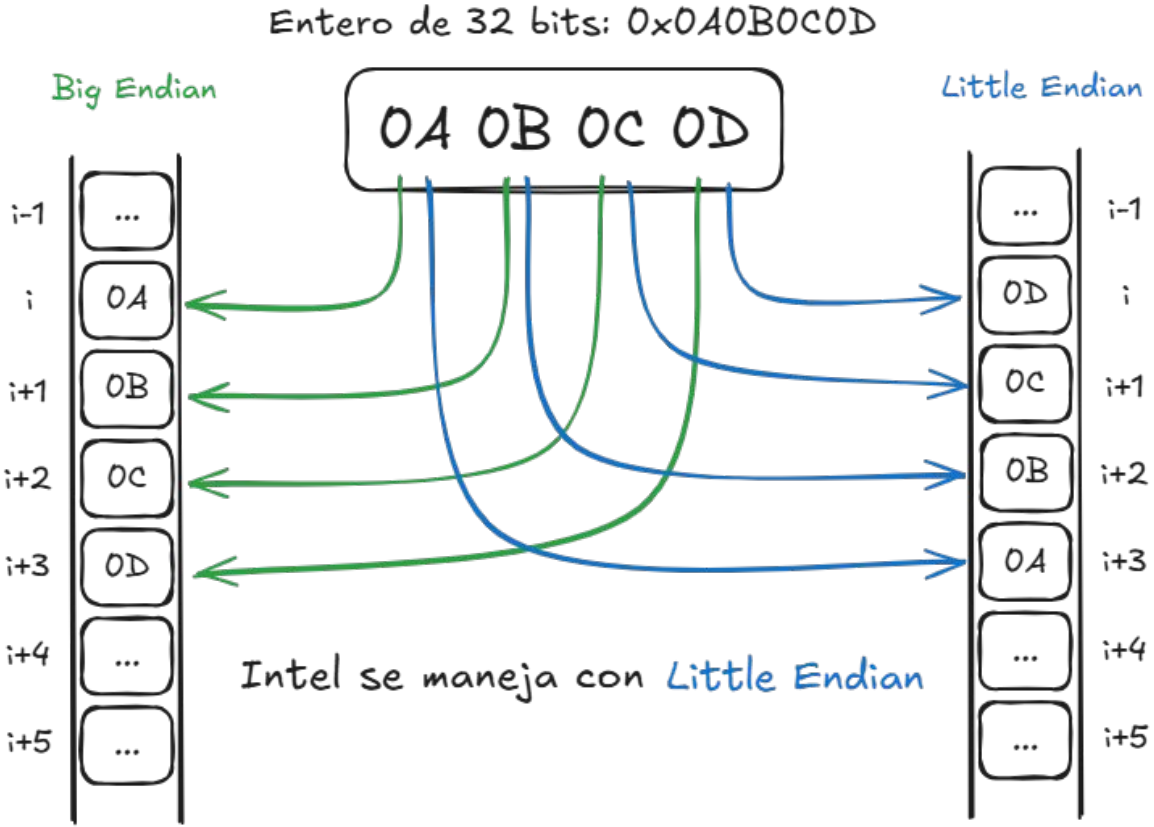
Representación de números de punto flotante:



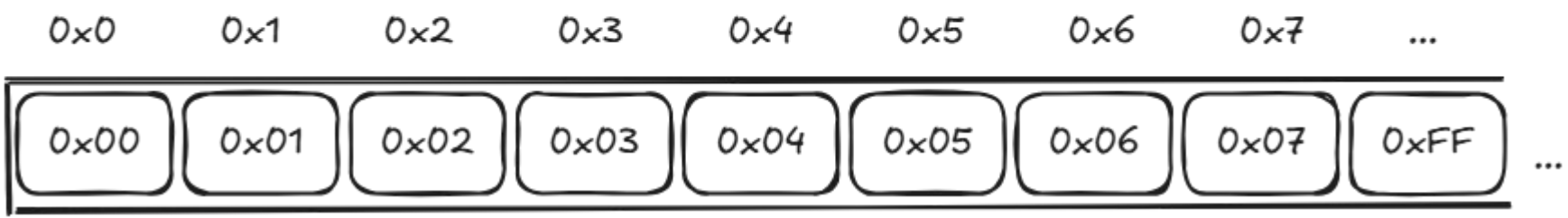
Patrones especiales:



Representación de datos en memoria:

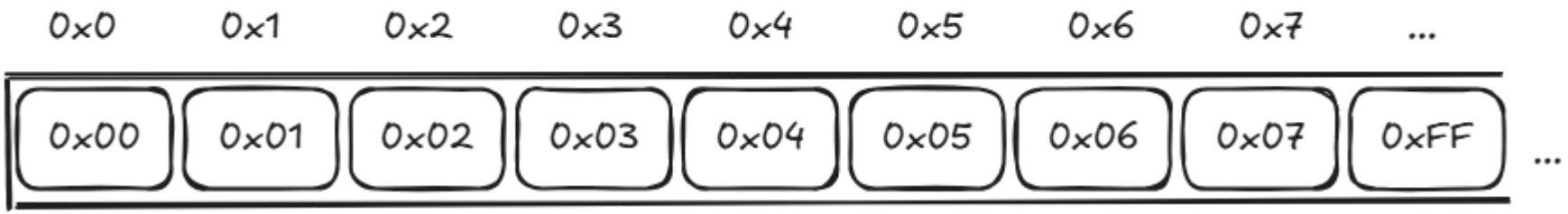


```
mi_dato: db 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
```

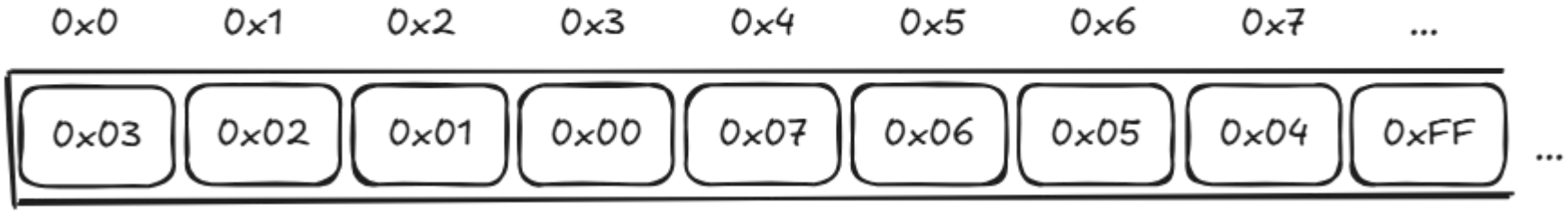



```
mi_dato: dd 0x00010203, 0x04050607
```

mi_dato: db 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07



mi_dato: dd 0x00010203, 0x04050607



Para definir floats basta con expresarlos como tales, respetando su tamaño (Dword para Float, Quadword para Double)

