

Modelo de Ejecución SIMD en IA-32

Alejandro Furfaro

7 de mayo de 2020

Agenda



- 1 Fundamentos
 - Sobre Fourier, Lagrange, Laplace y los grupos de Whatsapp
- 2 Procesamiento de Señales digitales
 - Digitalización de la señal
 - Arquitecturas de Procesamiento de una señal digital
- 3 Modelo de ejecución SIMD
 - Un modelo de paralelización
- 4 Implementaciones SIMD en x86
 - Arquitectura completa
 - Números Reales
 - Formatos de Punto Fijo
 - Formatos de Punto Flotante
 - Codificación de Números Reales
 - Extensiones AVX
- 5 Instrucciones
 - Transferencias (las mas comunes)
 - Aritmética en algoritmos DSP
 - Instrucciones de punto flotante
 - Instrucciones para manejo de enteros para SSEn
 - Instrucciones para manejo de

1 Fundamentos

- Sobre Fourier, Lagrange, Laplace y los grupos de Whatsapp

2 Procesamiento de Señales digitales

3 Modelo de ejecución SIMD

4 Implementaciones SIMD en x86

5 Instrucciones

Epochas en las que no habia Whatsapp



Jean-Baptiste Joseph Fourier, Joseph-Louis Lagrange. y Pierre-Simon Laplace...

Serie de Fourier

- Partió de demostrar que las funciones:

$$\sin(n \cdot x), \cos(n \cdot x), \forall n \in \text{entero} \quad (1)$$

Son funciones ortogonales, es decir, permiten calcular cualquier otra función mediante cálculos del tipo:

Su demostración es sumamente compleja y no es el objetivo de este curso.

Serie de Fourier

- Partió de demostrar que las funciones:

$$\sin(n \cdot x), \cos(n \cdot x), \forall n \in \text{entero} \quad (1)$$

Son funciones ortogonales, es decir, permiten calcular cualquier otra función mediante cálculos del tipo:

$$f(t) \sim \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2n\pi}{T}t\right) + b_n \sin\left(\frac{2n\pi}{T}t\right) \right]$$

$$a_0 = \frac{2}{T} \int_{-T/2}^{T/2} f(t) dt, \quad a_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \cos\left(\frac{2n\pi}{T}t\right) dt, \quad b_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \sin\left(\frac{2n\pi}{T}t\right) dt.$$

Su demostración es sumamente compleja y no es el objetivo de este curso.

Señal rectangular

La función rectangular es una función discontinua definida mediante la siguiente expresión.

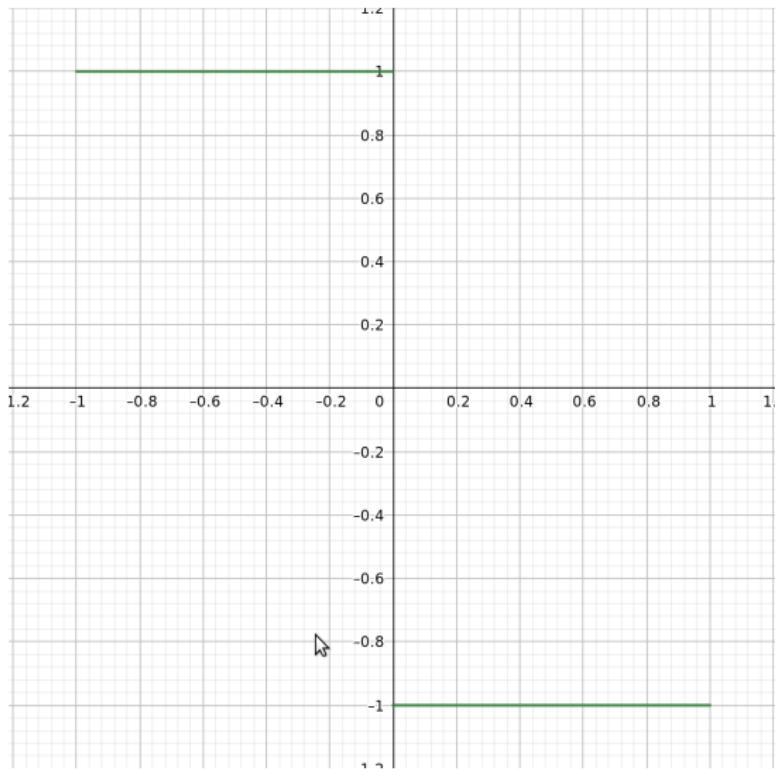
$$f(t) = \begin{cases} -1 & -1 \leq t \leq 0 \\ 1 & 0 \leq t \leq 1 \end{cases}$$

O más genéricamente:

$$f(t) = \begin{cases} -1 & -T/2 \leq t \leq 0 \\ 1 & 0 \leq t \leq T/2 \end{cases}; \text{ con } T = 2\pi/\omega_0$$

Siendo T, el período de la función, y ω_0 la frecuencia angular medida en radianes

Gráficamente...



Coeficiente a_0 de Fourier

$$a_0 = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cdot dt$$

$$a_0 = \frac{2}{T} \left[\int_{-\frac{T}{2}}^0 -dt + \int_{-0}^{\frac{T}{2}} dt \right] = \frac{2}{T} \cdot \left[-t \Big|_{-\frac{T}{2}}^0 + t \Big|_0^{\frac{T}{2}} \right]$$

$$a_0 = \frac{2}{T} \left[(0 \cdot (-1)) - \left(-\frac{T}{2} \cdot (-1) \right) + \left(\frac{T}{2} \right) + (0) \right]$$

$a_0 = 0$

(2)

Coeficientes a_n de Fourier

$$a_n = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cdot \cos(n\omega_0 t) \cdot dt$$

$$a_n = \frac{2}{T} \cdot \left[\int_{-\frac{T}{2}}^0 (-1) \cos(n\omega_0 t) \cdot dt + \int_{-0}^{\frac{T}{2}} (1) \cos(n\omega_0 t) \cdot dt \right]$$

$$a_n = \frac{2}{T} \cdot \left[-\frac{1}{n\omega_0} \cdot \sin(n\omega_0 t) \Big|_{\frac{T}{2}}^0 + \frac{1}{n\omega_0} \cdot \sin(n\omega_0 t) \Big|_0^{\frac{T}{2}} \right]$$

$a_0 = 0, \forall n \neq 0$

(3)

Coeficientes b_n de Fourier

$$b_n = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cdot \sin(n\omega_0 t) \cdot dt = \frac{2}{T} \left[\int_{-\frac{T}{2}}^0 (-1) \sin(n\omega_0 t) \cdot dt + \int_{-0}^{\frac{T}{2}} (1) \sin(n\omega_0 t) \cdot dt \right]$$

$$b_n = \frac{2}{T} \cdot \left[\frac{1}{n\omega_0} \cdot \cos(n\omega_0 t) \Big|_{\frac{T}{2}}^0 - \frac{1}{n\omega_0} \cdot \cos(n\omega_0 t) \Big|_0^{\frac{T}{2}} \right]$$

$$b_n = \frac{2}{\frac{2\pi}{\omega_0}} \cdot \frac{1}{n\omega_0} \cdot \left[\cos 0 + \cos(n\omega_0 \frac{T}{2}) - \left(\cos(n\omega_0 \frac{T}{2}) - \cos 0 \right) \right]$$

$$b_n = \frac{2}{n\pi} \cdot \left[(1 - \cos(n\pi)) - (\cos(n\pi) - 1) \right]$$

$$b_n = \frac{2}{n\pi} [1 - (-1)^n], \forall n \neq 0$$

(4)

Resultado

- La ecuación 4 es distinta de cero para los valores de n impares. Por lo tanto la función descompuesta en serie de Fourier queda del siguiente modo:

$$b_n = \frac{2}{1\pi} [1 - (-1)^1] \cdot \operatorname{sen}(1 \cdot \omega_0 t) + \frac{2}{3\pi} [1 - (-1)^3] \cdot \operatorname{sen}(3 \cdot \omega_0 t) + \\ + \frac{2}{5\pi} [1 - (-1)^5] \cdot \operatorname{sen}(5 \cdot \omega_0 t) + \frac{2}{7\pi} [1 - (-1)^7] \cdot \operatorname{sen}(7 \cdot \omega_0 t) + \dots$$

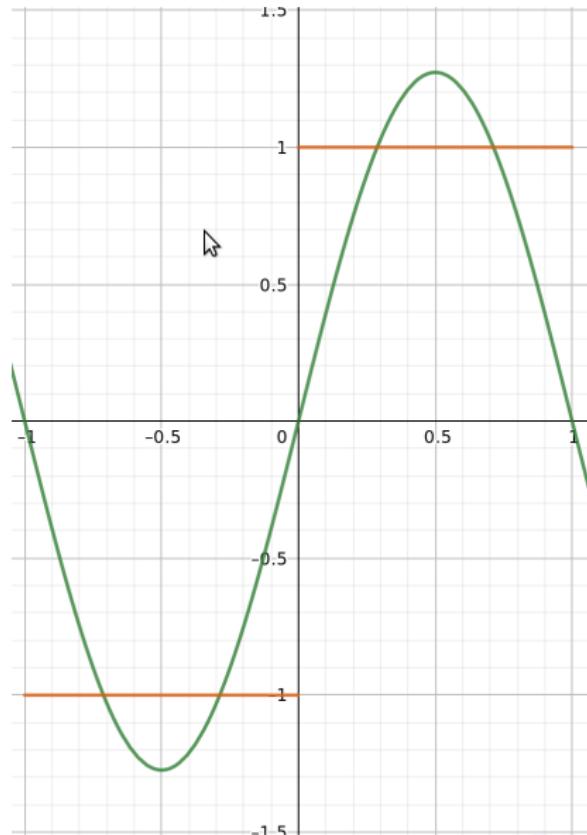
$$b_n = \frac{2}{1\pi} [2] \cdot \operatorname{sen}(1 \cdot \omega_0 t) + \frac{2}{3\pi} [2] \cdot \operatorname{sen}(3 \cdot \omega_0 t) + \frac{2}{5\pi} [2] \cdot \operatorname{sen}(5 \cdot \omega_0 t) + \\ + \frac{2}{7\pi} [2] \cdot \operatorname{sen}(7 \cdot \omega_0 t) + \dots$$

$$b_n = \frac{4}{\pi} \left[\operatorname{sen}(\omega_0 t) + \frac{1}{3} \cdot \operatorname{sen}(3 \cdot \omega_0 t) + \frac{1}{5} \cdot \operatorname{sen}(5 \cdot \omega_0 t) + \frac{1}{7} \cdot \operatorname{sen}(7 \cdot \omega_0 t) + \dots \right]$$

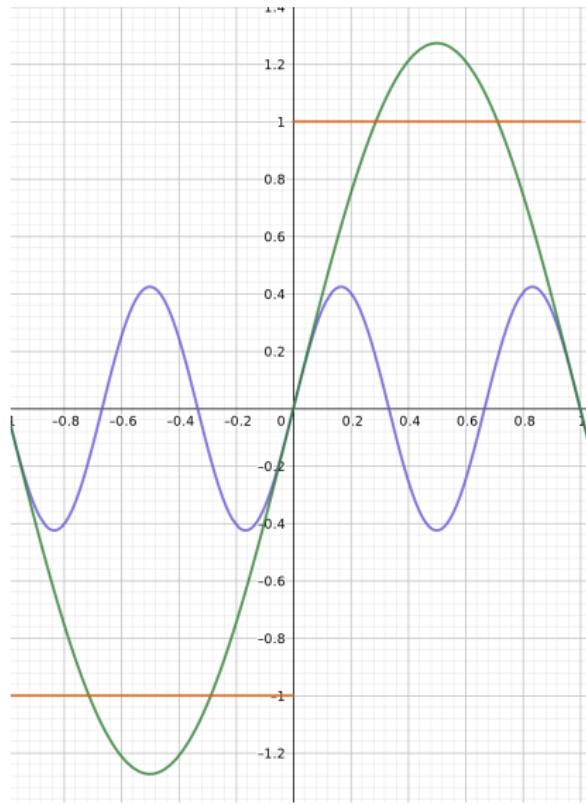
Donde, $\omega_0 = \frac{2 \cdot \pi}{T}$, y $T = (1) - (-1) = 2 \Rightarrow \omega_0 = \pi$. (5)

$$b_n = \frac{4}{\pi} \left[\operatorname{sen}(\pi \cdot t) + \frac{1}{3} \cdot \operatorname{sen}(3\pi \cdot t) + \frac{1}{5} \cdot \operatorname{sen}(5\pi \cdot t) + \frac{1}{7} \cdot \operatorname{sen}(7\pi \cdot t) + \dots \right]$$

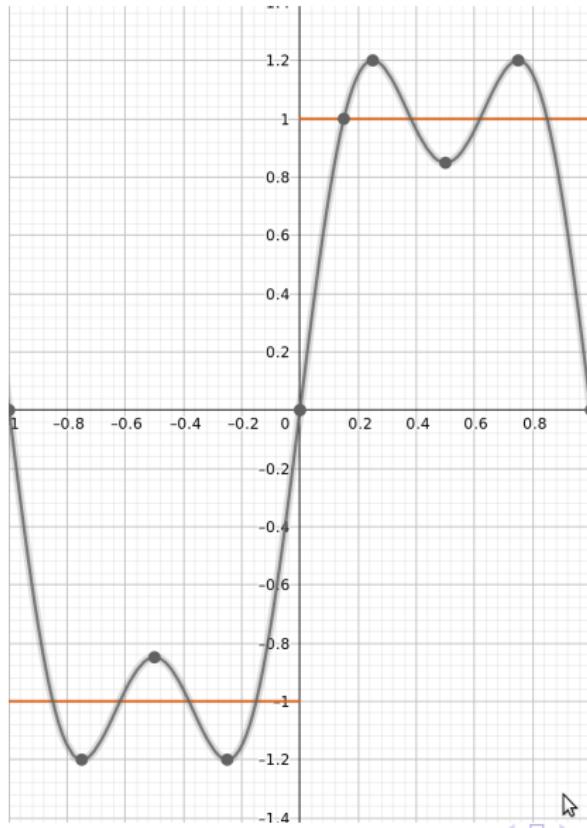
Aproximando con solo la fundamental...



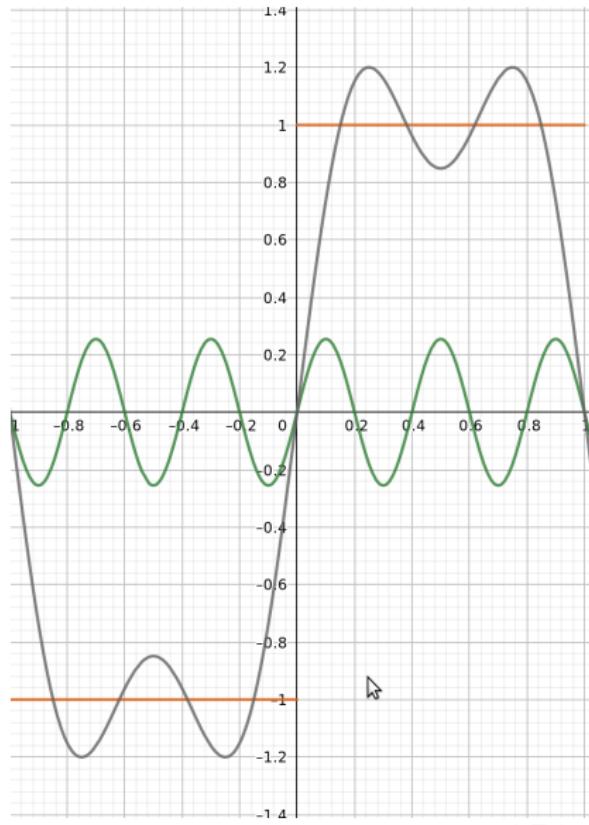
Aproximando con la fundamental y la 3er. Armónica



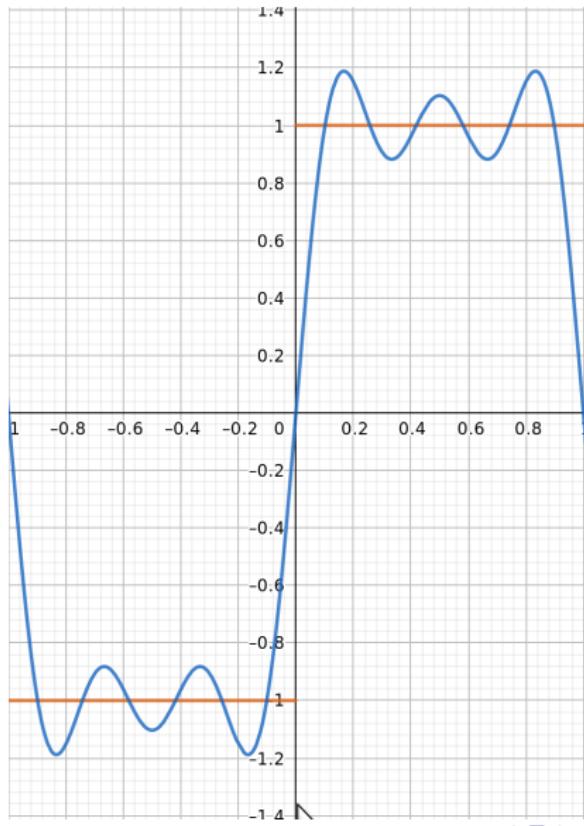
Sumando la fundamental y la 3er. Armónica



