

SIMD Parte 2

Conversiones, Shuffles y Blends

David Alejandro González Márquez

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Agenda

- Problemas de Precisión
- Instrucciones de Shuffle
- Instrucciones de Blend
- Instrucciones de Conversión

Problemas de Precisión

- No todos los números pueden ser representados de **forma exacta** en punto flotante

Problemas de Precisión

- No todos los números pueden ser representados de **forma exacta** en punto flotante

Por ejemplo: El número 1,1 no puede ser representado en Float de forma exacta, siendo el más aproximado: 1,100000023841858

Problemas de Precisión

- No todos los números pueden ser representados de **forma exacta** en punto flotante

Por ejemplo: El número 1,1 no puede ser representado en Float de forma exacta, siendo el más aproximado: 1,100000023841858

- Las operaciones en enteros no necesariamente generan resultados que caben en un entero del mismo tamaño

Problemas de Precisión

- No todos los números pueden ser representados de **forma exacta** en punto flotante

Por ejemplo: El número 1,1 no puede ser representado en Float de forma exacta, siendo el más aproximado: 1,100000023841858

- Las operaciones en enteros no necesariamente generan resultados que caben en un entero del mismo tamaño

Por ejemplo: La operación en bytes $0xFE \cdot 0x10$ da como resultado $0x0FE0$, no entra en un byte

Problemas de Precisión

- No todos los números pueden ser representados de **forma exacta** en punto flotante

Por ejemplo: El número 1,1 no puede ser representado en Float de forma exacta, siendo el más aproximado: 1,100000023841858

- Las operaciones en enteros no necesariamente generan resultados que caben en un entero del mismo tamaño

Por ejemplo: La operación en bytes $0xFE \cdot 0x10$ da como resultado $0x0FE0$, no entra en un byte

- Incluso es muy simple perder precisión si nuestros datos temporales son enteros

Problemas de Precisión

- No todos los números pueden ser representados de **forma exacta** en punto flotante

Por ejemplo: El número 1,1 no puede ser representado en Float de forma exacta, siendo el más aproximado: 1,100000023841858

- Las operaciones en enteros no necesariamente generan resultados que caben en un entero del mismo tamaño

Por ejemplo: La operación en bytes $0xFE \cdot 0x10$ da como resultado $0x0FE0$, no entra en un byte

- Incluso es muy simple perder precisión si nuestros datos temporales son enteros

Por ejemplo: Si hacemos la operación $(0xFE + 0x11)/0x02$ en enteros, el resultado es $0x87$, siendo el correcto $0x87, 8$

Problemas de Precisión

Moraleja,

- Antes de hacer cualquier cálculo, **analizar** detalladamente los pasos a realizar

Problemas de Precisión

Moraleja,

- Antes de hacer cualquier cálculo, **analizar** detalladamente los pasos a realizar
- Entender cuándo se **pierde precisión** y decidir qué hacer al respecto

Problemas de Precisión

Moraleja,

- Antes de hacer cualquier cálculo, **analizar** detalladamente los pasos a realizar
- Entender cuándo se **pierde precisión** y decidir qué hacer al respecto
- Las operaciones en enteros son **exactas** en enteros

Problemas de Precisión

Moraleja,

- Antes de hacer cualquier cálculo, **analizar** detalladamente los pasos a realizar
- Entender cuándo se **pierde precisión** y decidir qué hacer al respecto
- Las operaciones en enteros son **exactas** en enteros
- Las operaciones en punto flotante son siempre **aproximadas**

Problemas de Precisión

Moraleja,

- Antes de hacer cualquier cálculo, **analizar** detalladamente los pasos a realizar
- Entender cuándo se **pierde precisión** y decidir qué hacer al respecto
- Las operaciones en enteros son **exactas** en enteros
- Las operaciones en punto flotante son siempre **aproximadas**
- Las operaciones de conversión son **muy costosas**

Problemas de Precisión

Moraleja,

- Antes de hacer cualquier cálculo, **analizar** detalladamente los pasos a realizar
- Entender cuándo se **pierde precisión** y decidir qué hacer al respecto
- Las operaciones en enteros son **exactas** en enteros
- Las operaciones en punto flotante son siempre **aproximadas**
- Las operaciones de conversión son **muy costosas**
- Los registros tienen un tipo de datos oculto al programador, usar instrucciones en enteros en punto flotante o a la inversa, implica una **penalidad**

Problemas de Precisión

Moraleja,

- Antes de hacer cualquier cálculo, **analizar** detalladamente los pasos a realizar
- Entender cuándo se **pierde precisión** y decidir qué hacer al respecto
- Las operaciones en enteros son **exactas** en enteros
- Las operaciones en punto flotante son siempre **aproximadas**
- Las operaciones de conversión son **muy costosas**
- Los registros tienen un tipo de datos oculto al programador, usar instrucciones en enteros en punto flotante o a la inversa, implica una **penalidad**
- El **costo** de las operaciones depende de cada una, ya sea en punto flotante o enteros

Shuffles

Las instrucciones de *Shuffle* permiten **reordenar** datos en registros.

Sus parámetros serán el **registro a reordenar** y una **máscara** que indicará cómo hacerlo.

Shuffles

Las instrucciones de *Shuffle* permiten **reordenar** datos en registros.

Sus parámetros serán el **registro a reordenar** y una **máscara** que indicará cómo hacerlo.

- PSHUFB - Shuffle Packed Bytes
- PSHUFW - Shuffles high 16bit values
- PSHUFLW - Shuffles low 16bit values
- PSHUFD - Shuffle Packed Doublewords

- SHUFPS - Shuffle Packed Single FP Values
- SHUFPD - Shuffle Packed Double FP Values

Shuffles

PSHUFB — Packed Shuffle Bytes

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 38 00 /r ¹ PSHUFB <i>mm1</i> , <i>mm2/m64</i>	RM	V/V	SSSE3	Shuffle bytes in <i>mm1</i> according to contents of <i>mm2/m64</i> .
66 0F 38 00 /r PSHUFB <i>xmm1</i> , <i>xmm2/m128</i>	RM	V/V	SSSE3	Shuffle bytes in <i>xmm1</i> according to contents of <i>xmm2/m128</i> .
VEX.NDS.128.66.0F38.WIG 00 /r VPSHUFB <i>xmm1</i> , <i>xmm2</i> , <i>xmm3/m128</i>	RVM	V/V	AVX	Shuffle bytes in <i>xmm2</i> according to contents of <i>xmm3/m128</i> .
VEX.NDS.256.66.0F38.WIG 00 /r VPSHUFB <i>ymm1</i> , <i>ymm2</i> , <i>ymm3/m256</i>	RVM	V/V	AVX2	Shuffle bytes in <i>ymm2</i> according to contents of <i>ymm3/m256</i> .

Shuffles

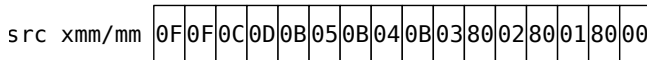
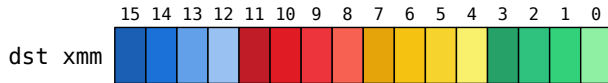
PSHUFB — Packed Shuffle Bytes

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 38 00 /r ¹ PSHUFB <i>mm1</i> , <i>mm2/m64</i>	RM	V/V	SSSE3	Shuffle bytes in <i>mm1</i> according to contents of <i>mm2/m64</i> .
66 0F 38 00 /r PSHUFB <i>xmm1</i> , <i>xmm2/m128</i>	RM	V/V	SSSE3	Shuffle bytes in <i>xmm1</i> according to contents of <i>xmm2/m128</i> .
VEX.NDS.128.66.0F38.WIG 00 /r VPSHUFB <i>xmm1</i> , <i>xmm2</i> , <i>xmm3/m128</i>	RVM	V/V	AVX	Shuffle bytes in <i>xmm2</i> according to contents of <i>xmm3/m128</i> .
VEX.NDS.256.66.0F38.WIG 00 /r VPSHUFB <i>ymm1</i> , <i>ymm2</i> , <i>ymm3/m256</i>	RVM	V/V	AVX2	Shuffle bytes in <i>ymm2</i> according to contents of <i>ymm3/m256</i> .

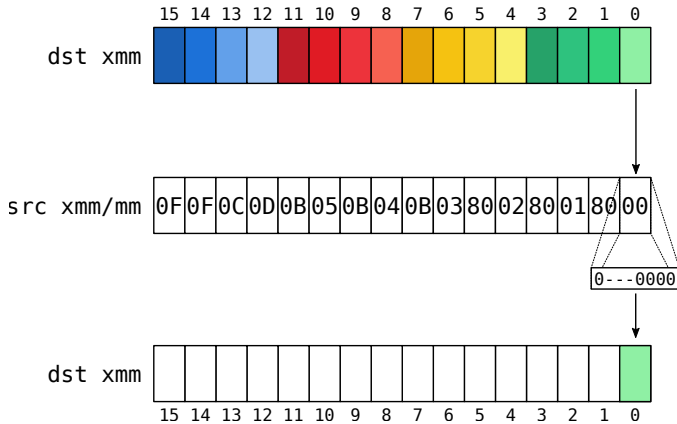
PSHUFB (with 128 bit operands)

```
for i = 0 to 15 {  
    if (SRC[(i * 8)+7] = 1 ) then  
        DEST[(i*8)+7..(i*8)+0] ← 0;  
    else  
        index[3..0] ← SRC[(i*8)+3 .. (i*8)+0];  
        DEST[(i*8)+7..(i*8)+0] ← DEST[(index*8+7)..(index*8+0)];  
    endif  
}  
DEST[VLMAX-1:128] ← 0
```

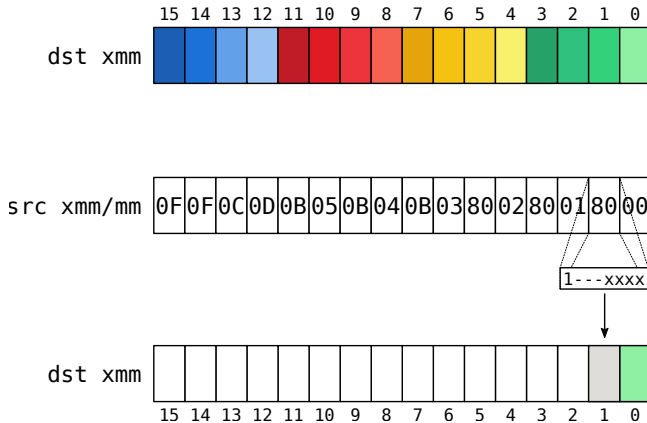
Ejemplo - PSHUFB dst, src



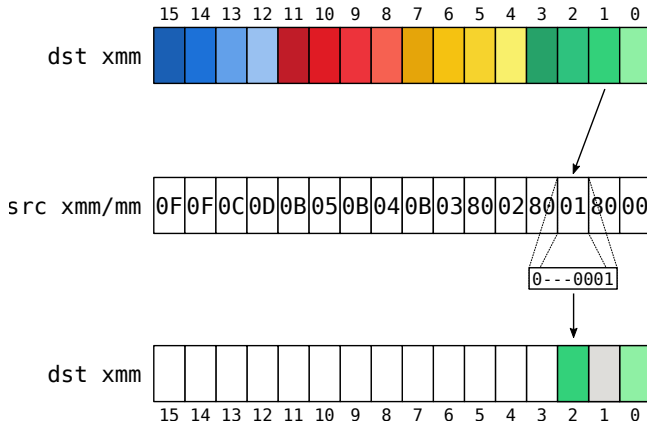
Ejemplo - PSHUFB dst, src



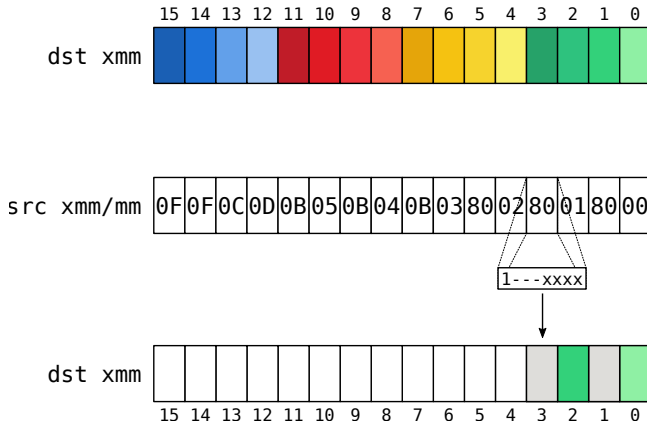
Ejemplo - PSHUFB dst, src



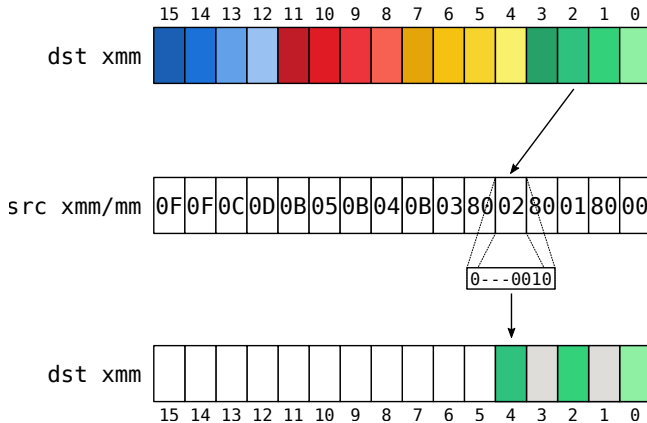
Ejemplo - PSHUFB dst, src



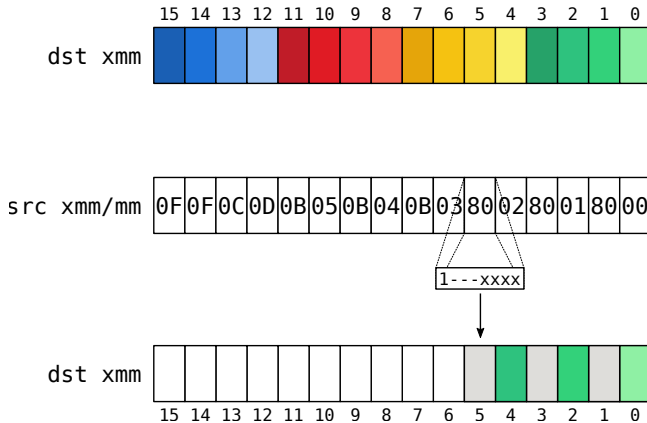
Ejemplo - PSHUFB dst, src



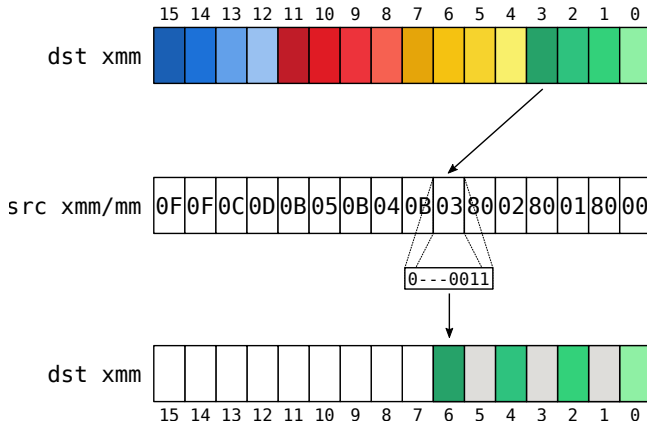
Ejemplo - PSHUFB dst, src



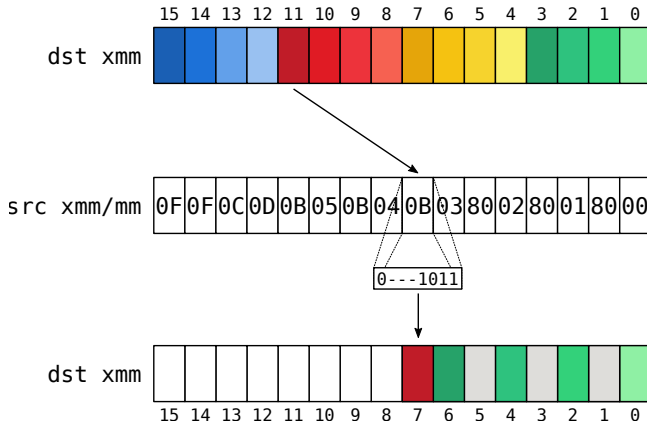
Ejemplo - PSHUFB dst, src



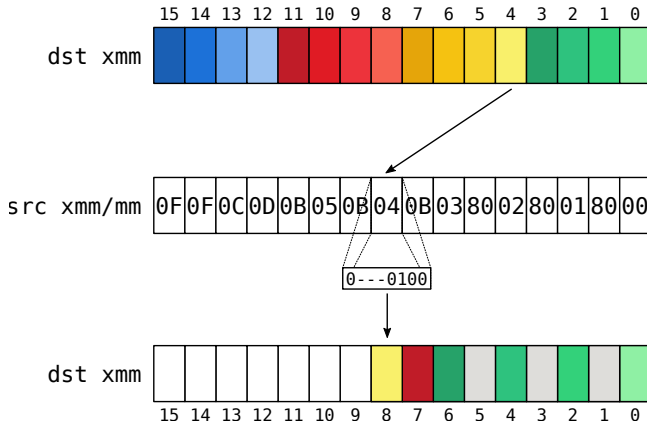
Ejemplo - PSHUFB dst, src



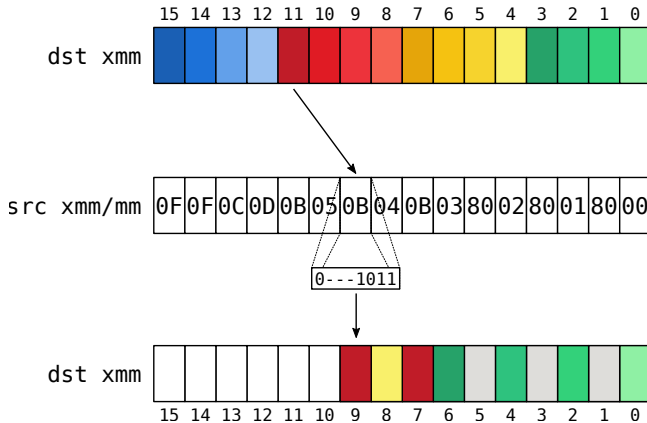
Ejemplo - PSHUFB dst, src



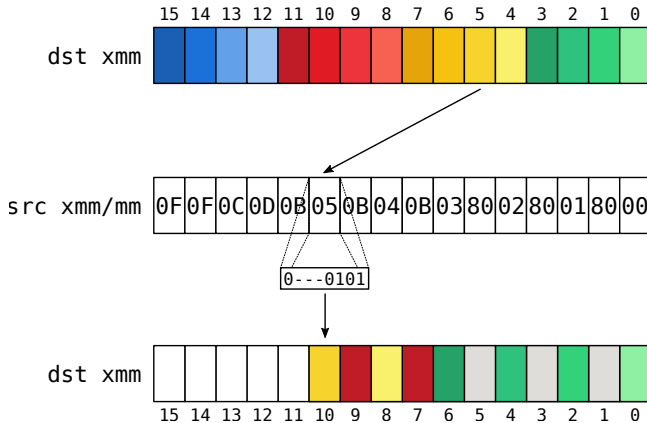
Ejemplo - PSHUFB dst, src



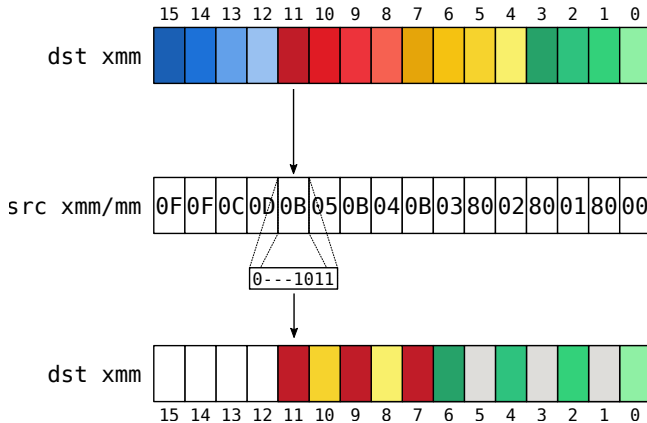
Ejemplo - PSHUFB dst, src



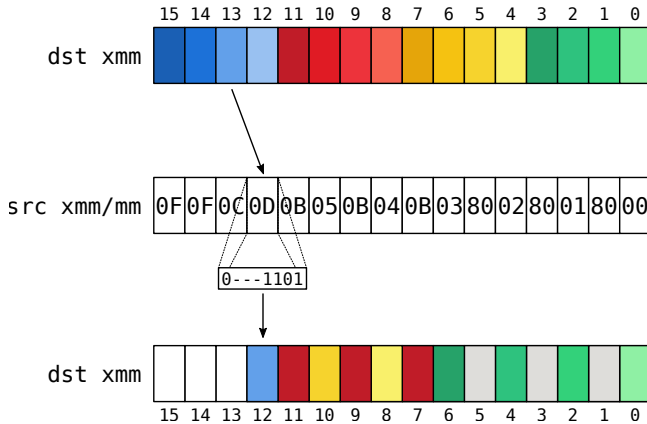
Ejemplo - PSHUFB dst, src



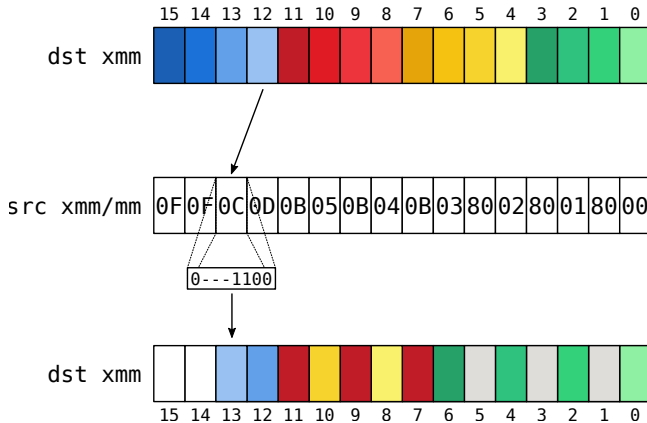
Ejemplo - PSHUFB dst, src



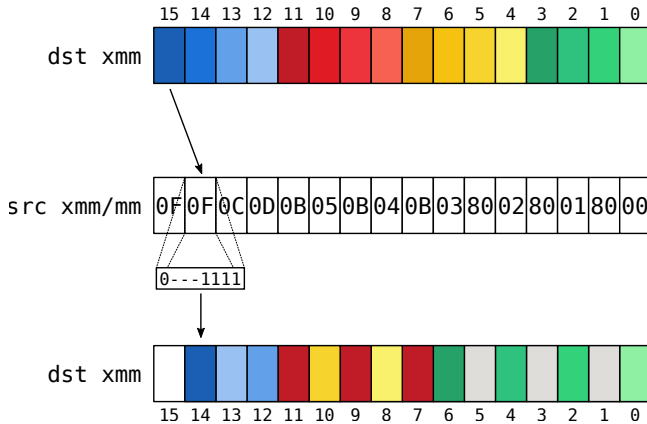
Ejemplo-PSHUFB dst, src



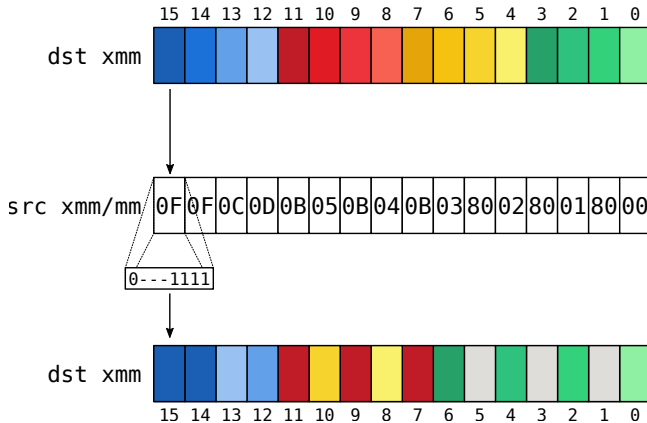
Ejemplo - PSHUFB dst, src



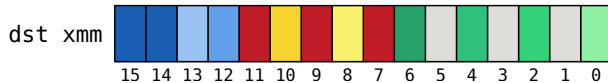
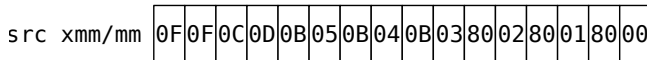
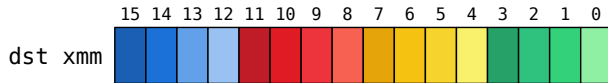
Ejemplo - PSHUFB dst, src



Ejemplo - PSHUFB dst, src



Ejemplo - PSHUFB dst, src



Shuffles

PSHUFLW—Shuffle Packed Low Words

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F2 0F 70 /r ib PSHUFLW <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	SSE2	Shuffle the low words in <i>xmm2/m128</i> based on the encoding in <i>imm8</i> and store the result in <i>xmm1</i> .
VEX.128.F2.0F.WIG 70 /r ib VPSHUFLW <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	AVX	Shuffle the low words in <i>xmm2/m128</i> based on the encoding in <i>imm8</i> and store the result in <i>xmm1</i> .
VEX.256.F2.0F.WIG 70 /r ib VPSHUFLW <i>ymm1</i> , <i>ymm2/m256</i> , <i>imm8</i>	RMI	V/V	AVX2	Shuffle the low words in <i>ymm2/m256</i> based on the encoding in <i>imm8</i> and store the result in <i>ymm1</i> .

Shuffles

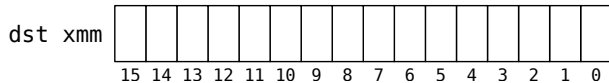
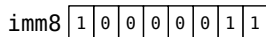
PSHUFLW—Shuffle Packed Low Words

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F2 0F 70 /r ib PSHUFLW <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	SSE2	Shuffle the low words in <i>xmm2/m128</i> based on the encoding in <i>imm8</i> and store the result in <i>xmm1</i> .
VEX.128.F2.0F.WIG 70 /r ib VPSHUFLW <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	AVX	Shuffle the low words in <i>xmm2/m128</i> based on the encoding in <i>imm8</i> and store the result in <i>xmm1</i> .
VEX.256.F2.0F.WIG 70 /r ib VPSHUFLW <i>ymm1</i> , <i>ymm2/m256</i> , <i>imm8</i>	RMI	V/V	AVX2	Shuffle the low words in <i>ymm2/m256</i> based on the encoding in <i>imm8</i> and store the result in <i>ymm1</i> .

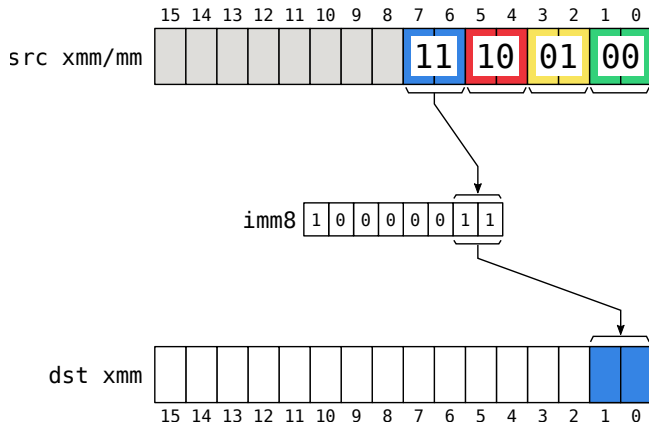
PSHUFLW (128-bit Legacy SSE version)

$\text{DEST}[15:0] \leftarrow (\text{SRC} \gg (\text{imm}[1:0] * 16))[15:0]$
 $\text{DEST}[31:16] \leftarrow (\text{SRC} \gg (\text{imm}[3:2] * 16))[15:0]$
 $\text{DEST}[47:32] \leftarrow (\text{SRC} \gg (\text{imm}[5:4] * 16))[15:0]$
 $\text{DEST}[63:48] \leftarrow (\text{SRC} \gg (\text{imm}[7:6] * 16))[15:0]$
 $\text{DEST}[127:64] \leftarrow \text{SRC}[127:64]$
 $\text{DEST}[\text{VLMAX}-1:128]$ (Unmodified)

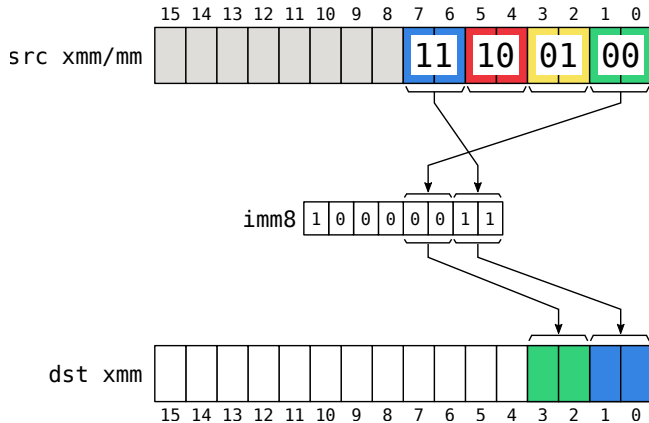
Ejemplo - PSHUFLW dst, src , imm8



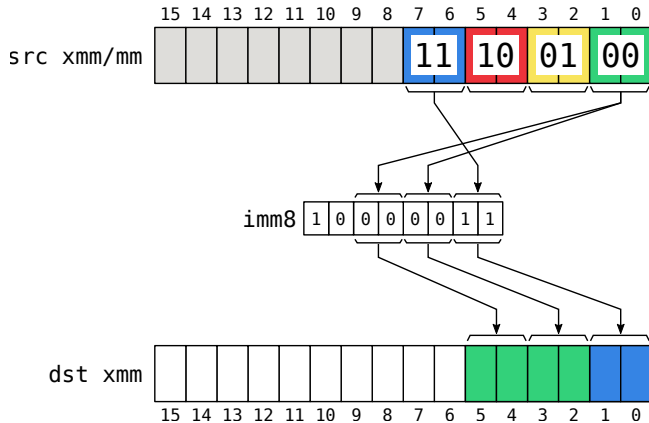
Ejemplo-PSHUFLW dst, src , imm8



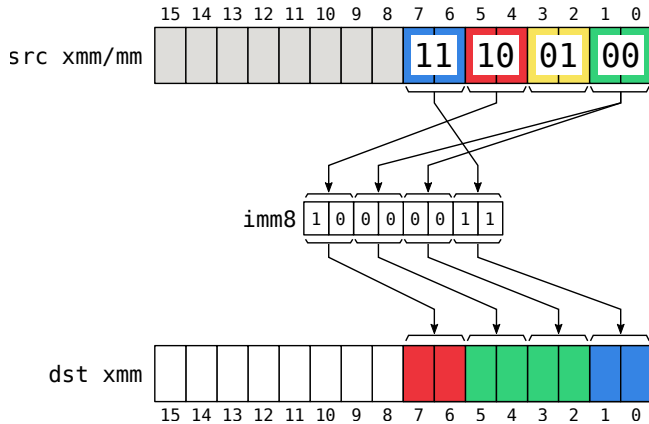
Ejemplo-PSHUFLW dst, src , imm8



Ejemplo-PSHUFLW dst, src , imm8



Ejemplo-PSHUFLW dst, src , imm8



Shuffles

PSHUFHW—Shuffle Packed High Words

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 70 /r ib PSHUFHW <i>xmm1, xmm2/m128, imm8</i>	RMI	V/V	SSE2	Shuffle the high words in <i>xmm2/m128</i> based on the encoding in <i>imm8</i> and store the result in <i>xmm1</i> .
VEX.128.F3.0F.WIG 70 /r ib VPSHUFHW <i>xmm1, xmm2/m128, imm8</i>	RMI	V/V	AVX	Shuffle the high words in <i>xmm2/m128</i> based on the encoding in <i>imm8</i> and store the result in <i>xmm1</i> .
VEX.256.F3.0F.WIG 70 /r ib VPSHUFHW <i>ymm1, ymm2/m256, imm8</i>	RMI	V/V	AVX2	Shuffle the high words in <i>ymm2/m256</i> based on the encoding in <i>imm8</i> and store the result in <i>ymm1</i> .

Shuffles

PSHUFHW—Shuffle Packed High Words

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
F3 0F 70 /r ib PSHUFHW <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	SSE2	Shuffle the high words in <i>xmm2/m128</i> based on the encoding in <i>imm8</i> and store the result in <i>xmm1</i> .
VEX.128.F3.0F.WIG 70 /r ib VPSHUFHW <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	AVX	Shuffle the high words in <i>xmm2/m128</i> based on the encoding in <i>imm8</i> and store the result in <i>xmm1</i> .
VEX.256.F3.0F.WIG 70 /r ib VPSHUFHW <i>ymm1</i> , <i>ymm2/m256</i> , <i>imm8</i>	RMI	V/V	AVX2	Shuffle the high words in <i>ymm2/m256</i> based on the encoding in <i>imm8</i> and store the result in <i>ymm1</i> .

PSHUFHW (128-bit Legacy SSE version)

DEST[63:0] \leftarrow SRC[63:0]

DEST[79:64] \leftarrow (SRC \gg (imm[1:0] * 16))[79:64]

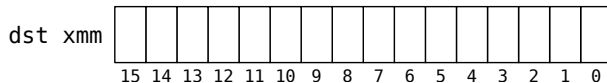
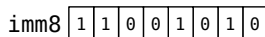
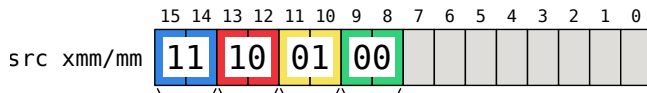
DEST[95:80] \leftarrow (SRC \gg (imm[3:2] * 16))[79:64]

DEST[111:96] \leftarrow (SRC \gg (imm[5:4] * 16))[79:64]

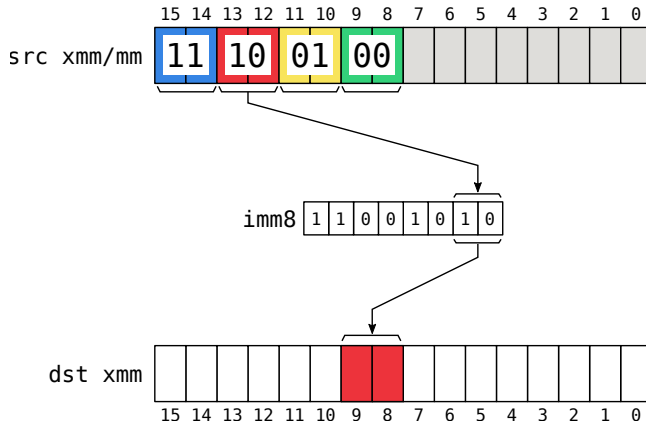
DEST[127:112] \leftarrow (SRC \gg (imm[7:6] * 16))[79:64]

DEST[VLMAX-1:128] (Unmodified)

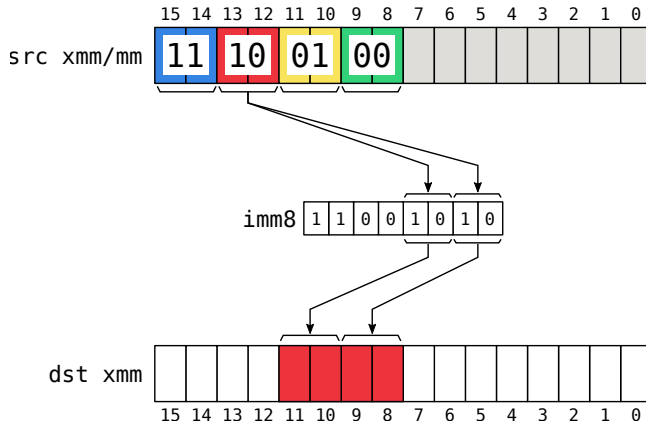
Ejemplo - PSHUFW dst, src , imm8



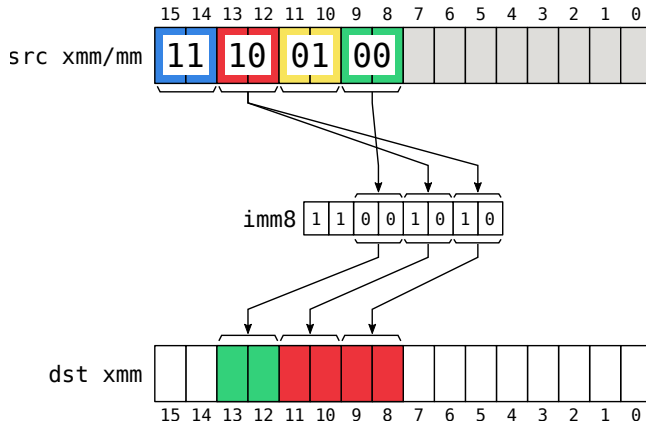
Ejemplo-PSHUFHW dst, src , imm8



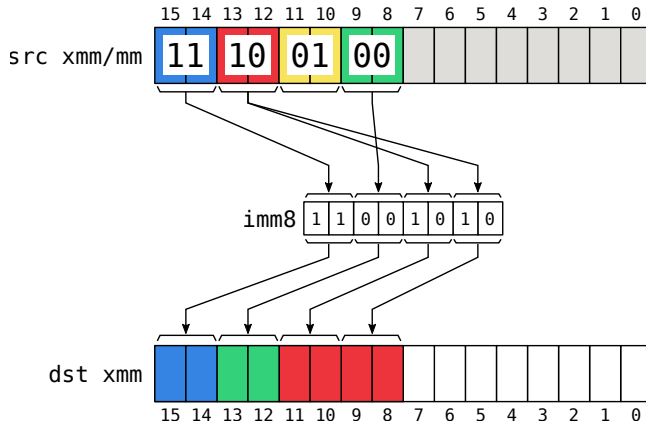
Ejemplo-PSHUFHW dst, src , imm8



Ejemplo-PSHUFHW dst, src , imm8



Ejemplo-PSHUFHW dst, src , imm8



Shuffles

PSHUFD—Shuffle Packed Doublewords

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 70 /r ib PSHUFD <i>xmm1, xmm2/m128, imm8</i>	RMI	V/V	SSE2	Shuffle the doublewords in <i>xmm2/m128</i> based on the encoding in <i>imm8</i> and store the result in <i>xmm1</i> .
VEX.128.66.0F.WIG 70 /r ib VPSHUFD <i>xmm1, xmm2/m128, imm8</i>	RMI	V/V	AVX	Shuffle the doublewords in <i>xmm2/m128</i> based on the encoding in <i>imm8</i> and store the result in <i>xmm1</i> .
VEX.256.66.0F.WIG 70 /r ib VPSHUFD <i>ymm1, ymm2/m256, imm8</i>	RMI	V/V	AVX2	Shuffle the doublewords in <i>ymm2/m256</i> based on the encoding in <i>imm8</i> and store the result in <i>ymm1</i> .

Shuffles

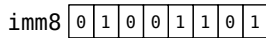
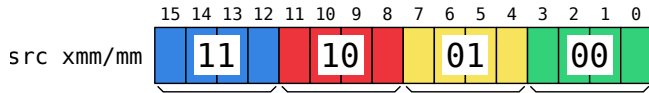
PSHUFD—Shuffle Packed Doublewords

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 70 /r ib PSHUFD <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	SSE2	Shuffle the doublewords in <i>xmm2/m128</i> based on the encoding in <i>imm8</i> and store the result in <i>xmm1</i> .
VEX.128.66.0F.WIG 70 /r ib VPSHUFD <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	AVX	Shuffle the doublewords in <i>xmm2/m128</i> based on the encoding in <i>imm8</i> and store the result in <i>xmm1</i> .
VEX.256.66.0F.WIG 70 /r ib VPSHUFD <i>ymm1</i> , <i>ymm2/m256</i> , <i>imm8</i>	RMI	V/V	AVX2	Shuffle the doublewords in <i>ymm2/m256</i> based on the encoding in <i>imm8</i> and store the result in <i>ymm1</i> .

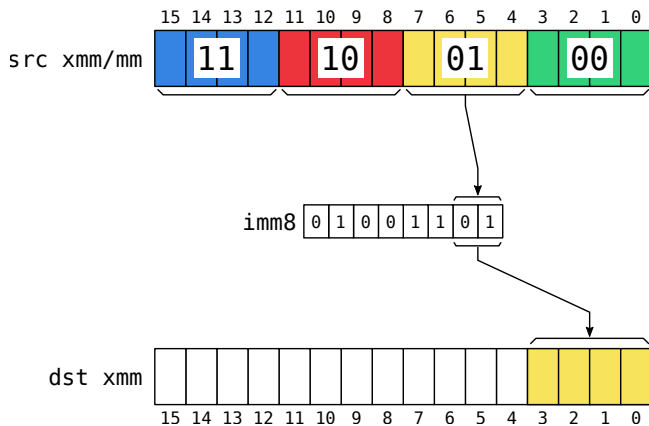
PSHUFD (128-bit Legacy SSE version)

```
DEST[31:0] ← (SRC >> (ORDER[1:0] * 32))[31:0];  
DEST[63:32] ← (SRC >> (ORDER[3:2] * 32))[31:0];  
DEST[95:64] ← (SRC >> (ORDER[5:4] * 32))[31:0];  
DEST[127:96] ← (SRC >> (ORDER[7:6] * 32))[31:0];  
DEST[VLMAX-1:128] (Unmodified)
```

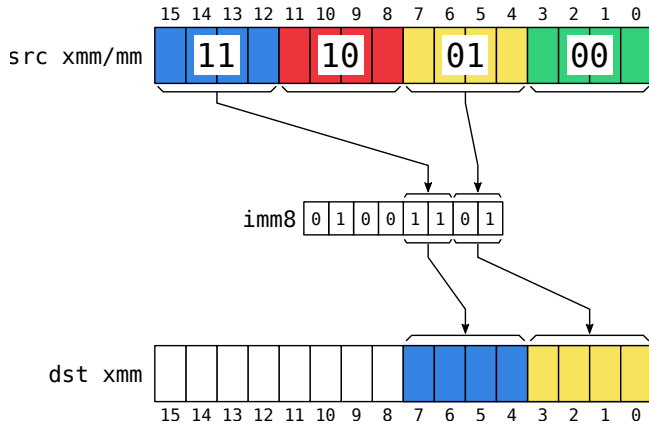
Ejemplo-PSHUFD dst, src , imm8



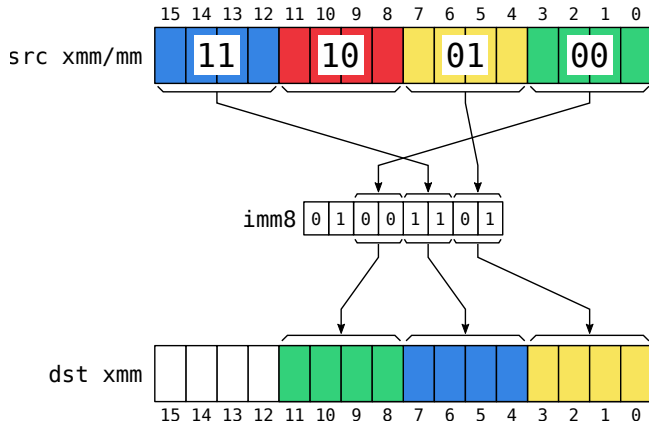
Ejemplo-PSHUFD dst, src , imm8



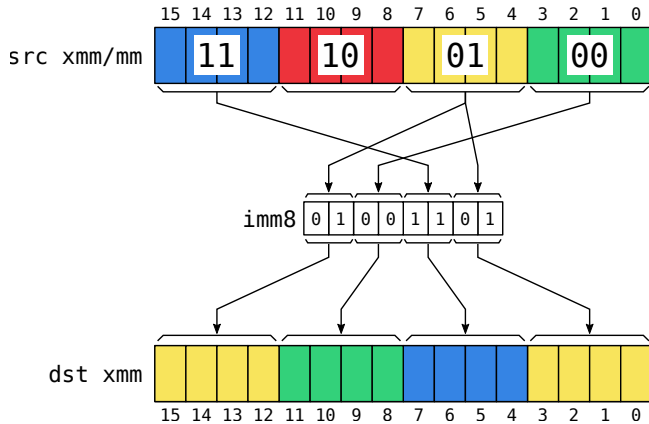
Ejemplo-PSHUFD dst, src , imm8



Ejemplo-PSHUFD dst, src , imm8



Ejemplo-PSHUFD dst, src , imm8



Shuffles

SHUFPS—Shuffle Packed Single-Precision Floating-Point Values

Opcode*/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
OF C6 /r ib SHUFPS <i>xmm1, xmm2/m128, imm8</i>	RMI	V/V	SSE	Shuffle packed single-precision floating-point values selected by <i>imm8</i> from <i>xmm1</i> and <i>xmm1</i> ² <i>m128</i> to <i>xmm1</i> .
VEX.NDS.128.OF.WIG C6 /r ib VSHUFPS <i>xmm1, xmm2, xmm3/m128, imm8</i>	RVMI	V/V	AVX	Shuffle Packed single-precision floating-point values selected by <i>imm8</i> from <i>xmm2</i> and <i>xmm3/mem</i> .
VEX.NDS.256.OF.WIG C6 /r ib VSHUFPS <i>ymm1, ymm2, ymm3/m256, imm8</i>	RVMI	V/V	AVX	Shuffle Packed single-precision floating-point values selected by <i>imm8</i> from <i>ymm2</i> and <i>ymm3/mem</i> .

Shuffles

SHUFPS—Shuffle Packed Single-Precision Floating-Point Values

Opcode*/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
OF C6 /r ib SHUFPS <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	SSE	Shuffle packed single-precision floating-point values selected by <i>imm8</i> from <i>xmm1</i> and <i>xmm2/m128</i> to <i>xmm1</i> .
VEX.NDS.128.OF.WIG C6 /r ib VSHUFPS <i>xmm1</i> , <i>xmm2</i> , <i>xmm3/m128</i> , <i>imm8</i>	RVMI	V/V	AVX	Shuffle Packed single-precision floating-point values selected by <i>imm8</i> from <i>xmm2</i> and <i>xmm3/mem</i> .
VEX.NDS.256.OF.WIG C6 /r ib VSHUFPS <i>ymm1</i> , <i>ymm2</i> , <i>ymm3/m256</i> , <i>imm8</i>	RVMI	V/V	AVX	Shuffle Packed single-precision floating-point values selected by <i>imm8</i> from <i>ymm2</i> and <i>ymm3/mem</i> .

SHUFPS (128-bit Legacy SSE version)

~~DEST~~
~~DEST~~
~~DEST~~
~~DEST~~
~~DEST~~

~~DEST[31:0] ← Select4(SRC1[127:0], imm8[1:0]);~~
~~DEST[63:32] ← Select4(SRC1[127:0], imm8[3:2]);~~
~~DEST[95:64] ← Select4(SRC2[127:0], imm8[5:4]);~~
~~DEST[127:96] ← Select4(SRC2[127:0], imm8[7:6]);~~
~~DEST[VLMAX-1:128] (Unmodified)~~

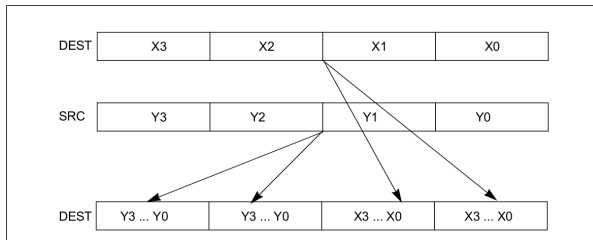
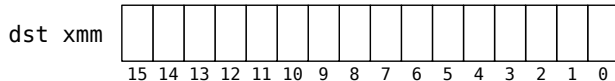
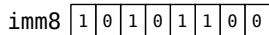
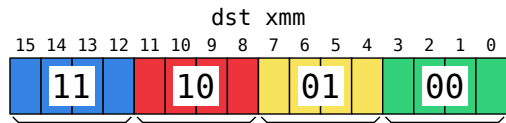
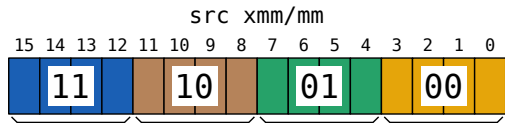
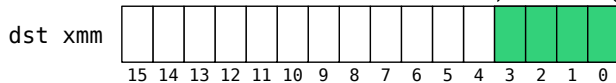
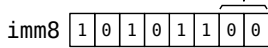
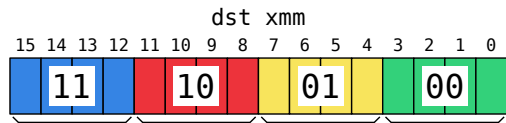
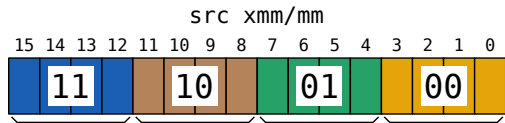


Figure 4-22. SHUFPS Shuffle Operation

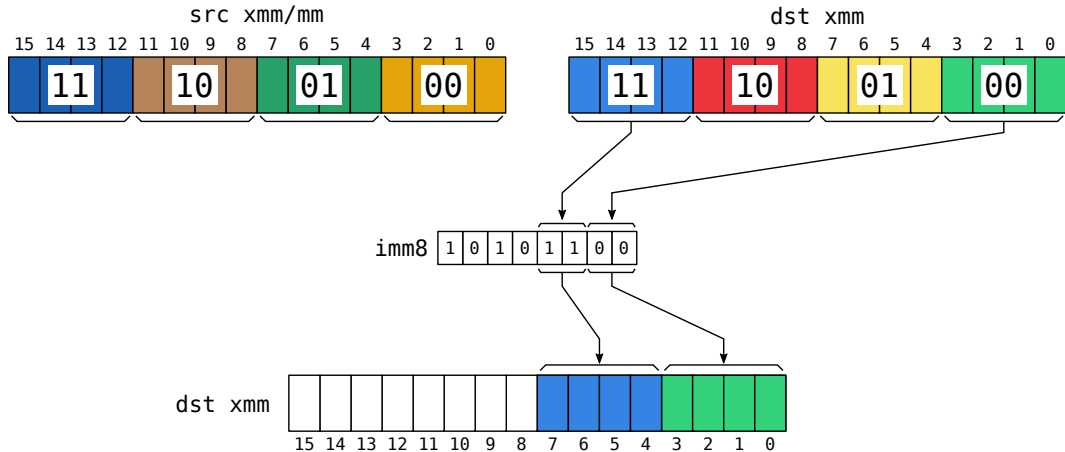
Ejemplo - SHUFPS dst, src , imm8



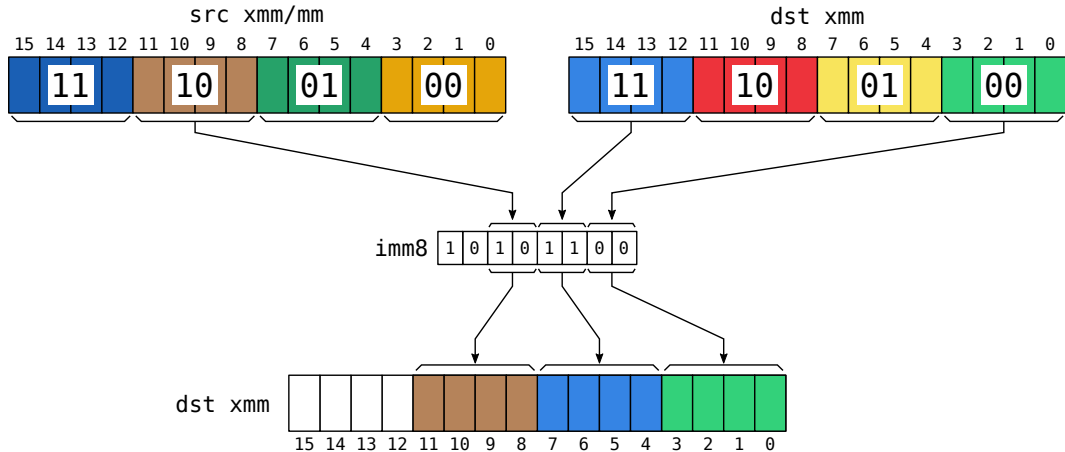
Ejemplo - SHUFPS dst, src , imm8



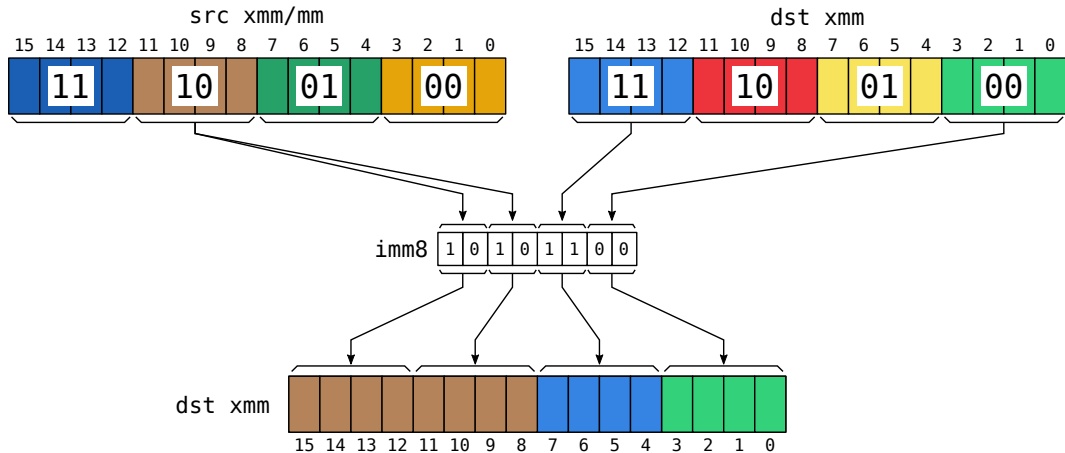
Ejemplo-SHUFPS dst, src , imm8



Ejemplo-SHUFPS dst, src , imm8



Ejemplo-SHUFPS dst, src , imm8



Shuffles

SHUFPD—Shuffle Packed Double-Precision Floating-Point Values

Opcode*/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F C6 /r ib SHUFPD <i>xmm1, xmm2/m128, imm8</i>	RMI	V/V	SSE2	Shuffle packed double-precision floating-point values selected by <i>imm8</i> from <i>xmm1</i> and <i>xmm2/m128</i> to <i>xmm1</i> .
VEX.NDS.128.66.0F.WIG C6 /r ib VSHUFPD <i>xmm1, xmm2, xmm3/m128, imm8</i>	RVMI	V/V	AVX	Shuffle Packed double-precision floating-point values selected by <i>imm8</i> from <i>xmm2</i> and <i>xmm3/mem</i> .
VEX.NDS.256.66.0F.WIG C6 /r ib VSHUFPD <i>ymm1, ymm2, ymm3/m256, imm8</i>	RVMI	V/V	AVX	Shuffle Packed double-precision floating-point values selected by <i>imm8</i> from <i>ymm2</i> and <i>ymm3/mem</i> .

SHUFPD (128-bit Legacy SSE version)

IF $IMM0[0] = 0$
 THEN $DEST[63:0] \leftarrow \text{DEST}[63:0]$
 ELSE $DEST[63:0] \leftarrow \text{SRC1}[127:64] \text{ FI};$
 IF $IMM0[1] = 0$
 THEN $DEST[127:64] \leftarrow \text{SRC2}[63:0]$
 ELSE $DEST[127:64] \leftarrow \text{SRC2}[127:64] \text{ FI};$
 $DEST[VLMAX-1:128]$ (Unmodified)

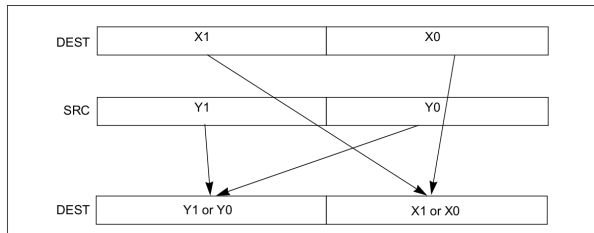
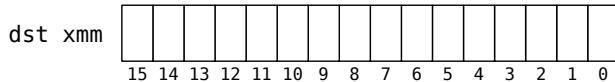
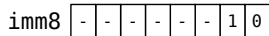
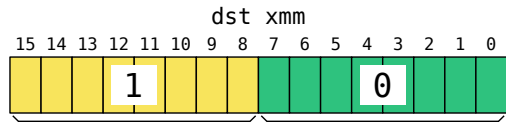
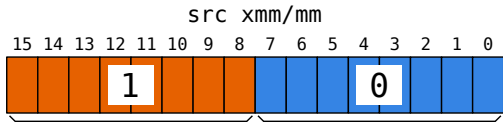
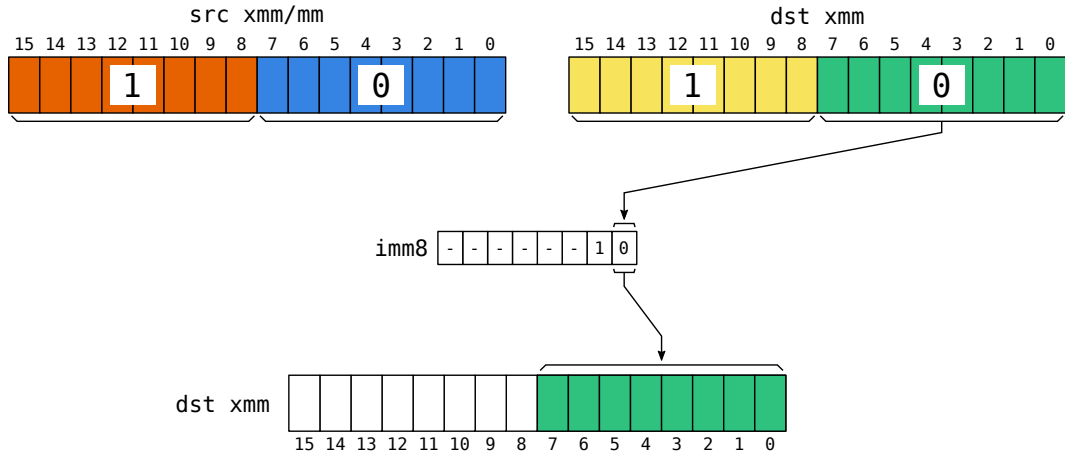


Figure 4-21. SHUFPD Shuffle Operation

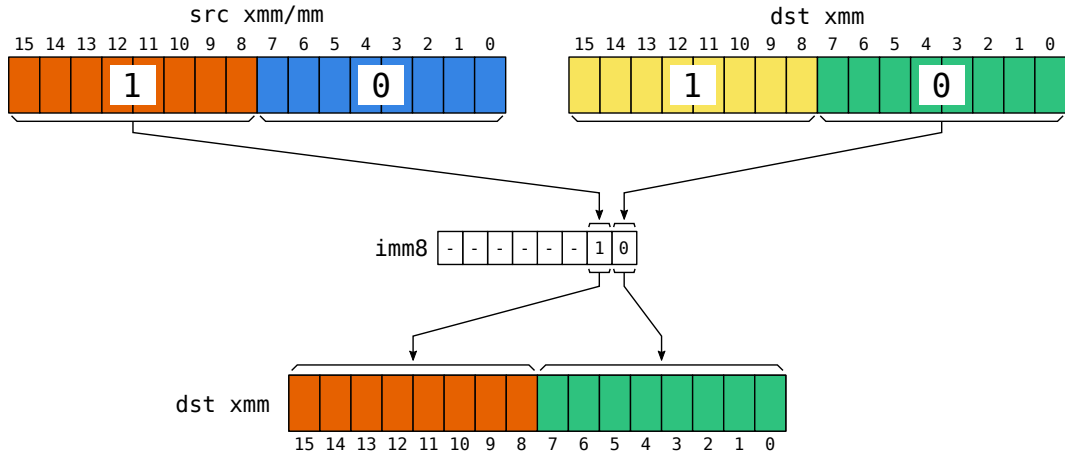
Ejemplo - SHUFPD dst, src , imm8



Ejemplo - SHUFPD dst, src , imm8



Ejemplo - SHUFPD dst, src , imm8



Insert/Extract

Las instrucciones de *Insert* y *Extract*, permiten como su nombre lo indica, **insertar** y **extraer** valores dentro de un registro.

Insert/Extract

Las instrucciones de *Insert* y *Extract*, permiten como su nombre lo indica, **insertar** y **extraer** valores dentro de un registro.

- INSERTPS - Insert Packed Single FP Value
- EXTRACTPS - Extract Packed Single FP Value
- PINSRB - Insert Byte
- PINSRW - Insert Word
- PINSRD - Insert Dword
- PINSRQ - Insert Qword
- PEXTRB - Extract Byte
- PEXTRW - Extract Word
- PEXTRD - Extract Dword
- PEXTRQ - Extract Qword

Insert Extract

INSERTPS — Insert Packed Single Precision Floating-Point Value

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 3A 21 /r ib INSERTPS <i>xmm1</i> , <i>xmm2/m32</i> , <i>imm8</i>	RMI	V/V	SSE4_1	Insert a single precision floating-point value selected by <i>imm8</i> from <i>xmm2/m32</i> into <i>xmm1</i> at the specified destination element specified by <i>imm8</i> and zero out destination elements in <i>xmm1</i> as indicated in <i>imm8</i> .
VEX.NDS.128.66.0F3A.WIG 21 /r ib VINSERTPS <i>xmm1</i> , <i>xmm2</i> , <i>xmm3/m32</i> , <i>imm8</i>	RVMI	V/V	AVX	Insert a single precision floating point value selected by <i>imm8</i> from <i>xmm3/m32</i> and merge into <i>xmm2</i> at the specified destination element specified by <i>imm8</i> and zero out destination elements in <i>xmm1</i> as indicated in <i>imm8</i> .

Insert Extract

INSERTPS — Insert Packed Single Precision Floating-Point Value

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 3A 21 /r ib INSERTPS <i>xmm1</i> , <i>xmm2/m32</i> , <i>imm8</i>	RMI	V/V	SSE4_1	Insert a single precision floating-point value selected by <i>imm8</i> from <i>xmm2/m32</i> into <i>xmm1</i> at the specified destination element specified by <i>imm8</i> and zero out destination elements in <i>xmm1</i> as indicated in <i>imm8</i> .
VEX.NDS.128.66.0F3A.WIG 21 /r ib VINSERTPS <i>xmm1</i> , <i>xmm2</i> , <i>xmm3/m32</i> , <i>imm8</i>	RVMI	V/V	AVX	Insert a single precision floating point value selected by <i>imm8</i> from <i>xmm3/m32</i> and merge into <i>xmm2</i> at the specified destination element specified by <i>imm8</i> and zero out destination elements in <i>xmm1</i> as indicated in <i>imm8</i> .

INSERTPS (128-bit Legacy SSE version)

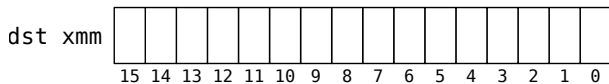
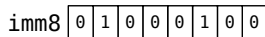
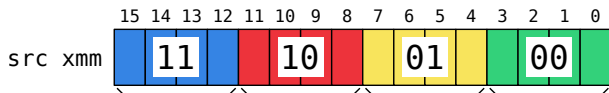
```
IF (SRC = REG) THEN COUNT_S ← imm8[7:6]
    ELSE COUNT_S ← 0
COUNT_D ← imm8[5:4]
ZMASK ← imm8[3:0]
CASE (COUNT_S) OF
    0: TMP ← SRC[31:0]
    1: TMP ← SRC[63:32]
    2: TMP ← SRC[95:64]
    3: TMP ← SRC[127:96]
ESAC;
```

CASE (COUNT_D) OF

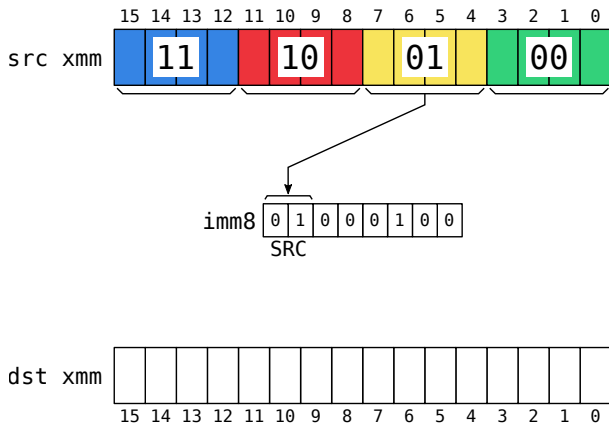
```
0: TMP2[31:0] ← TMP
    TMP2[127:32] ← DEST[127:32]
1: TMP2[63:32] ← TMP
    TMP2[31:0] ← DEST[31:0]
    TMP2[127:64] ← DEST[127:64]
2: TMP2[95:64] ← TMP
    TMP2[63:0] ← DEST[63:0]
    TMP2[127:96] ← DEST[127:96]
3: TMP2[127:96] ← TMP
    TMP2[95:0] ← DEST[95:0]
ESAC;
```

```
IF (ZMASK[0] = 1) THEN DEST[31:0] ← 00000000H
    ELSE DEST[31:0] ← TMP2[31:0]
IF (ZMASK[1] = 1) THEN DEST[63:32] ← 00000000H
    ELSE DEST[63:32] ← TMP2[63:32]
IF (ZMASK[2] = 1) THEN DEST[95:64] ← 00000000H
    ELSE DEST[95:64] ← TMP2[95:64]
IF (ZMASK[3] = 1) THEN DEST[127:96] ← 00000000H
    ELSE DEST[127:96] ← TMP2[127:96]
DEST[VLMAX-1:128] (Unmodified)
```

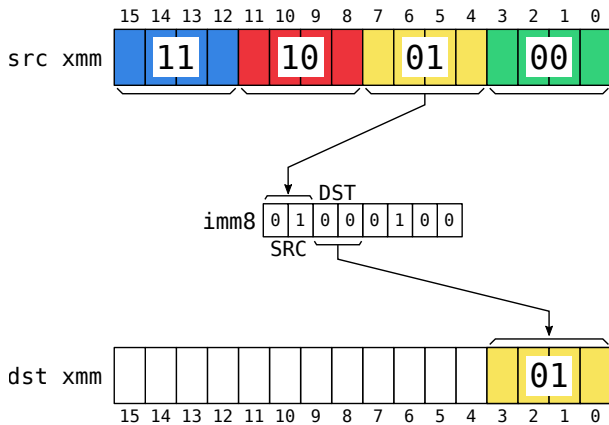
Ejemplo - INSERTPS dst, src , imm8



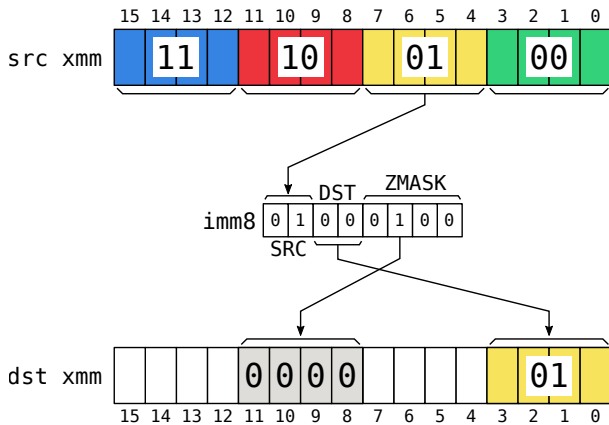
Ejemplo - INSERTPS dst, src , imm8



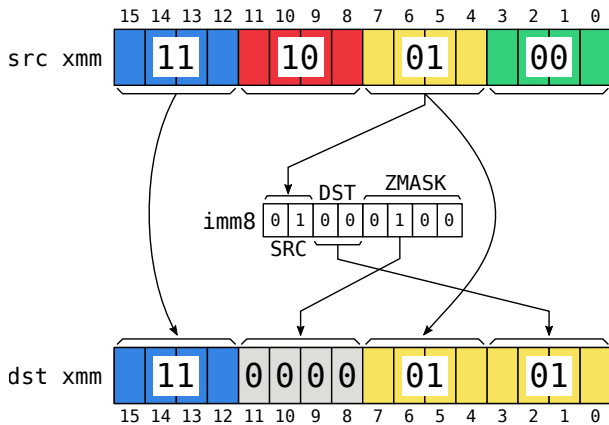
Ejemplo - INSERTPS dst, src , imm8



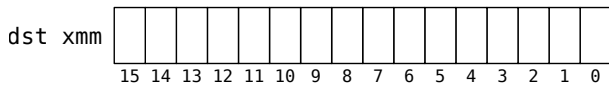
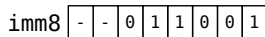
Ejemplo - INSERTPS dst, src , imm8



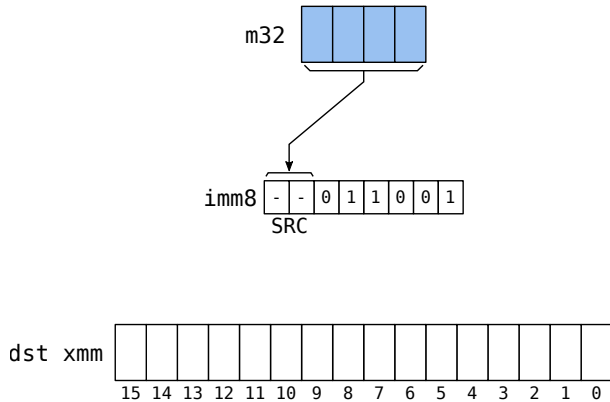
Ejemplo - INSERTPS dst, src , imm8



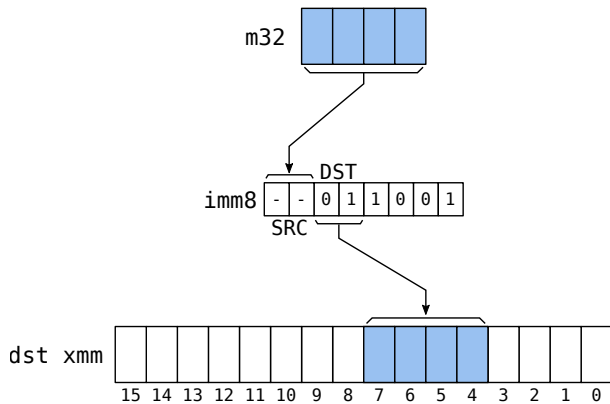
Ejemplo - INSERTPS dst, src , imm8



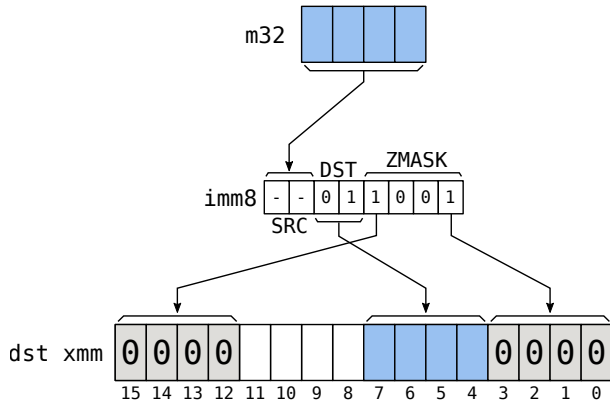
Ejemplo - INSERTPS dst, src , imm8



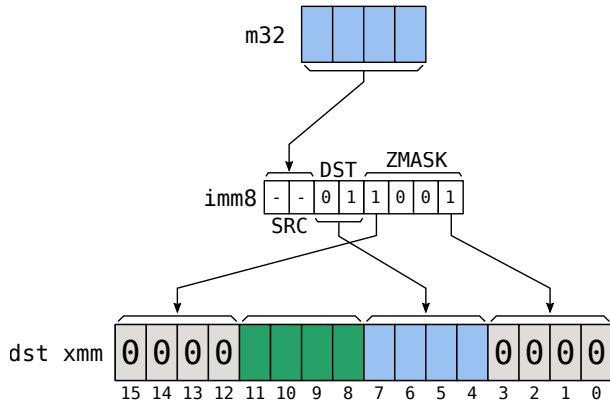
Ejemplo - INSERTPS dst, src , imm8



Ejemplo - INSERTPS dst, src , imm8



Ejemplo - INSERTPS dst, src , imm8



Insert Extract

EXTRACTPS — Extract Packed Single Precision Floating-Point Value

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 3A 17 /r ib EXTRACTPS <i>reg/m32, xmm2, imm8</i>	MRI	V/V	SSE4_1	Extract a single-precision floating-point value from <i>xmm2</i> at the source offset specified by <i>imm8</i> and store the result to <i>reg or m32</i> . The upper 32 bits of <i>r64</i> is zeroed if <i>reg</i> is <i>r64</i> .
VEX.128.66.0F3A.WIG 17 /r ib VEXTRACTPS <i>r/m32, xmm1, imm8</i>	MRI	V/V	AVX	Extract one single-precision floating-point value from <i>xmm1</i> at the offset specified by <i>imm8</i> and store the result in <i>reg or m32</i> . Zero extend the results in 64-bit register if applicable.

Insert Extract

EXTRACTPS — Extract Packed Single Precision Floating-Point Value

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 3A 17 /r ib EXTRACTPS <i>reg/m32, xmm2, imm8</i>	MRI	V/V	SSE4_1	Extract a single-precision floating-point value from <i>xmm2</i> at the source offset specified by <i>imm8</i> and store the result to <i>reg or m32</i> . The upper 32 bits of <i>r64</i> is zeroed if <i>reg</i> is <i>r64</i> .
VEX.128.66.0F3A.WIG 17 /r ib VEXTRACTPS <i>r/m32, xmm1, imm8</i>	MRI	V/V	AVX	Extract one single-precision floating-point value from <i>xmm1</i> at the offset specified by <i>imm8</i> and store the result in <i>reg or m32</i> . Zero extend the results in 64-bit register if applicable.

EXTRACTPS (128-bit Legacy SSE version)

SRC_OFFSET ← IMM8[1:0]

IF (64-Bit Mode and DEST is register)

DEST[31:0] ← (SRC[127:0] » (SRC_OFFSET*32)) AND 0FFFFFFFh

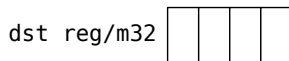
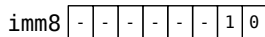
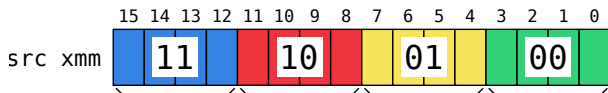
DEST[63:32] ← 0

ELSE

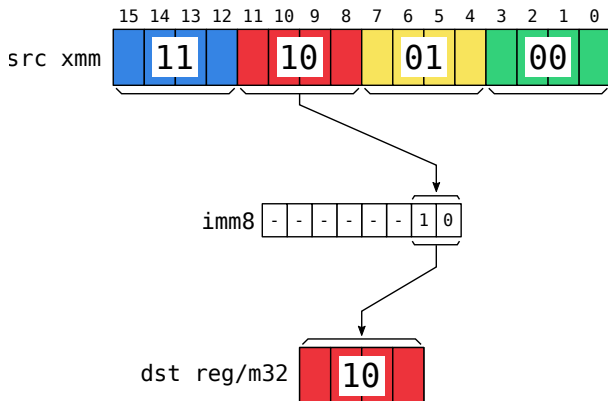
DEST[31:0] ← (SRC[127:0] » (SRC_OFFSET*32)) AND 0FFFFFFFh

FI

Ejemplo-EXTRACTPS dst, src , imm8



Ejemplo-EXTRACTPS dst, src , imm8



Insert Extract

PINSRB/PINSRD/PINSRQ — Insert Byte/Dword/Qword

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 OF 3A 20 /r ib PINSRB <i>xmm1</i> , <i>r32/m8</i> , <i>imm8</i>	RMI	V/V	SSE4_1	Insert a byte integer value from <i>r32/m8</i> into <i>xmm1</i> at the destination element in <i>xmm1</i> specified by <i>imm8</i> .
66 OF 3A 22 /r ib PINSRD <i>xmm1</i> , <i>r/m32</i> , <i>imm8</i>	RMI	V/V	SSE4_1	Insert a dword integer value from <i>r/m32</i> into the <i>xmm1</i> at the destination element specified by <i>imm8</i> .
66 REX.W OF 3A 22 /r ib PINSRQ <i>xmm1</i> , <i>r/m64</i> , <i>imm8</i>	RMI	V/N. E.	SSE4_1	Insert a qword integer value from <i>r/m64</i> into the <i>xmm1</i> at the destination element specified by <i>imm8</i> .
VEX.NDS.128.66.0F3A.W0 20 /r ib VPINSRB <i>xmm1</i> , <i>xmm2</i> , <i>r32/m8</i> , <i>imm8</i>	RVMI	V ¹ /V	AVX	Merge a byte integer value from <i>r32/m8</i> and rest from <i>xmm2</i> into <i>xmm1</i> at the byte offset in <i>imm8</i> .
VEX.NDS.128.66.0F3A.W0 22 /r ib VPINSRD <i>xmm1</i> , <i>xmm2</i> , <i>r/m32</i> , <i>imm8</i>	RVMI	V/V	AVX	Insert a dword integer value from <i>r32/m32</i> and rest from <i>xmm2</i> into <i>xmm1</i> at the dword offset in <i>imm8</i> .
VEX.NDS.128.66.0F3A.W1 22 /r ib VPINSRQ <i>xmm1</i> , <i>xmm2</i> , <i>r/m64</i> , <i>imm8</i>	RVMI	V/I	AVX	Insert a qword integer value from <i>r64/m64</i> and rest from <i>xmm2</i> into <i>xmm1</i> at the qword offset in <i>imm8</i> .

Insert Extract

PINSRB/PINSRD/PINSRQ – Insert Byte/Dword/Qword

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 OF 3A 20 /r ib PINSRB <i>xmm1</i> , <i>r32/m8</i> , <i>imm8</i>	RMI	V/V	SSE4_1	Insert a byte integer value from <i>r32/m8</i> into <i>xmm1</i> at the destination element in <i>xmm1</i> specified by <i>imm8</i> .
66 OF 3A 22 /r ib PINSRD <i>xmm1</i> , <i>r/m32</i> , <i>imm8</i>	RMI	V/V	SSE4_1	Insert a dword integer value from <i>r/m32</i> into the <i>xmm1</i> at the destination element specified by <i>imm8</i> .
66 REX.W OF 3A 22 /r ib PINSRQ <i>xmm1</i> , <i>r/m64</i> , <i>imm8</i>	RMI	V/N. E.	SSE4_1	Insert a qword integer value from <i>r/m64</i> into the <i>xmm1</i> at the destination element specified by <i>imm8</i> .
VEX.NDS.128.66.0F3A.W0 20 /r ib VPINSRB <i>xmm1</i> , <i>xmm2</i> , <i>r32/m8</i> , <i>imm8</i>	RVMI			<p>CASE OF</p> <p>PINSRB: $SEL \leftarrow COUNT[3:0];$ $MASK \leftarrow (OFFH \ll (SEL * 8));$ $TEMP \leftarrow (((SRC[7:0] \ll (SEL * 8)) \text{ AND } MASK);$ PINSRD: $SEL \leftarrow COUNT[1:0];$ $MASK \leftarrow (OFFFFFFFFFH \ll (SEL * 32));$ $TEMP \leftarrow (((SRC \ll (SEL * 32)) \text{ AND } MASK) ;$ PINSRQ: $SEL \leftarrow COUNT[0]$ $MASK \leftarrow (OFFFFFFFFFFFFFFFFFFFH \ll (SEL * 64));$ $TEMP \leftarrow (((SRC \ll (SEL * 32)) \text{ AND } MASK) ;$ ESAC; $DEST \leftarrow ((DEST \text{ AND } \text{NOT } MASK) \text{ OR } TEMP);$</p>
VEX.NDS.128.66.0F3A.W0 22 /r ib VPINSRD <i>xmm1</i> , <i>xmm2</i> , <i>r/m32</i> , <i>imm8</i>	RVMI			
VEX.NDS.128.66.0F3A.W1 22 /r ib VPINSRQ <i>xmm1</i> , <i>xmm2</i> , <i>r/m64</i> , <i>imm8</i>	RVMI			

Insert Extract

PINSRW—Insert Word

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F C4 /r ib ¹ PINSRW <i>mm</i> , <i>r32/m16</i> , <i>imm8</i>	RMI	V/V	SSE	Insert the low word from <i>r32</i> or from <i>m16</i> into <i>mm</i> at the word position specified by <i>imm8</i> .
66 0F C4 /r ib PINSRW <i>xmm</i> , <i>r32/m16</i> , <i>imm8</i>	RMI	V/V	SSE2	Move the low word of <i>r32</i> or from <i>m16</i> into <i>xmm</i> at the word position specified by <i>imm8</i> .
VEX.NDS.128.66.0F.W0 C4 /r ib VPINSRW <i>xmm1</i> , <i>xmm2</i> , <i>r32/m16</i> , <i>imm8</i>	RVMI	V ² /V	AVX	Insert a word integer value from <i>r32/m16</i> and rest from <i>xmm2</i> into <i>xmm1</i> at the word offset in <i>imm8</i> .

Insert Extract

PINSRW—Insert Word

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F C4 /r ib ¹ PINSRW <i>mm</i> , <i>r32/m16</i> , <i>imm8</i>	RMI	V/V	SSE	Insert the low word from <i>r32</i> or from <i>m16</i> into <i>mm</i> at the word position specified by <i>imm8</i> .
66 0F C4 /r ib PINSRW <i>xmm</i> , <i>r32/m16</i> , <i>imm8</i>	RMI	V/V	SSE2	Move the low word of <i>r32</i> or from <i>m16</i> into <i>xmm</i> at the word position specified by <i>imm8</i> .
VEX.NDS.128.66.0F.W0 C4 /r ib VPINSRW <i>xmm1</i> , <i>xmm2</i> , <i>r32/m16</i> , <i>imm8</i>	RVMI	V ² /V	AVX	Insert a word integer value from <i>r32/m16</i> and rest from <i>xmm2</i> into <i>xmm1</i> at the word offset in <i>imm8</i> .

PINSRW (with 128-bit source operand)

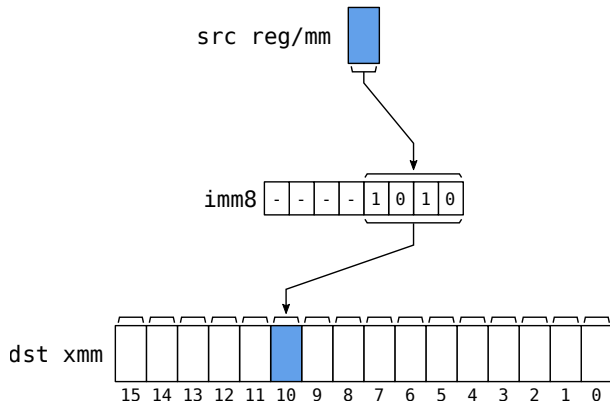
SEL ← COUNT AND 7H;

CASE (Determine word position) OF

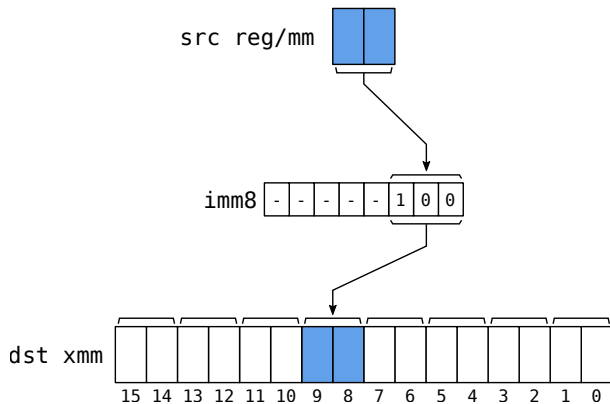
SEL ← 0: MASK ← 0000000000000000000000000000FFFFH;
 SEL ← 1: MASK ← 0000000000000000000000000000FFFF0000H;
 SEL ← 2: MASK ← 0000000000000000000000000000FFFF00000000H;
 SEL ← 3: MASK ← 0000000000000000000000000000FFFF000000000000H;
 SEL ← 4: MASK ← 000000000000FFFF00000000000000000000000H;
 SEL ← 5: MASK ← 00000000FFFF000000000000000000000000H;
 SEL ← 6: MASK ← 0000FFFF0000000000000000000000000000H;
 SEL ← 7: MASK ← FFFF00000000000000000000000000000000H;

DEST ← (DEST AND NOT MASK) OR (((SRC << (SEL * 16)) AND MASK);

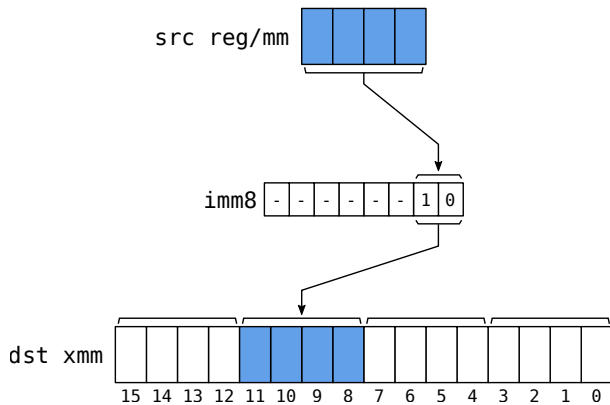
Ejemplo - PINSRB dst, src , imm8



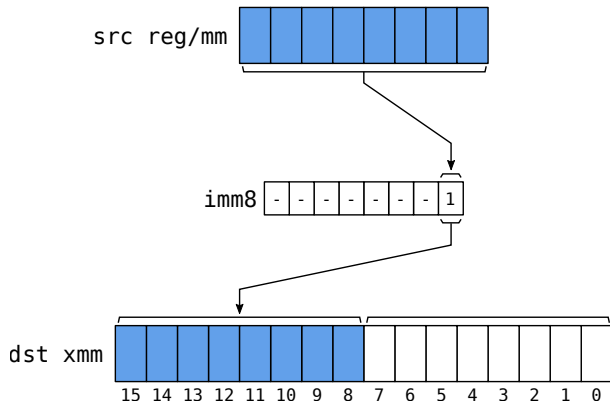
Ejemplo - PINSRW dst, src , imm8



Ejemplo-PINSRD dst, src , imm8



Ejemplo-PINSRQ dst, src , imm8



Insert Extract

PEXTRB/PEXTRD/PEXTRQ — Extract Byte/Dword/Qword

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 3A 14 /r ib PEXTRB <i>reg/m8, xmm2, imm8</i>	MRI	V/V	SSE4_1	Extract a byte integer value from <i>xmm2</i> at the source byte offset specified by <i>imm8</i> into <i>reg</i> or <i>m8</i> . The upper bits of r32 or r64 are zeroed.
66 0F 3A 16 /r ib PEXTRD <i>r/m32, xmm2, imm8</i>	MRI	V/V	SSE4_1	Extract a dword integer value from <i>xmm2</i> at the source dword offset specified by <i>imm8</i> into <i>r/m32</i> .
66 REX.W 0F 3A 16 /r ib PEXTRQ <i>r/m64, xmm2, imm8</i>	MRI	V/N.E.	SSE4_1	Extract a qword integer value from <i>xmm2</i> at the source qword offset specified by <i>imm8</i> into <i>r/m64</i> .
VEX.128.66.0F3A.W0 14 /r ib VPEXTRB <i>reg/m8, xmm2, imm8</i>	MRI	V ¹ /V	AVX	Extract a byte integer value from <i>xmm2</i> at the source byte offset specified by <i>imm8</i> into <i>reg</i> or <i>m8</i> . The upper bits of r64/r32 is filled with zeros.
VEX.128.66.0F3A.W0 16 /r ib VPEXTRD <i>r32/m32, xmm2, imm8</i>	MRI	V/V	AVX	Extract a dword integer value from <i>xmm2</i> at the source dword offset specified by <i>imm8</i> into <i>r32/m32</i> .
VEX.128.66.0F3A.W1 16 /r ib VPEXTRQ <i>r64/m64, xmm2, imm8</i>	MRI	V/i	AVX	Extract a qword integer value from <i>xmm2</i> at the source dword offset specified by <i>imm8</i> into <i>r64/m64</i> .

Insert Extract

PEXTRB/PEXTRD/PEXTRQ — Extract Byte/Dword/Qword

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 3A 14 /r ib PEXTRB <i>reg/m8, xmm2, imm8</i>	MRI	V/V	SSE4_1	Extract a byte integer value from <i>xmm2</i> at the source byte offset specified by <i>imm8</i> into <i>reg</i>
66 0F 3A 16 /r ib PEXTRD <i>r/m32, xmm2, imm8</i>	MRI			CASE of PEXTRB: SEL ← COUNT[3:0]; TEMP ← (Src >> SEL*8) AND FFH; IF (DEST = Mem8) THEN Mem8 ← TEMP[7:0]; ELSE IF (64-Bit Mode and 64-bit register selected) THEN R64[7:0] ← TEMP[7:0]; r64[63:8] ← ZERO_FILL; }; ELSE R32[7:0] ← TEMP[7:0]; r32[31:8] ← ZERO_FILL; }; FI; PEXTRD:SEL ← COUNT[1:0]; TEMP ← (Src >> SEL*32) AND FFFF_FFFFH; DEST ← TEMP; PEXTRQ: SEL ← COUNT[0]; TEMP ← (Src >> SEL*64); DEST ← TEMP; EASC:
66 REX.W 0F 3A 16 /r ib PEXTRQ <i>r/m64, xmm2, imm8</i>	MRI			
VEX.128.66.0F3A.W0 14 /r ib VPEXTRB <i>reg/m8, xmm2, imm8</i>	MRI			
VEX.128.66.0F3A.W0 16 /r ib VPEXTRD <i>r32/m32, xmm2, imm8</i>	MRI			
VEX.128.66.0F3A.W1 16 /r ib VPEXTRQ <i>r64/m64, xmm2, imm8</i>	MRI			

Insert Extract

PEXTRW—Extract Word

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F C5 /r ib ¹ PEXTRW <i>reg, mm, imm8</i>	RMI	V/V	SSE	Extract the word specified by <i>imm8</i> from <i>mm</i> and move it to <i>reg</i> , bits 15-0. The upper bits of r32 or r64 is zeroed.
66 0F C5 /r ib PEXTRW <i>reg, xmm, imm8</i>	RMI	V/V	SSE2	Extract the word specified by <i>imm8</i> from <i>xmm</i> and move it to <i>reg</i> , bits 15-0. The upper bits of r32 or r64 is zeroed.
66 0F 3A 15 /r ib PEXTRW <i>reg/m16, xmm, imm8</i>	MRI	V/V	SSE4_1	Extract the word specified by <i>imm8</i> from <i>xmm</i> and copy it to lowest 16 bits of <i>reg</i> or <i>m16</i> . Zero-extend the result in the destination, r32 or r64.
VEX.128.66.0F.W0 C5 /r ib VPEXTRW <i>reg, xmm1, imm8</i>	RMI	V ² /V	AVX	Extract the word specified by <i>imm8</i> from <i>xmm1</i> and move it to <i>reg</i> , bits 15:0. Zero-extend the result. The upper bits of r64/r32 is filled with zeros.
VEX.128.66.0F3A.W0 15 /r ib VPEXTRW <i>reg/m16, xmm2, imm8</i>	MRI	V/V	AVX	Extract a word integer value from <i>xmm2</i> at the source word offset specified by <i>imm8</i> into <i>reg</i> or <i>m16</i> . The upper bits of r64/r32 is filled with zeros.

Insert Extract

PEXTRW—Extract Word

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F C5 /r ib ¹ PEXTRW <i>reg, mm, imm8</i>	RMI	V/V	SSE	Extract the word specified by <i>imm8</i> from <i>mm</i> and move it to <i>reg</i> , bits 15-0. The upper bits of r32 or r64 is zeroed.
66 0F C5 /r ib PEXTRW <i>reg, xmm, imm8</i>	RMI	V/V	SSE2	Extract the word specified by <i>imm8</i> from <i>xmm</i> and move it to <i>reg</i> , bits 15-0. The upper bits of r32 or r64 is zeroed.

```

IF (DEST = Mem16)
THEN
    SEL ← COUNT[2:0];
    TEMP ← (Src >> SEL*16) AND FFFFH;
    Mem16 ← TEMP[15:0];
ELSE IF (64-Bit Mode and destination is a general-purpose register)
THEN
    FOR (PEXTRW instruction with 64-bit source operand)
    { SEL ← COUNT[1:0];
      TEMP ← (SRC >> (SEL * 16)) AND FFFFH;
      r64[15:0] ← TEMP[15:0];
      r64[63:16] ← ZERO_FILL; };
    FOR (PEXTRW instruction with 128-bit source operand)
    { SEL ← COUNT[2:0];
      TEMP ← (SRC >> (SEL * 16)) AND FFFFH;
      r64[15:0] ← TEMP[15:0];
      r64[63:16] ← ZERO_FILL; }

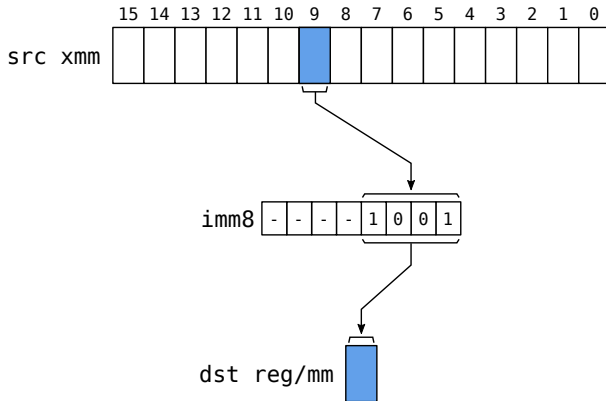
```

```

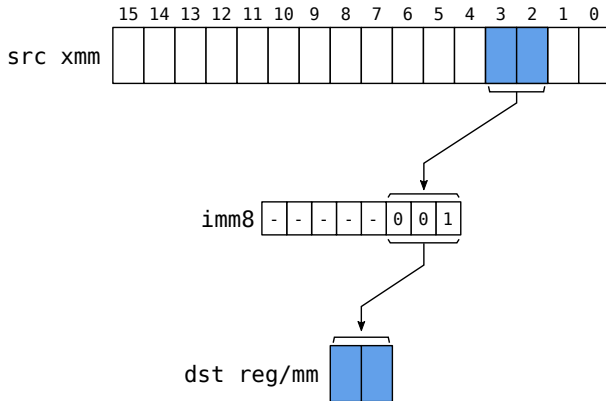
ELSE
    FOR (PEXTRW instruction with 64-bit source operand)
    { SEL ← COUNT[1:0];
      TEMP ← (SRC >> (SEL * 16)) AND FFFFH;
      r32[15:0] ← TEMP[15:0];
      r32[31:16] ← ZERO_FILL; };
    FOR (PEXTRW instruction with 128-bit source operand)
    { SEL ← COUNT[2:0];
      TEMP ← (SRC >> (SEL * 16)) AND FFFFH;
      r32[15:0] ← TEMP[15:0];
      r32[31:16] ← ZERO_FILL; };
FI;
FI;

```

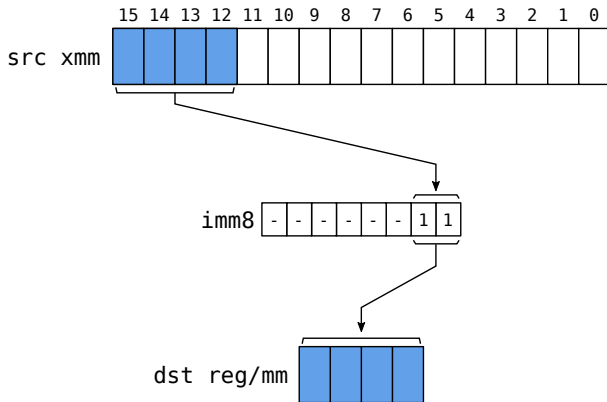
Ejemplo - PEXTRB dst, src , imm8



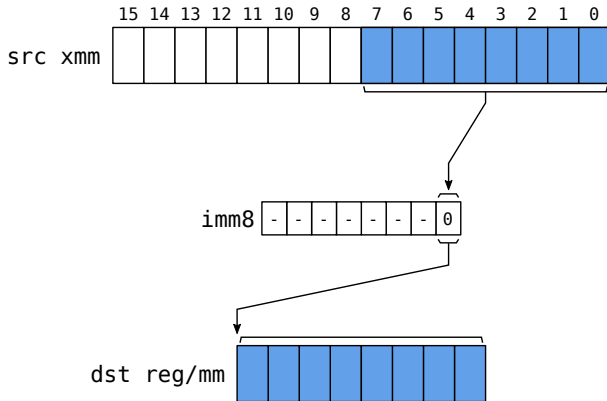
Ejemplo - PEXTRW dst, src , imm8



Ejemplo-PEXTRD dst, src , imm8



Ejemplo-PEXTRQ dst, src , imm8



Blend

Las instrucciones de *Blend* permiten **mezclar** registros dependiendo del valor de sus datos.
Usando tanto inmediatos como otros registros.

Blend

Las instrucciones de *Blend* permiten **mezclar** registros dependiendo del valor de sus datos. Usando tanto inmediatos como otros registros.

- BLENDPS - Blend Packed Single FP Values
- BLENDPD - Blend Packed Double FP Values
- BLENDVPS - Variable Blend Packed Single FP Values
- BLENDVPD - Variable Blend Packed Double FP Values
- PBLENDW - Blend Packed Words
- PBLENDVB - Variable Blend Packed Bytes

Blend

BLENDPS — Blend Packed Single Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 3A 0C /r ib BLENDPS <i>xmm1, xmm2/m128, imm8</i>	RMI	V/V	SSE4_1	Select packed single precision floating-point values from <i>xmm1</i> and <i>xmm2/m128</i> from mask specified in <i>imm8</i> and store the values into <i>xmm1</i> .

BLENDPD — Blend Packed Double Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 3A 0D /r ib BLENDPD <i>xmm1, xmm2/m128, imm8</i>	RMI	V/V	SSE4_1	Select packed DP-FP values from <i>xmm1</i> and <i>xmm2/m128</i> from mask specified in <i>imm8</i> and store the values into <i>xmm1</i> .

Blend

BLENDPS — Blend Packed Single Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 3A 0C /r ib BLENDPS <i>xmm1, xmm2/m128, imm8</i>	RMI	V/V	SSE4_1	Select packed single precision floating-point values from <i>xmm1</i> and <i>xmm2/m128</i> from mask specified in <i>imm8</i> and store the values into <i>xmm1</i> .

BLENDPD — Blend Packed Double Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 3A 0D /r ib BLENDPD <i>xmm1, xmm2/m128, imm8</i>	RMI	V/V	SSE4_1	Select packed DP-FP values from <i>xmm1</i> and <i>xmm2/m128</i> from mask specified in <i>imm8</i> and store the values into <i>xmm1</i> .

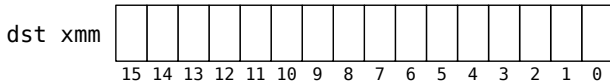
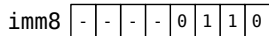
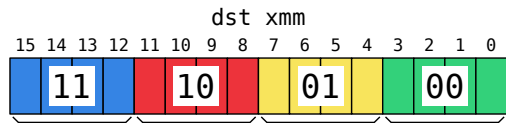
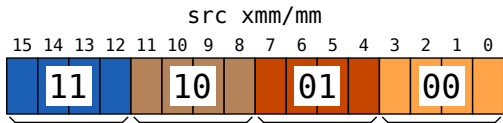
BLENDPS

```
IF (IMM8[0] = 0) THEN DEST[31:0] ← DEST[31:0]
    ELSE DEST [31:0] ← SRC[31:0] FI
IF (IMM8[1] = 0) THEN DEST[63:32] ← DEST[63:32]
    ELSE DEST [63:32] ← SRC[63:32] FI
IF (IMM8[2] = 0) THEN DEST[95:64] ← DEST[95:64]
    ELSE DEST [95:64] ← SRC[95:64] FI
IF (IMM8[3] = 0) THEN DEST[127:96] ← DEST[127:96]
    ELSE DEST [127:96] ← SRC[127:96] FI
DEST[VLMAX-1:128] (Unmodified)
```

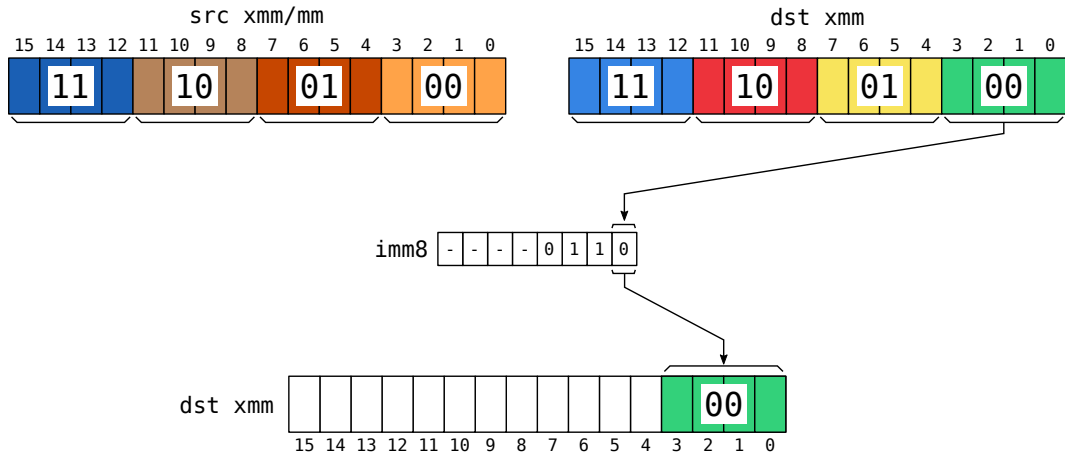
BLENDPD

```
IF (IMM8[0] = 0) THEN DEST[63:0] ← DEST[63:0]
    ELSE DEST [63:0] ← SRC[63:0] FI
IF (IMM8[1] = 0) THEN DEST[127:64] ← DEST[127:64]
    ELSE DEST [127:64] ← SRC[127:64] FI
DEST[VLMAX-1:128] (Unmodified)
```

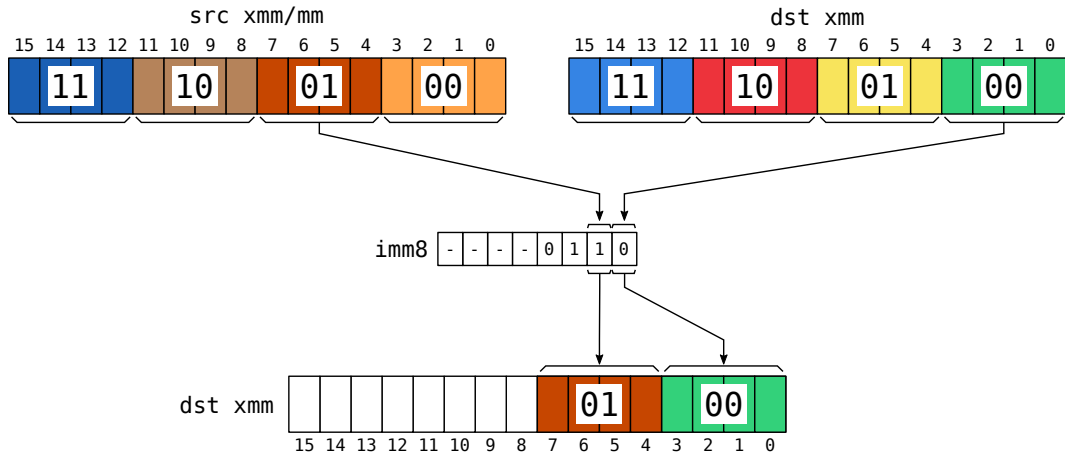
Ejemplo-BLENDPS dst, src , imm8



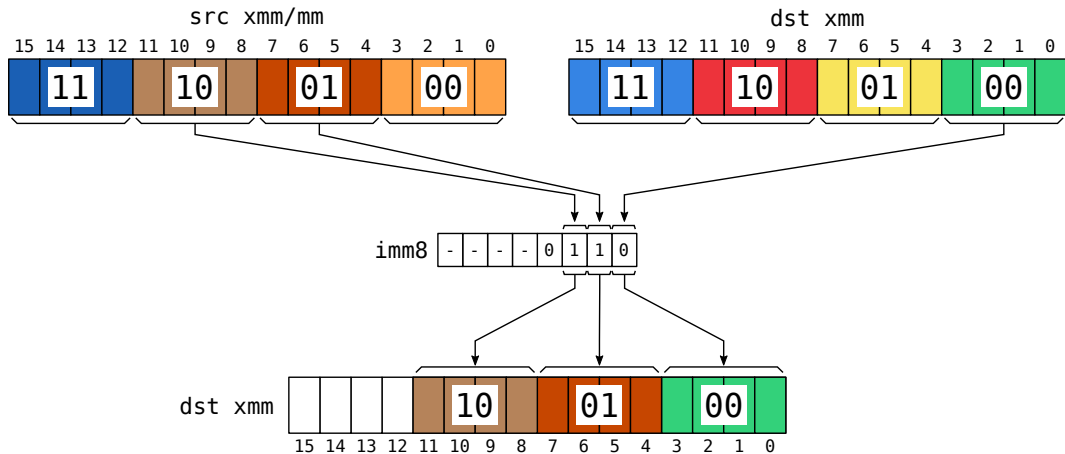
Ejemplo-BLENDPS dst, src , imm8



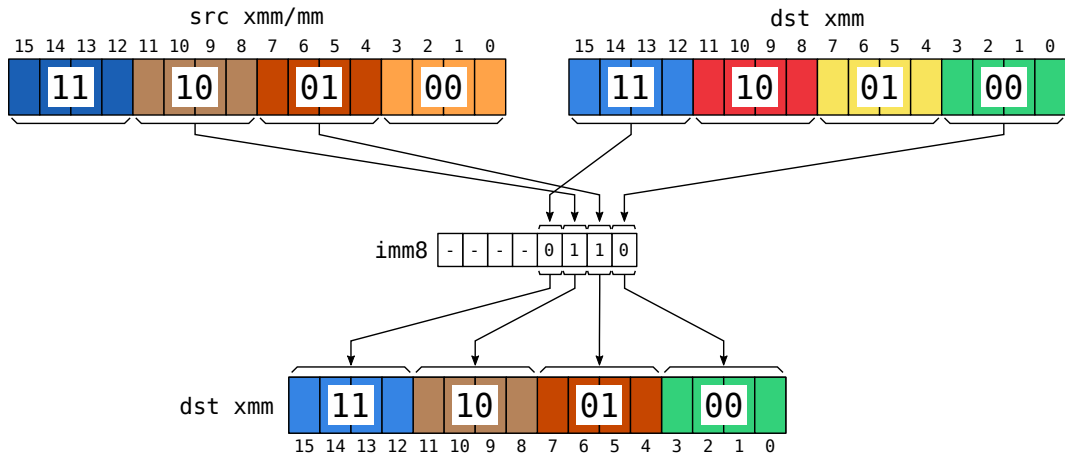
Ejemplo-BLENDPS dst, src , imm8



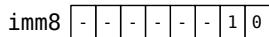
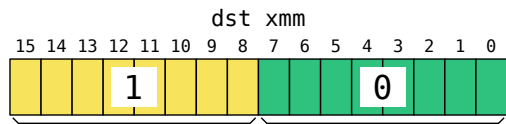
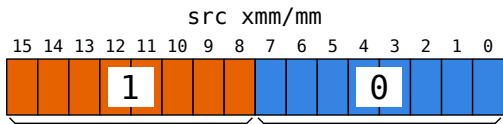
Ejemplo-BLENDPS dst, src , imm8



Ejemplo-BLENDPS dst, src , imm8

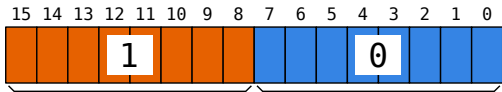


Ejemplo-BLENDPD dst, src , imm8

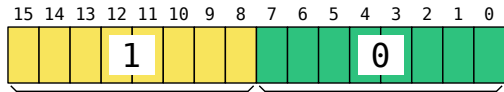


Ejemplo-BLENDPD dst, src , imm8

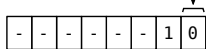
src xmm/mm



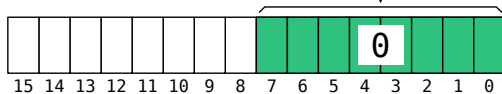
dst xmm



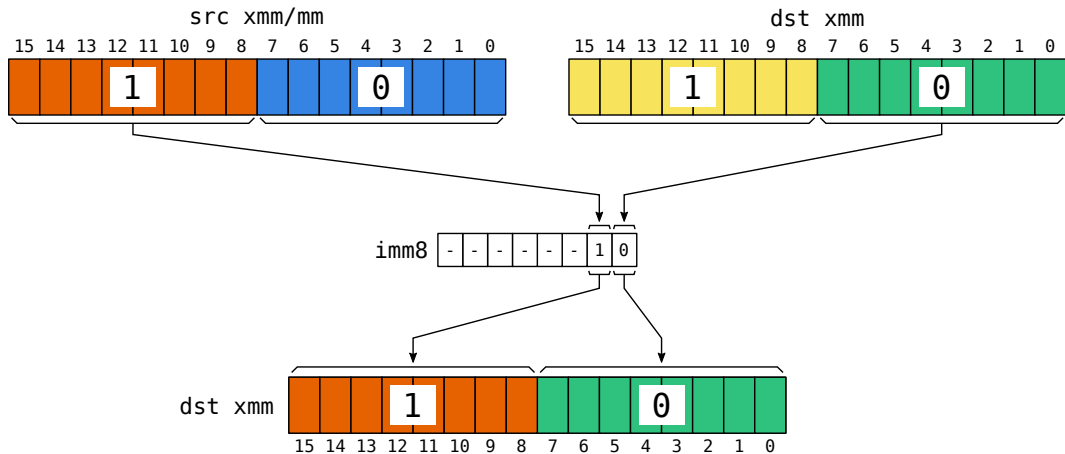
imm8



dst xmm



Ejemplo-BLENDPD dst, src , imm8



Blend

BLENDVPS — Variable Blend Packed Single Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 38 14 /r BLENDVPS <i>xmm1</i> , <i>xmm2/m128</i> , < <i>XMM0</i> >	RMO	V/V	SSE4_1	Select packed single precision floating-point values from <i>xmm1</i> and <i>xmm2/m128</i> from mask specified in <i>XMM0</i> and store the values into <i>xmm1</i> .

BLENDVPD — Variable Blend Packed Double Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 38 15 /r BLENDVPD <i>xmm1</i> , <i>xmm2/m128</i> , < <i>XMM0</i> >	RMO	V/V	SSE4_1	Select packed DP FP values from <i>xmm1</i> and <i>xmm2</i> from mask specified in <i>XMM0</i> and store the values in <i>xmm1</i> .

BLENDVPS — Variable Blend Packed Single Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 38 14 /r BLENDVPS <i>xmm1</i> , <i>xmm2/m128</i> , < <i>XMM0</i> >	RMO	V/V	SSE4_1	Select packed single precision floating-point values from <i>xmm1</i> and <i>xmm2/m128</i> from mask specified in <i>XMM0</i> and store the values into <i>xmm1</i> .

BLENDVPD — Variable Blend Packed Double Precision Floating-Point Values

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 38 15 /r BLENDVPD <i>xmm1</i> , <i>xmm2/m128</i> , < <i>XMM0</i> >	RMO	V/V	SSE4_1	Select packed DP FP values from <i>xmm1</i> and <i>xmm2</i> from mask specified in <i>XMM0</i> and store the values in <i>xmm1</i> .

BLENDVPS

BLENDVPS (128-bit Legacy SSE version)

MASK ← XMM0

IF (MASK[31] = 0) THEN DEST[31:0] ← DEST[31:0]

ELSE DEST [31:0] ← SRC[31:0] FI

IF (MASK[63] = 0) THEN DEST[63:32] ← DEST[63:32]

ELSE DEST [63:32] ← SRC[63:32] FI

IF (MASK[95] = 0) THEN DEST[95:64] ← DEST[95:64]

ELSE DEST [95:64] ← SRC[95:64] FI

IF (MASK[127] = 0) THEN DEST[127:96] ← DEST[127:96]

ELSE DEST [127:96] ← SRC[127:96] FI

DEST[VLMAX-1:128] (Unmodified)

BLENDVPD

BLENDVPD (128-bit Legacy SSE version)

MASK ← XMM0

IF (MASK[63] = 0) THEN DEST[63:0] ← DEST[63:0]

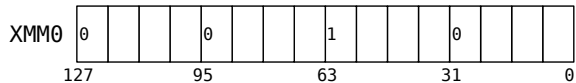
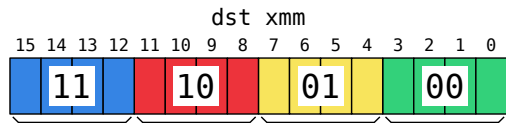
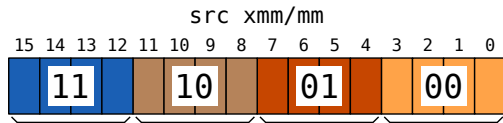
ELSE DEST [63:0] ← SRC[63:0] FI

IF (MASK[127] = 0) THEN DEST[127:64] ← DEST[127:64]

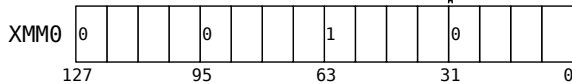
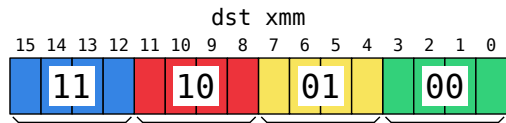
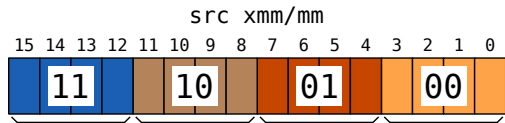
ELSE DEST [127:64] ← SRC[127:64] FI

DEST[VLMAX-1:128] (Unmodified)

Ejemplo-BLENDVPS dst, src , imm8

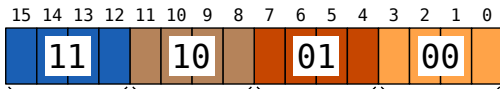


Ejemplo-BLENDVPS dst, src , imm8

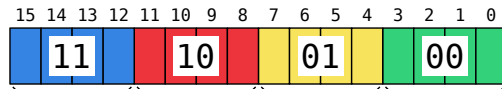


Ejemplo-BLENDVPS dst, src , imm8

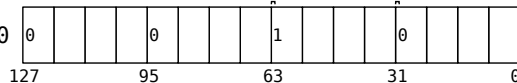
src xmm/mm



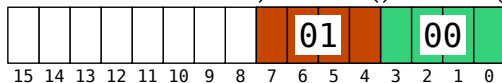
dst xmm



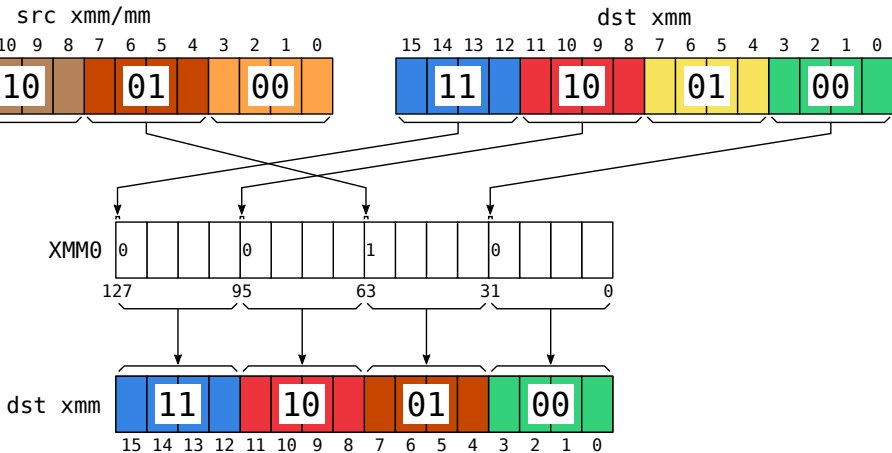
XMM0



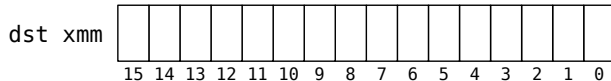
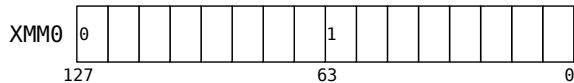
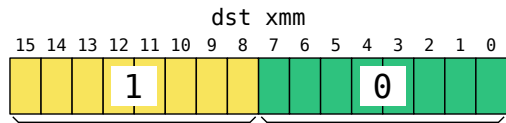
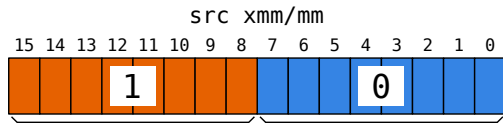
dst xmm



Ejemplo-BLENDVPS dst, src , imm8

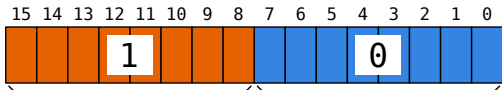


Ejemplo - BLENDVPD dst, src , imm8

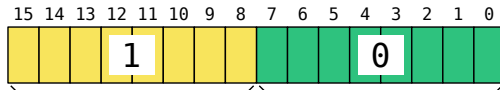


Ejemplo-BLENDVPD dst, src , imm8

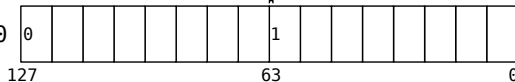
src xmm/mm



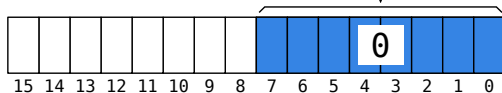
dst xmm



XMM0

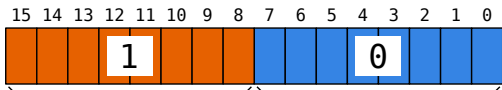


dst xmm

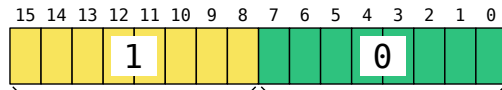


Ejemplo-BLENDVPD dst, src , imm8

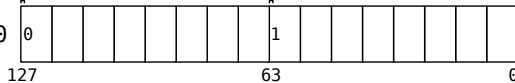
src xmm/mm



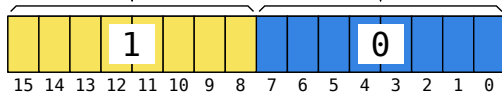
dst xmm



XMM0



dst xmm



Blend

PBLENDW — Blend Packed Words

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 3A 0E /r ib PBLENDW <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	SSE4_1	Select words from <i>xmm1</i> and <i>xmm2/m128</i> from mask specified in <i>imm8</i> and store the values into <i>xmm1</i> .

PBLENDW — Blend Packed Words

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 3A 0E /r ib PBLENDW <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	SSE4_1	Select words from <i>xmm1</i> and <i>xmm2/m128</i> from mask specified in <i>imm8</i> and store the values into <i>xmm1</i> .

Operation

PBLENDW (128-bit Legacy SSE version)

IF (*imm8*[0] = 1) THEN *DEST*[15:0] ← *SRC*[15:0]

ELSE *DEST*[15:0] ← *DEST*[15:0]

IF (*imm8*[1] = 1) THEN *DEST*[31:16] ← *SRC*[31:16]

ELSE *DEST*[31:16] ← *DEST*[31:16]

IF (*imm8*[2] = 1) THEN *DEST*[47:32] ← *SRC*[47:32]

ELSE *DEST*[47:32] ← *DEST*[47:32]

IF (*imm8*[3] = 1) THEN *DEST*[63:48] ← *SRC*[63:48]

ELSE *DEST*[63:48] ← *DEST*[63:48]

IF (*imm8*[4] = 1) THEN *DEST*[79:64] ← *SRC*[79:64]

ELSE *DEST*[79:64] ← *DEST*[79:64]

IF (*imm8*[5] = 1) THEN *DEST*[95:80] ← *SRC*[95:80]

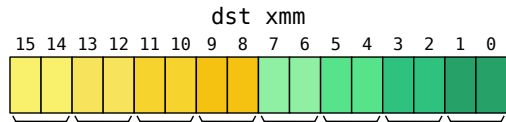
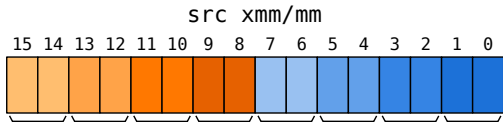
ELSE *DEST*[95:80] ← *DEST*[95:80]

IF (*imm8*[6] = 1) THEN *DEST*[111:96] ← *SRC*[111:96]

ELSE *DEST*[111:96] ← *DEST*[111:96]

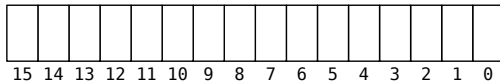
IF (*imm8*[7] = 1) THEN *DEST*[127:112] ← *SRC*[127:112]

Ejemplo-PBLENDW dst, src , imm8

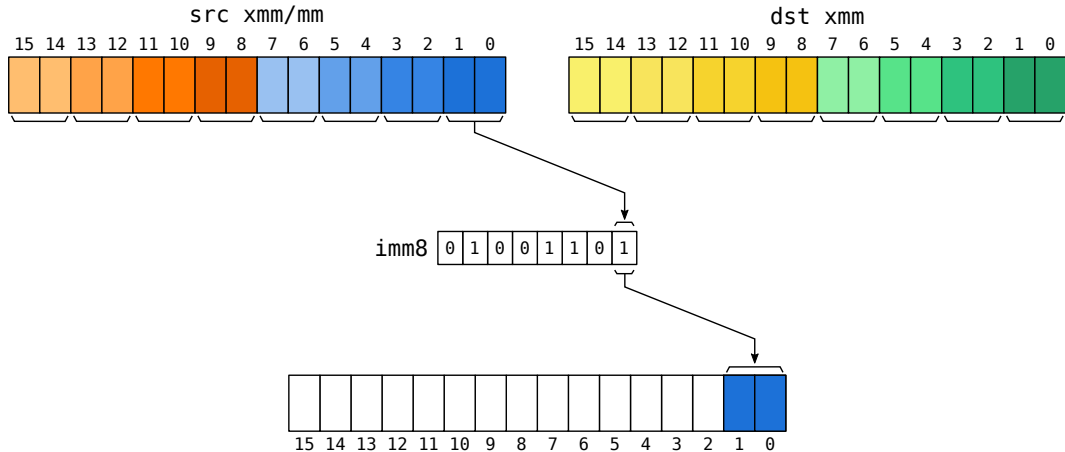


imm8

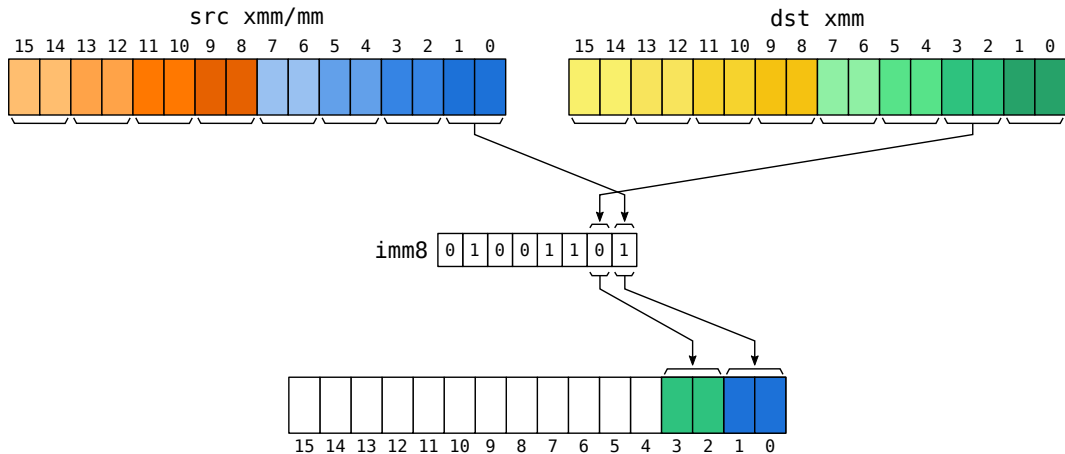
0	1	0	0	1	1	0	1
---	---	---	---	---	---	---	---



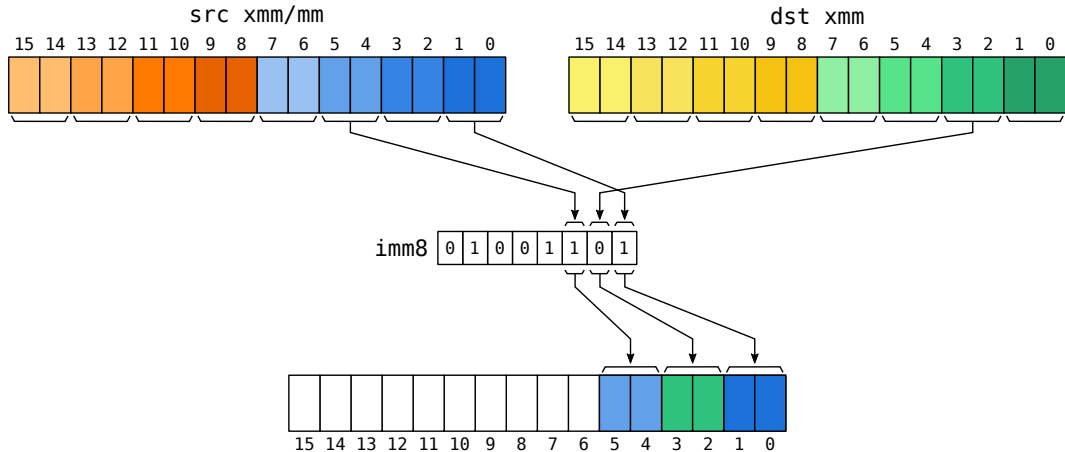
Ejemplo-PBLENDW dst, src , imm8



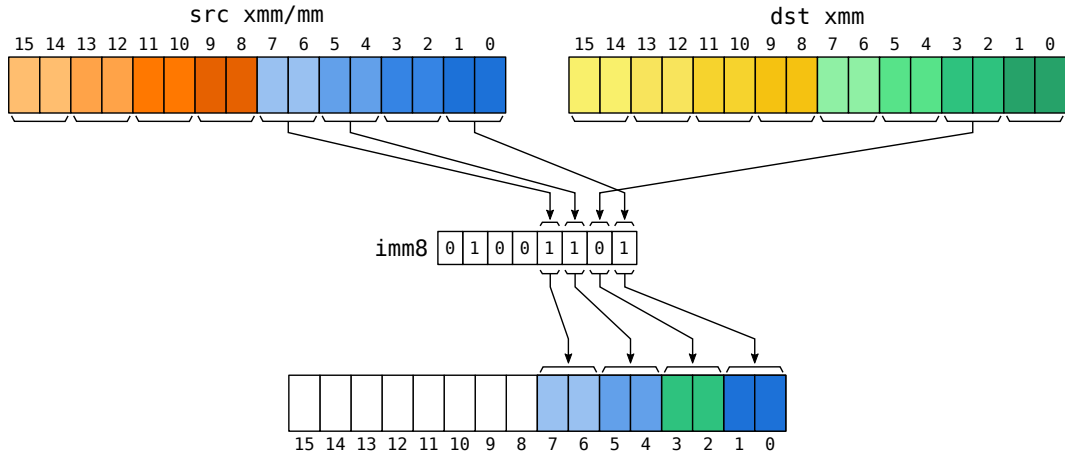
Ejemplo-PBLENDW dst, src , imm8



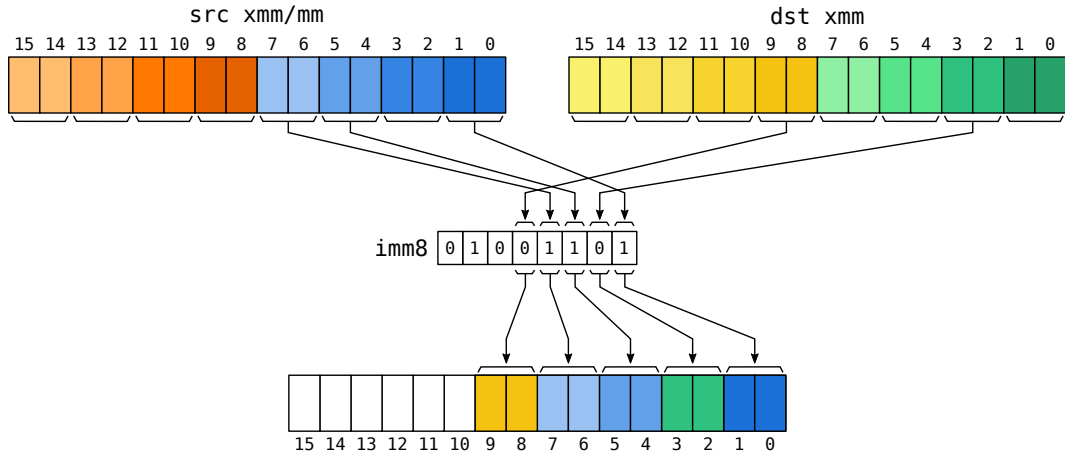
Ejemplo-PBLENDW dst, src , imm8



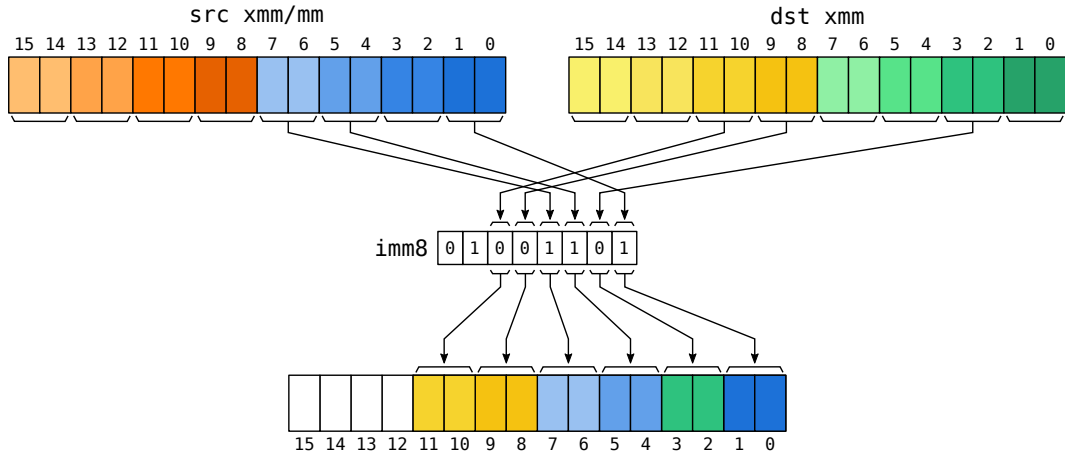
Ejemplo-PBLENDW dst, src , imm8



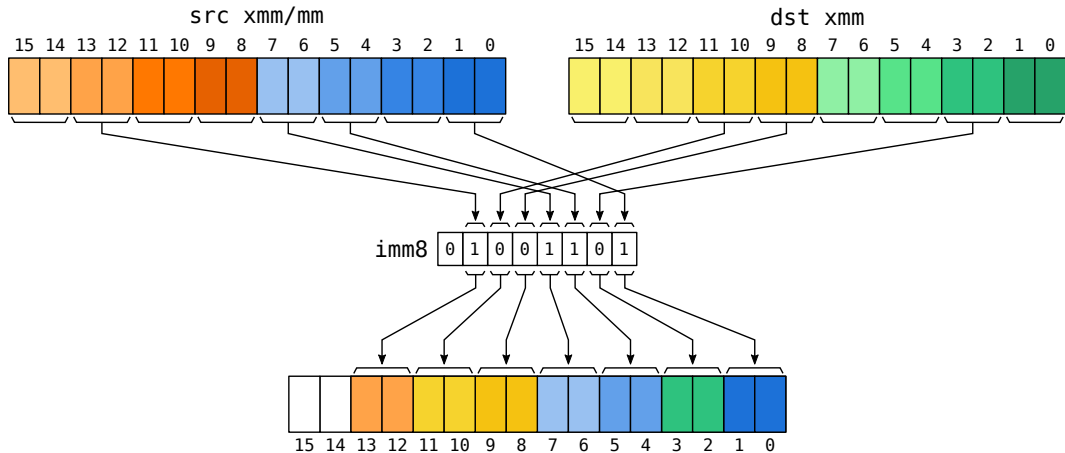
Ejemplo-PBLENDW dst, src , imm8



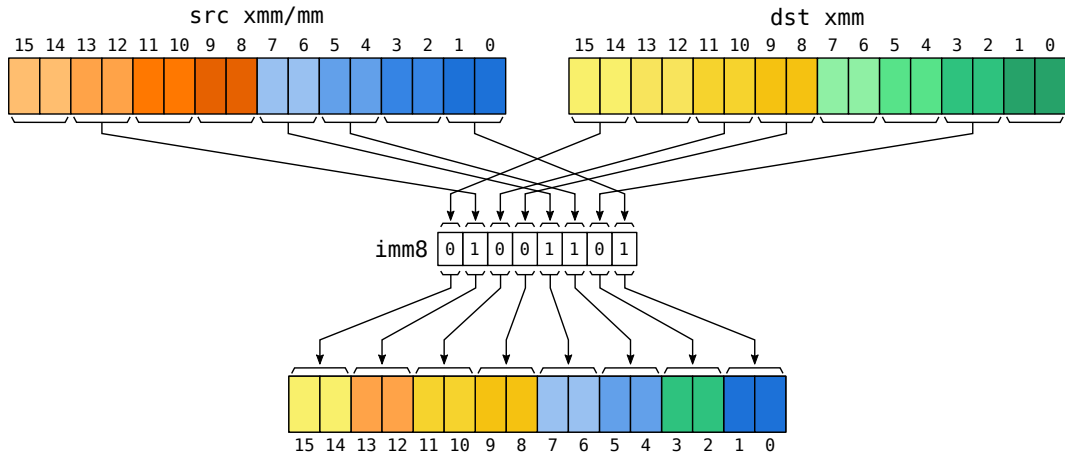
Ejemplo-PBLENDW dst, src , imm8



Ejemplo-PBLENDW dst, src , imm8



Ejemplo-PBLENDW dst, src , imm8



Blend

PBLENDVB — Variable Blend Packed Bytes

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 38 10 /r PBLENDVB <i>xmm1</i> , <i>xmm2/m128</i> , < <i>XMM0</i> >	RM	V/V	SSE4_1	Select byte values from <i>xmm1</i> and <i>xmm2/m128</i> from mask specified in the high bit of each byte in <i>XMM0</i> and store the values into <i>xmm1</i> .

PBLENDVB – Variable Blend Packed Bytes

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 38 10 /r PBLENDVB <i>xmm1</i> , <i>xmm2/m128</i> , < <i>XMM0</i> >	RM	V/V	SSE4_1	Select byte values from <i>xmm1</i> and <i>xmm2/m128</i> from mask specified in the high bit of each byte in <i>XMM0</i> and store the values into <i>xmm1</i> .

Operation

PBLENDVB (128-bit Legacy SSE version)

MASK ← XMM0

IF (MASK[7] = 1) THEN DEST[7:0] ← SRC[7:0];

ELSE DEST[7:0] ← DEST[7:0];

IF (MASK[15] = 1) THEN DEST[15:8] ← SRC[15:8];

ELSE DEST[15:8] ← DEST[15:8];

IF (MASK[23] = 1) THEN DEST[23:16] ← SRC[23:16];

ELSE DEST[23:16] ← DEST[23:16];

IF (MASK[31] = 1) THEN DEST[31:24] ← SRC[31:24];

ELSE DEST[31:24] ← DEST[31:24];

IF (MASK[39] = 1) THEN DEST[39:32] ← SRC[39:32];

ELSE DEST[39:32] ← DEST[39:32];

IF (MASK[47] = 1) THEN DEST[47:40] ← SRC[47:40];

ELSE DEST[47:40] ← DEST[47:40];

IF (MASK[55] = 1) THEN DEST[55:48] ← SRC[55:48];

ELSE DEST[55:48] ← DEST[55:48];

IF (MASK[63] = 1) THEN DEST[63:56] ← SRC[63:56];

ELSE DEST[63:56] ← DEST[63:56];

IF (MASK[71] = 1) THEN DEST[71:64] ← SRC[71:64];

ELSE DEST[71:64] ← DEST[71:64];

IF (MASK[79] = 1) THEN DEST[79:72] ← SRC[79:72];

ELSE DEST[79:72] ← DEST[79:72];

IF (MASK[87] = 1) THEN DEST[87:80] ← SRC[87:80];

ELSE DEST[87:80] ← DEST[87:80];

IF (MASK[95] = 1) THEN DEST[95:88] ← SRC[95:88];

ELSE DEST[95:88] ← DEST[95:88];

IF (MASK[103] = 1) THEN DEST[103:96] ← SRC[103:96];

ELSE DEST[103:96] ← DEST[103:96];

IF (MASK[111] = 1) THEN DEST[111:104] ← SRC[111:104];

ELSE DEST[111:104] ← DEST[111:104];

IF (MASK[119] = 1) THEN DEST[119:112] ← SRC[119:112];

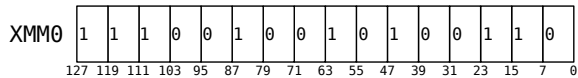
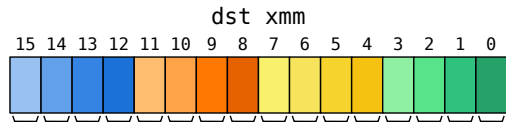
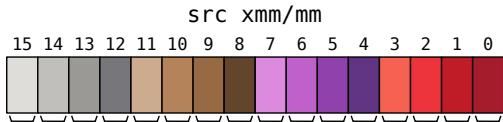
ELSE DEST[119:112] ← DEST[119:112];

IF (MASK[127] = 1) THEN DEST[127:120] ← SRC[127:120];

ELSE DEST[127:120] ← DEST[127:120];

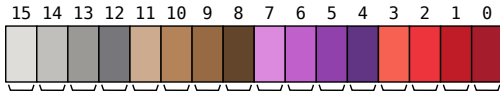
DEST[VLMAX-1:128] (Unmodified)

Ejemplo - PBLENDVB dst, src

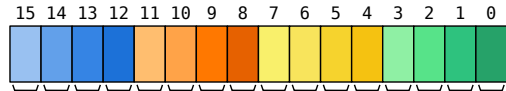


Ejemplo - PBLENDVB dst, src

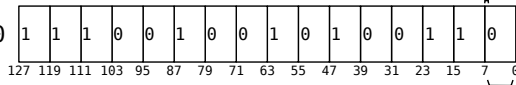
src xmm/mm



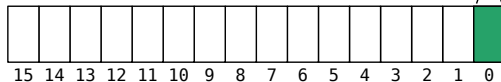
dst xmm



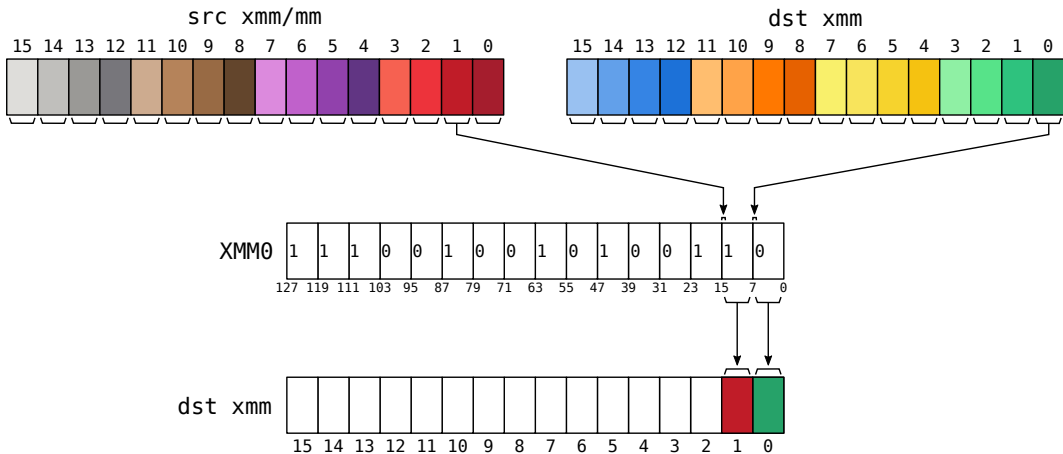
XMM0



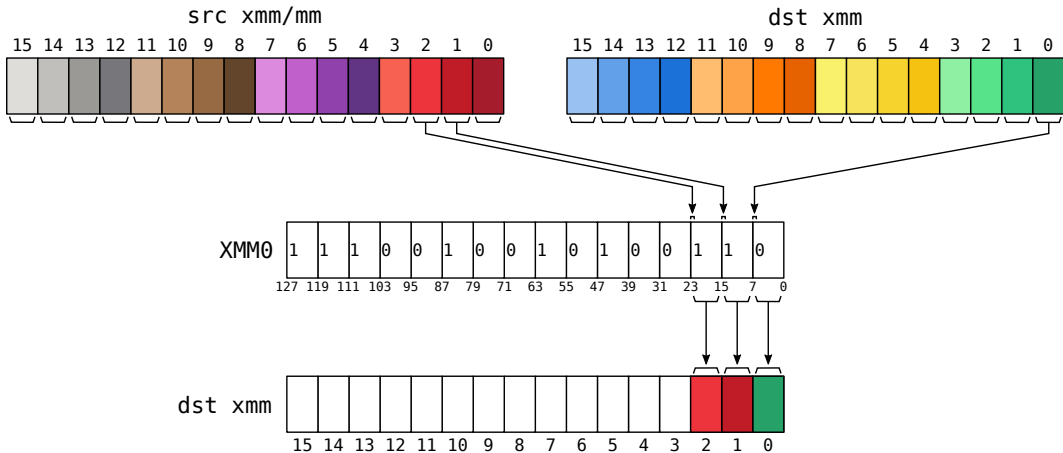
dst xmm



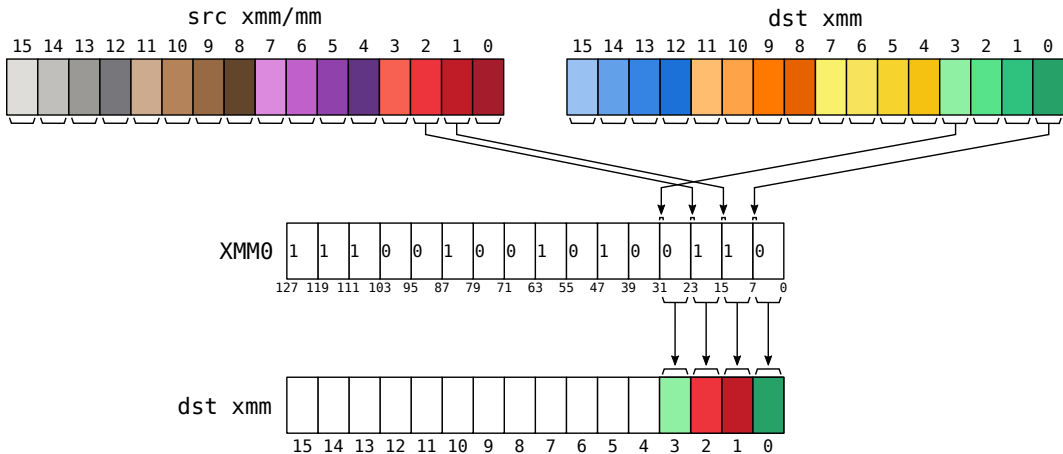
Ejemplo - PBLENDVB dst, src



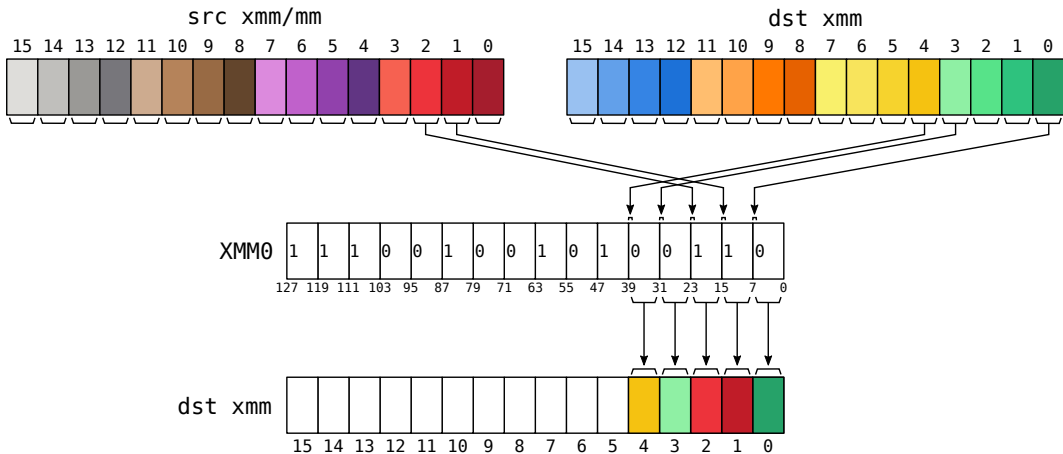
Ejemplo - PBLENDVB dst, src



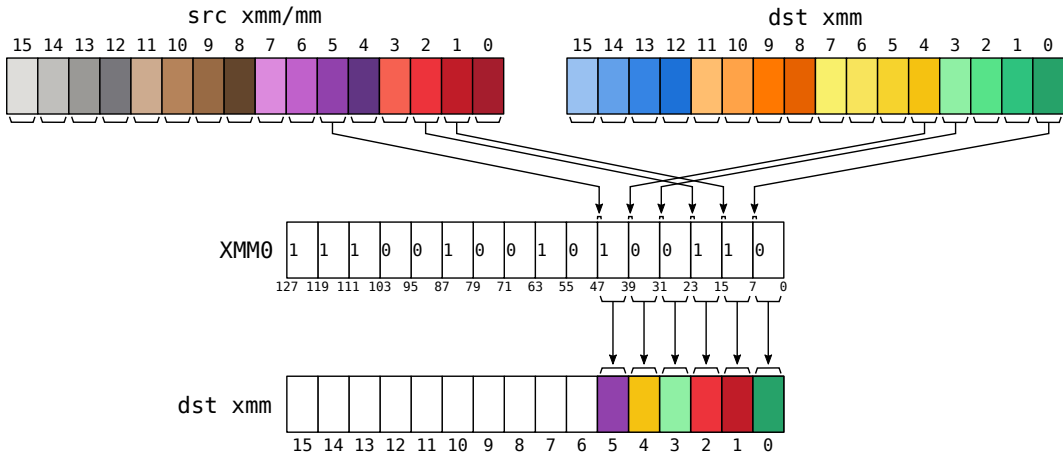
Ejemplo - PBLENDVB dst, src



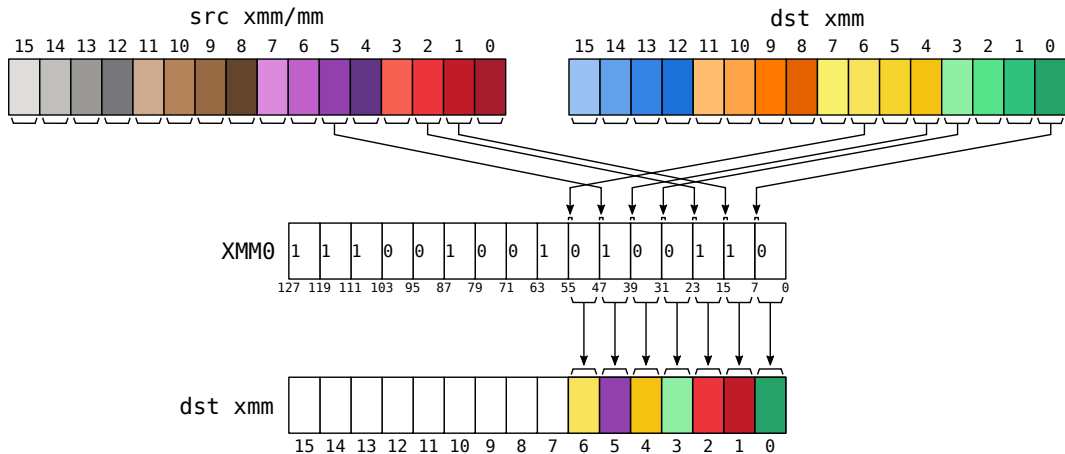
Ejemplo - PBLENDVB dst, src



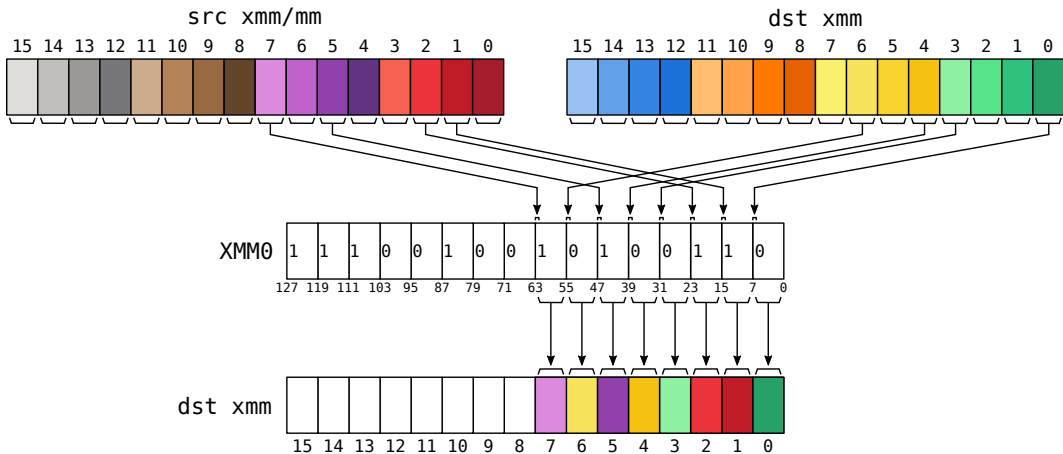
Ejemplo - PBLENDVB dst, src



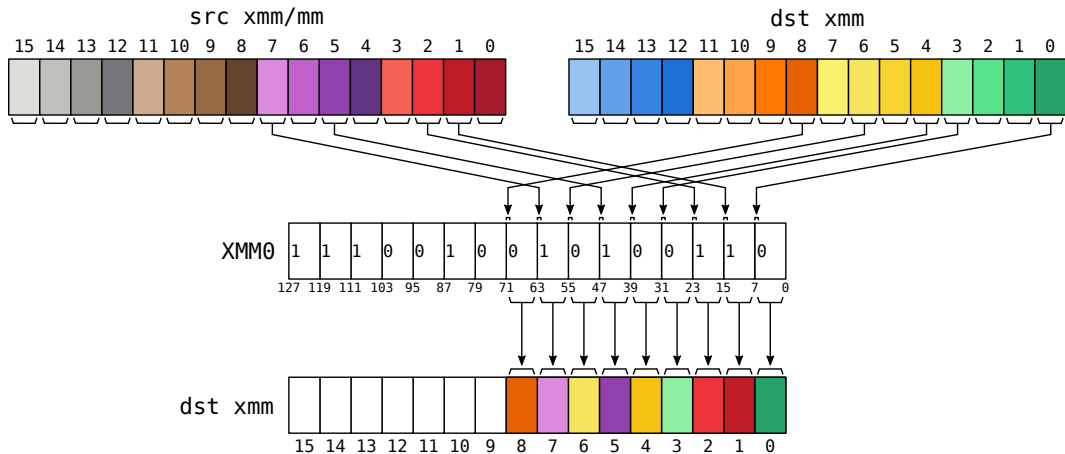
Ejemplo - PBLENDVB dst, src



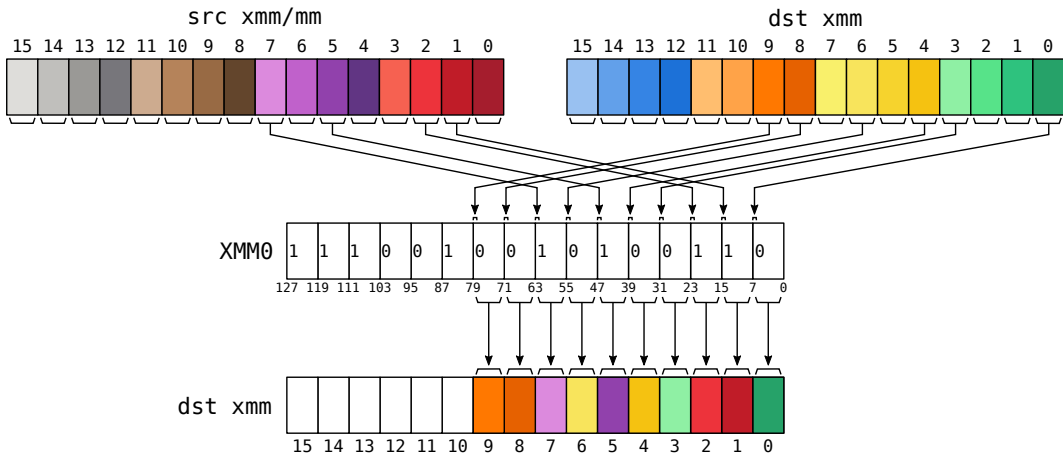
Ejemplo - PBLENDVB dst, src



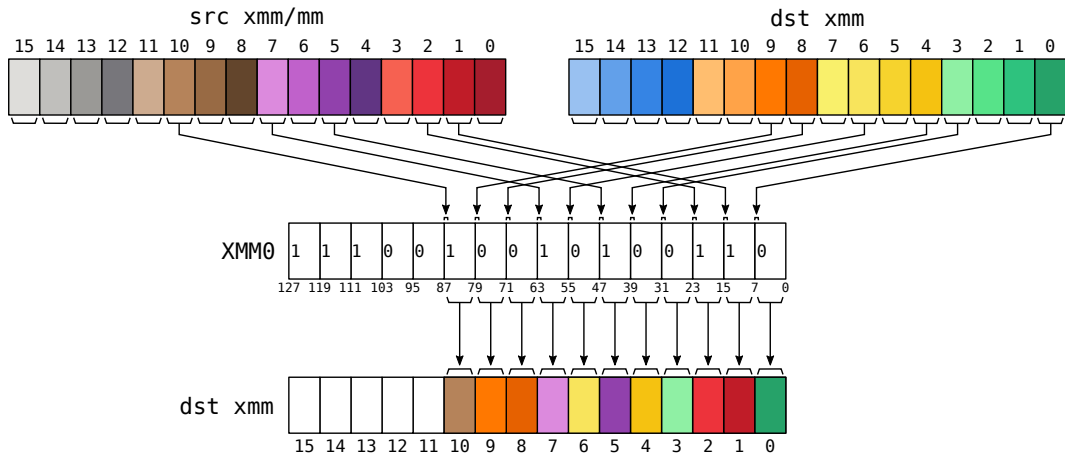
Ejemplo - PBLENDVB dst, src



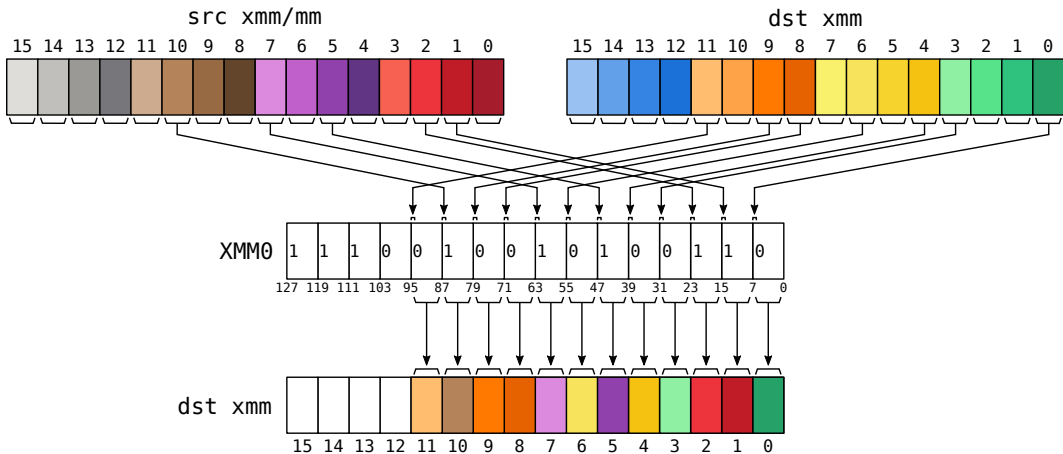
Ejemplo - PBLENDVB dst, src



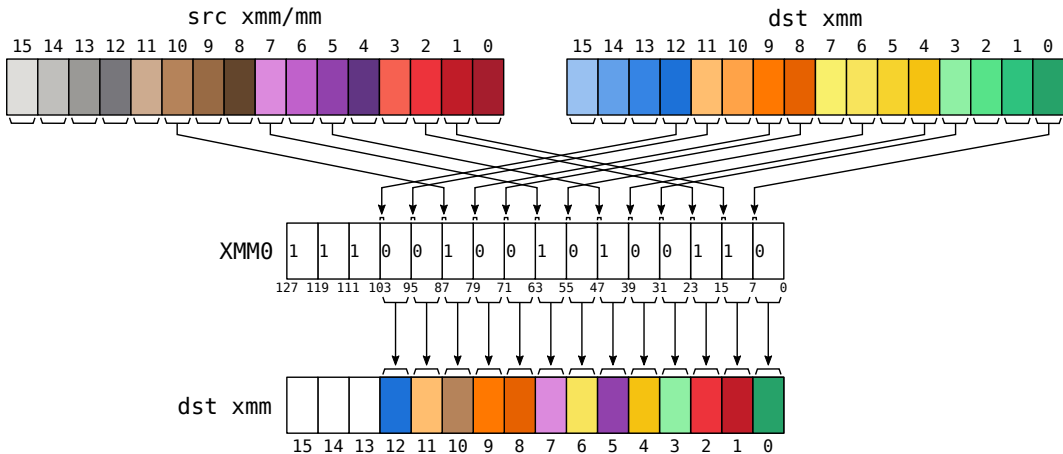
Ejemplo - PBLENDVB dst, src



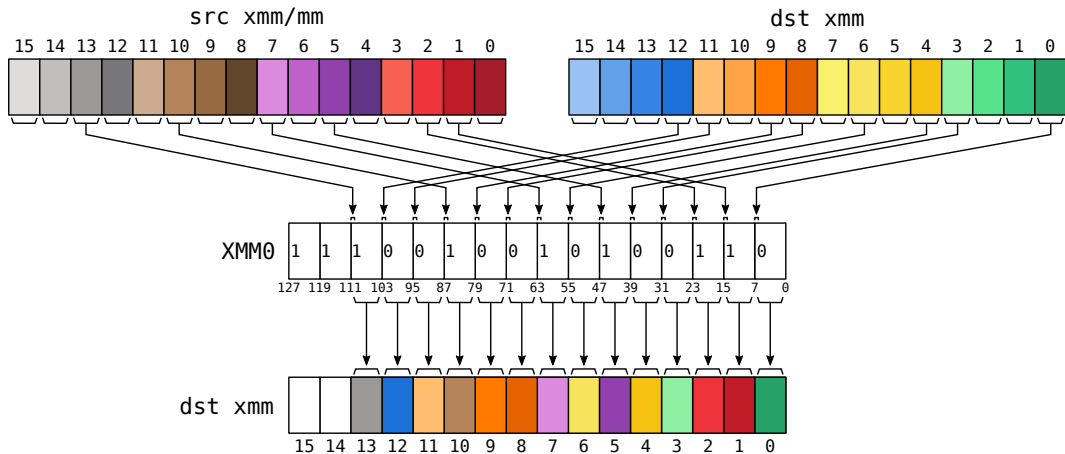
Ejemplo - PBLENDVB dst, src



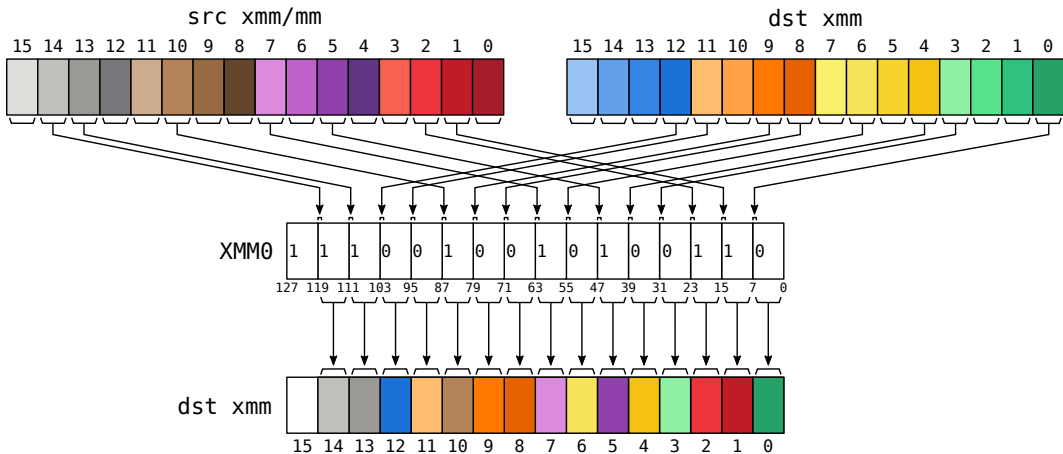
Ejemplo - PBLENDVB dst, src



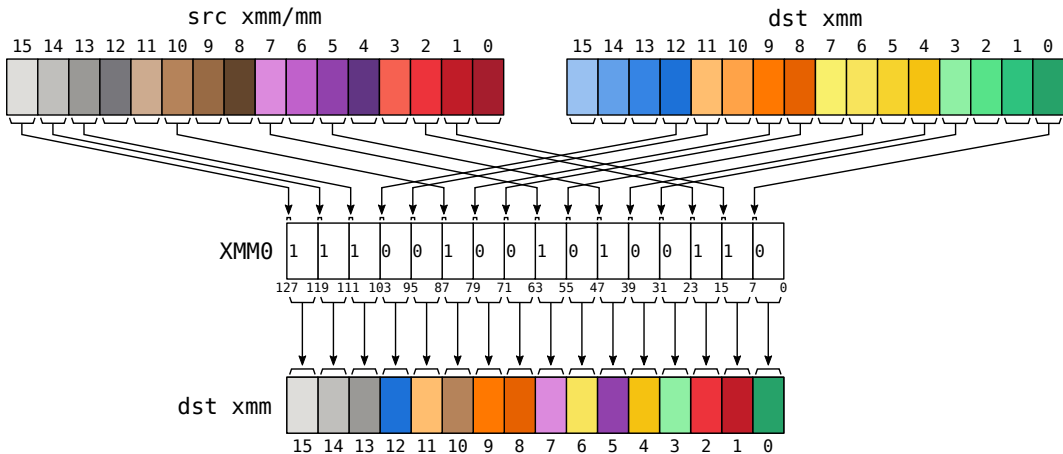
Ejemplo - PBLENDVB dst, src



Ejemplo - PBLENDVB dst, src



Ejemplo - PBLENDVB dst, src



Conversiones

Las instrucciones de conversión son de la forma: `CVTxx2yy`

Donde `xx` e `yy` pueden valer:

ps - Packed Single FP

ss - Scalar Single FP

pd - Packed Double FP

sd - Scalar Double FP

pi - Packed Integer

si - Scalar Integer

dq - Packed Dword

Conversiones

Las instrucciones de conversión son de la forma: **CVTxx2yy**

Donde **xx** e **yy** pueden valer:

ps - Packed Single FP	pd - Packed Double FP	pi - Packed Integer
ss - Scalar Single FP	sd - Scalar Double FP	si - Scalar Integer
		dq - Packed Dword

Instrucciones solo de punto flotante

- **CVTSD2SS** - Scalar Double FP to Scalar Single FP (1X) → **CVTSD2SS** xmm1, xmm2/m64
- **CVTSS2SD** - Scalar Single FP to Scalar Double FP (1X) → **CVTSS2SD** xmm1, xmm2/m32

Conversiones

Las instrucciones de conversión son de la forma: **CVTxx2yy**

Donde **xx** e **yy** pueden valer:

ps - Packed Single FP	pd - Packed Double FP	pi - Packed Integer
ss - Scalar Single FP	sd - Scalar Double FP	si - Scalar Integer
		dq - Packed Dword

Instrucciones solo de punto flotante

- **CVTSD2SS** - Scalar Double FP to Scalar Single FP (1X) → **CVTSD2SS** xmm1, xmm2/m64
- **CVTSS2SD** - Scalar Single FP to Scalar Double FP (1X) → **CVTSS2SD** xmm1, xmm2/m32
- **CVTPD2PS** - Packed Double FP to Packed Single FP (2X) → **CVTPD2PS** xmm1, xmm2/m128
- **CVTPS2PD** - Packed Single FP to Packed Double FP (2X) → **CVTPS2PD** xmm1, xmm2/m64

Conversiones

Instrucciones entre enteros y punto flotante

- **CVT*SI*2*SS*** - Dword Integer to Scalar Single FP → **CVT*SI*2*SS*** xmm, r/m32
- **CVT*SS*2*SI*** - Scalar Single FP to Dword Integer → **CVT*SS*2*SI*** r32, xmm/m32
- **CVT*SI*2*SD*** - Dword Integer to Scalar Double FP → **CVT*SI*2*SD*** xmm, r/m64
- **CVT*SD*2*SI*** - Scalar Double FP to Dword Integer → **CVT*SD*2*SI*** r64, xmm/m64

Conversiones

Instrucciones entre enteros y punto flotante

- **CVTSI2SS** - Dword Integer to Scalar Single FP → **CVTSI2SS** xmm, r/m32
- **CVTSS2SI** - Scalar Single FP to Dword Integer → **CVTSS2SI** r32, xmm/m32
- **CVTSI2SD** - Dword Integer to Scalar Double FP → **CVTSI2SD** xmm, r/m64
- **CVTSD2SI** - Scalar Double FP to Dword Integer → **CVTSD2SI** r64, xmm/m64
- **CVTDQ2PS** - Packed Dword Integers to Packed Single FP (4X) → **CVTDQ2PS** xmm1, xmm2/m128
- **CVTPS2DQ** - Packed Single FP to Packed Dword Integers (4X) → **CVTPS2DQ** xmm1, xmm2/m128
- **CVTDQ2PD** - Packed Dword Integers to Packed Double FP (2X) → **CVTDQ2PD** xmm1, xmm2/m64
- **CVTPD2DQ** - Packed Double FP to Packed Dword Integers (2X) → **CVTPD2DQ** xmm1, xmm2/m128

Conversiones

Instrucciones de redondeo

- **ROUNDSS** - Round Scalar Single FP to Integer → `ROUNDSS xmm1, xmm2/m32, imm8`
- **ROUNDSD** - Round Scalar Double FP to Integer → `ROUNDSD xmm1, xmm2/m64, imm8`
- **ROUNDPS** - Round Packed Single FP to Integer (4X) → `ROUNDPS xmm1, xmm2/m128, imm8`
- **ROUNDPD** - Round Packed Double FP to Integer (2X) → `ROUNDPD xmm1, xmm2/m128, imm8`

El parámetro inmediato indica el tipo de redondeo.

Conversiones

Instrucciones de redondeo

- **ROUNDSS** - Round Scalar Single FP to Integer → **ROUNDSS** xmm1, xmm2/m32, imm8
- **ROUNDSD** - Round Scalar Double FP to Integer → **ROUNDSD** xmm1, xmm2/m64, imm8
- **ROUNDPS** - Round Packed Single FP to Integer (4X) → **ROUNDPS** xmm1, xmm2/m128, imm8
- **ROUNDPD** - Round Packed Double FP to Integer (2X) → **ROUNDPD** xmm1, xmm2/m128, imm8

El parámetro inmediato indica el tipo de redondeo.

Instrucciones de truncado

- **CVTTSS2SI** - Truncation Scalar Single FP to Dword Integer (1X) → **CVTTSS2SI** r32, xmm/m32
- **CVTTSD2SI** - Truncation Scalar Double FP to Signed Integer (1X) → **CVTTSD2SI** r32, xmm/m64
- **CVTTPS2DQ** - Truncation Packed Single FP to Packed Dword Int. (4X) → **CVTTPS2DQ** xmm1, xmm2/m128

Ejercicios

- 1 Sea un vector a de n valores punto flotante de 32 bits.
Realizar la siguiente operación: $\sqrt{a[i * 2] \cdot 0,7 + a[i * 2 + 1] \cdot 0,3} \cdot 255$
Donde i itera entre 0 y $n/2$. Almacenar el resultado sobre el mismo vector en double y considerar que $n \equiv 0 \pmod{4}$.
- 2 Sea un vector que contiene exactamente 10 valores enteros sin signo de 3 bytes cada uno. Realizar la sumatoria de los mismos y almacenar el resultado en un double.
- 3 Sea un vector de n valores enteros de 32 bits almacenados en big-endian. Convertir cada uno de los valores a double, considerar que $n \equiv 0 \pmod{4}$. Almacenarlos en un nuevo vector de forma que primero se guarden los valores de índice par y luego los de índice impar. Es decir: $XYXY \dots XYXY \rightarrow XXXX \dots YYYY$

¡Gracias!

Recuerden leer los comentarios al final de este video por aclaraciones o fe de erratas.