Combinar imágenes

Ejercicio SIMD del recuperatorio 2c 2023

En Orga2 llamamos `combinarImagenes` al filtro que dadas dos imágenes (con formato BGRA) de igual tamaño, devuelve una tercera que combina las dos de la siguiente forma:

$$
\begin{aligned}
res[ij]_B &= A[ij]_B + B[ij]_R \\
res[ij]_G &= \begin{cases} A[ij]_G - B[ij]_G & \text{si } A[ij]_G > B[ij]_G \\ promedio(A[ij]_G,\ B[ij]_G) & \text{si no} \end{cases} \\
res[ij]_R &= B[ij]_B - A[ij]_R
\end{aligned}
$$

El valor de la componente de transparencia en el resultado debe ser 255 para todos los pixeles. Implementar el filtro de manera que procese de al menos **dos píxeles** simultáneamente.

```
void combinarImagenes(uint8_t *src1, uint8_t *src2, uint8_t *dst,
                      uint32_t width, uint32_t height)
```

Notar que:

- Se recibe un puntero al espacio de memoria donde debe guardarse el resultado, es decir, esa memoria ya fue pedida (o alocada)

- Para el `promedio` se puede utilizar la instrucción pavgb/pavgw, o calcularlo siguiendo la fórmula $\frac{(A[ij]_G + B[ij]_G + 1)}{2}$

```
void combinarImagenes(uint8_t *src1, uint8_t *src2, uint8_t *dst,
                      uint32_t width, uint32_t height)
```

Notar que:

- Se recibe un puntero al espacio de memoria donde debe guardarse el resultado, es decir, esa memoria ya fue pedida (o alocada)

¿Cantidad de píxeles?

¿Cantidad de iteraciones?

```
void combinarImagenes(uint8_t *src1, uint8_t *src2, uint8_t *dst,
                      uint32_t width, uint32_t height)
```

Notar que:

- Se recibe un puntero al espacio de memoria donde debe guardarse el resultado, es decir, esa memoria ya fue pedida (o alocada)

¿Cantidad de píxeles? width*height

¿Cantidad de iteraciones?

```
void combinarImagenes(uint8_t *src1, uint8_t *src2, uint8_t *dst,
                      uint32_t width, uint32_t height)
```

Notar que:

- Se recibe un puntero al espacio de memoria donde debe guardarse el resultado, es decir, esa memoria ya fue pedida (o alocada)

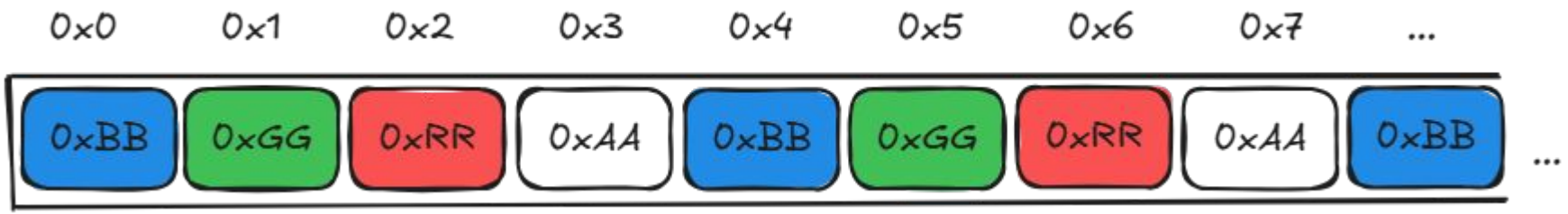
¿Cantidad de píxeles? width*height

¿Cantidad de iteraciones? (width*height / 4)

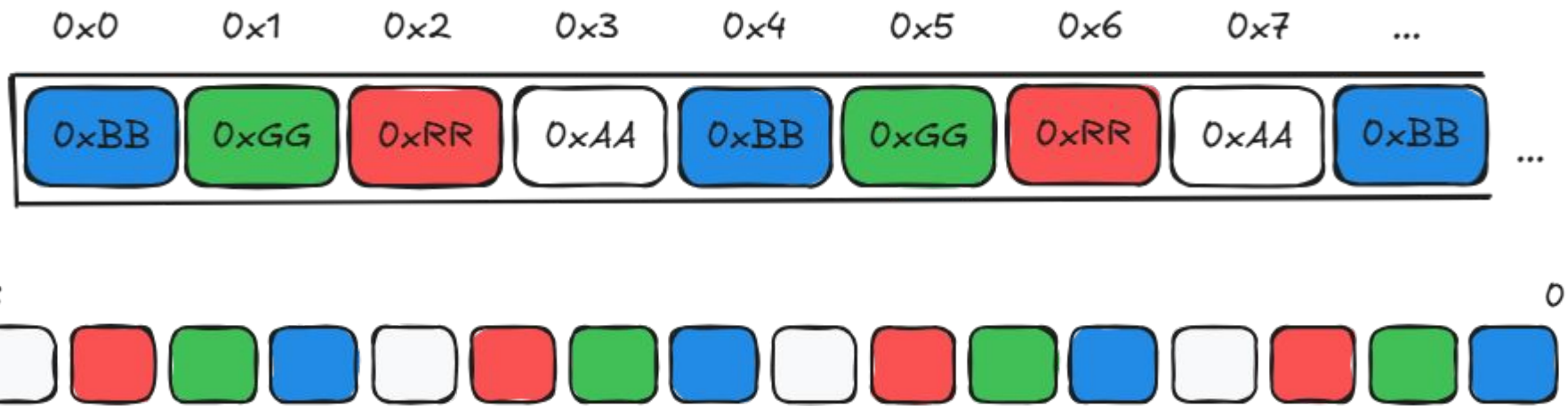
```
combinarImagenes_asm:

push rbp
mov rbp, rsp
mov r10, rdx

mov eax, r8d
mul ecx
shr eax, 2
mov r8d, eax ; r8w = cantidad iteraciones

xor rax, rax

.ciclo:
    movdqu xmm1, [rdi + rax]
    movdqu xmm2, [rsi + rax]

    movdqu [r10 + rax], xmm3 ; guardo resultado en dst

    add rax, 16

    dec r8w
    jnz .ciclo

pop rbp
ret
```

```
//**************************************
//Declaración de estructuras
//**************************************
typedef struct bgra_t {
    unsigned char b, g, r, a;
} __attribute__((packed)) bgra_t;
```

```c
//**************************************
//Declaración de estructuras
//**************************************
typedef struct bgra_t {
    unsigned char b, g, r, a;
} __attribute__((packed)) bgra_t;
```



```
//**************************************
//Declaración de estructuras
//**************************************
typedef struct bgra_t {
    unsigned char b, g, r, a;
} __attribute__((packed)) bgra_t;
```