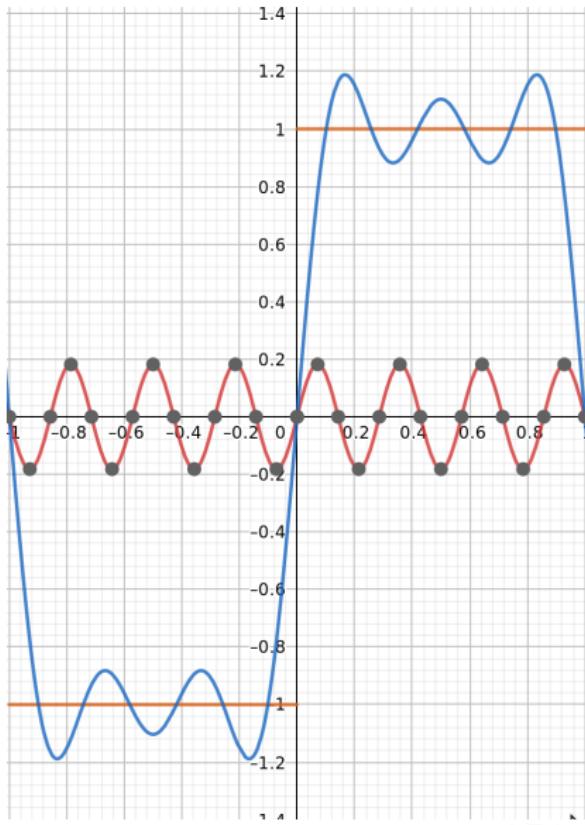
Agregando la 5ta. Armónica



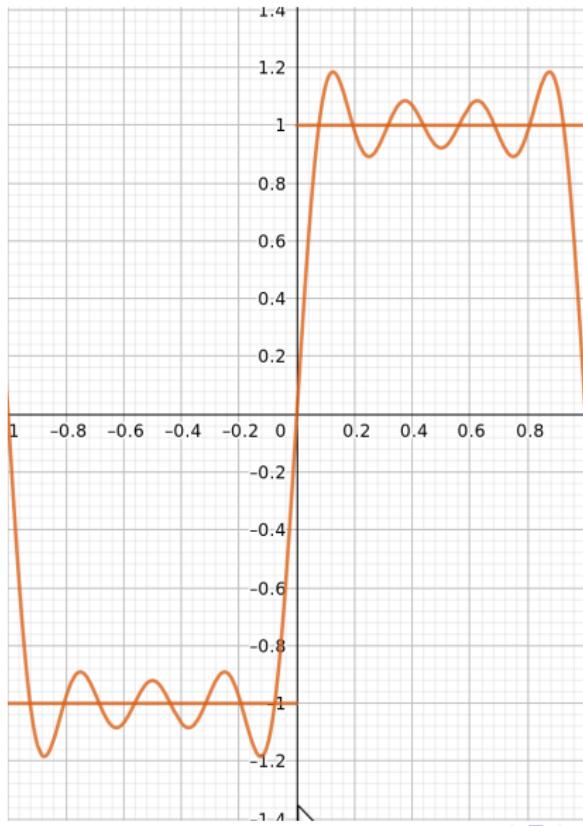
Sumando la 5ta. Armónica



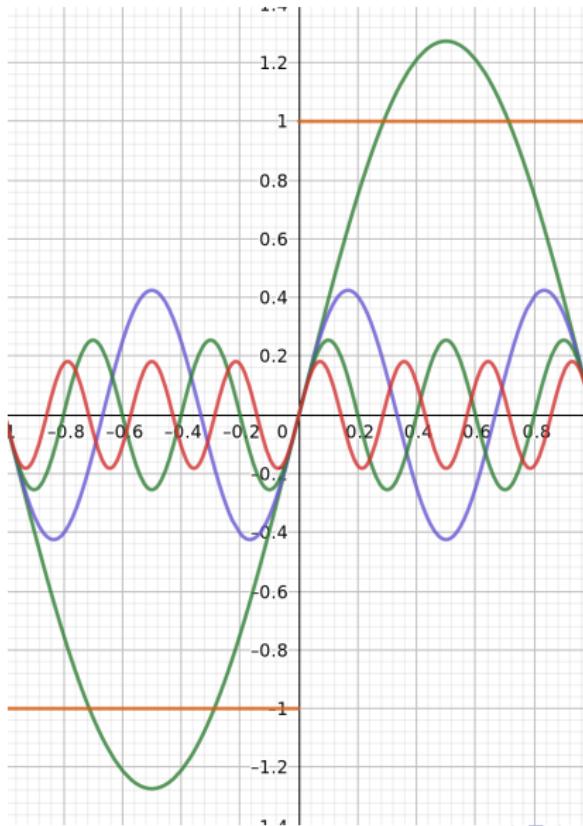
Agregando la 7ma. Armónica



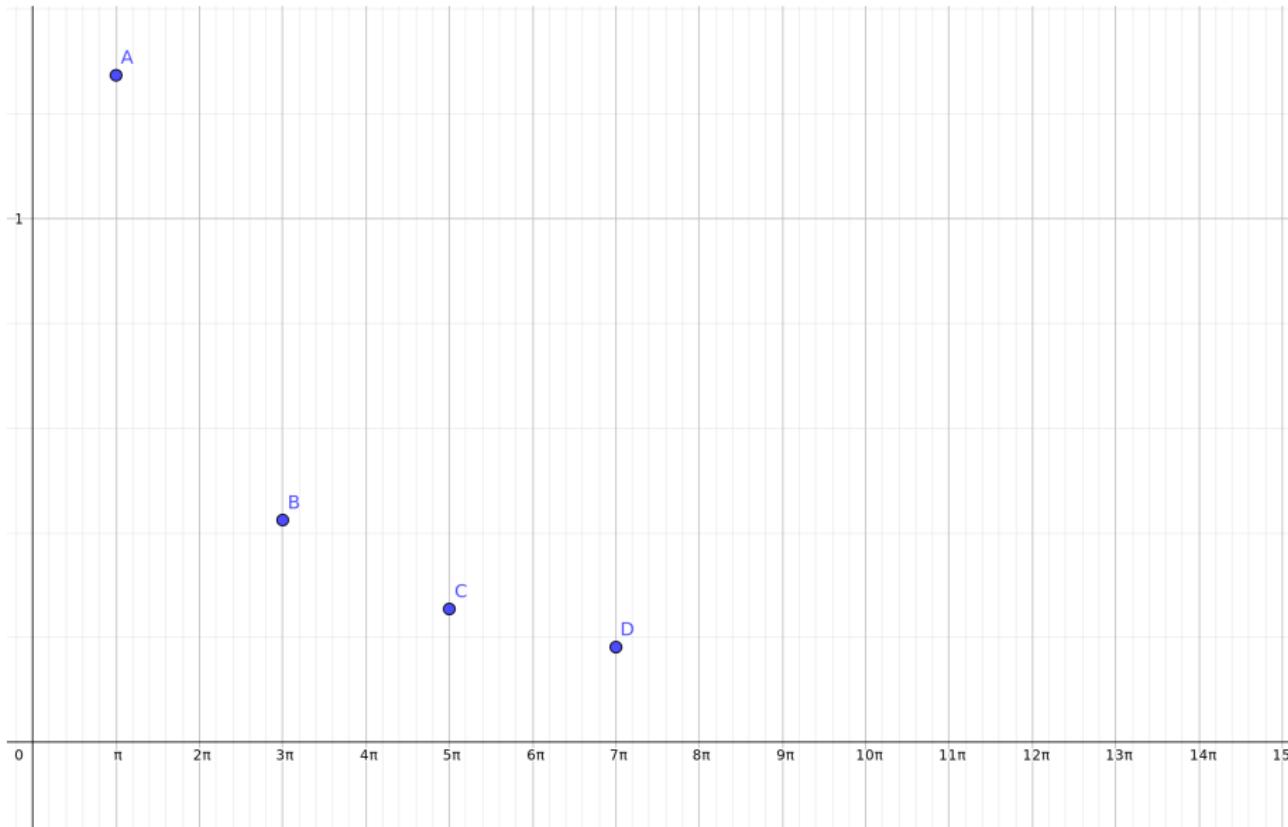
Sumando la 7ma. Armónica



Representación en el dominio del tiempo



Representación en el dominio de la frecuencia



Observaciones

Observaciones

- Permite decomponer cualquier señal periódica en señales muy simples de ser analizadas: *seno* y *coseno*.

Observaciones

- Permite decomponer cualquier señal periódica en señales muy simples de ser analizadas: *seno* y *coseno*.
- El requisito de periodicidad es absolutamente flexible: una grabación de audio es finalmente una secuencia sonidos en un tiempo finito, de modo que repetirla infinitamente la convierte en una señal periódica.

Observaciones

- Permite decomponer cualquier señal periódica en señales muy simples de ser analizadas: *seno* y *coseno*.
- El requisito de periodicidad es absolutamente flexible: una grabación de audio es finalmente una secuencia sonidos en un tiempo finito, de modo que repetirla infinitamente la convierte en una señal periódica.
- Pasamos de representar una señal continua en el dominio del tiempo en una señal discreta en el dominio de la frecuencia.

Observaciones

- Permite decomponer cualquier señal periódica en señales muy simples de ser analizadas: *seno* y *coseno*.
- El requisito de periodicidad es absolutamente flexible: una grabación de audio es finalmente una secuencia sonidos en un tiempo finito, de modo que repetirla infinitamente la convierte en una señal periódica.
- Pasamos de representar una señal continua en el dominio del tiempo en una señal discreta en el dominio de la frecuencia.
- Eliminar la componentes armónicas a partir de un cierto orden hace que la señal en el dominio de la frecuencia tenga una cantidad finita (y por lo tanto computarizable) de componentes de frecuencias.

Observaciones

- Permite decomponer cualquier señal periódica en señales muy simples de ser analizadas: *seno* y *coseno*.
- El requisito de periodicidad es absolutamente flexible: una grabación de audio es finalmente una secuencia sonidos en un tiempo finito, de modo que repetirla infinitamente la convierte en una señal periódica.
- Pasamos de representar una señal continua en el dominio del tiempo en una señal discreta en el dominio de la frecuencia.
- Eliminar la componentes armónicas a partir de un cierto orden hace que la señal en el dominio de la frecuencia tenga una cantidad finita (y por lo tanto computarizable) de componentes de frecuencias.
- Esto, evidentemente, introduce un error.

Observaciones

- Permite decomponer cualquier señal periódica en señales muy simples de ser analizadas: *seno* y *coseno*.
- El requisito de periodicidad es absolutamente flexible: una grabación de audio es finalmente una secuencia sonidos en un tiempo finito, de modo que repetirla infinitamente la convierte en una señal periódica.
- Pasamos de representar una señal continua en el dominio del tiempo en una señal discreta en el dominio de la frecuencia.
- Eliminar las componentes armónicas a partir de un cierto orden hace que la señal en el dominio de la frecuencia tenga una cantidad finita (y por lo tanto computarizable) de componentes de frecuencias.
- Esto, evidentemente, introduce un error.
- Por lo general las armónicas de ordenes mayores tienden a tener amplitudes muy pequeñas.

Observaciones

- Permite decomponer cualquier señal periódica en señales muy simples de ser analizadas: *seno* y *coseno*.
- El requisito de periodicidad es absolutamente flexible: una grabación de audio es finalmente una secuencia sonidos en un tiempo finito, de modo que repetirla infinitamente la convierte en una señal periódica.
- Pasamos de representar una señal continua en el dominio del tiempo en una señal discreta en el dominio de la frecuencia.
- Eliminar las componentes armónicas a partir de un cierto orden hace que la señal en el dominio de la frecuencia tenga una cantidad finita (y por lo tanto computarizable) de componentes de frecuencias.
- Esto, evidentemente, introduce un error.
- Por lo general las armónicas de ordenes mayores tienden a tener amplitudes muy pequeñas.
- Sin embargo, el error debido al corte de frecuencias no es apreciable en la práctica.

1 Fundamentos

2 Procesamiento de Señales digitales

- Digitalización de la señal
- Arquitecturas de Procesamiento de una señal digital

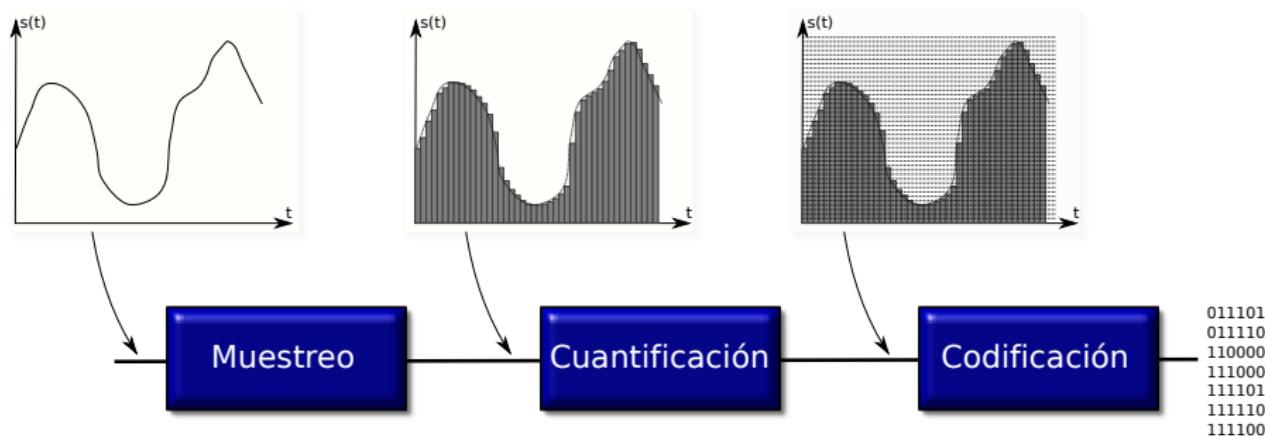
3 Modelo de ejecución SIMD

4 Implementaciones SIMD en x86

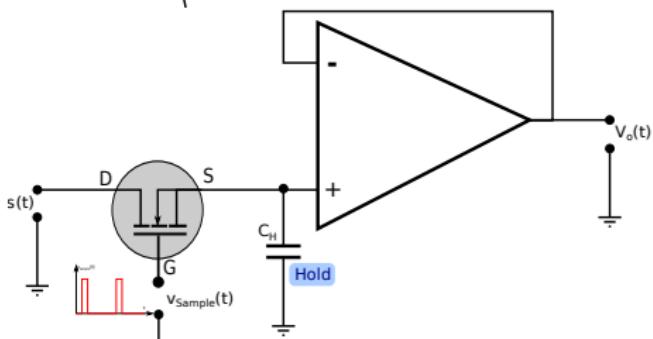
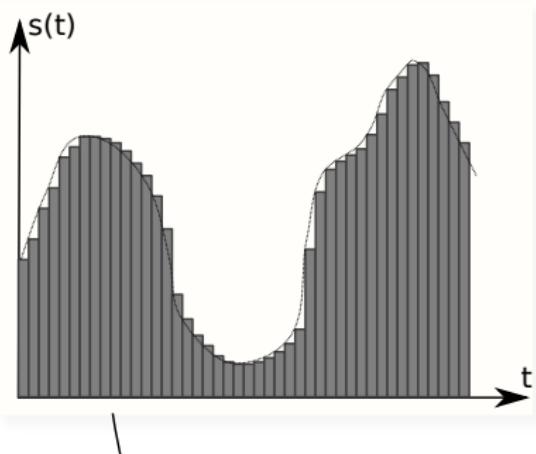
5 Instrucciones

Señal digitalizada

- El proceso de digitalización de una señal responde al siguiente modelo.



Muestreo y retención

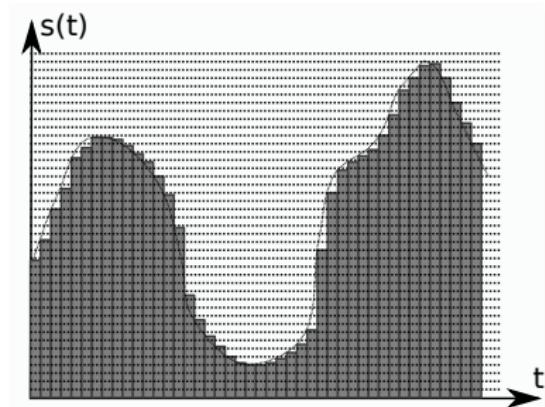


- Se toma un valor instantáneo de la señal y se “retiene” el valor de tensión en una capacidad, con un circuito resistivo de descarga de muy alta resistencia.

- *Observar que el error aumenta en los cambios abruptos de señal.*

Cuantificación y Codificación

- Se llevan a cabo en un Conversor Analógico Digital



- Se puede aproximar por redondeo o truncamiento
- *Independientemente del método el error aumenta en las zonas donde la derivada de la señal es más alta*

1 Fundamentos

2 Procesamiento de Señales digitales

- Digitalización de la señal
- Arquitecturas de Procesamiento de una señal digital

3 Modelo de ejecución SIMD

4 Implementaciones SIMD en x86

5 Instrucciones

Procesador de Señales Digitales

- Es una CPU de propósito dedicado, diseñada para realizar cálculos y procesamiento de un único tipo de datos: secuencias de valores correspondientes a la codificación de las muestras de una señal de entrada.
- Su arquitectura está pensada para optimizar el procesamiento de datos que no son de gran tamaño (8, 16, 24, o a lo sumo 32 bits). Se trata de pixeles de una imagen, o de valores instantáneos de audio, o de señales médicas o de mapas térmicos, etc.
- La característica distintiva de este tipo de datos reside en sus algoritmos de cálculo: Normalmente se requiere procesar no solo el valor actual sino la combinación del valor actual con n valores anteriores en el tiempo, o vecinos (en el caso de una imagen lo que llamaremos N_8)

Procesador de Señales Digitales

- En general una operación muy frecuente son los filtros de convolución, dados por una expresión del tipo:

$$y[n] = \sum_{i=0}^N a_i \cdot x[n-i] = a_0 \cdot x[n] + a_1 \cdot x[n-1] + a_2 \cdot x[n-2] + \dots + a_N \cdot x[n-N]$$

- En general se deben resolver sumas de productos y acumular su resultado.
- Para optimizar se deberían leer y procesar varios datos en paralelo
- Las técnicas de paralelismo que se desarrollaron en estos procesadores se denominaron por tal razón **Data Level Parallelism**.