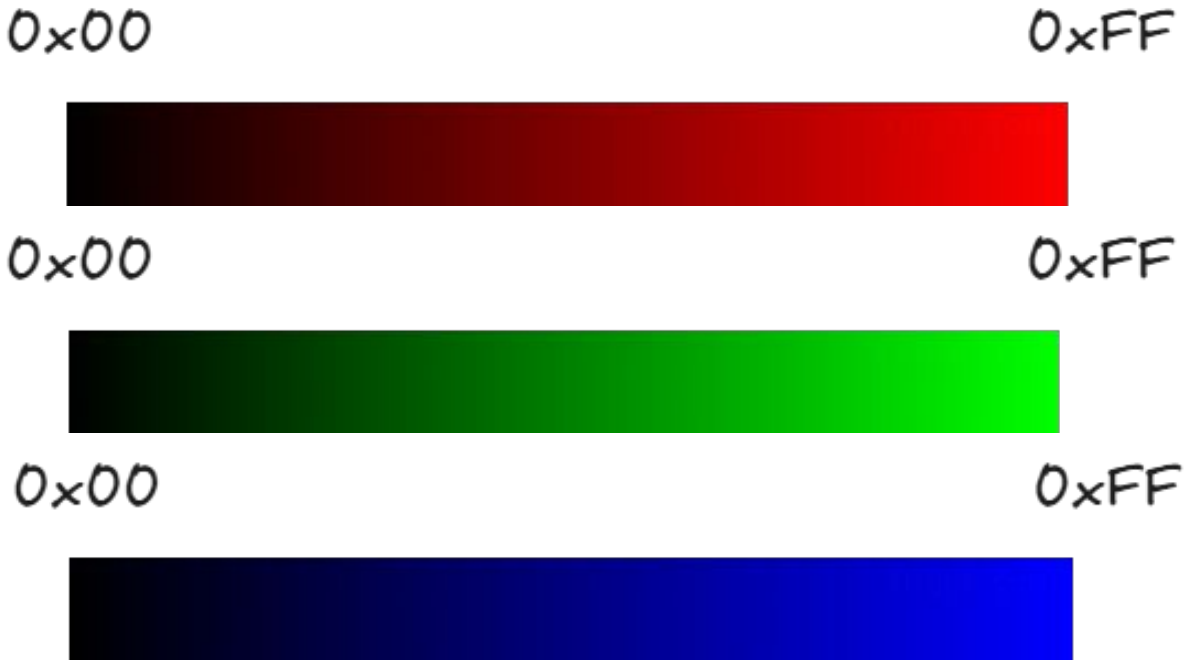
```
mi_float: dd 0.1, 0.2, 0.3, 0.4
mi_double: dq 0.1, 0.2, 0.3, 0.4
```

¿Qué es un pixel?

Un pixel (picture - element) es una muestra de luz en un punto que representamos digitalmente. Si la representamos con un único entero de 8 bits, solo podremos determinar la intensidad de la luz y formar imágenes en blanco y negro

¿Qué es un pixel?

Si en cambio dividimos la muestra de luz en tres canales, rojo, verde y azul, podemos determinar la intensidad de cada uno y formar imágenes a color

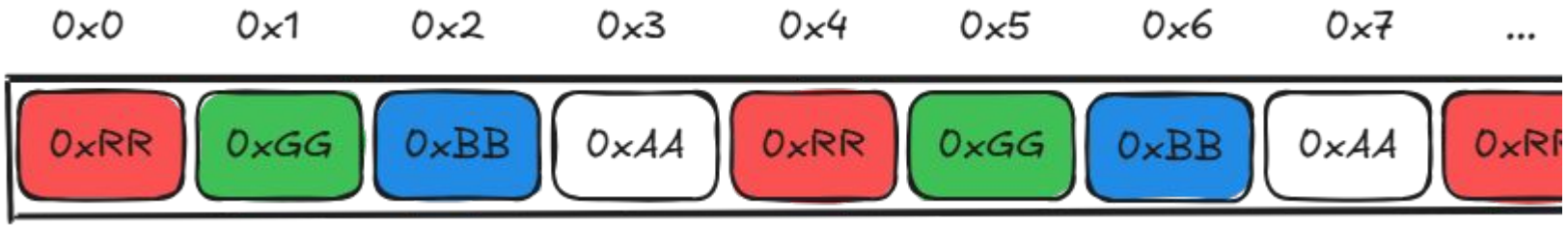


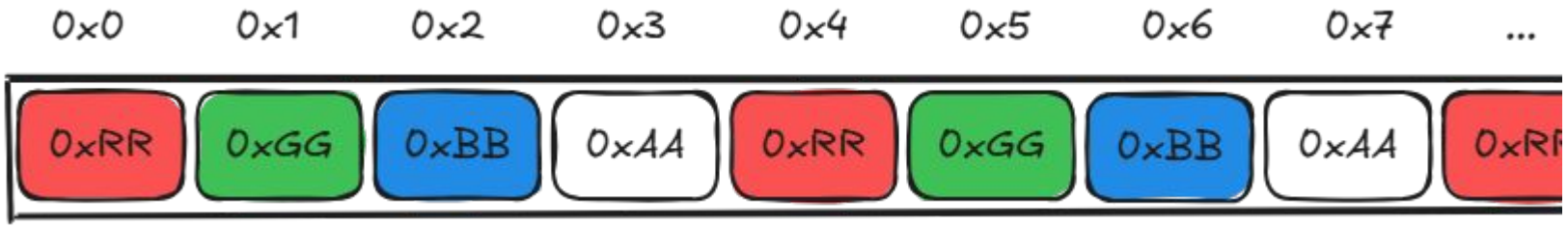
Un formato muy común, añade un cuarto canal, llamado alfa, que determina la transparencia (u opacidad) del pixel, donde 0xFF representa un color totalmente opaco y 0x00 uno totalmente transparente.

Este es el formato conocido como RGBA:

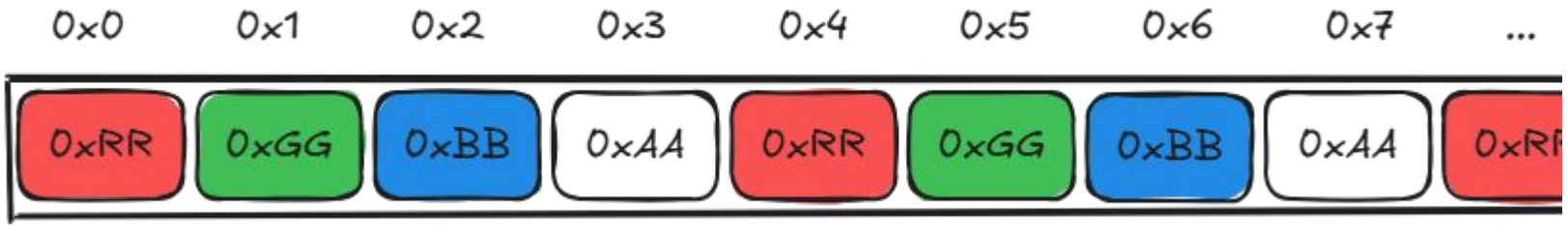
```
typedef struct rgba_pixfmt{

        uint8_t r;
        uint8_t g;
        uint8_t b;
        uint8_t a;

} rgba_t
```

¿Cómo va a lucir en memoria?


0x0
0x1
0x2
0x3
0x4
0x5
0x6
0x7
...
0xRR
0xGG
0xBB
0xAA
0xRR
0xGG
0xBB
0xAA
0xRR



¿Y si queremos definir un pixel como constante en ASM?