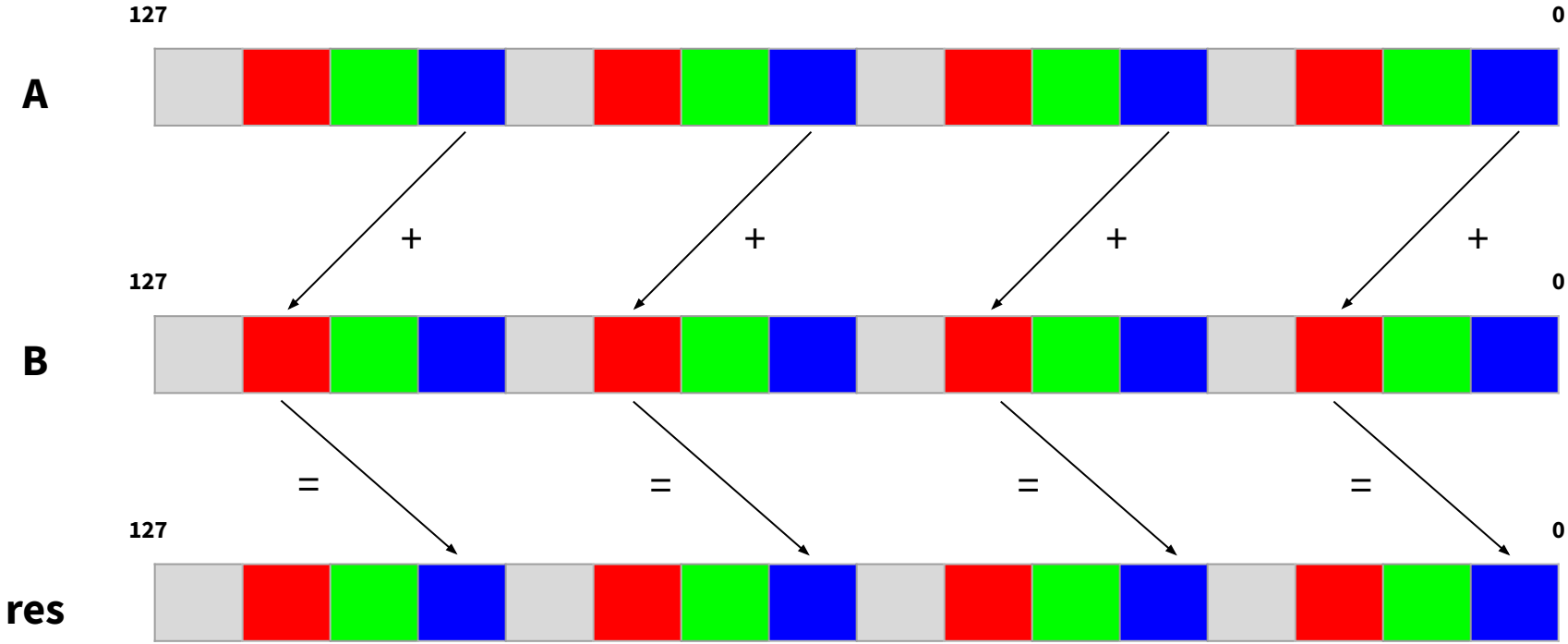


En Orga2 llamamos `combinarImagenes` al filtro que dadas dos imágenes (con formato BGRA) de igual tamaño, devuelve una tercera que combina las dos de la siguiente forma:

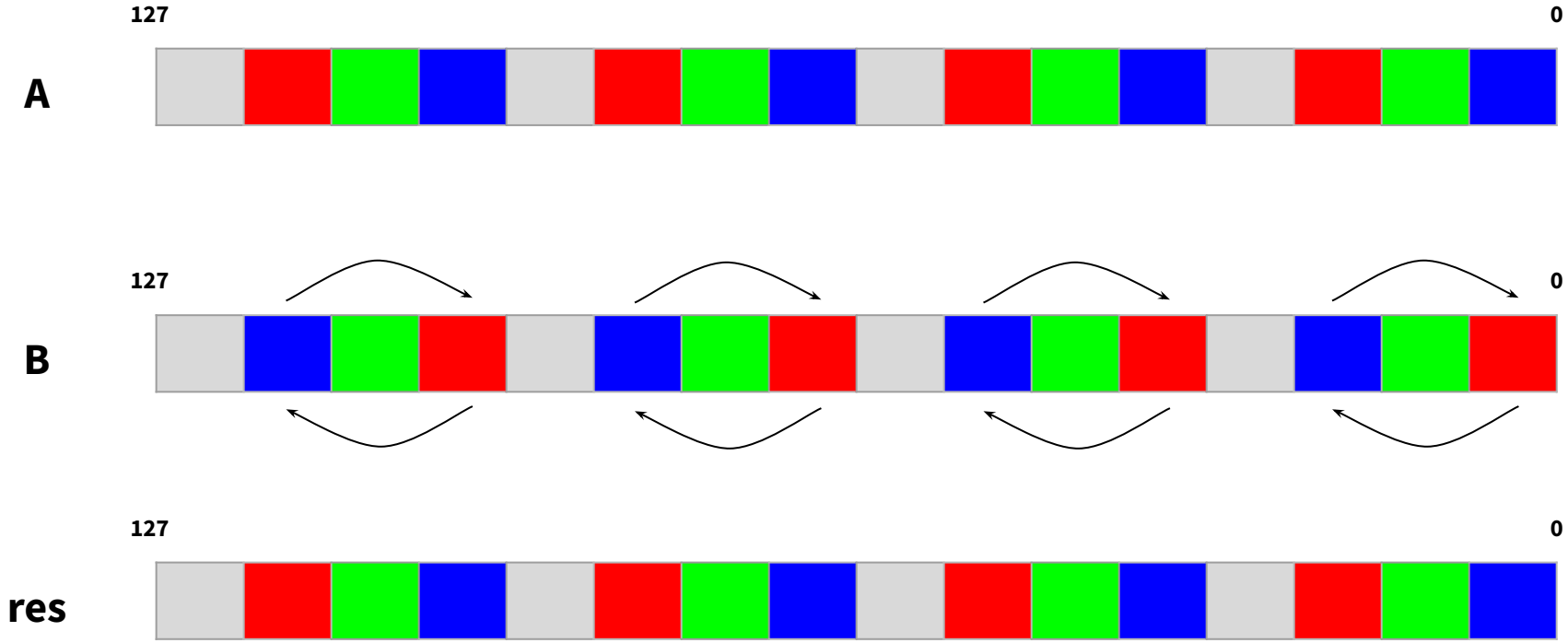
$$
\begin{aligned}
res[ij]_B &= A[ij]_B + B[ij]_R \\
res[ij]_G &= \begin{cases} A[ij]_G - B[ij]_G & \text{si } A[ij]_G > B[ij]_G \\ promedio(A[ij]_G,\ B[ij]_G) & \text{si no} \end{cases} \\
res[ij]_R &= B[ij]_B - A[ij]_R
\end{aligned}
$$

El valor de la componente de transparencia en el resultado debe ser 255 para todos los pixeles. Implementar el filtro de manera que procese de al menos **dos píxeles** simultáneamente.

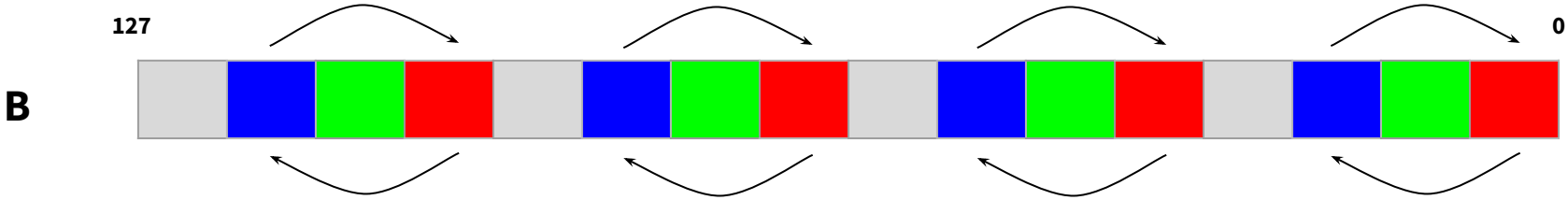
$$res[ij]_B = A[ij]_B + B[ij]_R$$



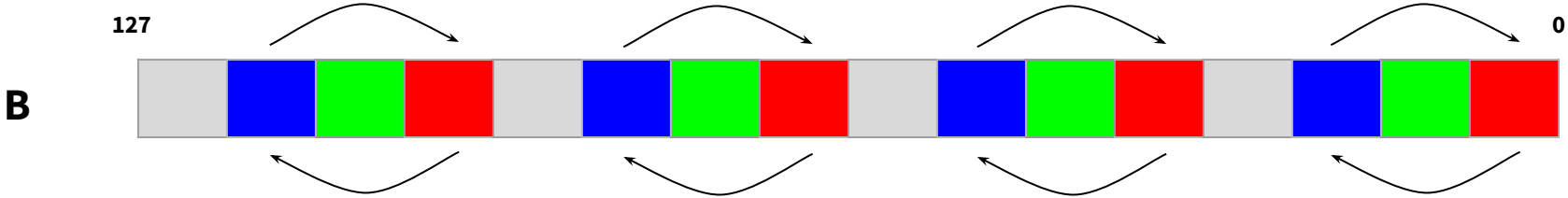
$$res[ij]_B = A[ij]_B + B[ij]_R$$



$$res[ij]_B = A[ij]_B + B[ij]_R$$

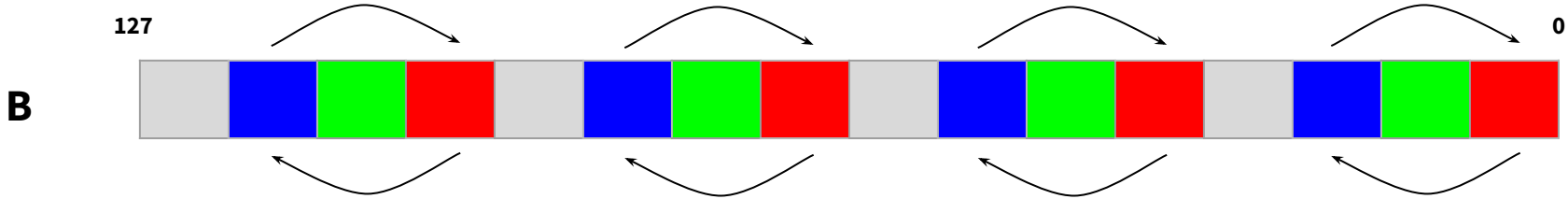


$$res[ij]_B = A[ij]_B + B[ij]_R$$




```
shuffle_img2: DB 0x02, 0x01, 0x00, 0x03, 0x06, 0x05, 0x04, 0x07, 0x0A, 0x09, 0x08, 0x0B, 0x0E, 0x0D, 0x0C, 0x0F
```

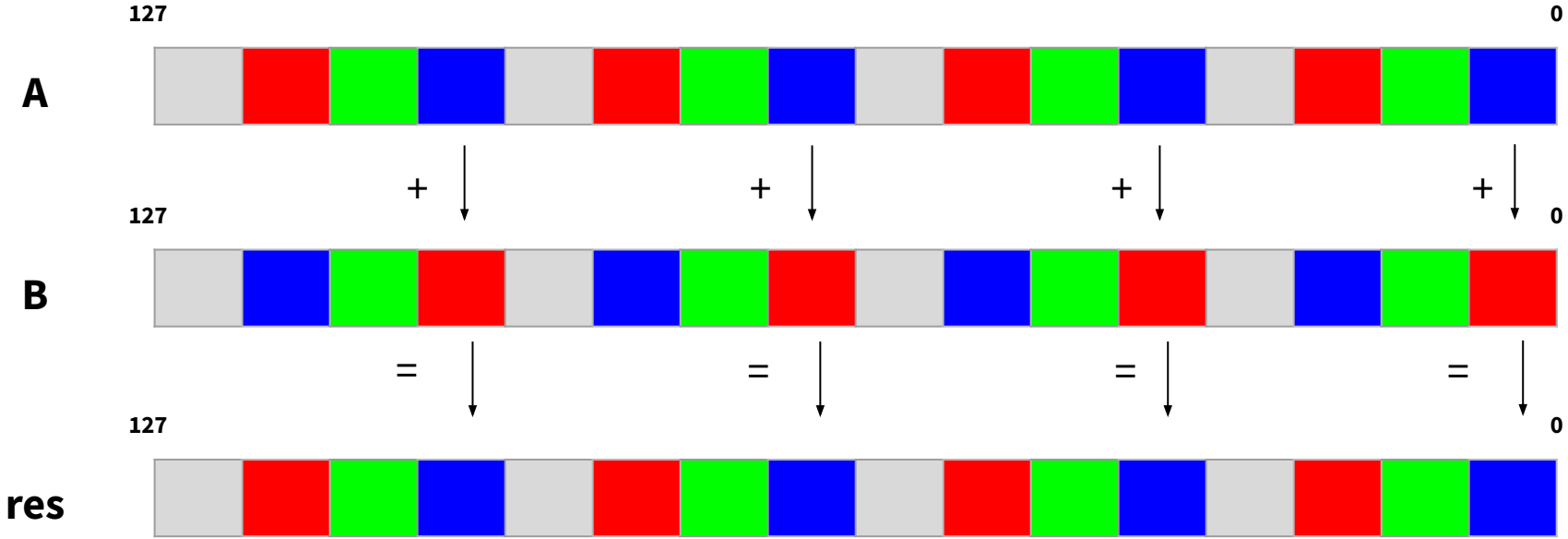
$$res[ij]_B = A[ij]_B + B[ij]_R$$




```
shuffle_img2: DB 0x02, 0x01, 0x00, 0x03, 0x06, 0x05, 0x04, 0x07, 0x0A, 0x09, 0x08, 0x0B, 0x0E, 0x0D, 0x0C, 0x0F
```

movdqu xmm, [shuffle_img2]
pshufb B , xmm

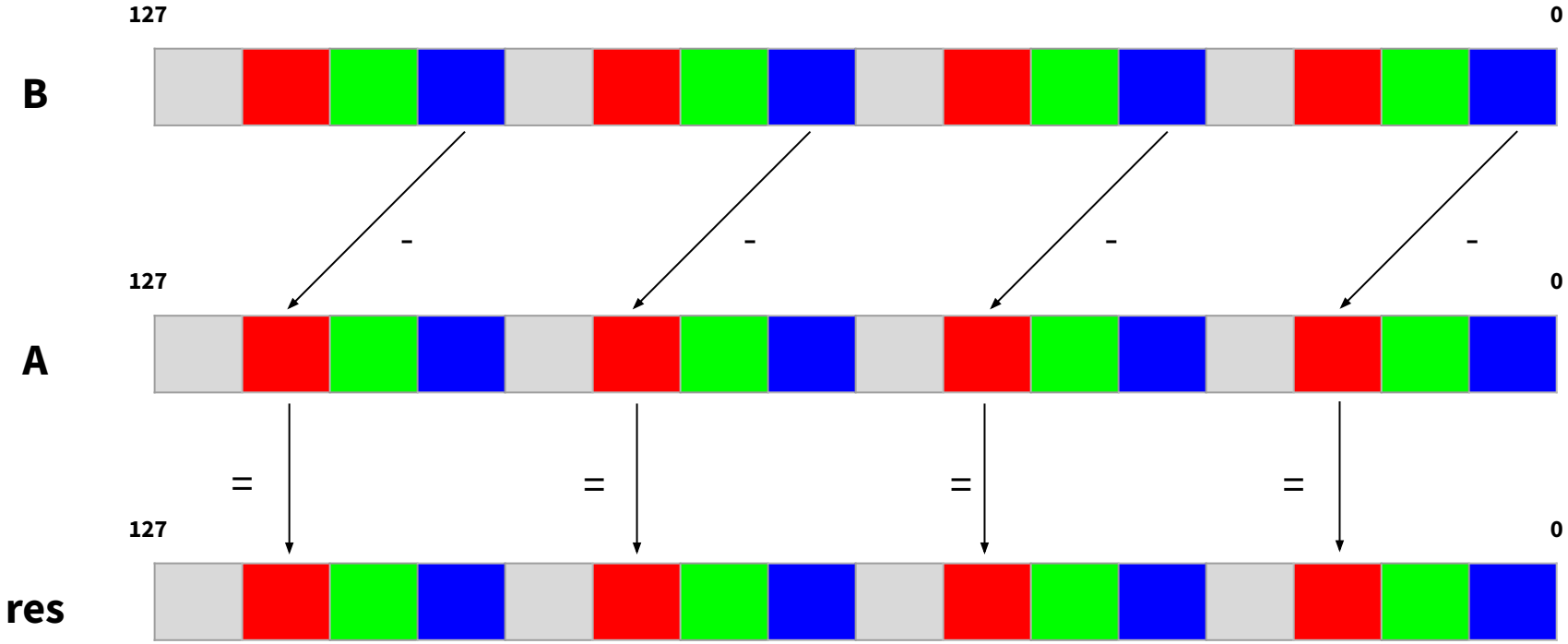
$$res[ij]_B = A[ij]_B + B[ij]_R$$



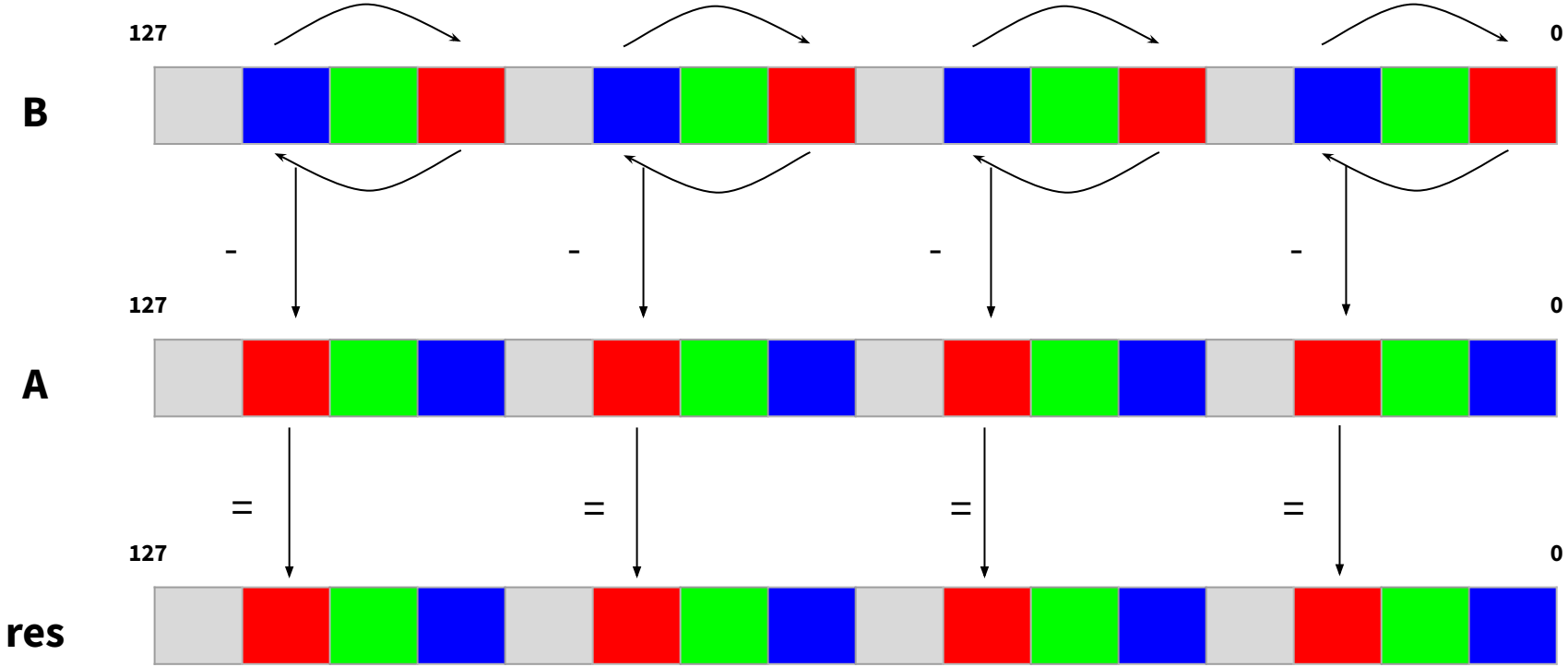
paddusb A, B

```
combinarImagenes_asm:

push rbp
mov rbp, rsp
mov r10, rdx

mov eax, r8d
mul ecx
shr eax, 2
mov r8d, eax ; r8w = cantidad iteraciones

xor rax, rax

movdqu xmm9, [shuffle_img2]

.ciclo:
    movdqu xmm1, [rdi + rax]
    movdqu xmm2, [rsi + rax]

    pshufb xmm2, xmm9

    ; componente blue
    movdqu xmm3, xmm1
    paddusb xmm3, xmm2
```

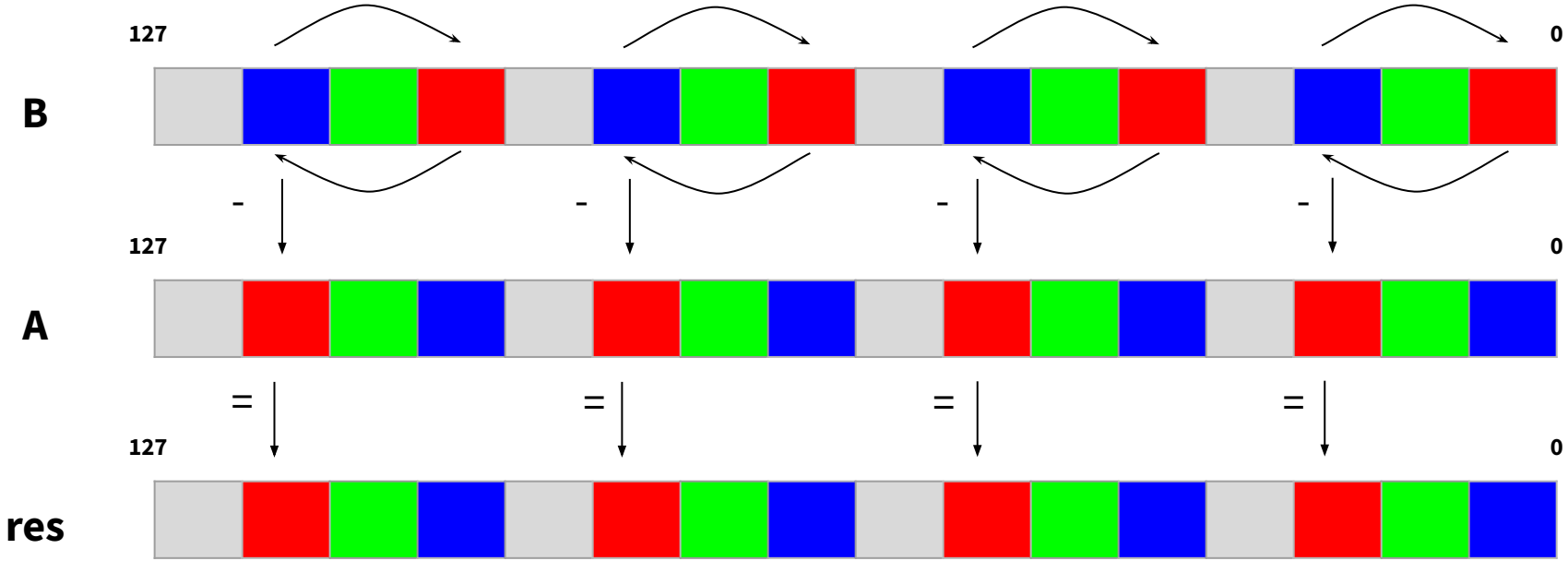
$$res[ij]_R = B[ij]_B - A[ij]_R$$



$$res[ij]_R = B[ij]_B - A[ij]_R$$



$$res[ij]_R = B[ij]_B - A[ij]_R$$



psubusb B, A

```
.ciclo:
    movdqu xmm1, [rdi + rax]
    movdqu xmm2, [rsi + rax]

    pshufb xmm2, xmm9

    ; componente blue
    movdqu xmm3, xmm1
    paddusb xmm3, xmm2

    ; componente red
    movdqu xmm5, xmm2
    psubusb xmm5, xmm1
```

$$res[ij]_G = \begin{cases} A[ij]_G - B[ij]_G & \text{si } A[ij]_G > B[ij]_G \\ promedio(A[ij]_G,\ B[ij]_G) & \text{si no} \end{cases}$$

- Para el `promedio` se puede utilizar la instrucción pavgb/pavgw, o calcularlo siguiendo la fórmula $\frac{(A[ij]_G + B[ij]_G + 1)}{2}$

$$res[ij]_G = \begin{cases} A[ij]_G - B[ij]_G & \text{si } A[ij]_G > B[ij]_G \\ promedio(A[ij]_G,\ B[ij]_G) & \text{si no} \end{cases}$$

- Para el `promedio` se puede utilizar la instrucción pavgb/pavgw, o calcularlo siguiendo la fórmula $\frac{(A[ij]_G + B[ij]_G + 1)}{2}$

pavgb A, B
psubusb A, B

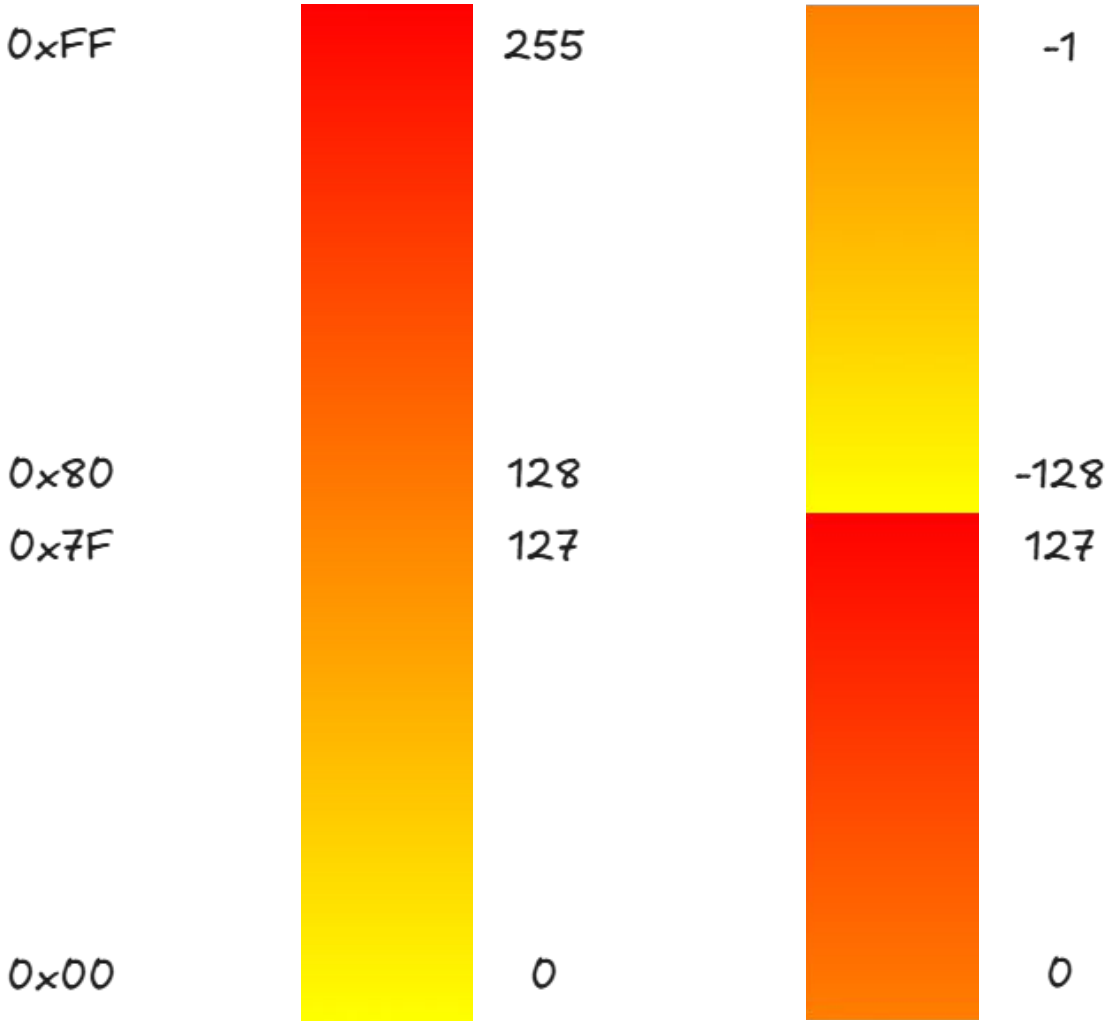
El promedio no tiene mucha ciencia con esa instrucción
La resta tampoco, pero…
¿Cómo haríamos la comparación?

$$res[ij]_G = \begin{cases} A[ij]_G - B[ij]_G & \text{si } A[ij]_G > B[ij]_G \\ promedio(A[ij]_G,\ B[ij]_G) & \text{si no} \end{cases}$$

- Para el `promedio` se puede utilizar la instrucción pavgb/pavgw, o calcularlo siguiendo la fórmula $\frac{(A[ij]_G + B[ij]_G + 1)}{2}$

El promedio no tiene mucha ciencia con esa instrucción
La resta tampoco, pero…
¿Cómo haríamos la comparación?

¡Ojo que PCMPGTB es una comparación signada!


0xFF
255
-1
0x80
128
-128
0x7F
127
127
0x00
0
0

Si el valor es menor a 128, cuando le sumemos 128 va a quedar en el rango de los negativos

Si el valor es mayor a 128, cuando le sumemos 128 va a dar la vuelta y va a quedar en el rango de los positivos

Manteniéndose así la relación de orden de los números, pero ahora en la interpretación signada

Si el valor es menor a 128, cuando le sumemos 128 va a quedar en el rango de los negativos

Si el valor es mayor a 128, cuando le sumemos 128 va a dar la vuelta y va a quedar en el rango de los positivos

Manteniéndose así la relación de orden de los números, pero ahora en la interpretación signada

```
todos128: times 16 db 128


movdqa xmm8, [todos128]
movdqu xmm0, xmm1 ;xmm1 tiene pixeles de A
movdqa xmm5, xmm2 ;xmm1 tiene pixeles de B
paddb xmm5, xmm8
paddb xmm0, xmm8

pcmpgtb xmm0, xmm5
```

```
; componente green
;xmm1 tiene pixeles de A
;xmm2 tiene pixeles de B
movdqu xmm4, xmm1
movdqu xmm0, xmm1
movdqu xmm6, xmm1
pavgb xmm4, xmm2    ; average de a componente
psubusb xmm6, xmm2 ; resta
movdqa xmm5, xmm2
paddb xmm5, xmm8
paddb xmm0, xmm8

pcmpgtb xmm0, xmm5
```

Tenemos los promedios en xmm4 y las restas en xmm6

¿Cómo usamos la máscara en xmm0 para quedarnos con los resultados que queremos de cada uno?

PBLENDVB — Variable Blend Packed Bytes

Opcode/Instruction	Op/En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 38 10 /r PBLENDVB xmm1, xmm2/m128, <XMM0>	RM	V/V	SSE4_1	Select byte values from xmm1 and xmm2/m128 from mask specified in the high bit of each byte in XMM0 and store the values into xmm1.
VEX.128.66.0F3A.W0 4C /r /is4 VPBLENDVB xmm1, xmm2, xmm3/m128, xmm4	RVMR	V/V	AVX	Select byte values from xmm2 and xmm3/m128 using mask bits in the specified mask register, xmm4, and store the values into xmm1.
VEX.256.66.0F3A.W0 4C /r /is4 VPBLENDVB ymm1, ymm2, ymm3/m256, ymm4	RVMR	V/V	AVX2	Select byte values from ymm2 and ymm3/m256 from mask specified in the high bit of each byte in ymm4 and store the values into ymm1.

Instruction Operand Encoding ¶

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r, w)	ModRM:r/m (r)	<XMM0>	N/A
RVMR	ModRM:reg (w)	VEX.vvvv (r)	ModRM:r/m (r)	imm8[7:4]

Description ¶

Conditionally copies byte elements from the source operand (second operand) to the destination operand (first operand) depending on mask bits defined in the implicit third register argument, XMM0. The mask bits are the most significant bit in each byte element of the XMM0 register.

If a mask bit is “1", then the corresponding byte element in the source operand is copied to the destination, else the byte element in the destination operand is left unchanged.

```
; componente green
;xmm1 tiene pixeles de A
;xmm2 tiene pixeles de B
movdqu xmm4, xmm1
movdqu xmm0, xmm1
movdqu xmm6, xmm1
pavgb xmm4, xmm2   ; average de a componente
psubusb xmm6, xmm2 ; resta
movdqa xmm5, xmm2
paddb xmm5, xmm8
paddb xmm0, xmm8

pcmpgtb xmm0, xmm5
pblendvb xmm4, xmm6 ; dejo el average si no se cumple que es mas grande
```

```
pshufb xmm2, xmm9
; componente blue
movdqu xmm3, xmm1
paddusb xmm3, xmm2

; componente green
;xmm1 tiene pixeles de A
;xmm2 tiene pixeles de B
movdqu xmm4, xmm1
movdqu xmm0, xmm1
movdqu xmm6, xmm1
pavgb xmm4, xmm2   ; average de a componente
psubusb xmm6, xmm2 ; resta
movdqa xmm5, xmm2
paddb xmm5, xmm8
paddb xmm0, xmm8

pcmpgtb xmm0, xmm5
pblendvb xmm4, xmm6

; componente red
movdqu xmm5, xmm2
psubusb xmm5, xmm1
```

Tenemos los azules finales en xmm3, los verdes en xmm4 y los rojos en xmm5

¿Cómo podemos combinarlos en un solo registro?

```
maskblend1: DB 0x00, 0x80, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00
maskblend2: DB 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x80, 0x00
maskAlpha: DB 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0xFF
```

```
movdqa xmm7, [maskAlpha]

; mergeo las tres componentes
movdqa xmm0, [maskblend1]
pblendvb xmm3, xmm4

movdqa xmm0, [maskblend2]
pblendvb xmm3, xmm5

por xmm3, xmm7
```

```asm
combinarImagenes_asm:
push rbp
mov rbp, rsp
mov r10, rdx

mov eax, r8d
mul ecx
shr eax, 2
mov r8d, eax ; r8w = cantidad iteraciones

xor rax, rax

movdqa xmm7, [maskAlpha]
movdqa xmm8, [todos128]
movdqu xmm9, [shuffle_img2]

.ciclo:
    movdqu xmm1, [rdi + rax]
    movdqu xmm2, [rsi + rax]

    pshufb xmm2, xmm9
    ; componente blue
    movdqu xmm3, xmm1
    paddusb xmm3, xmm2

    ; componente green
    ;xmm1 tiene pixeles de A
    ;xmm2 tiene pixeles de B
    movdqu xmm4, xmm1
    movdqu xmm0, xmm1
    movdqu xmm6, xmm1
    pavgb xmm4, xmm2    ; average
    psubusb xmm6, xmm2 ; resta
    movdqa xmm5, xmm2
    paddb xmm5, xmm8
    paddb xmm0, xmm8

    pcmpgtb xmm0, xmm5
    pblendvb xmm4, xmm6

    ; componente red
    movdqu xmm5, xmm2
    psubusb xmm5, xmm1

    ; mergeo las tres componentes
    movdqa xmm0, [maskblend1]
    pblendvb xmm3, xmm4

    movdqa xmm0, [maskblend2]
    pblendvb xmm3, xmm5

    por xmm3, xmm7
    movdqu [r10 + rax], xmm3

    add rax, 16

    dec r8w
    jnz .ciclo

pop rbp
ret
```