Arquitectura de un Procesador de Señales Digitales

- En general el diseño de un Procesador de Señales Digitales concentra sus esfuerzos en resolver en paralelo:

Arquitectura de un Procesador de Señales Digitales

- En general el diseño de un Procesador de Señales Digitales concentra sus esfuerzos en resolver en paralelo:

El acceso a los operandos Para este fin se implementan buses paralelos con una cantidad de líneas de datos superior al ancho de palabra de la CPU

Arquitectura de un Procesador de Señales Digitales

- En general el diseño de un Procesador de Señales Digitales concentra sus esfuerzos en resolver en paralelo:

El acceso a los operandos Para este fin se implementan buses paralelos con una cantidad de líneas de datos superior al ancho de palabra de la CPU

El almacenamiento de resultados Se resuelve mediante buses dedicados para manejar la salida de la ALU hacia memoria o registros (con las consideraciones que la concurrencia de accesos debe tener en cuenta en el hardware)

Arquitectura de un Procesador de Señales Digitales

- En general el diseño de un Procesador de Señales Digitales concentra sus esfuerzos en resolver en paralelo:

El acceso a los operandos Para este fin se implementan buses paralelos con una cantidad de líneas de datos superior al ancho de palabra de la CPU

El almacenamiento de resultados Se resuelve mediante buses dedicados para manejar la salida de la ALU hacia memoria o registros (con las consideraciones que la concurrencia de accesos debe tener en cuenta en el hardware)

El procesamiento de la mayor cantidad de datos posible Los primeros pasos se conocen como VLIW (Very Large Instruction Word), que derivó en el modelo de ejecución SIMD (Single Instruction Multiple Data)

1 Fundamentos

2 Procesamiento de Señales digitales

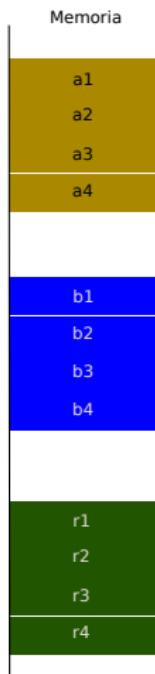
3 Modelo de ejecución SIMD

- Un modelo de paralelización

4 Implementaciones SIMD en x86

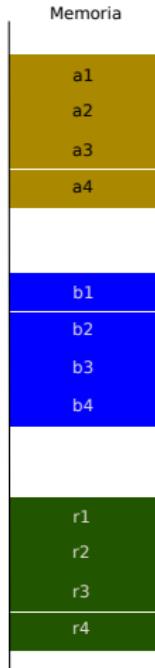
5 Instrucciones

Single Instruction Multiple Data



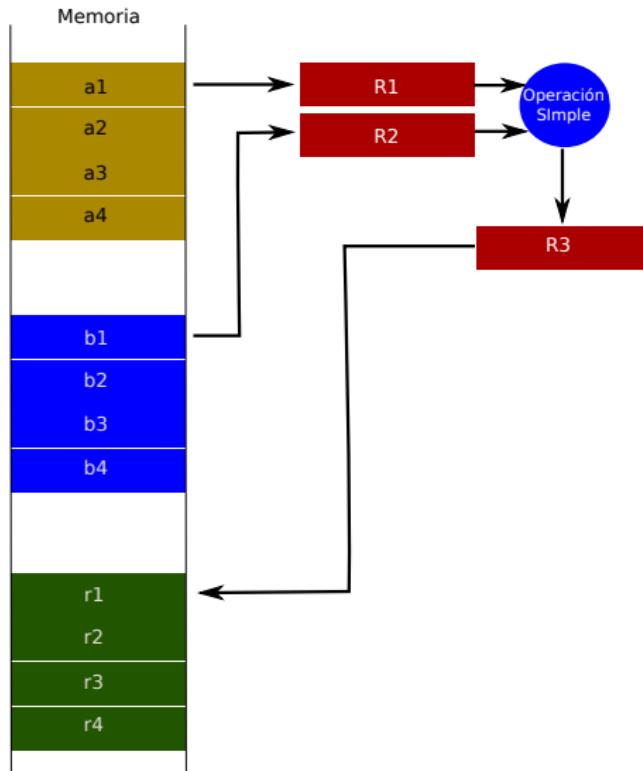
- Se trata de un modelo de ejecución capaz de computar una sola operación sobre un conjunto de múltiples datos.
- Se refiere a esta técnica como paralelismo a nivel de datos.
- Es particularmente útil para procesar audio, video, o imágenes en donde se aplican algoritmos repetitivos sobre sets de datos del mismo formato y que se procesan en conjunto, como por ejemplo en filtros, compresores, codificadores en donde la salida depende de los últimos n valores de muestras tomados.
- La figura muestra el layout típico de variables memoria para aplicar este modelo

Como sería la vida sin SIMD?

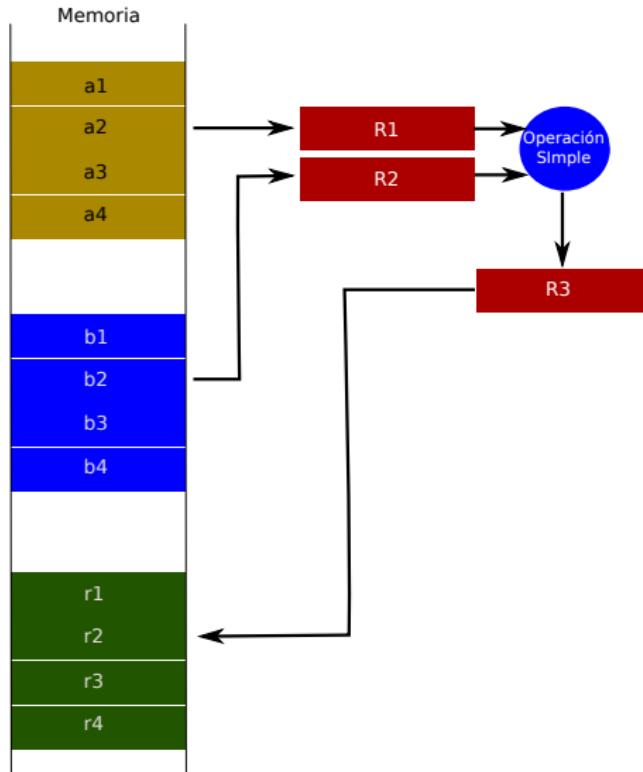


- En contraposición a SIMD el modelo previo es SISD (**S**ingle **I**nstruction **S**ingle **D**ata)
- Considerando el sector de memoria de la figura, nos proponemos realizar una operación aritmética o lógica sobre las cadenas de datos a_n y b_n , almacenando el resultado en r_n
- Si un procesador no dispone de un modelo arquitectural que le permita implementar paralelismo a nivel de datos, como SIMD, esta operación implica un loop, ya que solo puede procesar un dato por cada instrucción (SISD).

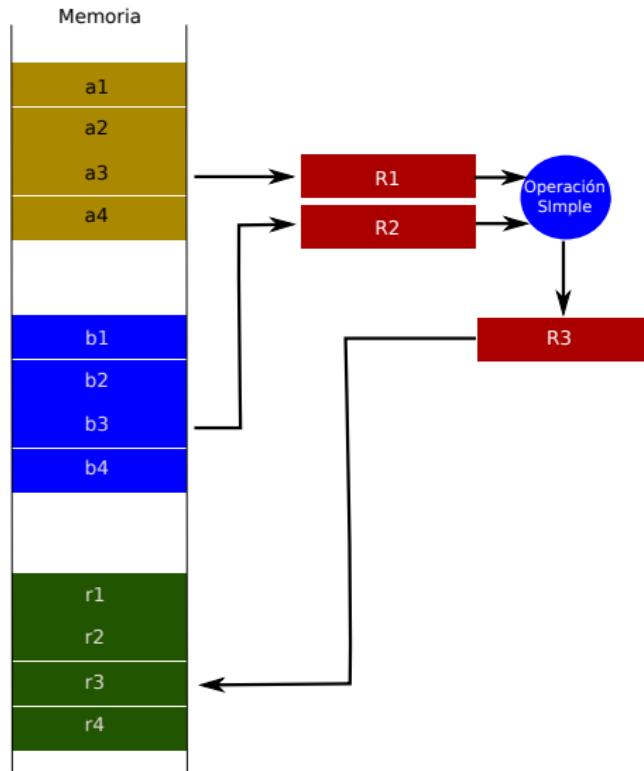
Single Instruction Single Data



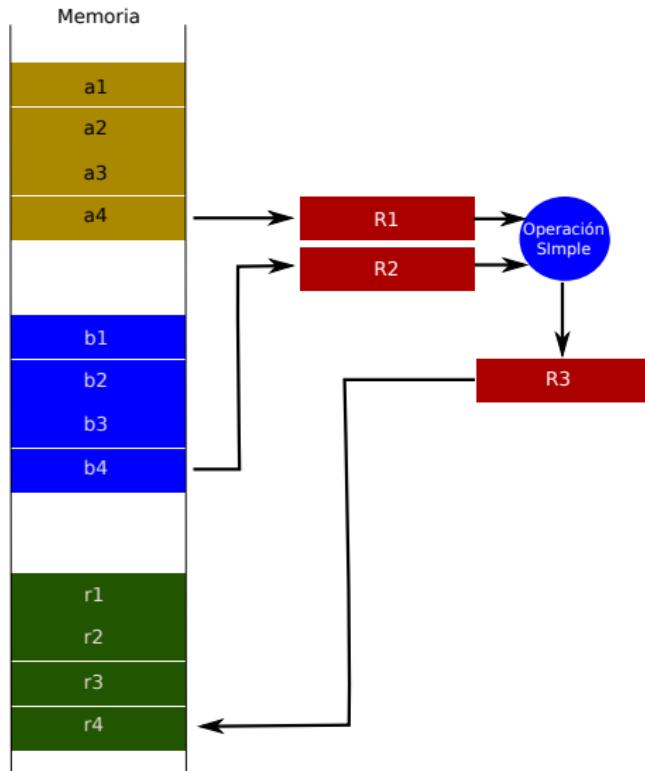
Single Instruction Single Data



Single Instruction Single Data

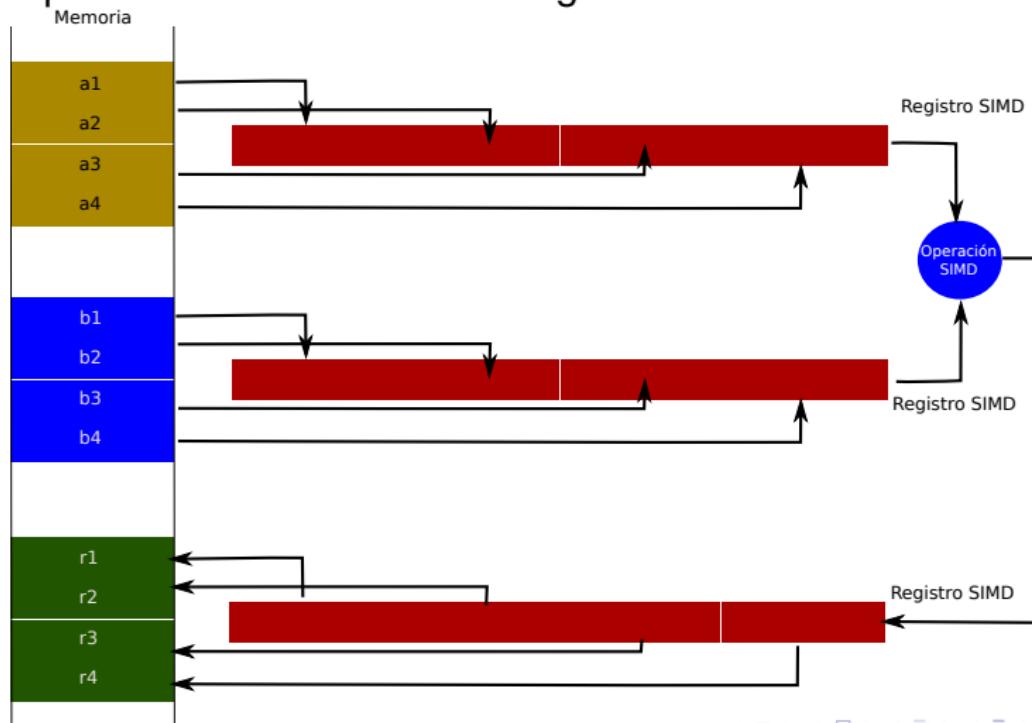


Single Instruction Single Data

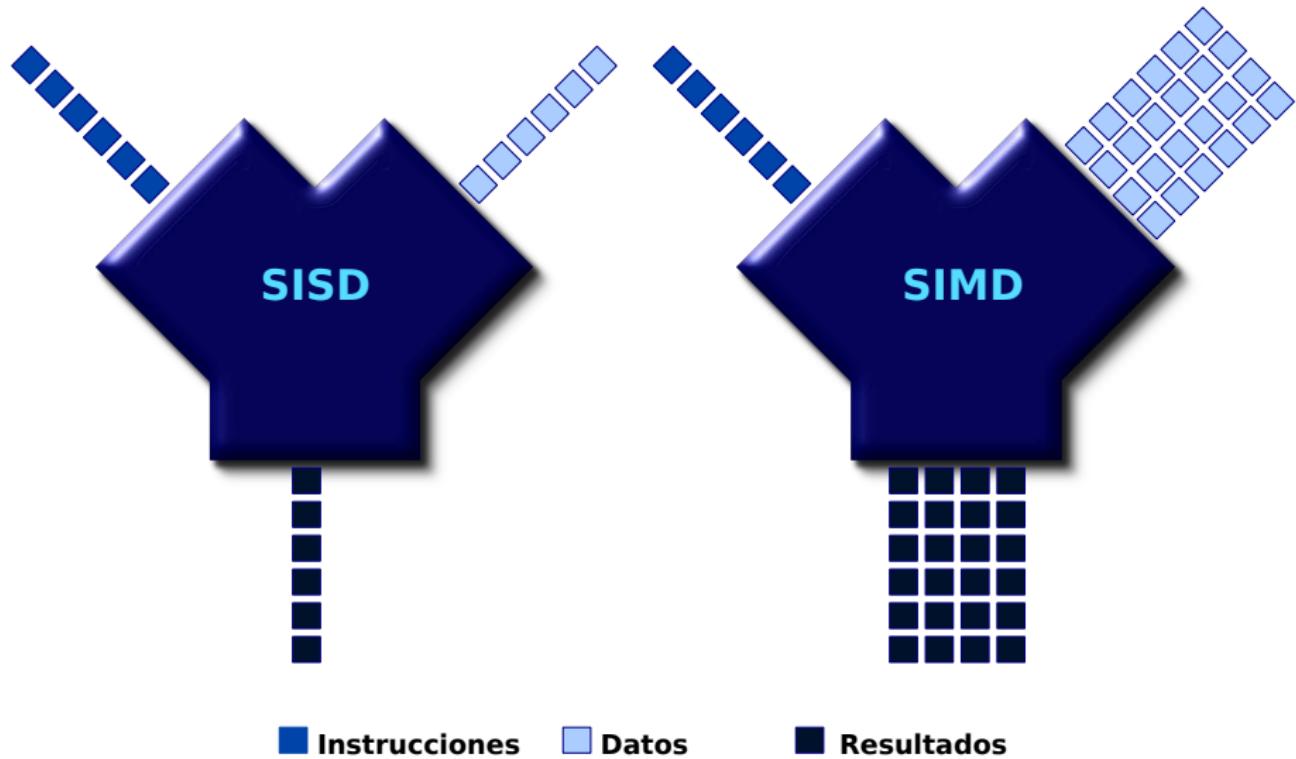


Single Instruction Multiple Data

- Cada Registro se carga en una sola instrucción
- La operación se efectúa en la segunda instrucción

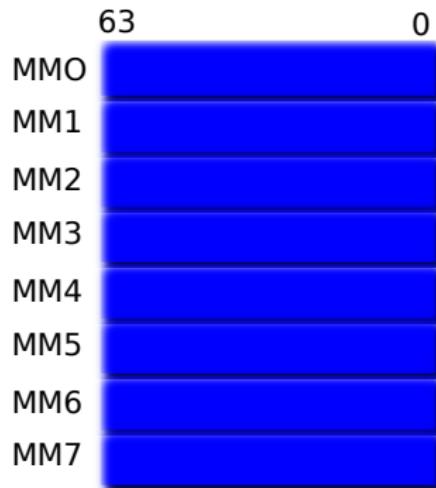


Resumiendo, SIMD vs. SISD

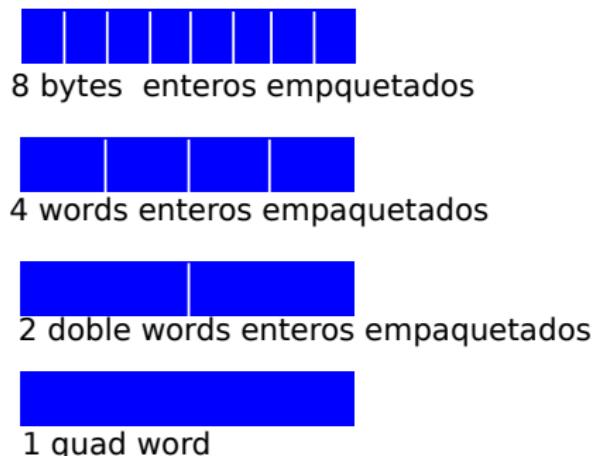


Multimedia extensions - MMX

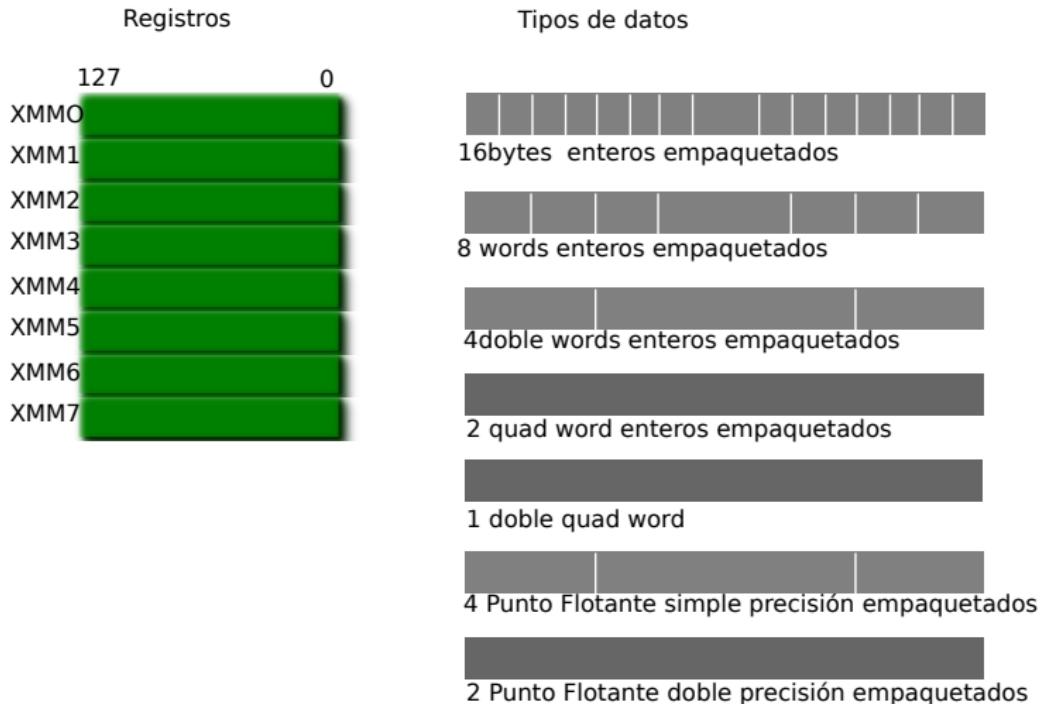
Registros



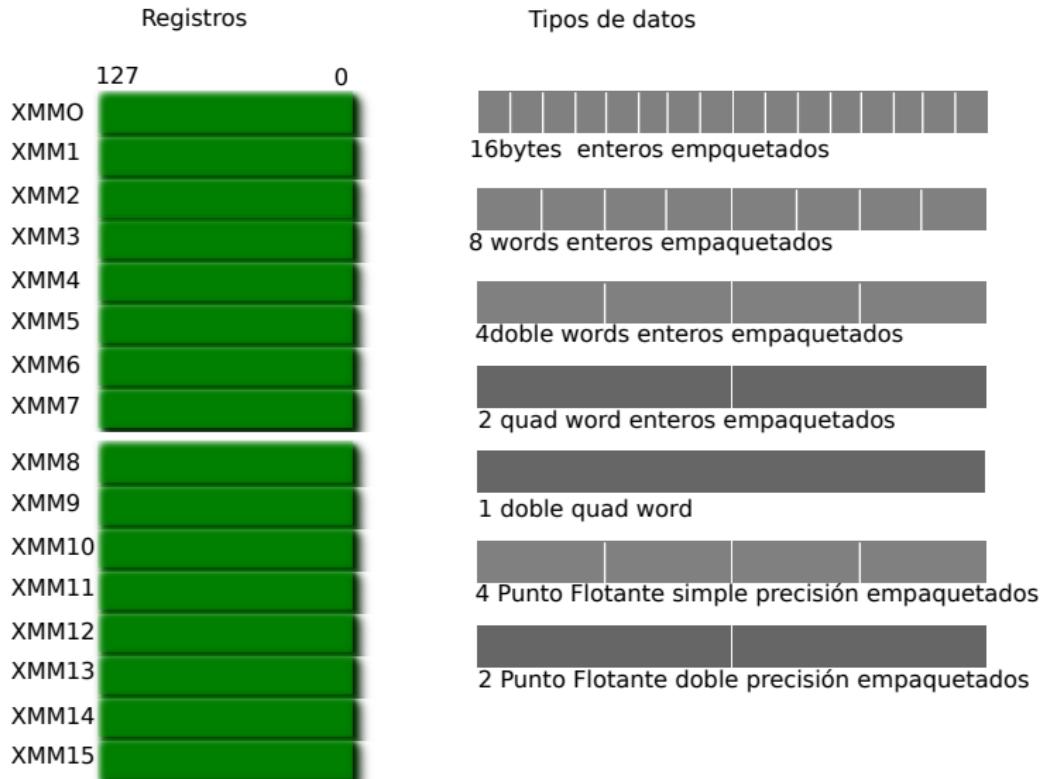
Tipos de datos



Streaming SIMD Extension



Streaming SIMD Extension en Modo 64 bits



1 Fundamentos

2 Procesamiento de Señales digitales

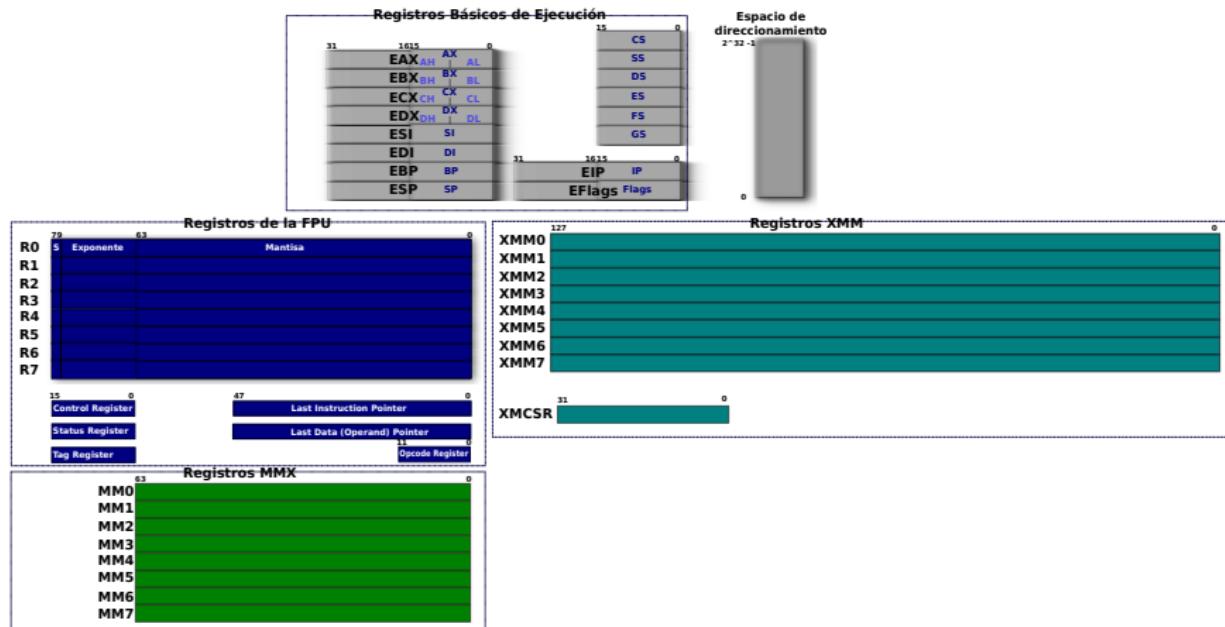
3 Modelo de ejecución SIMD

4 Implementaciones SIMD en x86

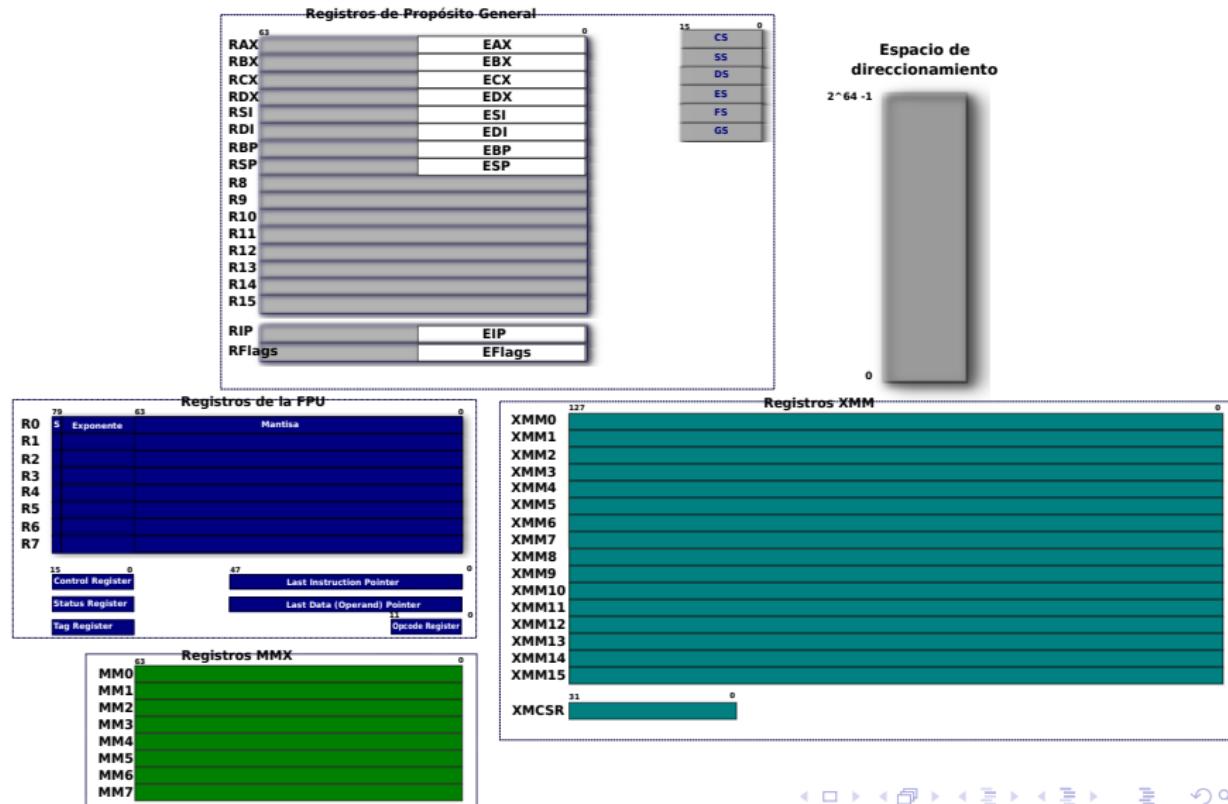
- Arquitectura completa
- Números Reales
- Formatos de Punto Fijo
- Formatos de Punto Flotante
- Codificación de Números Reales
- Extensiones AVX

5 Instrucciones

ISA en 32 bits



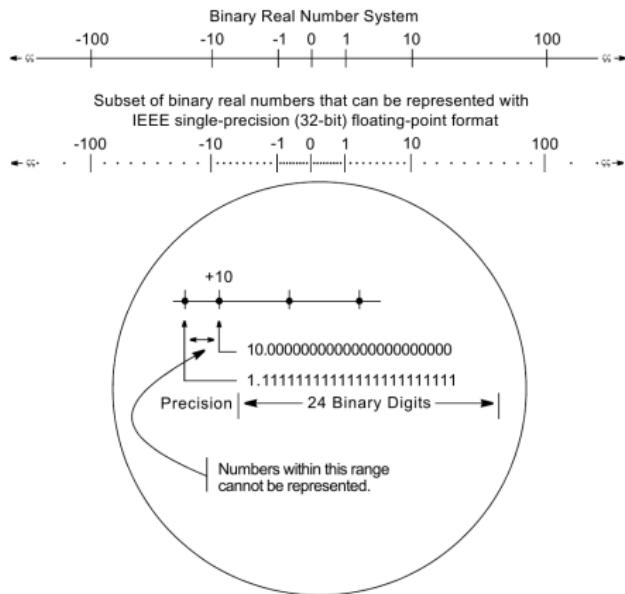
ISA en 64 bits



- 1 Fundamentos
- 2 Procesamiento de Señales digitales
- 3 Modelo de ejecución SIMD
- 4 Implementaciones SIMD en x86
 - Arquitectura completa
 - **Números Reales**
 - Formatos de Punto Fijo
 - Formatos de Punto Flotante
 - Codificación de Números Reales
 - Extensiones AVX
- 5 Instrucciones

Números Reales

- El rango de los números reales comprende desde $-\infty$ hasta $+\infty$.
- Los registros de un procesador tienen resolución finita.
- Por lo tanto un computador solo puede representar un subconjunto de \mathbb{R} .
- Además, no es solo un tema de magnitud sino de resolución.



Representación binaria de Números Reales

Formatos

En general podemos formalizar la representación de un número real expresado en los siguientes formatos:

- ① Punto Fijo
- ② Punto Flotante

Notación Científica

- Para el caso de los números reales se trabaja en **notación científica**.

$$n = \pm f * 10^e$$

Notación Científica

- Para el caso de los números reales se trabaja en **notación científica**.

$$n = \pm f * 10^e$$

$$-725,832 = -7,25832 * 10^2 = -725,832 * 10^0$$

Notación Científica

- Para el caso de los números reales se trabaja en **notación científica**.

$$n = \pm f * 10^e$$

$$-725,832 = -7,25832 * 10^2 = -725,832 * 10^0$$

$$3,14 = 0,314 * 10^1 = 3,14 * 10^0$$

Notación Científica

- Para el caso de los números reales se trabaja en **notación científica**.

$$n = \pm f * 10^e$$

$$-725,832 = -7,25832 * 10^2 = -725,832 * 10^0$$

$$3,14 = 0,314 * 10^1 = 3,14 * 10^0$$

$$0,000001 = 0,1 * 10^{-5} = 1,0 * 10^{-6}$$

Notación Científica

- Para el caso de los números reales se trabaja en **notación científica**.

$$n = \pm f * 10^e$$

$$-725,832 = -7,25832 * 10^2 = -725,832 * 10^0$$

$$3,14 = 0,314 * 10^1 = 3,14 * 10^0$$

$$0,000001 = 0,1 * 10^{-5} = 1,0 * 10^{-6}$$

$$1941 = 0,1941 * 10^4 = 1,941 * 10^3$$

Notación Científica

- Para el caso de los números reales se trabaja en **notación científica**.

$$n = \pm f * 10^e$$

$$-725,832 = -7,25832 * 10^2 = -725,832 * 10^0$$

$$3,14 = 0,314 * 10^1 = 3,14 * 10^0$$

$$0,000001 = 0,1 * 10^{-5} = 1,0 * 10^{-6}$$

$$1941 = 0,1941 * 10^4 = 1,941 * 10^3$$

- Para unificar la representación se recurre a la **notación científica normalizada**, en donde:

$0,1 \leq f < 1$, y **e** es un entero con signo.

Notación Científica en el sistema Binario

- En el sistema binario la expresión de un número en notación científica normalizada es:

$$n = \pm f * 2^e$$

- En donde:

$0,5 \leq f < 1$, y e es un entero con signo.

1 Fundamentos

2 Procesamiento de Señales digitales

3 Modelo de ejecución SIMD

4 Implementaciones SIMD en x86

- Arquitectura completa
- Números Reales
- **Formatos de Punto Fijo**
- Formatos de Punto Flotante
- Codificación de Números Reales
- Extensiones AVX

5 Instrucciones

Representación binaria en Punto Fijo con signo

- Se representan mediante una expresión del tipo:

$$(a_n a_{n-1} \dots a_0.a_{-1} a_{-2} \dots a_{-m})_2 = (-1)^s * (a_n * 2^n + \dots + a_0 * 2^0 + a_{-1} * 2^{-1} + a_{-2} * 2^{-2} + \dots + a_{-m} * 2^{-m})$$

- Donde:

- s es el signo: **0** si el número es positivo y **1** si es negativo
 - $a_i \in \mathbb{Z}$ y $0 \leq a_i \leq 1, \forall i = -m, -1, 0, 1, \dots n$

- Distancia entre dos números consecutivos es 2^{-m} .

- *Deja de ser un rango continuo de números para transformarse en un rango discreto.*

1 Fundamentos

2 Procesamiento de Señales digitales

3 Modelo de ejecución SIMD

4 Implementaciones SIMD en x86

- Arquitectura completa
- Números Reales
- Formatos de Punto Fijo
- Formatos de Punto Flotante**
- Codificación de Números Reales
- Extensiones AVX

5 Instrucciones

Representación en Punto Flotante

- Se representan con los pares de valores (m, e) , denotando:

$$(m, e) = m * b^e$$

- En donde:

- 1 **m** llamado mantisa, y que representa un número fraccionario.
- 2 **e** llamado exponente, al cual se debe elevar la base numérica **(b)** de representación para obtener el valor real.

Representación en Punto Flotante

Representación en Punto Flotante

- Mantisa y exponente pueden representarse:

Representación en Punto Flotante

- Mantisa y exponente pueden representarse:
 - 1 con signo.

Representación en Punto Flotante

- Mantisa y exponente pueden representarse:
 - 1 con signo.
 - 2 sin signo.

Representación en Punto Flotante

- Mantisa y exponente pueden representarse:
 - 1 con signo.
 - 2 sin signo.
 - 3 con notación complemento.

Representación en Punto Flotante

- Mantisa y exponente pueden representarse:
 - 1 con signo.
 - 2 sin signo.
 - 3 con notación complemento.
 - 4 con notación exceso m.

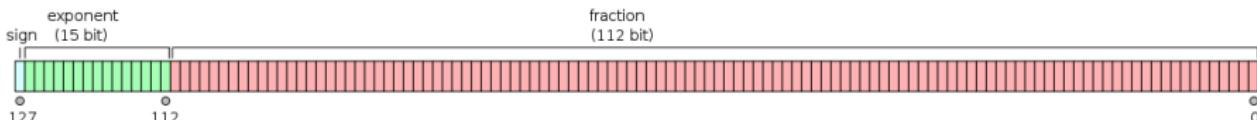
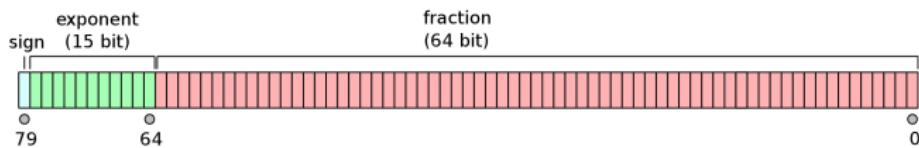
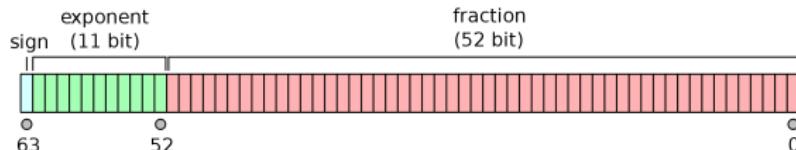
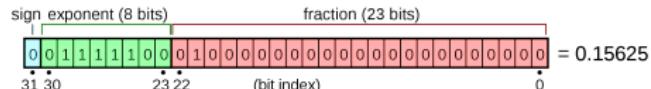
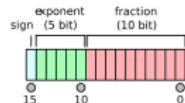
Representación en Punto Flotante

- Mantisa y exponente pueden representarse:
 - 1 con signo.
 - 2 sin signo.
 - 3 con notación complemento.
 - 4 con notación exceso m.
- Para que las representaciones sean únicas, la mantisa deberá estar normalizada.

Punto Flotante: Formato IEEE 754

- IEEE (Institute of Electrical and Electronic Engineers).
- El Standard IEEE 754 para punto flotante binario es el mas ampliamente utilizado. En este Standard se especifican los formatos para 32 bits, 64 bits, y 80 bits.
- En 2008 se introdujeron un formato de 16 bits y el de 80 fue reemplazado por uno de 128 bits (IEEE 754-2008).

Formatos IEEE 754



IEEE 754: Rangos

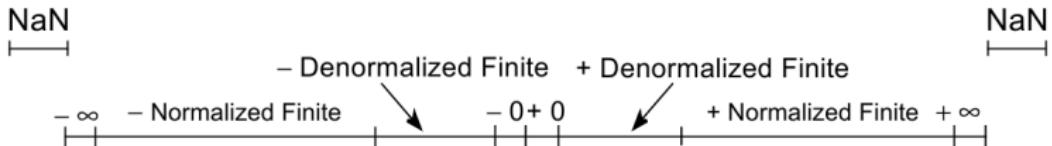
Data Type	Length	Precision (Bits)	Approximate Normalized Range	
			Binary	Decimal
Single Precision	32	24	2^{-126} to 2^{127}	1.18×10^{-38} to 3.40×10^{38}
Double Precision	64	53	2^{-1022} to 2^{1023}	2.23×10^{-308} to 1.79×10^{308}
Double Extended Precision	80	64	2^{-16382} to 2^{16383}	3.37×10^{-4932} to 1.18×10^{4932}

- 1 Fundamentos
- 2 Procesamiento de Señales digitales
- 3 Modelo de ejecución SIMD
- 4 Implementaciones SIMD en x86
 - Arquitectura completa
 - Números Reales
 - Formatos de Punto Fijo
 - Formatos de Punto Flotante
 - Codificación de Números Reales
 - Extensiones AVX
- 5 Instrucciones

Codificación de los números normalizados... y los demás

- Ceros signados
- Números finitos de-normalizados
- Números finitos normalizados
- Infinitos signados
- NaNs (Not a Number)
- Números Indefinidos

Códigos para cada caso



Real Number and NaN Encodings For 32-Bit Floating-Point Format

S	E	Sig ¹	
1	0	0.000...	- 0

S	E	Sig ¹	
+ 0	0	0.000...	

1	0	0.XXX... ²	- Denormalized Finite
---	---	-----------------------	-----------------------

+Denormalized Finite	0	0	0.XXX... ²
----------------------	---	---	-----------------------

1	1...254	1.XXX...	- Normalized Finite
---	---------	----------	---------------------

+Normalized Finite	0	1...254	1.XXX...
--------------------	---	---------	----------

1	255	1.000...	- ∞
---	-----	----------	------------

+ ∞	0	255	1.000...
------------	---	-----	----------

X ³	255	1.0XX... ²	SNaN
----------------	-----	-----------------------	------

SNaN	X ³	255	1.0XX... ²
------	----------------	-----	-----------------------

X ³	255	1.1XX...	QNaN
----------------	-----	----------	------

QNaN	X ³	255	1.1XX...
------	----------------	-----	----------

NOTES:

1. Integer bit of fraction implied for single-precision floating-point format.
2. Fraction must be non-zero.
3. Sign bit ignored.

Ceros Signados

Ceros Signados

- Una operación puede dar +0 o -0 en función del bit de signo.

Ceros Signados

- Una operación puede dar +0 o -0 en función del bit de signo.
- En ambos casos el valor es el mismo.

Ceros Signados

- Una operación puede dar +0 o -0 en función del bit de signo.
- En ambos casos el valor es el mismo.
- El signo de un resultado cero depende de la operación en sí y del modo de redondeo utilizado.

Ceros Signados

- Una operación puede dar +0 o -0 en función del bit de signo.
- En ambos casos el valor es el mismo.
- El signo de un resultado cero depende de la operación en sí y del modo de redondeo utilizado.
- Los ceros signados ayudan a interpretar el intervalo aritmético en el que se ubicaría el resultado si la precisión aritmética fuese mayor.

Ceros Signados

- Una operación puede dar +0 o -0 en función del bit de signo.
- En ambos casos el valor es el mismo.
- El signo de un resultado cero depende de la operación en sí y del modo de redondeo utilizado.
- Los ceros signados ayudan a interpretar el intervalo aritmético en el que se ubicaría el resultado si la precisión aritmética fuese mayor.
- Indica la dirección desde la cual ocurrió el redondeo a cero, o el signo de un infinito que fue invertido.

Números Finitos Normalizados

Números Finitos Normalizados

- El rango de éstos números se compone de todos los valores finitos distintos de cero codificables en formato de números reales entre 0 y $\pm\infty$.

Números Finitos Normalizados

- El rango de éstos números se compone de todos los valores finitos distintos de cero codificables en formato de números reales entre 0 y $\pm\infty$.
- En el formato de punto flotante simple precisión estos números se componen de todos aquellos cuyos exponentes desplazados van de 1 a 254, (no desplazados van de -126 a 127).

Números Finitos Normalizados

- El rango de éstos números se compone de todos los valores finitos distintos de cero codificables en formato de números reales entre 0 y $\pm\infty$.
- En el formato de punto flotante simple precisión estos números se componen de todos aquellos cuyos exponentes desplazados van de 1 a 254, (no desplazados van de -126 a 127).
- Cuando se aproximan a cero, estos números no pueden seguir expresándose en este formato, ya que el rango del exponente no puede compensar el desplazamiento a izquierda del punto decimal.

Números Finitos Normalizados

- El rango de éstos números se compone de todos los valores finitos distintos de cero codificables en formato de números reales entre 0 y $\pm\infty$.
- En el formato de punto flotante simple precisión estos números se componen de todos aquellos cuyos exponentes desplazados van de 1 a 254, (no desplazados van de -126 a 127).
- Cuando se aproximan a cero, estos números no pueden seguir expresándose en este formato, ya que el rango del exponente no puede compensar el desplazamiento a izquierda del punto decimal.
- Cuando se llega a un exponente cero en un número normalizado, se pasa al rango de-normalizado.

Números Finitos Normalizados

Números Finitos Normalizados

- En general las operaciones entre números normalizados arrojan como resultado otro número normalizado.

Números Finitos Normalizados

- En general las operaciones entre números normalizados arrojan como resultado otro número normalizado.
- Si hay underflow se pasa a trabajar en formato de-normalizado.

Números Finitos Normalizados

- En general las operaciones entre números normalizados arrojan como resultado otro número normalizado.
- Si hay underflow se pasa a trabajar en formato de-normalizado.
- Para el resultado de una operación que requiere como exponente -129, tenemos el siguiente proceso de denormalización para representarlo con exponente -126 (extremo inferior del rango).

Números Finitos Normalizados

- En general las operaciones entre números normalizados arrojan como resultado otro número normalizado.
- Si hay underflow se pasa a trabajar en formato de-normalizado.
- Para el resultado de una operación que requiere como exponente -129, tenemos el siguiente proceso de denormalización para representarlo con exponente -126 (extremo inferior del rango).

Operation	Sign	Exponent*	Significand
True Result	0	-129	1.0101110000...00
Denormalize	0	-128	0.10101110000...00
Denormalize	0	-127	0.01010111000...00
Denormalize	0	-126	0.00101011100...00
Denormal Result	0	-126	0.00101011100...00

Números Finitos Normalizados

- En general las operaciones entre números normalizados arrojan como resultado otro número normalizado.
- Si hay underflow se pasa a trabajar en formato de-normalizado.
- Para el resultado de una operación que requiere como exponente -129, tenemos el siguiente proceso de denormalización para representarlo con exponente -126 (extremo inferior del rango).

Operation	Sign	Exponent*	Significand
True Result	0	-129	1.0101110000...00
Denormalize	0	-128	0.10101110000...00
Denormalize	0	-127	0.01010111000...00
Denormalize	0	-126	0.00101011100...00
Denormal Result	0	-126	0.00101011100...00

- Observar como se pierden los tres bits menos significativos de la mantisa.

Sobre infinitos y demás “cosas raras”

Sobre infinitos y demás “cosas raras”

- Infinitos signados

Sobre infinitos y demás “cosas raras”

- Infinitos signados

- $-\infty$ y $+\infty$, representan los máximos números reales positivo y negativo representables en formato de punto flotante.

Sobre infinitos y demás “cosas raras”

- Infinitos signados

- $-\infty$ y $+\infty$, representan los máximos números reales positivo y negativo representables en formato de punto flotante.
- La mantisa siempre es 1.000....00, y el máximo exponente desplazado representable (p. Ej. 255 para precisión simple).

Sobre infinitos y demás “cosas raras”

- Infinitos signados

- $-\infty$ y $+\infty$, representan los máximos números reales positivo y negativo representables en formato de punto flotante.
- La mantisa siempre es 1.000....00, y el máximo exponente desplazado representable (p. Ej. 255 para precisión simple).

- NaNs

Sobre infinitos y demás “cosas raras”

- Infinitos signados

- $-\infty$ y $+\infty$, representan los máximos números reales positivo y negativo representables en formato de punto flotante.
- La mantisa siempre es 1.000....00, y el máximo exponente desplazado representable (p. Ej. 255 para precisión simple).

- NaNs

- **NaN = Not a Number.**

Sobre infinitos y demás “cosas raras”

- Infinitos signados

- $-\infty$ y $+\infty$, representan los máximos números reales positivo y negativo representables en formato de punto flotante.
- La mantisa siempre es 1.000....00, y el máximo exponente desplazado representable (p. Ej. 255 para precisión simple).

- NaNs

- NaN = **Not a Number**.
- No son parte del rango de números reales.

Sobre infinitos y demás “cosas raras”

- Infinitos signados

- $-\infty$ y $+\infty$, representan los máximos números reales positivo y negativo representables en formato de punto flotante.
- La mantisa siempre es 1.000....00, y el máximo exponente desplazado representable (p. Ej. 255 para precisión simple).

- NaNs

- NaN = **Not a Number**.
- No son parte del rango de números reales.
- **QNaN**: Quiet NaN tiene el bit mas significativo fraccional seteado. Pueden propagarse por posteriores operaciones sin generar una excepción.

Sobre infinitos y demás “cosas raras”

- Infinitos signados

- $-\infty$ y $+\infty$, representan los máximos números reales positivo y negativo representables en formato de punto flotante.
- La mantisa siempre es 1.000....00, y el máximo exponente desplazado representable (p. Ej. 255 para precisión simple).

- NaNs

- **NaN = Not a Number.**
- No son parte del rango de números reales.
- **QNaN:** Quiet NaN tiene el bit mas significativo fraccional seteado. Pueden propagarse por posteriores operaciones sin generar una excepción.
- **SNaN:** Signaled NaN. Tiene en cero el bit fraccional mas significativo. Resulta de una operación inválida de punto flotante.

Información de control y estado

- Cuando se introducen las extensiones SSE, con el Pentium III, Intel incluye el registro **MXCSR**.

	31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		F Z	R C	P M	U M	O M	Z M	D M	I M	D A Z	P E	U E	O E	Z E	D E	I E		
Flush to Zero																		
Rounding Control																		
Precision Mask																		
Underflow Mask																		
Overflow Mask																		
Divide-by-Zero Mask																		
Denormal Operation Mask																		
Invalid Operation Mask																		
Denormals Are Zeros*																		
Precision Flag																		
Underflow Flag																		
Overflow Flag																		
Divide-by-Zero Flag																		
Denormal Flag																		
Invalid Operation Flag																		

* The denormals-are-zeros flag was introduced in the Pentium 4 and Intel Xeon processor.



Registro MXCSR

Registro MXCSR

- El bit 15 (FZ) del registro **MXCSR** habilita el modo “flush-to-zero”, que controla la respuesta a la condición de underflow de una instrucción SIMD de punto flotante.

Registro MXCSR

- El bit 15 (FZ) del registro **MXCSR** habilita el modo “flush-to-zero”, que controla la respuesta a la condición de underflow de una instrucción SIMD de punto flotante.
- Cuando se enmascara la excepción de underflow excepción y se habilita el modo “flush-to-zero”, el procesador realiza las siguientes operaciones cuando detecta una condición de underflow en punto flotante:

Registro MXCSR

- El bit 15 (FZ) del registro **MXCSR** habilita el modo “flush-to-zero”, que controla la respuesta a la condición de underflow de una instrucción SIMD de punto flotante.
- Cuando se enmascara la excepción de underflow excepción y se habilita el modo “flush-to-zero”, el procesador realiza las siguientes operaciones cuando detecta una condición de underflow en punto flotante:
 - Retorna cero con el signo del resultado correcto.

Registro MXCSR

- El bit 15 (FZ) del registro **MXCSR** habilita el modo “flush-to-zero”, que controla la respuesta a la condición de underflow de una instrucción SIMD de punto flotante.
- Cuando se enmascara la excepción de underflow excepción y se habilita el modo “flush-to-zero”, el procesador realiza las siguientes operaciones cuando detecta una condición de underflow en punto flotante:
 - Retorna cero con el signo del resultado correcto.
 - Pone en '1' los flags de las excepciones de precisión y de underflow.

Registro MXCSR

Registro MXCSR

- Si no se enmascara la excepción de underflow, se ignora el bit “flush-to-zero”. El modo “flush-to-zero” no es compatible con el Standard IEEE 754.

Registro MXCSR

- Si no se enmascara la excepción de underflow, se ignora el bit “flush-to-zero”. El modo “flush-to-zero” no es compatible con el Standard IEEE 754.
- La respuesta indicada por el IEEE a un underflow es entregar el resultado denormalizado.

Registro MXCSR

- Si no se enmascara la excepción de underflow, se ignora el bit “flush-to-zero”. El modo “flush-to-zero” no es compatible con el Standard IEEE 754.
- La respuesta indicada por el IEEE a un underflow es entregar el resultado denormalizado.
- El modo “flush-to-zero” se provee en principio por razones de performance.

Registro MXCSR

- Si no se enmascara la excepción de underflow, se ignora el bit “flush-to-zero”. El modo “flush-to-zero” no es compatible con el Standard IEEE 754.
- La respuesta indicada por el IEEE a un underflow es entregar el resultado denormalizado.
- El modo “flush-to-zero” se provee en principio por razones de performance.
- Al costo de alguna pérdida de precisión, puede obtenerse una ejecución mas rápida para aplicaciones en donde la condición de underflow es común y es tolerable el redondeo a cero cuando se produce el underflow.

Registro MXCSR

- Si no se enmascara la excepción de underflow, se ignora el bit “flush-to-zero”. El modo “flush-to-zero” no es compatible con el Standard IEEE 754.
- La respuesta indicada por el IEEE a un underflow es entregar el resultado denormalizado.
- El modo “flush-to-zero” se provee en principio por razones de performance.
- Al costo de alguna pérdida de precisión, puede obtenerse una ejecución mas rápida para aplicaciones en donde la condición de underflow es común y es tolerable el redondeo a cero cuando se produce el underflow.
- El bit “flush-to-zero” se limpia durante el arranque o en el reset del procesador, y se deshabilita el modo “flush-to-zero”.

- 1 Fundamentos
- 2 Procesamiento de Señales digitales
- 3 Modelo de ejecución SIMD
- 4 Implementaciones SIMD en x86
 - Arquitectura completa
 - Números Reales
 - Formatos de Punto Fijo
 - Formatos de Punto Flotante
 - Codificación de Números Reales
 - **Extensiones AVX**
- 5 Instrucciones

Operando con vectores de 256 bits de ancho

Operando con vectores de 256 bits de ancho

- Soporta vectores de 256 bits de ancho con un set de registros denominados **YMM**.

Operando con vectores de 256 bits de ancho

- Soporta vectores de 256 bits de ancho con un set de registros denominados **YMM**.
- Acelera hasta 2X las instrucciones de punto flotante de 256 bits, respecto de las mismas instrucciones de 128 bits SSE.

Operando con vectores de 256 bits de ancho

- Soporta vectores de 256 bits de ancho con un set de registros denominados **YMM**.
- Acelera hasta 2X las instrucciones de punto flotante de 256 bits, respecto de las mismas instrucciones de 128 bits SSE.
- Mejora las instrucciones SSE legacy de 128 bits permitiendo trabajar con tres operandos y simplifica la vectorización de expresiones en compiladores de lenguajes de alto nivel.

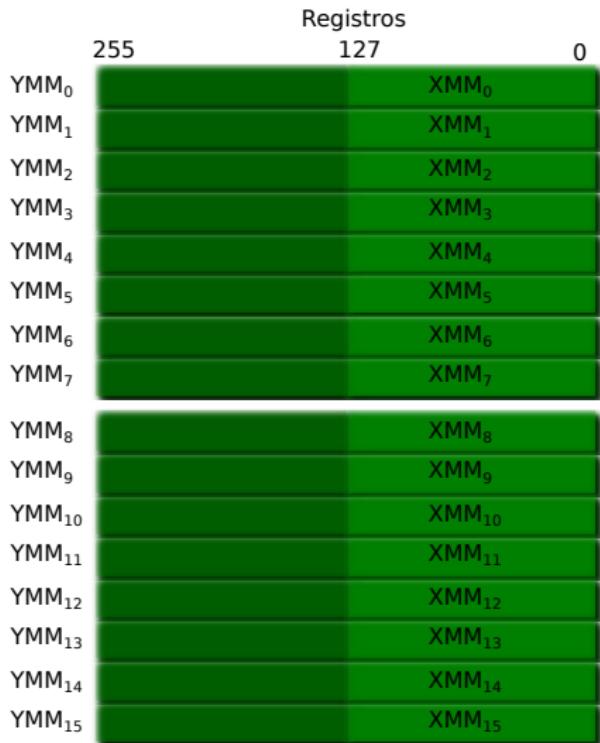
Operando con vectores de 256 bits de ancho

- Soporta vectores de 256 bits de ancho con un set de registros denominados **YMM**.
- Acelera hasta 2X las instrucciones de punto flotante de 256 bits, respecto de las mismas instrucciones de 128 bits SSE.
- Mejora las instrucciones SSE legacy de 128 bits permitiendo trabajar con tres operandos y simplifica la vectorización de expresiones en compiladores de lenguajes de alto nivel.
- El prefijo **VEX** soporta operaciones generales de tres operandos lo que mejora la sintaxis haciéndola mas flexible para codificar las nuevas instrucciones.

Operando con vectores de 256 bits de ancho

- Soporta vectores de 256 bits de ancho con un set de registros denominados **YMM**.
- Acelera hasta 2X las instrucciones de punto flotante de 256 bits, respecto de las mismas instrucciones de 128 bits SSE.
- Mejora las instrucciones SSE legacy de 128 bits permitiendo trabajar con tres operandos y simplifica la vectorización de expresiones en compiladores de lenguajes de alto nivel.
- El prefijo **VEX** soporta operaciones generales de tres operandos lo que mejora la sintaxis haciéndola mas flexible para codificar las nuevas instrucciones.
- La mayoría de las instrucciones **AVX** de 128 y 256 bits codificadas con el prefijo **VEX** (tanto de carga como de almacenamiento en memoria) no estás restringidas a alineación a 16 o 32 bytes.

Registros YMM



Registros YMM

- Las extensiones **AVX** introducen una extensión en el tamaño de los registros SIMD desde 128 bits a 256 bits. Los registros **XMM** de 128 bits, pasan a ser la parte baja del registro **YMM** correspondiente.

	255	Registros	127	0
YMM ₀			XMM ₀	
YMM ₁			XMM ₁	
YMM ₂			XMM ₂	
YMM ₃			XMM ₃	
YMM ₄			XMM ₄	
YMM ₅			XMM ₅	
YMM ₆			XMM ₆	
YMM ₇			XMM ₇	
YMM ₈			XMM ₈	
YMM ₉			XMM ₉	
YMM ₁₀			XMM ₁₀	
YMM ₁₁			XMM ₁₁	
YMM ₁₂			XMM ₁₂	
YMM ₁₃			XMM ₁₃	
YMM ₁₄			XMM ₁₄	
YMM ₁₅			XMM ₁₅	

Registros YMM

- Las extensiones **AVX** introducen una extensión en el tamaño de los registros SIMD desde 128 bits a 256 bits. Los registros **XMM** de 128 bits, pasan a ser la parte baja del registro **YMM** correspondiente.

- Por lo demás soporta las mismas instrucciones solo que se extiende a tres y hasta cuatro operandos.

	Registros
YMM ₀	XMM ₀
YMM ₁	XMM ₁
YMM ₂	XMM ₂
YMM ₃	XMM ₃
YMM ₄	XMM ₄
YMM ₅	XMM ₅
YMM ₆	XMM ₆
YMM ₇	XMM ₇
YMM ₈	XMM ₈
YMM ₉	XMM ₉
YMM ₁₀	XMM ₁₀
YMM ₁₁	XMM ₁₁
YMM ₁₂	XMM ₁₂
YMM ₁₃	XMM ₁₃
YMM ₁₄	XMM ₁₄
YMM ₁₅	XMM ₁₅

1 Fundamentos

2 Procesamiento de Señales digitales

3 Modelo de ejecución SIMD

4 Implementaciones SIMD en x86

5 Instrucciones

- Transferencias (las mas comunes)
- Aritmética en algoritmos DSP
- Instrucciones de punto flotante
- Instrucciones para manejo de enteros para SSEn
- Instrucciones para manejo de cacheabilidad

Instrucciones de movimiento

- MOVD Mover un valor de 32 bits a la parte baja de un operando de 64 o 128 bits
- MOVDQ Mover un valor de 64 bits a la parte baja de un operando de 128 bits.
- MOVDQ2Q Mover un valor de 64 bits a un registro MMX desde la parte baja de un registro XMM.
- MOVDQA Mover un valor de 64 bits **alineado**.
- MOVDQU Mover un valor de 64 bits **desalineado**.
- MOVAPD Mover 128 bits de Punto Flotante Doble Precisión empaquetados **alineados**.
- MOVAPS Mover 128 bits de Punto Flotante Simple Precisión empaquetados **desalineados**.

Ejemplos de Instrucciones de movimiento

```
1 ; /////////////////////////////////
2     movd    mm4, ebx          ;mm4[31-0] <- ebx
3     movd    xmm1, dword [esi] ;xmm1[31-0] <- [esi]
4     movq    xmm8, r9          ;xmm8[63-0] <- r9
5     movq    qword [ebp+12], xmm0; [ebp+12] <- xmm0 [63-0]
6     movdq2q mm5,xmm12        ;mm5 <- xmm12 [63-0]
7     movdqa  [esi], xmm5       ;Si esi no contiene un valor
8                           ;múltiplo de 16, entonces la
9                           ;variable no está alineada.
10                          ;Se genera una excepción #GP
11     movdqu xmm9,dqword[rbx]  Si rbx contiene un valor
12                           ;múltiplo de 16, entonces la
13                           ;variable está alineada.
14                           ;Se genera una excepción #GP
```

1 Fundamentos

2 Procesamiento de Señales digitales

3 Modelo de ejecución SIMD

4 Implementaciones SIMD en x86

5 Instrucciones

- Transferencias (las mas comunes)
- **Aritmética en algoritmos DSP**
- Instrucciones de punto flotante
- Instrucciones para manejo de enteros para SSEn
- Instrucciones para manejo de cacheabilidad

Acumulación de resultados sobre un registro

Acumulación de resultados sobre un registro

- A medida que acumulamos resultados en un registro, llegamos inexorablemente al punto en el que el rango del resultado excede la capacidad de bits del registro de acumulación.

Acumulación de resultados sobre un registro

- A medida que acumulamos resultados en un registro, llegamos inexorablemente al punto en el que el rango del resultado excede la capacidad de bits del registro de acumulación.
- En este punto los procesadores convencionales indican la situación mediante un flag de overflow, y en el operando destino se almacena el valor de **desborde**.

Acumulación de resultados sobre un registro

- A medida que acumulamos resultados en un registro, llegamos inexorablemente al punto en el que el rango del resultado excede la capacidad de bits del registro de acumulación.
- En este punto los procesadores convencionales indican la situación mediante un flag de overflow, y en el operando destino se almacena el valor de **desborde**.
- La aplicación chequea dentro del lazo de cálculo este flag y de acuerdo a su estado decide si sigue la acumulación.

Acumulación de resultados sobre un registro

- A medida que acumulamos resultados en un registro, llegamos inexorablemente al punto en el que el rango del resultado excede la capacidad de bits del registro de acumulación.
- En este punto los procesadores convencionales indican la situación mediante un flag de overflow, y en el operando destino se almacena el valor de **desborde**.
- La aplicación chequea dentro del lazo de cálculo este flag y de acuerdo a su estado decide si sigue la acumulación.
- El costo de esta práctica de programación es tiempo de ejecución para analizar como tratar cada resultado parcial.

Manejo de los desbordes

Manejo de los desbordes

Aritmética de Desborde Del comportamiento planteado en el slide anterior, proviene el nombre que se le ha asignado a la forma de aritmética empleada por los procesadores convencionales: Al llegar al extremo superior del rango realizan una operación de que se denomina wraparound y consiste en resetear el contador al valor inicial y setear el flag de overflow). Si por el contrario se sustrae un registro al llegar a cero, si se pide una nueva sustracción se pasa al valor máximo y se sigue desde allí. El costo es comprobar en cada ciclo de un lazo el valor del flag para decidir si se continúa en el lazo o no.

Manejo de los desbordes

Aritmética de Desborde Del comportamiento planteado en el slide anterior, proviene el nombre que se le ha asignado a la forma de aritmética empleada por los procesadores convencionales: Al llegar al extremo superior del rango realizan una operación de que se denomina wraparound y consiste en resetear el contador al valor inicial y setear el flag de overflow). Si por el contrario se sustrae un registro al llegar a cero, si se pide una nueva sustracción se pasa al valor máximo y se sigue desde allí. El costo es comprobar en cada ciclo de un lazo el valor del flag para decidir si se continúa en el lazo o no.

Aritmética Saturada Al producirse una condición fuera de rango el operando destino mantiene el máximo o mínimo valor del rango (dependiendo si la condición se ha producido por exceso o por defecto).

Aritmética Saturada

Tipo de Dato	Límite Inferior		Límite Superior	
	Hexadecimal	Decimal	Hexadecimal	Decimal
byte signado	0x80	-128	0x7F	127
word signada	0x8000	-32768	0x7FFF	32767

Cuadro: Aritmética Saturada Signada

Tipo de Dato	Límite Inferior		Límite Superior	
	Hexadecimal	Decimal	Hexadecimal	Decimal
byte no signado	0x00	0	0xFF	255
word no signada	0x0000	0	0xFFFF	65535

Cuadro: Aritmética Saturada No Signada

Aritmética Saturada vs. Aritmética de Desborde

- Consideremos la siguiente suma empaquetada de 8 bytes.
- En el byte 6 puede observarse la diferencia entre ambas operaciones

Byte	7	6	5	4	3	2	1	0
oper 1	0x4D	0x23	0x9F	0xC0	0x11	0x4A	0x29	0x0B
oper 2	0x32	0xFF	0x1A	0x0D	0x3F	0xAF	0xB0	0x36
desb.	0x7F	0x22	0xB9	0xCD	0x50	0xF9	0xD9	0x41
sat.	0x7F	0xFF	0xB9	0xCD	0x50	0xF9	0xD9	0x41

Cuadro: Suma Empaquetada **saturada** vs. suma empaquetada **de desborde**

Cuando usar aritmética saturada o de desborde

Cuando usar aritmética saturada o de desborde

- En algunas aplicaciones la aritmética saturada provee soluciones más eficaces que la aritmética de desborde.

Cuando usar aritmética saturada o de desborde

- En algunas aplicaciones la aritmética saturada provee soluciones más eficaces que la aritmética de desborde.
- Al procesar video, cuando un nivel de negro satura, no tiene sentido seguir procesando la variable ya que no produce mas efecto sobre la salida.

Cuando usar aritmética saturada o de desborde

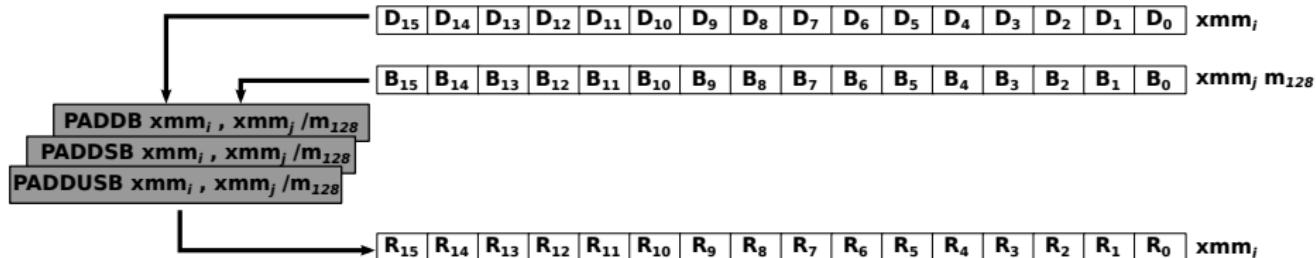
- En algunas aplicaciones la aritmética saturada provee soluciones más eficaces que la aritmética de desborde.
- Al procesar video, cuando un nivel de negro satura, no tiene sentido seguir procesando la variable ya que no produce mas efecto sobre la salida.
- Si utilizamos la habitual lógica de desborde, el valor remanente en el registro de resultado al saturar el negro nos lleva de vuelta al blanco, afectando el flag de overflow para prevenirnos de la situación, pero el número almacenado en el operando del resultado es mucho menor al máximo del rango.

Instrucciones de Aritmética entera

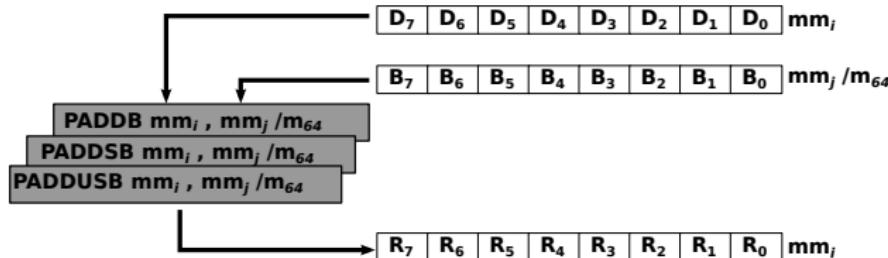
Aritmética	Desborde				Saturada Signada		Saturada No signada	
Operación	byte	word	dobleWord	QuadWord	byte	word	byte	word
Suma	PADDB	PADDW	PADDD	PADDQ	PADD\$B	PADD\$W	PADD\$UB	PADD\$UW
Resta	PSUBB	PSUBW	PSUBD	PSUBQ	PSUB\$B	PSUB\$W	PSUB\$UB	PSUB\$UW
Producto Enteros Signados		PMULLW PMULHW						
Producto Enteros No Signados		PMULLUW PMULHUW						
Producto Doble Words			PMULDQ PMU-LUDQ					
Suma de Prod.	PMADDUBSW PMADDWD							

Cuadro: Operaciones aritméticas

Suma empaquetada de bytes

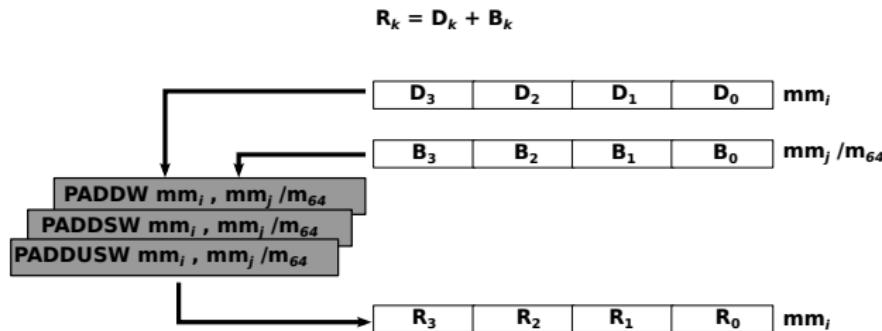
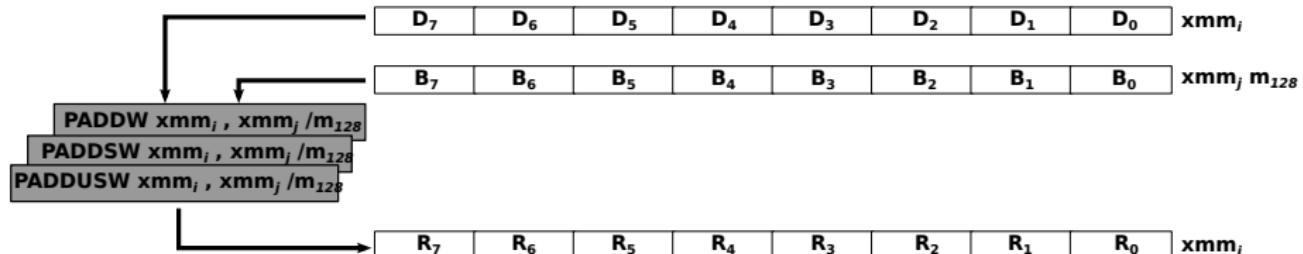


$$R_k = D_k + B_k$$

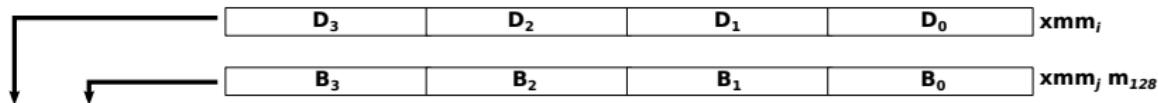


$$R_k = D_k + B_k$$

Suma empaquetada de words



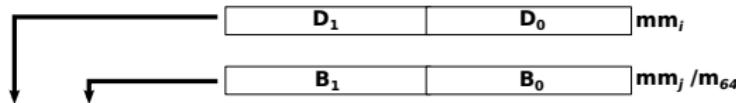
Suma empaquetada de double words



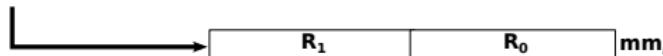
PADDD $xmm_i, xmm_j / m_{128}$



$$R_k = D_k + B_k$$

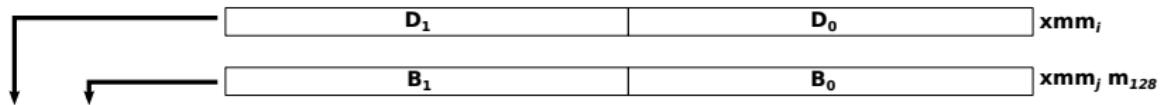


PADDD $mm_i, mm_j / m_{64}$



$$R_k = D_k + B_k$$

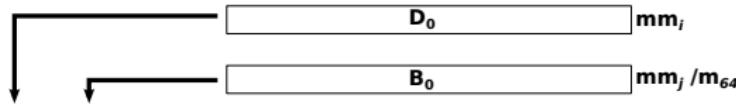
Suma empaquetada de quad words



PADDQ xmm_i , xmm_j /m₁₂₈



$$R_k = D_k + B_k$$

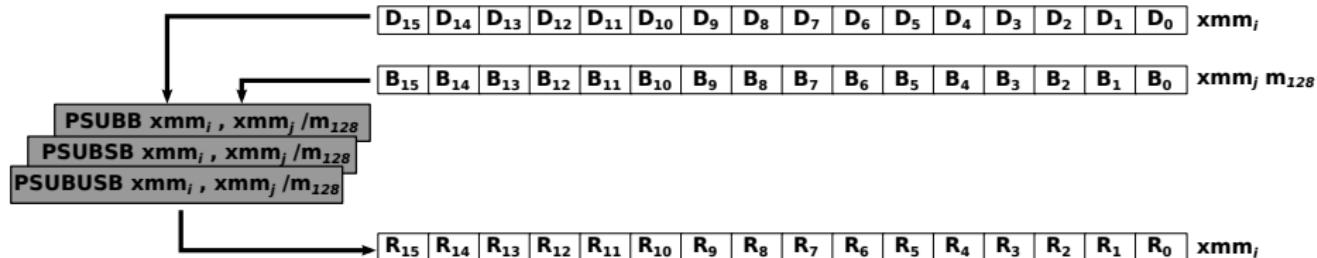


PADDQ mm_i , mm_j /m₆₄

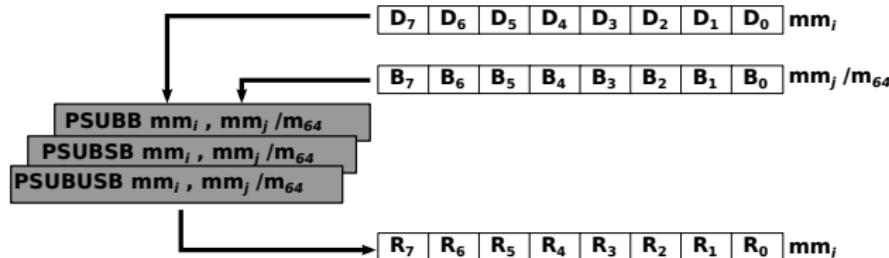


$$R_k = D_k + B_k$$

Resta empaquetada de bytes

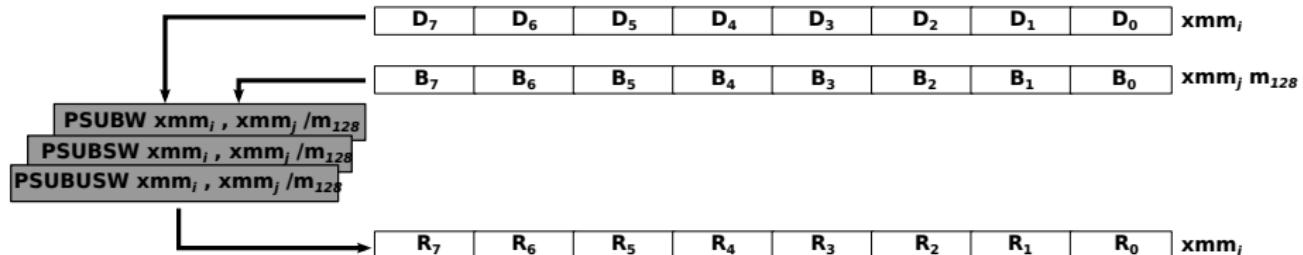


$$R_k = D_k - B_k$$

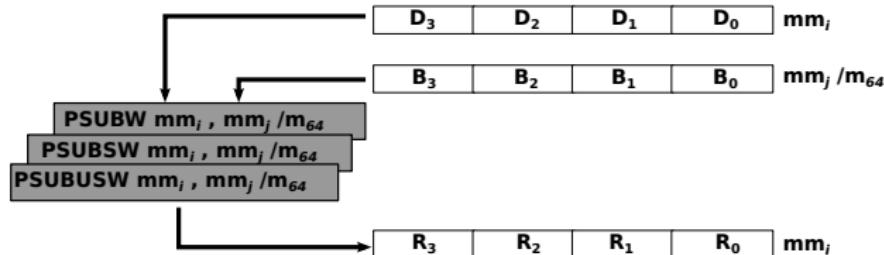


$$R_k = D_k - B_k$$

Resta empaquetada de words

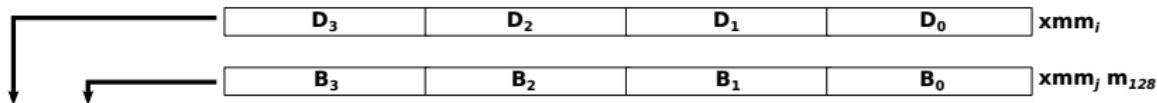


$$R_k = D_k - B_k$$



$$R_k = D_k - B_k$$

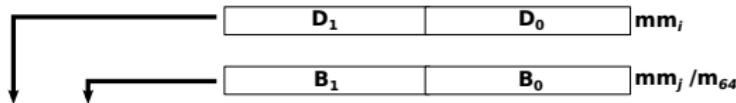
Resta empaquetada de double words



PSUBD xmm_i, xmm_j /m₁₂₈



$$R_k = D_k - B_k$$

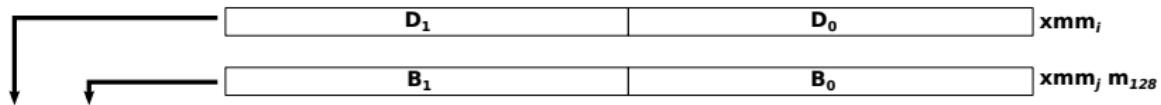


PSUBD mm_i, mm_j /m₆₄

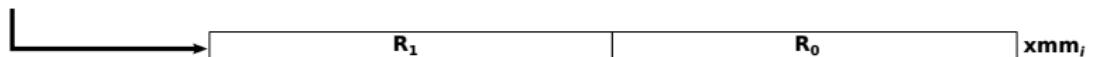


$$R_k = D_k - B_k$$

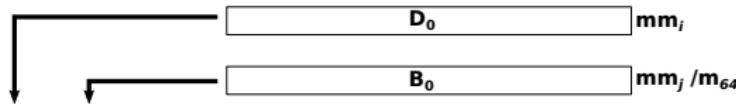
Resta empaquetada de quad words



PSUBQ xmm_i, xmm_j / m₁₂₈



$$R_k = D_k - B_k$$

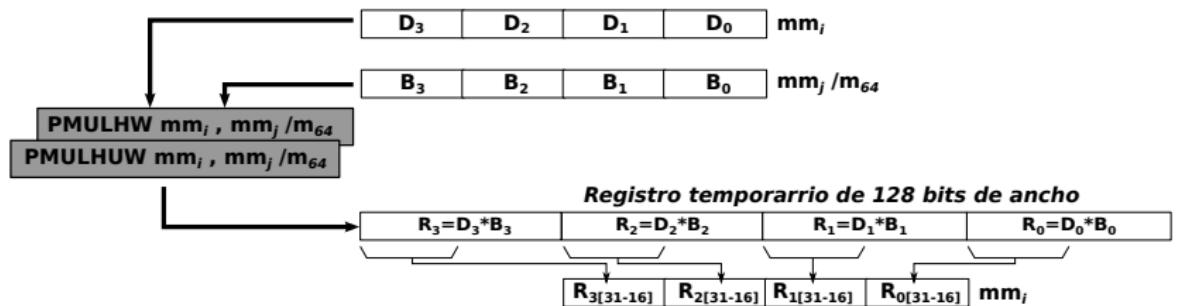
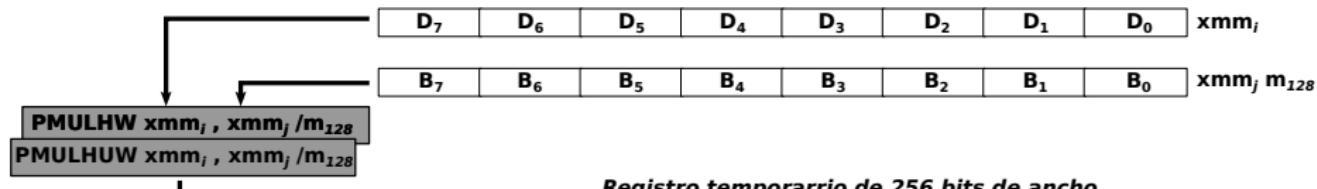


PSUBQ mm_i, mm_j / m₆₄

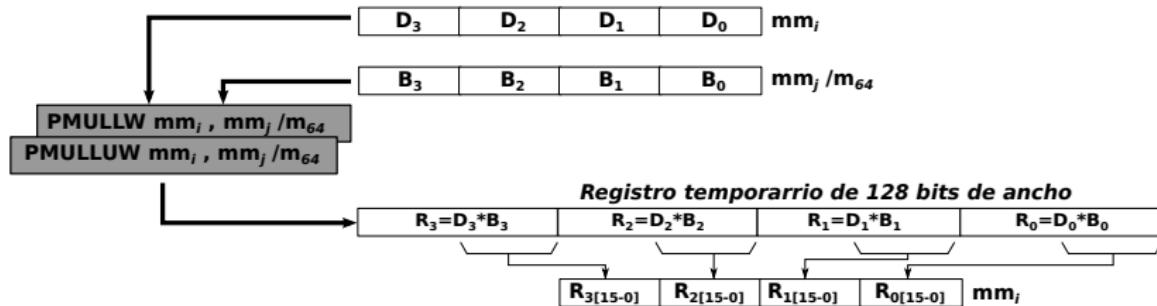
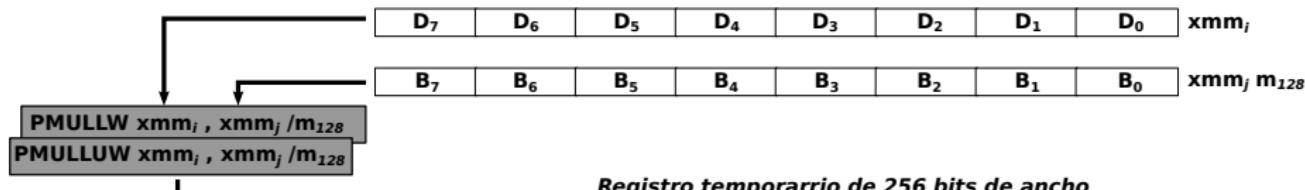


$$R_k = D_k - B_k$$

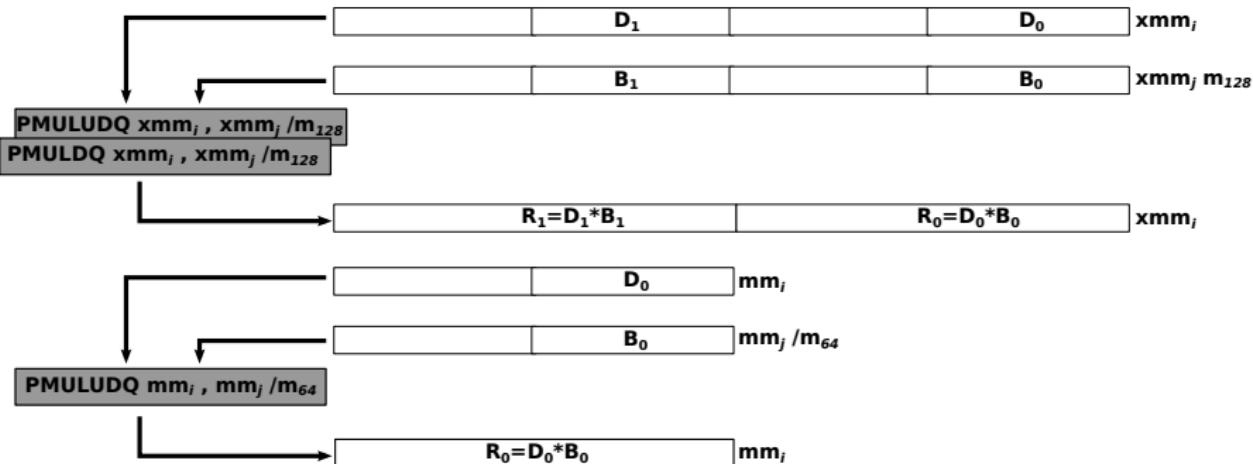
Multiplicación de words empaquetada



Multiplicación de words empaquetada

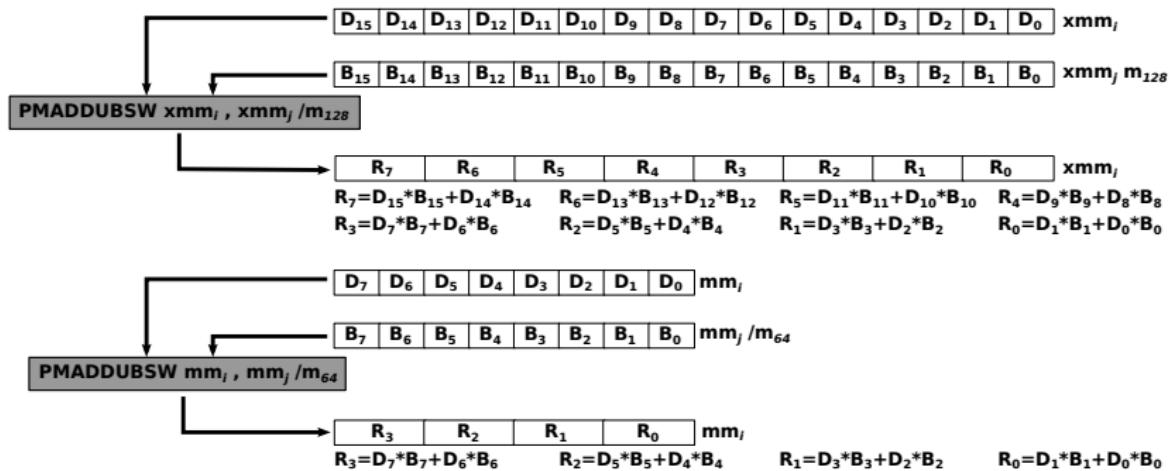


Multiplicación de doble words empaquetada



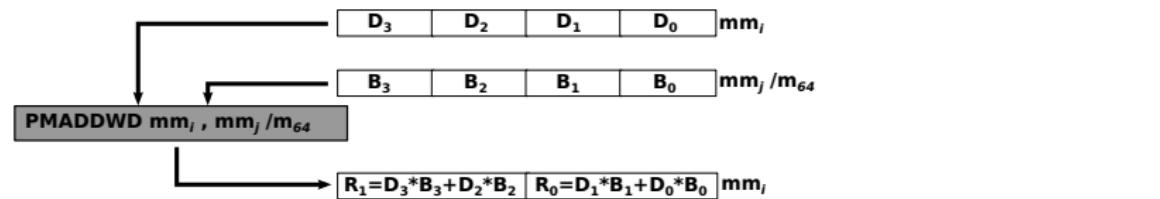
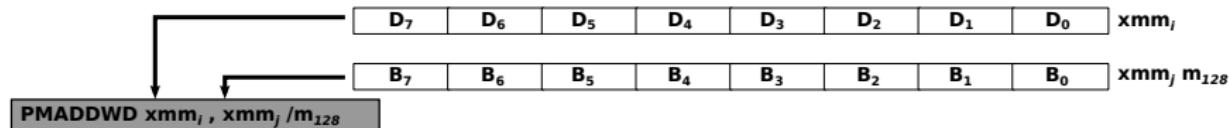
Suma de productos empaquetada

- El primer operando contiene bytes enteros no signados
- El segundo operando contiene bytes signados empaquetados
- Almacena en el operando destino words signadas obtenidas a partir de la suma saturada de los productos parciales



Suma de productos empaquetada

- Ambos operandos contienen words enteros signadas
- Almacena en el operando destino doble words signadas obtenidas a partir de la suma de los productos parciales



Aplicación: Diseño de un filtro FIR

- Un filtro FIR (por **F**inite **I**mpulse **R**esponse) son uno de los principales algoritmos de filtrado utilizados en DSP.
- La salida de un filtro de orden m viene dado por la siguiente expresión:

$$y[j] = \sum_{i=0}^m c_i \cdot x[j - i]$$

- El siguiente fragmento de código calcula 640 valores de salida de un filtro FIR de orden 63:

```
1 | for ( j = 0; j < 640; j++) {  
2 |   int s = 0; // s = acumulador  
3 |   for ( i = 0; i <= 63; i++)  
4 |     s += c[i] * x[i+j]; // x[] = valores de muestras de entrada  
5 |                           // c[] = coeficientes del filtro  
6 |   y[j] = s;           // y[] = Valores de salida  
7 }
```

- $x[]$ mantiene siempre la muestra actual en la entrada mas las 639 anteriores

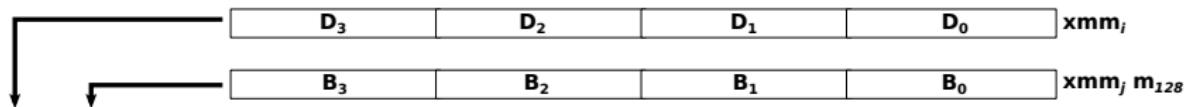
Aplicación: Diseño de un filtro FIR

- Escrito en SSE queda como a continuación:

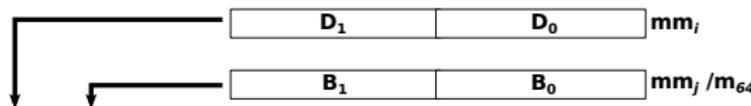
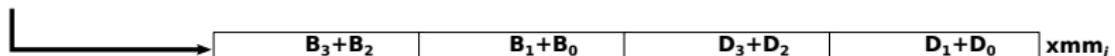
```
1 Lazo1:  
2     pxor xmm0, xmm0 ; inicializa en 0 4 acumuladores (c/u de 32 bits)  
3     sub ebp, ebp      ; inicializa Indice al buffer de coeficientes en 0  
4 Lazo2:  
5     movups xmm1, qword ptr [esi+ebp]; carga vector de 8 elementos en reg  
6     pmaddwd xmm1, qword ptr [edi+ebp]; .. multiplica de a paresmuestras y  
7     paddd xmm0, xmm1           ; acumula resultados en xmm0 (4 x 32)  
8     add ebp, 2*8    ;2 bytes = tama o del dato , 8 = cantidad de datos  
9     cmp ebp, 2*64   ; repite para los 64 coeficientes (tomados de a 8)  
10    jnz Lazo2  
11    phaddd xmm0, xmm0  
12    phaddd xmm0, xmm0 ;acumula cuatro valores parciales en un resultado  
13    psrad xmm0, 15    ;Lo escala a 16 Bits (tama o original del dato  
14    movd qword ptr [eax], xmm0 ; lo almacena (solo quedan los 16 bits men  
15    add esi, 2        ;incrementa el puntero al buffer de muestras  
16    add eax, 2        ;incrementa puntero al buffer de salida (Va a pisar lo  
17    sub ecx, 1        ;repeat for all input samples
```

- La relación de aceleración de este algoritmo respecto del C visto anteriormente es aproximadamente 7x.

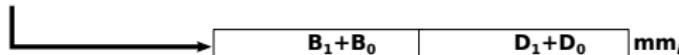
Como funciona la suma horizontal del código anterior



PHADDD xmm_i , xmm_j / m₁₂₈



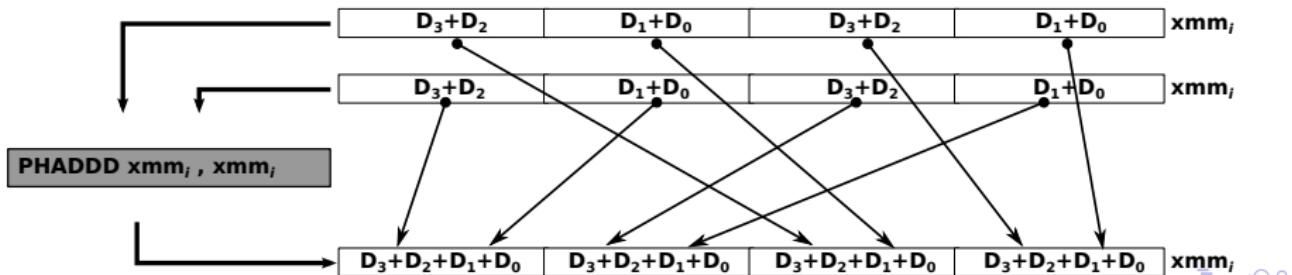
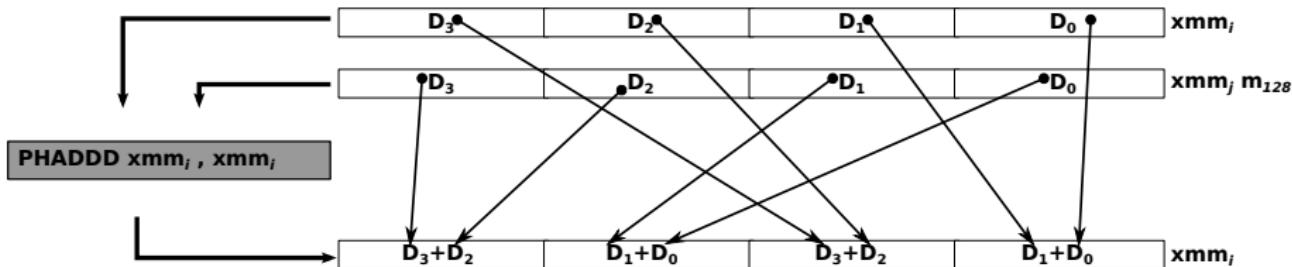
PHADDD mm_i , mm_j / m₆₄



- Consultar **PHADDW** (es idéntica para words), y **PHADDSW**, igual que PHADDW pero suma saturada

Como funciona la suma horizontal del código anterior

- En el caso del algoritmo se la usa dos veces consecutivas y utilizando el mismo registro en ambos operandos, con el siguiente resultado:



Consideraciones de implementación

Consideraciones de implementación

- En general los filtros trabajan con valores fraccionarios para los coeficientes.

Consideraciones de implementación

- En general los filtros trabajan con valores fraccionarios para los coeficientes.
- Sin embargo como su rango dinámico nos es muy grande por lo cual considerando datos de 16 bits puede trabajarse en punto fijo cuyo resultado matemáticamente es igual que trabajar con enteros afectados por un factor de escala definido por la posición del punto decimal dentro de los 16 bits. Es decir, trabajamos con operaciones de aritmética entera pero tratamos los datos como fraccionarios de punto fijo.

Consideraciones de implementación

- En general los filtros trabajan con valores fraccionarios para los coeficientes.
- Sin embargo como su rango dinámico nos es muy grande por lo cual considerando datos de 16 bits puede trabajarse en punto fijo cuyo resultado matemáticamente es igual que trabajar con enteros afectados por un factor de escala definido por la posición del punto decimal dentro de los 16 bits. Es decir, trabajamos con operaciones de aritmética entera pero tratamos los datos como fraccionarios de punto fijo.
- Como el rango dinámico de los coeficientes es numéricamente pequeños de modo que es de esperar que el resultado a enviar a la salida tenga una valor dentro del rango de la entrada. Por eso cada valor de salida se puede escalar a 16 bits al final de su cálculo.

Instrucciones Lógicas

Instrucciones Lógicas

- Tratan a los contenidos de los registros **MMX**, y **XMM** como datos puros de 64 y 128 bits respectivamente.

Instrucciones Lógicas

- Tratan a los contenidos de los registros **MMX**, y **XMM** como datos puros de 64 y 128 bits respectivamente.
- No tienen en cuenta ningún tipo de empaquetado de modo que resultan mas similares a las instrucciones correspondientes al modelo de registros de propósito general.

Instrucciones Lógicas

- Tratan a los contenidos de los registros **MMX**, y **XMM** como datos puros de 64 y 128 bits respectivamente.
- No tienen en cuenta ningún tipo de empaquetado de modo que resultan mas similares a las instrucciones correspondientes al modelo de registros de propósito general.
- Son cuatro instrucciones **PAND**, **PNAND**, **POR**, y **PXOR**.

Instrucciones Lógicas

- Tratan a los contenidos de los registros **MMX**, y **XMM** como datos puros de 64 y 128 bits respectivamente.
- No tienen en cuenta ningún tipo de empaquetado de modo que resultan mas similares a las instrucciones correspondientes al modelo de registros de propósito general.
- Son cuatro instrucciones **PAND**, **PNAND**, **POR**, y **PXOR**.
- La P inicial las identifica como instrucciones específicas para este tipo de registros.

Instrucciones Lógicas

- Tratan a los contenidos de los registros **MMX**, y **XMM** como datos puros de 64 y 128 bits respectivamente.
- No tienen en cuenta ningún tipo de empaquetado de modo que resultan mas similares a las instrucciones correspondientes al modelo de registros de propósito general.
- Son cuatro instrucciones **PAND**, **PNAND**, **POR**, y **PXOR**.
- La P inicial las identifica como instrucciones específicas para este tipo de registros.
- El operando fuente pueden ser cualquier registro **MMX**, **XMM** o cualquier dirección de memoria de 64 o 128 bits de ancho.

Instrucciones Lógicas

- Tratan a los contenidos de los registros **MMX**, y **XMM** como datos puros de 64 y 128 bits respectivamente.
- No tienen en cuenta ningún tipo de empaquetado de modo que resultan mas similares a las instrucciones correspondientes al modelo de registros de propósito general.
- Son cuatro instrucciones **PAND**, **PNAND**, **POR**, y **PXOR**.
- La P inicial las identifica como instrucciones específicas para este tipo de registros.
- El operando fuente pueden ser cualquier registro **MMX**, **XMM** o cualquier dirección de memoria de 64 o 128 bits de ancho.
- El operando destino solo puede ser un registro **MMX** o **XMM**.

Instrucciones de Desplazamiento

Instrucciones de Desplazamiento

- Desplaza a derecha o izquierda en la cantidad de bits que se indica cada operando empaquetado dentro del registro o dirección de memoria que se indica.

Instrucciones de Desplazamiento

- Desplaza a derecha o izquierda en la cantidad de bits que se indica cada operando empaquetado dentro del registro o dirección de memoria que se indica.
- **PSLLW, PSLLD, PSLLQ, y PSSLDQ** desplazan en forma lógica a izquierda (Packed Shift Left Logical) los respectivos operandos empaquetados (Words, Doble words, Quad Words o DobleQuad- Words).

Instrucciones de Desplazamiento

- Desplaza a derecha o izquierda en la cantidad de bits que se indica cada operando empaquetado dentro del registro o dirección de memoria que se indica.
- **PSLLW, PSLLD, PSLLQ, y PSSLDQ** desplazan en forma lógica a izquierda (Packed Shift Left Logical) los respectivos operandos empaquetados (Words, Doble words, Quad Words o DobleQuad- Words).
- **PSRLW, PSRLD, PSRLQ, PSRLDQ** realizan lo propio pero desplazando a la derecha (Packed Shift Right Logical).

Instrucciones de Desplazamiento

- Desplaza a derecha o izquierda en la cantidad de bits que se indica cada operando empaquetado dentro del registro o dirección de memoria que se indica.
- **PSLLW**, **PSLLD**, **PSLLO**, y **PSSLDQ** desplazan en forma lógica a izquierda (Packed Shift Left Logical) los respectivos operandos empaquetados (Words, Doble words, Quad Words o DobleQuad- Words).
- **PSRLW**, **PSRLD**, **PSRLQ**, **PSRLDQ** realizan lo propio pero desplazando a la derecha (Packed Shift Right Logical).
- **PSRAW** y **PSRAD** realizan un desplazamiento a derecha aritmético (conservan el signo de cada operando empaquetado). Solo se dispone de instrucciones de Word y doble words empaquetadas.

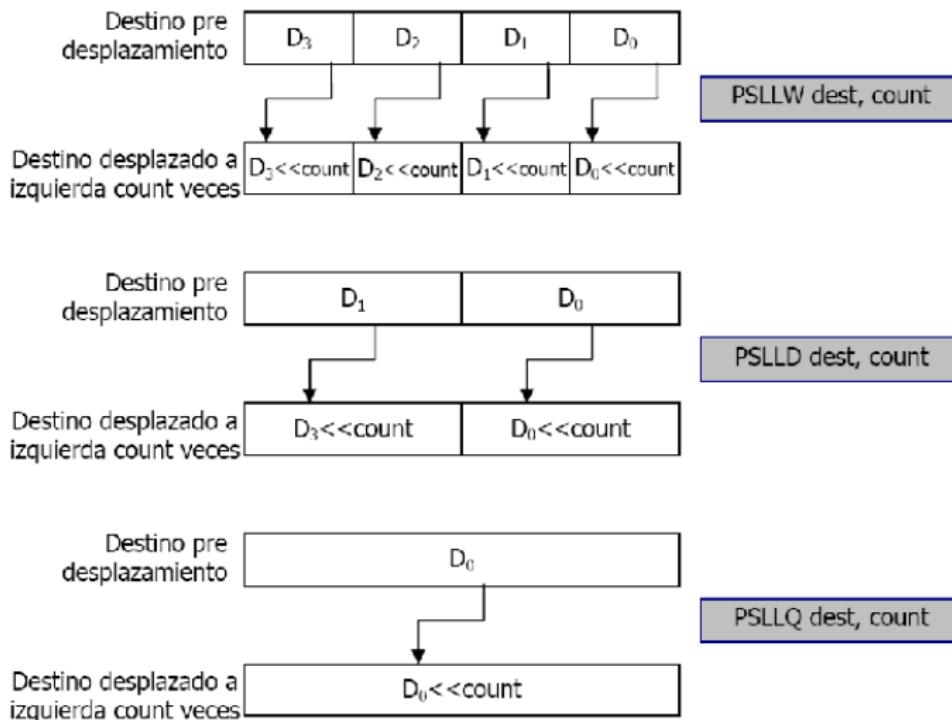
Instrucciones de Desplazamiento

- Desplaza a derecha o izquierda en la cantidad de bits que se indica cada operando empaquetado dentro del registro o dirección de memoria que se indica.
- **PSLLW**, **PSLLD**, **PSLLO**, y **PSSLDQ** desplazan en forma lógica a izquierda (Packed Shift Left Logical) los respectivos operandos empaquetados (Words, Doble words, Quad Words o DobleQuad- Words).
- **PSRLW**, **PSRLD**, **PSRLQ**, **PSRLDQ** realizan lo propio pero desplazando a la derecha (Packed Shift Right Logical).
- **PSRAW** y **PSRAD** realizan un desplazamiento a derecha aritmético (conservan el signo de cada operando empaquetado). Solo se dispone de instrucciones de Word y doble words empaquetadas.
- **PSSLDQ** y **PSRLDQ** trabajan solo con operandos de 128 bits como es lógico.

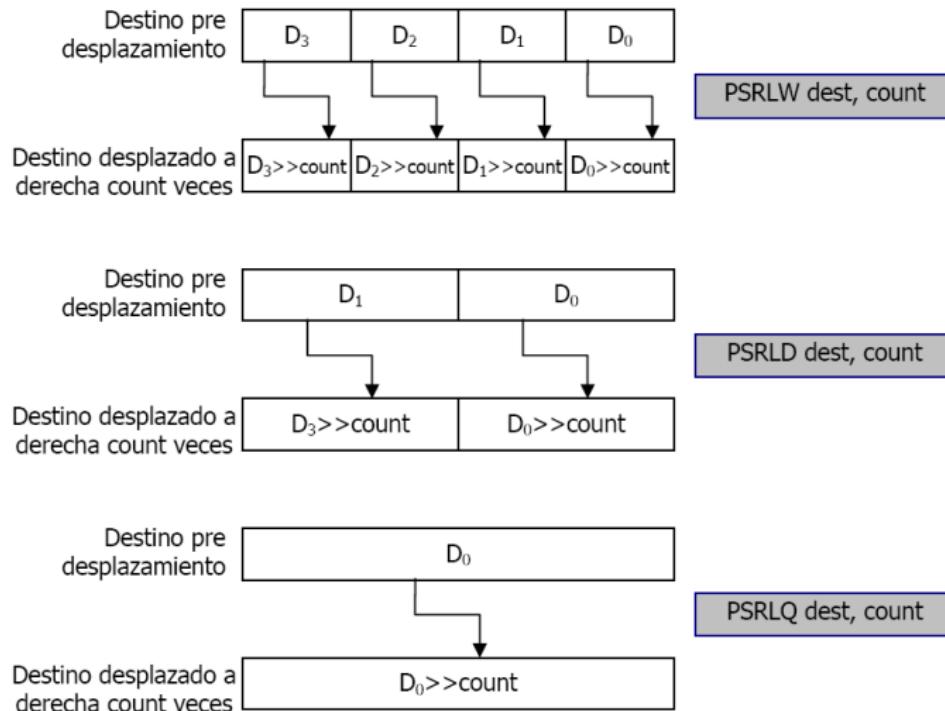
Instrucciones de Desplazamiento

- