

# SIMD Shuffle y Conversiones

## Organización del Computador II

Gonzalo Ciruelos

**Diapositivas por David Alejandro González Márquez**

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

20-04-2017

- Problemas de Precisión
- Instrucciones de Shuffle
- Instrucciones de Conversión
- Ejercicios

# Problemas de Precisión

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

**Por ejemplo:** El número 1,1 no es posible de 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  $0x87,8$

## 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**
- Usar operaciones de enteros sobre punto flotante o a la inversa, implica una **penalidad** en tiempo
- Las operaciones en punto flotante son **más costosas** que las de enteros.

Las instrucciones de *Shuffles* permiten **reordenar** datos en registros. Sus parámetros serán el **registro a reordenar** y una **máscara** que indicará cómo hacerlo.

- PSHUFB - Shuffle Packed Bytes
- PSHUFW - Shuffle Packed Words
- PSHUFD - Shuffle Packed Doublewords
  
- PSHUFHW - Shuffles high 16bit values
- PSHUFLW - Shuffles low 16bit values
  
- SHUFPS - Shuffle Packed Single FP Values
- SHUFPD - Shuffle Packed Double FP Values

Las instrucciones de *Shuffles* permiten **reordenar** datos en registros. Sus parámetros serán el **registro a reordenar** y una **máscara** que indicará cómo hacerlo.

- PSHUFB - Shuffle Packed Bytes
- PSHUFW - Shuffle Packed Words
- PSHUFD - Shuffle Packed Doublewords
  
- PSHUFW - Shuffles high 16bit values
- PSUFLW - Shuffles low 16bit values
  
- SHUFPS - Shuffle Packed Single FP Values
- SHUFPD - Shuffle Packed Double FP Values

## PSHUFB — Packed Shuffle Bytes

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F 38 00 /r <sup>1</sup> 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 ( $\text{SRC}[(i * 8) + 7] = 1$ ) then

$\text{DEST}[(i * 8) + 7 .. (i * 8) + 0] \leftarrow 0;$

else

$\text{index}[3..0] \leftarrow \text{SRC}[(i * 8) + 3 .. (i * 8) + 0];$

$\text{DEST}[(i * 8) + 7 .. (i * 8) + 0] \leftarrow \text{DEST}[(\text{index} * 8) + 7 .. (\text{index} * 8) + 0];$

endif

}

$\text{DEST}[\text{VLMAX} - 1 : 128] \leftarrow 0$

## PSHUFW—Shuffle Packed Words

Opcode/ Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
OF 70 /r ib PSHUFW <i>mm1</i> , <i>mm2/m64</i> , <i>imm8</i>	RMI	Valid	Valid	Shuffle the words in <i>mm2/m64</i> based on the encoding in <i>imm8</i> and store the result in <i>mm1</i> .

```
DEST[15:0] ← (SRC >> (ORDER[1:0] * 16))[15:0];  
DEST[31:16] ← (SRC >> (ORDER[3:2] * 16))[15:0];  
DEST[47:32] ← (SRC >> (ORDER[5:4] * 16))[15:0];  
DEST[63:48] ← (SRC >> (ORDER[7:6] * 16))[15:0];
```



## 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)

$\text{DEST}[31:0] \leftarrow (\text{SRC} \gg (\text{ORDER}[1:0] * 32))[31:0];$   
 $\text{DEST}[63:32] \leftarrow (\text{SRC} \gg (\text{ORDER}[3:2] * 32))[31:0];$   
 $\text{DEST}[95:64] \leftarrow (\text{SRC} \gg (\text{ORDER}[5:4] * 32))[31:0];$   
 $\text{DEST}[127:96] \leftarrow (\text{SRC} \gg (\text{ORDER}[7:6] * 32))[31:0];$   
 $\text{DEST}[\text{VLMAX}-1:128]$  (Unmodified)

## 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)

DEST[15:0]  $\leftarrow$  (SRC >> (imm[1:0] \* 16))[15:0]

DEST[31:16]  $\leftarrow$  (SRC >> (imm[3:2] \* 16))[15:0]

DEST[47:32]  $\leftarrow$  (SRC >> (imm[5:4] \* 16))[15:0]

DEST[63:48]  $\leftarrow$  (SRC >> (imm[7:6] \* 16))[15:0]

DEST[127:64]  $\leftarrow$  SRC[127:64]

DEST[VLMAX-1:128] (Unmodified)

## 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> .

## PSHUFHW (128-bit Legacy SSE version)

DEST[63:0] ← SRC[63:0]

DEST[79:64] ← (SRC >> (imm[1:0] \* 16))[79:64]

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

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

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

DEST[VLMAX-1:128] (Unmodified)

## 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>xmm2/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> .

### SHUFPS (128-bit Legacy SSE version)

$\text{DEST}[31:0] \leftarrow \text{Select4}(\text{SRC1}[127:0], \text{imm8}[1:0]);$   
 $\text{DEST}[63:32] \leftarrow \text{Select4}(\text{SRC1}[127:0], \text{imm8}[3:2]);$   
 $\text{DEST}[95:64] \leftarrow \text{Select4}(\text{SRC1}[127:0], \text{imm8}[5:4]);$   
 $\text{DEST}[127:96] \leftarrow \text{Select4}(\text{SRC1}[127:0], \text{imm8}[7:6]);$   
 $\text{DEST}[\text{VLMAX}-1:128]$  (Unmodified)

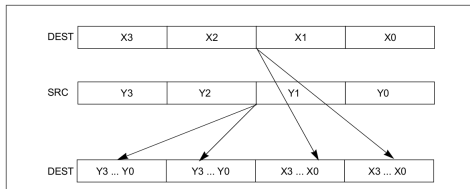


Figure 4-22. SHUFPS Shuffle Operation

## 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 IMMO[0] = 0
    THEN DEST[63:0] ← SRC1 DEST[63:0]
    ELSE DEST[63:0] ← SRC1 [127:64] F;
IF IMMO[1] = 0
    THEN DEST[127:64] ← SRC2[63:0]
    ELSE DEST[127:64] ← SRC2[127:64] F;
DEST[VLMAX-1:128] (Unmodified)
    
```

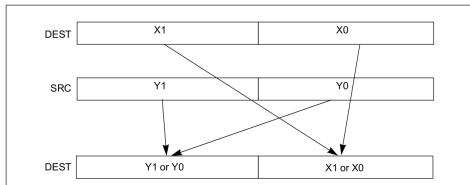


Figure 4-21. SHUFPD Shuffle Operation

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

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

## 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, xmm2/m32, 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, xmm2, xmm3/m32, 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)
```



## EXTRACTPS — Extract Packed Single Precision Floating-Point Value

Opcode/ Instruction	Op/ En	64/32-bit Mode	CPUID Feature Flag	Description
66 0F 3A 17 <i>/r ib</i> 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</i> or <i>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 <i>/r ib</i> 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</i> or <i>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

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

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 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 0F 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 0F 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 <sup>1</sup> /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	CASE OF PINSRB: SEL ← COUNT[3:0]; MASK ← (0FFH << (SEL * 8)); TEMP ← (((SRC[7:0] << (SEL * 8)) AND MASK); PINSRD: SEL ← COUNT[1:0]; MASK ← (0FFFFFFFH << (SEL * 32)); TEMP ← (((SRC << (SEL * 32)) AND MASK) ; PINSRQ: SEL ← COUNT[0] MASK ← (0FFFFFFFFFFFFFFFFFH << (SEL * 64)); TEMP ← (((SRC << (SEL * 32)) AND MASK) ; ESAC; DEST ← ((DEST AND NOT MASK) OR TEMP);	
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		

## PINSRW—Insert Word

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F C4 /r ib <sup>1</sup> 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> .
VE X.NDS.128.66.0F.W0 C4 /r ib VPINSRW <i>xmm1</i> , <i>xmm2</i> , <i>r32/m16</i> , <i>imm8</i>	RVMI	V <sup>2</sup> /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 ← 00000000000000000000000000FFFF0000H;  
 SEL ← 2: MASK ← 000000000000000000000000FFFF00000000H;  
 SEL ← 3: MASK ← 000000000000000000000000FFFF0000000000H;  
 SEL ← 4: MASK ← 000000000000000000000000FFFF00000000000H;  
 SEL ← 5: MASK ← 00000000FFFF00000000000000000000000H;  
 SEL ← 6: MASK ← 0000FFFF000000000000000000000000000H;  
 SEL ← 7: MASK ← FFFF0000000000000000000000000000000H;

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

## 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 <i>r32</i> or <i>r64</i> 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 <sup>1</sup> /V	AVX	CASE of PEXTRB: $SEL \leftarrow COUNT[3:0]$ ; $TEMP \leftarrow (Src \gg SEL * 8) \text{ AND } FFFH$ ; IF (DEST = Mem8) THEN $Mem8 \leftarrow TEMP[7:0]$ ; ELSE IF (64-Bit Mode and 64-bit register selected) THEN $R64[7:0] \leftarrow TEMP[7:0]$ ; $r64[63:8] \leftarrow ZERO\_FILL$ ; ; ELSE $R32[7:0] \leftarrow TEMP[7:0]$ ; $r32[31:8] \leftarrow ZERO\_FILL$ ; ; FI; PEXTRD: $SEL \leftarrow COUNT[1:0]$ ; $TEMP \leftarrow (Src \gg SEL * 32) \text{ AND } FFFF\_FFFFH$ ; DEST $\leftarrow TEMP$ ; PEXTRQ: $SEL \leftarrow COUNT[0]$ ; $TEMP \leftarrow (Src \gg SEL * 64)$ ; DEST $\leftarrow TEMP$ ;
VEX.128.66.0F3A.W0 16 /r ib VPEXTRD <i>r32/m32, xmm2, imm8</i>	MRI	V/V	AVX	
VEX.128.66.0F3A.W1 16 /r ib VPEXTRQ <i>r64/m64, xmm2, imm8</i>	MRI	V/i	AVX	

EAS:

# Insert/Extract

## PEXTRW—Extract Word

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
0F C5 /r ib <sup>1</sup> 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.

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;

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

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

## 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</i> , <i>xmm2/m128</i> , <i>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</i> , <i>xmm2/m128</i> , <i>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)

```



Las instrucciones de conversión son de la forma: `cvt $xx$ 2 $yy$`

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

- CVTSD2SI - Scalar Double FP to Dword Integer  
`CVTSD2SI r32, xmm/m64 | CVTSD2SI r64, xmm/m64`
- CVTSI2SD - Dword Integer to Scalar Double FP  
`CVTSI2SD xmm, r/m32 | CVTSI2SD xmm, r/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`

## Otras instrucciones de conversión (1):

- 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`
- 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`
- CVTTPS2DQ - Truncation Packed Single FP to Packed Dword Int. (4X)  
`CVTTPS2DQ xmm1, xmm2/m128`
- 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`

## Otras instrucciones de conversión (2):

- CVTSS2SI - Scalar Single FP to Dword Integer  
`CVTSS2SI r32, xmm/m32`
- CVTSD2SI - Scalar Double FP to Dword Integer  
`CVTSD2SI r32, xmm/m64`
- CVTPI2PS - Packed Dword Integers to Packed Single FP (2X)  
`CVTPI2PS xmm, mm/m64`
- CVTPI2PD - Packed Dword Integers to Packed Double FP (2X)  
`CVTPI2PD xmm, mm/m64`
- ROUNDPS - Round Packed Single FP to Integer (4X)  
`ROUNDPS xmm1, xmm2/m128, imm8`
- ROUNDDSS - Round Scalar Single FP to Integer  
`ROUNDDSS xmm1, xmm2/m32, imm8`
- ROUNDPS - Round Packed Double FP to Integer (2X)  
`ROUNDPS xmm1, xmm2/m128, imm8`
- ROUNDDSD - Round Scalar Double FP to Integer  
`ROUNDDSD xmm1, xmm2/m64, imm8`

- 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  $n \equiv 0 \pmod{4}$ .

# Ejercicio 1 - Solución

```
section .text
ej1: ; rdi = a, esi = n;
    push rbp
    mov rbp, rsp
    mov ecx, esi
    shr ecx, 2
    movdqu xmm8, [val0703]
    movdqu xmm9, [val255]
    .ciclo:
        movdqu xmm0, [rdi] ; xmm0 = | fp3 | fp2 | fp1 | fp0 |
        cvtPS2PD xmm1, xmm0 ; xmm1 = |    fp1    |    fp0    |
        psrldq xmm0, 8      ; xmm0 = | 0 | 0 | fp3 | fp2 |
        cvtPS2PD xmm2, xmm0 ; xmm2 = |    fp3    |    fp2    |
        mulpd xmm1, xmm8    ; xmm1 = | 0.3 * fp1 | 0.7 * fp0 |
        mulpd xmm2, xmm8    ; xmm2 = | 0.3 * fp3 | 0.7 * fp2 |

        movdqu xmm3, xmm1 ; xmm3 = | 0.3 * fp1 | 0.7 * fp0 |
        shufpd xmm3, xmm2, 1; xmm3 = | 0.7 * fp2 | 0.3 * fp1 |
        shufpd xmm1, xmm2, 2; xmm1 = | 0.3 * fp3 | 0.7 * fp0 |
        addpd xmm1, xmm3    ; xmm1 = | 0.7*fp2+3.0*fp3 | 0.7*fp0+0.3*fp1 |

        sqrtpd xmm1, xmm1 ; xmm1 = | sqrt(fp3_fp2) | sqrt(fp1_fp0) |
        mulpd xmm1, xmm9 ; xmm1 = | 255*sqrt(fp3_fp2) | 255*sqrt(fp1_fp0) |
        movdqu [rdi], xmm1
        add rdi, 16
    loop .ciclo
    pop rbp
    ret

section .rodata
val0703: dq 0.7, 0.3
val255:  dq 255.0, 255.0
```

# Ejercicio 1 - Solución (alternativa)

```
section .text
ej1: ; rdi = a, esi = n;
    push rbp
    mov rbp, rsp
    mov ecx, esi
    shr ecx, 2
    movdqu xmm8, [val0703]
    movdqu xmm9, [val255]
.ciclo:
    movdqu xmm0, [rdi] ; xmm0 = | fp3 | fp2 | fp1 | fp0 |
    cvtPS2PD xmm1, xmm0 ; xmm1 = |    fp1    |    fp0    |
    psrldq xmm0, 8      ; xmm0 = | 0 | 0 | fp3 | fp2 |
    cvtPS2PD xmm2, xmm0 ; xmm2 = |    fp3    |    fp2    |
    mulpd xmm1, xmm8    ; xmm1 = | 0.3 * fp1 | 0.7 * fp0 |
    mulpd xmm2, xmm8    ; xmm2 = | 0.3 * fp3 | 0.7 * fp2 |

    haddpd xmm1, xmm2   ; xmm1 = | 0.3*fp3+0.7*fp2 | 0.3*fp1+0.7*fp0 |

    sqrtpd xmm1, xmm1   ; xmm1 = | sqrt(fp3_fp2) | sqrt(fp1_fp0) |
    mulpd xmm1, xmm9    ; xmm1 = | 255*sqrt(fp3_fp2) | 255*sqrt(fp1_fp0) |
    movdqu [rdi], xmm1
    add rdi, 16
    loop .ciclo
    pop rbp
    ret
```

- 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  $a$  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.

## Ejercicio 2 - Solución

```
section .rodata
transform5a4: db 0x00,0x01,0x02,0xFF,0x03,0x04,0x05,0xFF,0x06,0x07,0x08,0xFF,0x09,0x0A,0x0B,0xFF
transform5a1: db 0x0C,0x0D,0x0E,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF

section .text
ej3: ; rdi = a, esi = n;
    push rbp
    mov rbp, rsp
    movdqu xmm8, [transform5a4]
    movdqu xmm9, [transform5a1]
    movdqu xmm0, [rdi]           ; xmm0 = | _ | xxx | xxx | xxx | xxx | xxx |
    movdqu xmm1, xmm0           ; xmm1 = | _ | xxx | xxx | xxx | xxx | xxx |
    movdqu xmm2, [rdi+10*3-16]   ; xmm2 = | yyy | yyy | yyy | yyy | yyy | _ |
    psrldq xmm2, 1               ; xmm2 = | _ | yyy | yyy | yyy | yyy | yyy |
    movdqu xmm3, xmm2           ; xmm2 = | _ | yyy | yyy | yyy | yyy | yyy |
    pshufb xmm0, xmm8            ; xmm0 = | 0xxx | 0xxx | 0xxx | 0xxx |
    pshufb xmm1, xmm9            ; xmm1 = | 0000 | 0000 | 0000 | 0xxx |
    pshufb xmm2, xmm8            ; xmm2 = | 0yyy | 0yyy | 0yyy | 0yyy |
    pshufb xmm3, xmm9            ; xmm3 = | 0000 | 0000 | 0000 | 0yyy |
    paddb xmm0, xmm1            ; xmm0 = | 0xxx | 0xxx | 0xxx | 0xxx+0xxx |
    paddb xmm2, xmm3            ; xmm2 = | 0yyy | 0yyy | 0yyy | 0yyy+0yyy |

    paddb xmm0, xmm2            ; xmm0 = | S3 | S2 | S1 | S0 |
    pshufd xmm1, xmm0, 1110b     ; xmm1 = | ---- | ---- | S3 | S2 |
    paddb xmm0, xmm1            ; xmm0 = | ---- | ---- | S1+S3 | S0+S2 |
    pshufd xmm1, xmm0, 0001b     ; xmm1 = | ---- | ---- | ---- | S1+S3 |
    paddb xmm0, xmm1            ; xmm0 = | ---- | ---- | ---- | S0+S1+S2+S3 |

    cvtdq2pd xmm0, xmm0         ; xmm0 = | _ | double(S0+S1+S2+S3) |
    pop rbp
    ret
```



# Ejercicio 2 - Solución (alternativa)

```
section .rodata
transform5a4: db 0x00,0x01,0x02,0xFF,0x03,0x04,0x05,0xFF,0x06,0x07,0x08,0xFF,0x09,0x0A,0x0B,0xFF
transform5a1: db 0x0C,0x0D,0x0E,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF

section .text
ej3: ; rdi = a, esi = n;
    push rbp
    mov rbp, rsp
    movdqu xmm8, [transform5a4]
    movdqu xmm9, [transform5a1]
    movdqu xmm0, [rdi]           ; xmm0 = |_|xxx|xxx|xxx|xxx|xxx|
    movdqu xmm1, xmm0           ; xmm1 = |_|xxx|xxx|xxx|xxx|xxx|
    movdqu xmm2, [rdi+10*3-16]   ; xmm2 = |yyy|yyy|yyy|yyy|yyy|_|
    psrldq xmm2, 1              ; xmm2 = |_|yyy|yyy|yyy|yyy|yyy|
    movdqu xmm3, xmm2           ; xmm2 = |_|yyy|yyy|yyy|yyy|yyy|
    pshufb xmm0, xmm8           ; xmm0 = |0xxx|0xxx|0xxx|0xxx|
    pshufb xmm1, xmm9           ; xmm1 = |0000|0000|0000|0xxx|
    pshufb xmm2, xmm8           ; xmm2 = |0yyy|0yyy|0yyy|0yyy|
    pshufb xmm3, xmm9           ; xmm3 = |0000|0000|0000|0yyy|
    paddb xmm0, xmm1            ; xmm0 = |0xxx|0xxx|0xxx|0xxx+0xxx|
    paddb xmm2, xmm3            ; xmm2 = |0yyy|0yyy|0yyy|0yyy+0yyy|

    phaddb xmm0, xmm2           ; xmm0 = |0xxx+0xxx|0xxx+0xxx+0xxx|0yyy+0yyy|0yyy+0yyy+0yyy|
    phaddb xmm0, xmm0           ; xmm0 = |_||_||0xxx+0xxx+0xxx+0xxx+0xxx|0yyy+0yyy+0yyy+0yyy+0yyy|
    phaddb xmm0, xmm0           ; xmm0 = |_||_||_||0xxx+0xxx+0xxx+0xxx+0xxx+0xxx+0yyy+0yyy+0yyy+0yyy+0yyy|

    cvtdq2pd xmm0, xmm0        ; xmm0 = |_||double(0xxx+0xxx+0xxx+0xxx+0xxx+0yyy+0yyy+0yyy+0yyy+0yyy)|
    pop rbp
    ret
```

¿Preguntas?