¿Y si queremos definir un pixel como constante en ASM?

mi_pixel_en_bytes: db 0xRR, 0xGG, 0xBB, 0xAA

mi_pixel_en_dwords: dd 0xAABBGGRR

LiveCoding: Pintar Pantalla

Paradigma SIMD (Single Instruction, Multiple Data):

Objetivo: Procesar grandes volúmenes de datos de forma eficiente

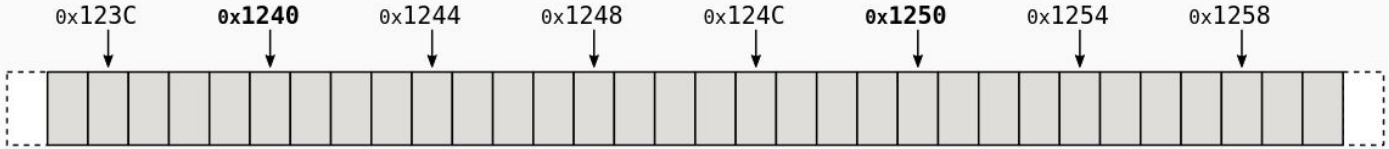
Estrategia: utilizar instrucciones vectoriales para procesar múltiples elementos en paralelo, aprovechando al máximo el ancho del bus de datos y evitando a toda costa las ramificaciones condicionales en el proceso.

Para esto disponemos de los 16 registros XMM y un amplio conjunto de instrucciones...

Instrucciones de lectura y escritura

MOVD	MOVQ	Move Doubleword/Quadword
MOVSS	MOVSD	Moves a 32bits Single FP/64bits Double FP
MOVDQA	MOVDQU	Moves aligned/unaligned double quadword
MOVAPS	MOVUPS	Moves 4 aligned/unaligned 32bit singles
MOVAPD	MOVUPD	Moves 2 aligned/unaligned 64bit doubles

Ejemplo:



`MOVD [0x123C], xmm0`	✓	`[0x123C]` ← `xmm0(32:0)`
`MOVQ xmm0, [0x1245]`	✓	`xmm0(64:0)` ← `[0x1245]`
`MOVDQA xmm0, [0x1245]`	×	Error dirección no alineada.
`MOVDQA [0x1250], xmm0`	✓	`[0x1250]` ← `xmm0(128:0)`
`MOVSS xmm0, [0x1248]`	✓	`xmm0(32:0)` ← `[0x1248]` ; sobre punto flotante
`MOVUPS [0x1258], xmm0`	✓	`[0x1258]` ← `xmm0(128:0)` ; sobre punto flotante

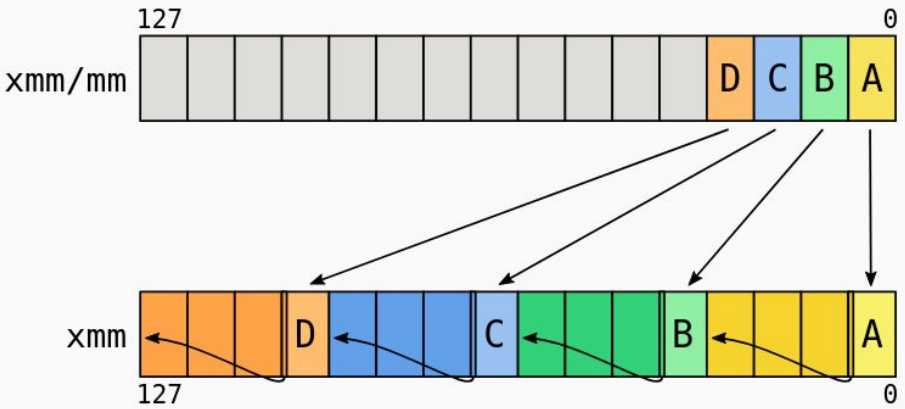
Instrucciones de lectura y escritura

PMOVSXBW	PMOVZXBW	packed sign/zero extension byte to word
PMOVSXBD	PMOVZXBD	packed sign/zero extension byte to dword
PMOVSXBQ	PMOVZXBQ	packed sign/zero extension byte to qword
PMOVSXWD	PMOVZXWD	packed sign/zero extension word to dword
PMOVSXWQ	PMOVZXWQ	packed sign/zero extension word to qword
PMOVSXDQ	PMOVZXDQ	packed sign/zero extension dword to qword

Ejemplos:

```
PMOVSXBD xmm0, xmm0
```

✓



Instrucciones de lectura y escritura

PMOVSXBW	PMOVZXBW	packed sign/zero extension byte to word
PMOVSXBD	PMOVZXBD	packed sign/zero extension byte to dword
PMOVSXBQ	PMOVZXBQ	packed sign/zero extension byte to qword
PMOVSXWD	PMOVZXWD	packed sign/zero extension word to dword
PMOVSXWQ	PMOVZXWQ	packed sign/zero extension word to qword
PMOVSXDQ	PMOVZXDQ	packed sign/zero extension dword to qword

Ejemplos:

```
PMOVSXBD xmm0, xmm0        ✓
PMOVZXWD xmm0, [data]      ✓
PMOVZXDQ xmm0, xmm1        ✓
PMOVZXQD xmm0, xmm0        ×   Instrucción invalida. No hay extension de Qword a DQword.
PMOVSXBD [data], xmm0      ×   Modo de direccionamiento invalido.
```


LiveCoding: Pintar Pantalla - Hacker Edition

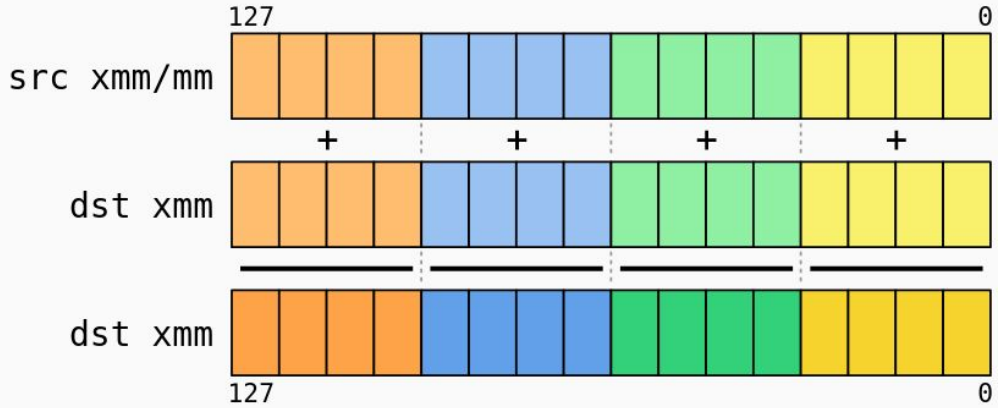
Operaciones aritméticas

PADDB	PADDW	PADDD	PADDQ	Add Integer
PSUBB	PSUBW	PSUBD	PSUBQ	Sub Integer
PMULHW	PMULLW			Mul Integer Word
PMULHD	PMULLD			Mul Integer Dword
PMINSB	PMAXSB	PMINUB	PMAXUB	Max and Min Integer
PMINSW	PMAXSW	PMINUW	PMAXUW	Max and Min Integer
PMINSD	PMAXSD	PMINUD	PMAXUD	Max and Min Integer

Ejemplos:

```
PADDD xmm0, xmm1
```

✓



Operaciones aritméticas

PADDB	PADDW	PADDD	PADDQ	Add Integer
PSUBB	PSUBW	PSUBD	PSUBQ	Sub Integer
PMULHW	PMULLW			Mul Integer Word
PMULHD	PMULLD			Mul Integer Dword
PMINSB	PMAXSB	PMINUB	PMAXUB	Max and Min Integer
PMINSW	PMAXSW	PMINUW	PMAXUW	Max and Min Integer
PMINSD	PMAXSD	PMINUD	PMAXUD	Max and Min Integer

Ejemplos:

```
PADDD xmm0, xmm1        ✓
PSUBW xmm0, [data]      ✓
PMULLD xmm0, xmm1       ✓
PMAXSW xmm0, [data]     ✓
PMINSB [data], xmm0     ×   Modo de direccionamiento invalido.
```

Operaciones aritméticas

PABSB	Absolute for 8 bit Integers
PABSW	Absolute for 16 bit Integers
PABSD	Absolute for 32 bit Integers

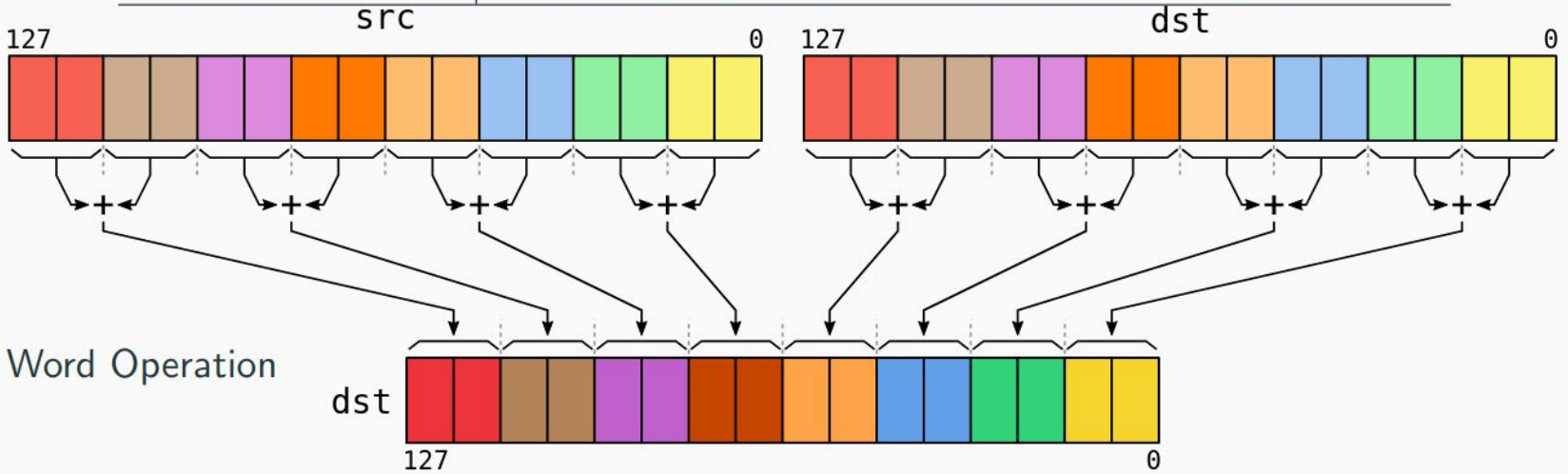
PADDSB	PADDSW	Add Int saturation
PADDUSB	PADDUSW	Add Int unsigned saturation
PSUBSB	PSUBSW	Sub Int saturation
PSUBUSB	PSUBUSW	Sub Int unsigned saturation

ADDPS	ADDSS	ADDPD	ADDSD	Addition of FP values
SUBPS	SUBSS	SUBPD	SUBSD	Subtraction of FP values
MULPS	MULSS	MULPD	MULSD	Multiply of FP values
DIVPS	DIVSS	DIVPD	DIVSD	Divition of FP values
MAXPS	MAXSS	MINPS	MINSS	Max and Min of Single FP values
MAXPD	MAXSD	MINPD	MINSD	Max and Min of Double FP values

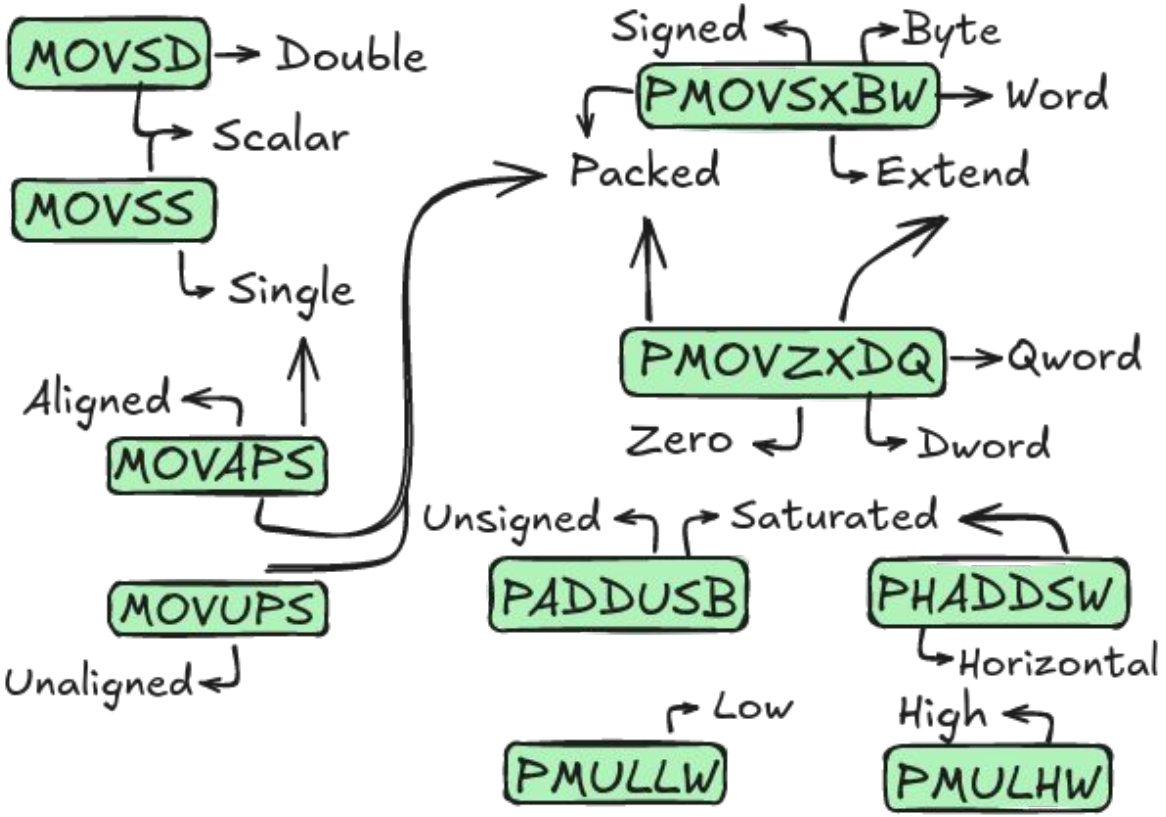
SQRTSS	SQRTPS	Square root of Scalar/Packed Single FP values
SQRTSD	SQRTPD	Square root of Scalar/Packed Double FP values

Operaciones aritméticas

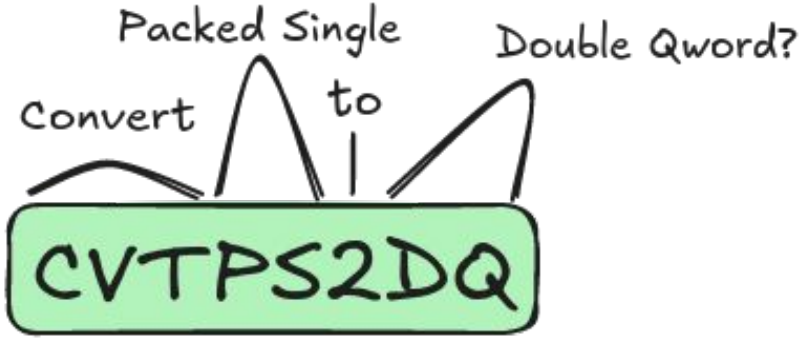
PHADDW	PHADDD	Horizontal addition of unsigned 16bit/32bit integers
PHADDSW		Horizontal saturated addition of 16bit integers
PHSUBW	PHSUBD	Horizontal subtraction of unsigned 16bit/32bit integers
PHSUBSW		Horizontal saturated subtraction of 16bit words
HADDPS	HADDPD	Packed Single/Double FP Horizontal Add
HSUBPS	HSUBPD	Packed Single/Double FP Horizontal Subtract



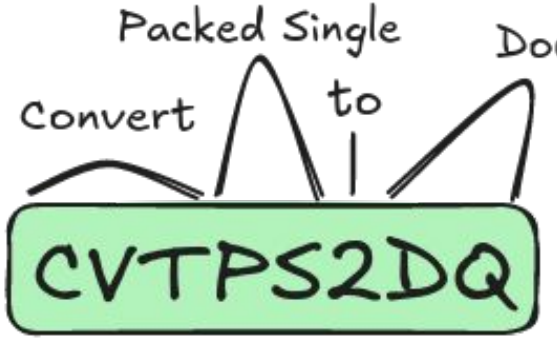
¿Reglas de nomenclatura? Ponele...



Converts y sus peculiaridades



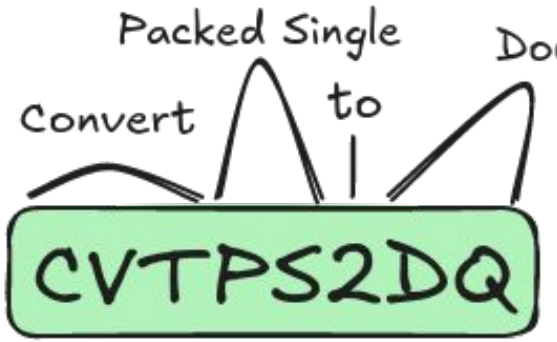
Converts y sus peculiaridades



NOP!

Packed Dwords
4 a 4

Converts y sus peculiaridades



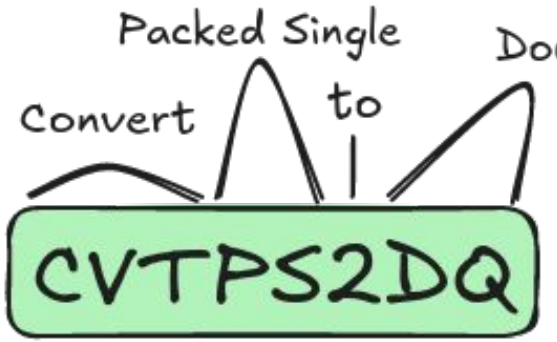
NOP!

Packed Dwords
4 a 4

CVTPS2PI

¿Y este?

Converts y sus peculiaridades



NOP!

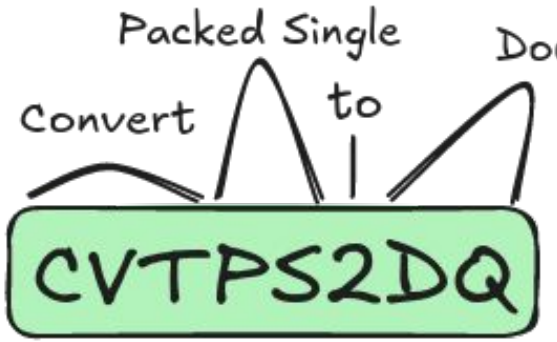
Packed Dwords
4 a 4

CVTPS2PI

¿Y este?

Packed intergers
o sea Dwords...
Pero 2 a 2 :)

Converts y sus peculiaridades



NOP!

Packed Dwords
4 a 4

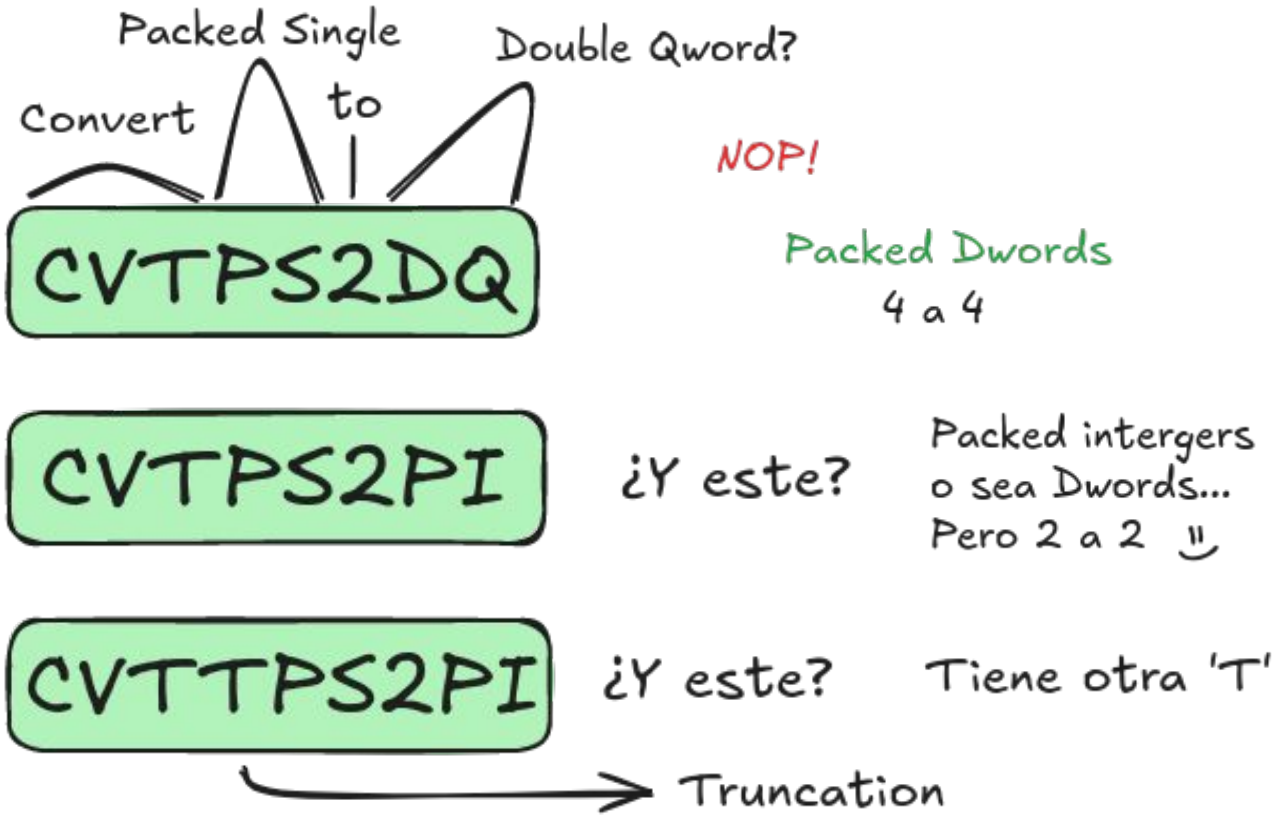
CVTPS2PI ¿Y este?

Packed intergers
o sea Dwords...
Pero 2 a 2 :)

CVTTPS2PI ¿Y este?

Tiene otra 'T'

Converts y sus peculiaridades



Converts y sus peculiaridades

Convert Packed Single to Double Qword?

CVTPS2DQ

NOP!

Packed Dwords
4 a 4

CVTPS2PI

¿Y este?

Packed intergers
o sea Dwords...
Pero 2 a 2 :)

CVTTPS2PI

¿Y este?

Tiene otra 'T'

→ Truncation

Moraleja...

Es bastante arbitrario todo, pero vale la pena tener en mente estos términos:

Z: zero

S: signed, scalar, single, saturated

U: unsigned, unaligned, unsaturated

A: aligned

P: packed

X: extend

H: high, horizontal

L: low

B: byte

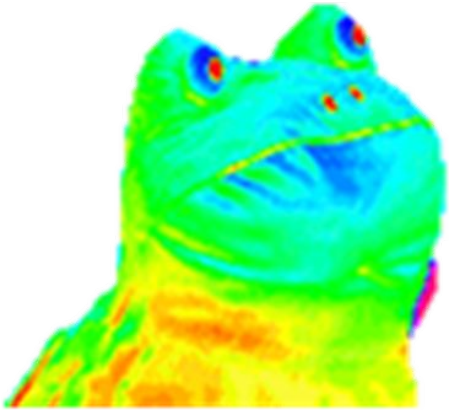
W: word

D: dword, double

Q: qword

DQ: double qword

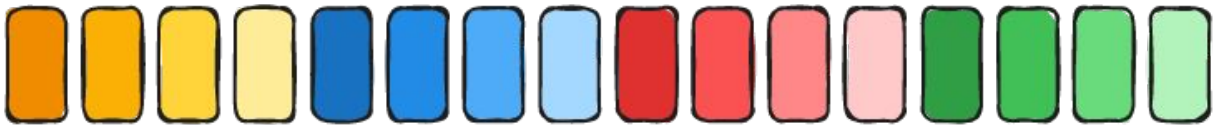
LiveCoding: Locura RGB



Shuffles - PSHUFB dst, src

Tenemos un registro xmm, cada pieza representa un byte
y el color representa al dato almacenado en el byte

xmm0 =



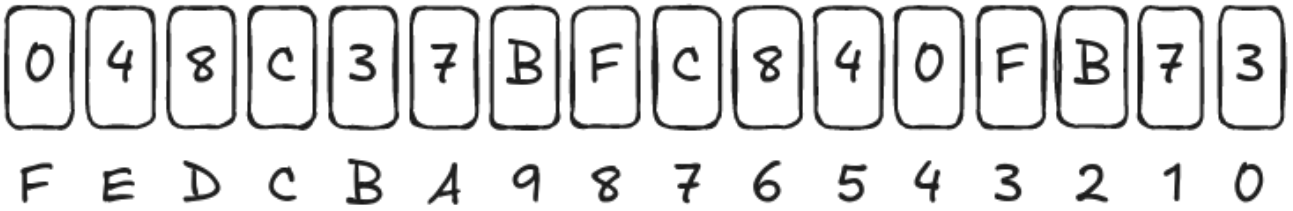
A cada elemento le corresponde una posición en el registro,
que notamos con un número del 0 al F (en hexa)

Con la posición 0 siendo la de más a la derecha
por corresponder a los bits menos significativos

Shuffles - PSHUFB dst, src

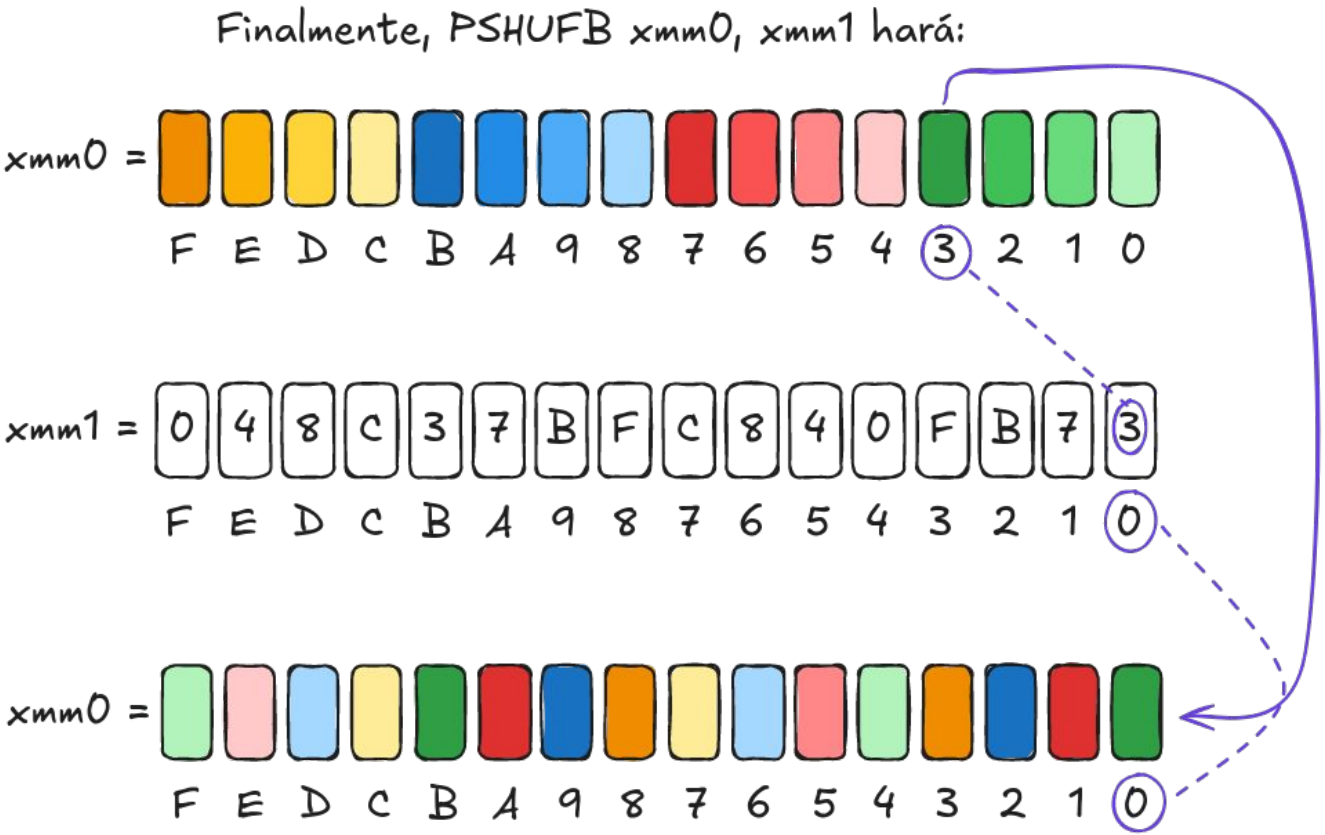
En otro xmm, vamos a colocar en cada byte un valor del 0 al F
que hará referencia a la posición del xmm anterior
cuyo valor queramos que se copie

xmm1 =



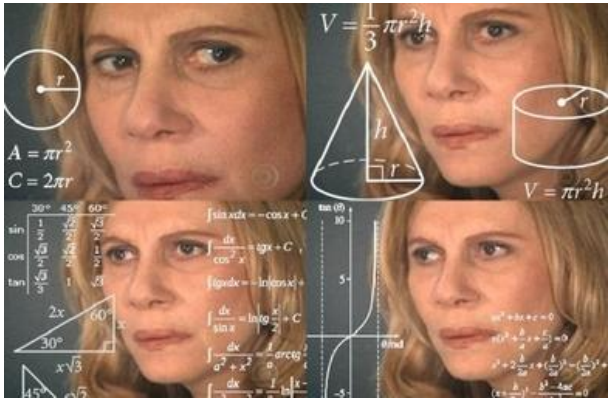
Notar que hay posiciones referenciadas múltiples veces
Y otras no referenciadas en absoluto

Shuffles - PSHUFB dst, src



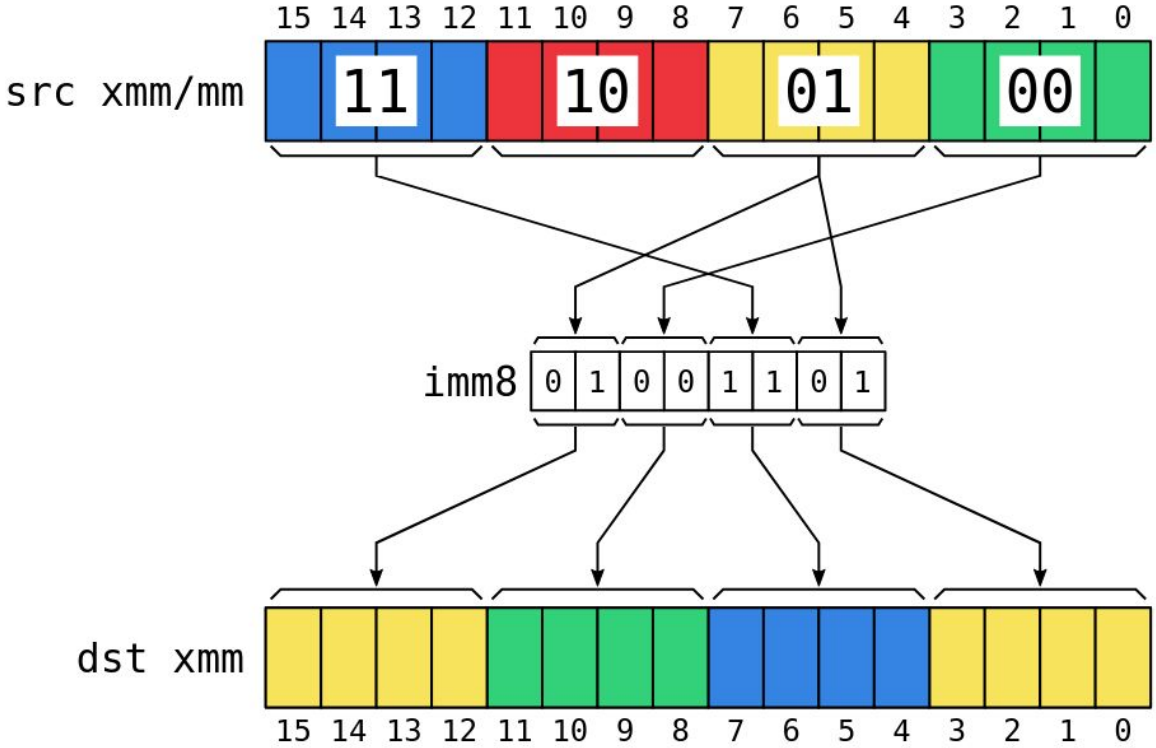
Shuffles - PSHUFB dst, src

Dato extra: si en el bit más significativo de algún elemento (byte) del **src** ponemos un '1' o, en otras palabras, en algún elemento del **src** tenemos un número negativo, entonces se escribirá un byte limpio (0x00) en el elemento correspondiente del **dst**.



LiveCoding: Locura RGB

Shuffles - PSHUFD dst, src, imm8



Shuffles

El resto de los shuffles los pueden investigar en el manual y en las diapos adicionales que subimos al campus. Todos tienen alguna peculiaridad así que revisen bien su comportamiento antes de usarlos.

Operaciones lógicas

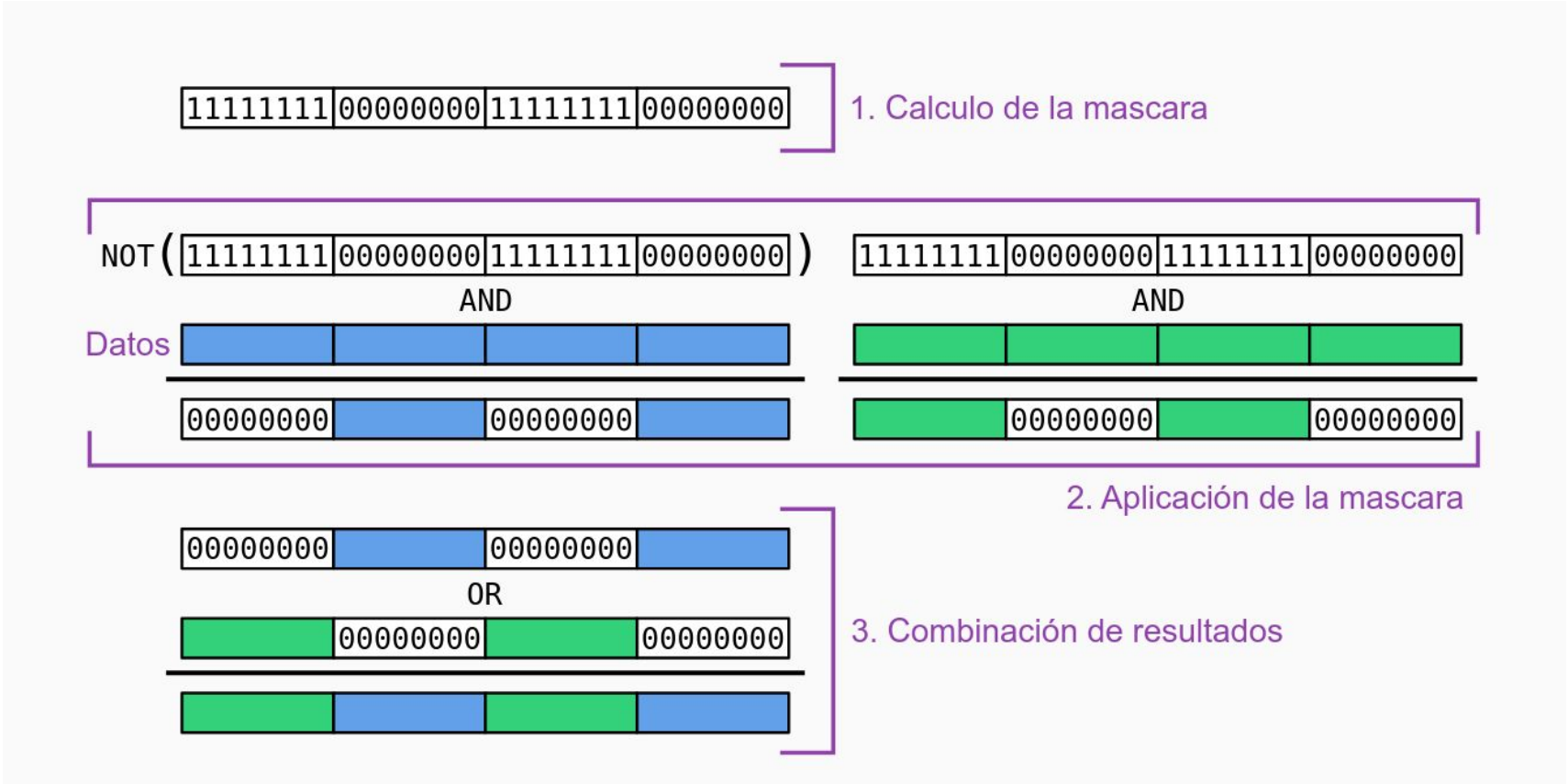
PAND	PANDN	POR	PXOR	Operaciones lógicas para enteros.
ANDPS	ANDNPS	ORPS	XORPS	Operaciones lógicas para *float*.
ANDPD	ANDNPD	ORPD	XORPD	Operaciones lógicas para *double*.

- Actúan lógicamente sobre todo el registro, sin importar el tamaño del operando.
- La distinción entre PS y PD se debe a meta información para el procesador.

PSLLW	PSLLD	PSLLQ	PSLLDQ*
PSRLW	PSRLD	PSRLQ	PSRLDQ*
PSRAW	PSRAD		

- Todos los shifts operan tanto de forma lógica como aritmética, a derecha e izquierda.
- Se limitan a realizar la operación sobre cada elemento dentro del registro según su tamaño.
- * En las operaciones indicadas, el parámetro es la cantidad de bytes del desplazamiento.

Máscaras


11111111 00000000 11111111 00000000
1. Calculo de la mascara
NOT (11111111 00000000 11111111 00000000)
AND
Datos
00000000 00000000
11111111 00000000 11111111 00000000
AND
00000000 00000000
2. Aplicación de la mascara
00000000 00000000
OR
00000000 00000000
3. Combinación de resultados

Hace un rato dijimos...

Estrategia: utilizar instrucciones vectoriales para procesar múltiples elementos en paralelo, aprovechando al máximo el ancho del bus de datos y **evitando a toda costa las ramificaciones condicionales** en el proceso.

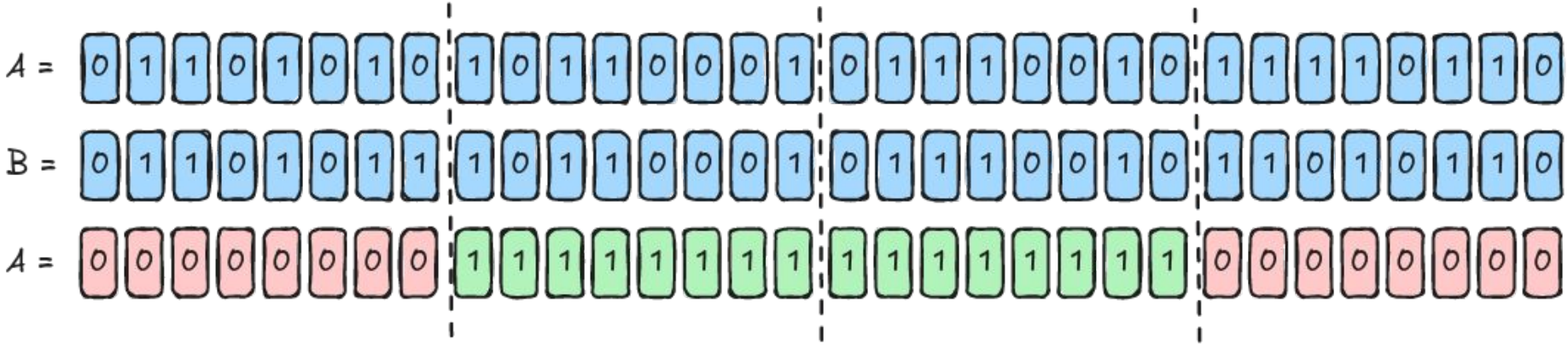
¿Y eso con qué se come?

Comparaciones

PCMPEQB	PCMPEQW	PCMPEQD	PCMPEQQ	Compare Packed Data for Equal
PCMPGTB	PCMPGTW	PCMPGTD	PCMPGTQ	Compare Packed Signed Int for Greater Than

Ejemplo en un registro más chico de lo normal:

PCMPEQB A, B

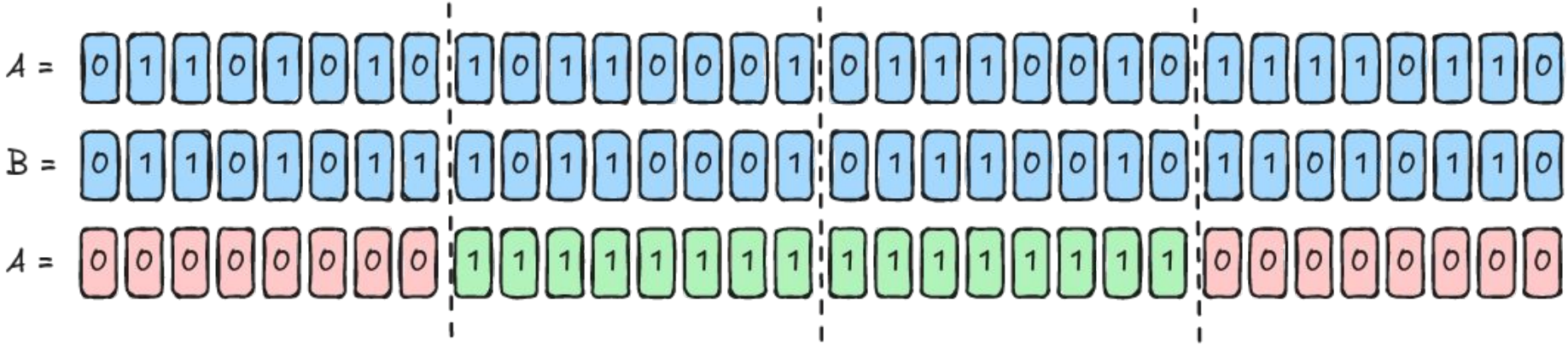


Comparaciones

PCMPEQB	PCMPEQW	PCMPEQD	PCMPEQQ	Compare Packed Data for Equal
PCMPGTB	PCMPGTW	PCMPGTD	PCMPGTQ	Compare Packed Signed Int for Greater Than

Ejemplo en un registro más chico de lo normal:

PCMPEQB A, B



Sepan que hay bastantes más instrucciones de comparación

LiveCoding: Dibujar círculo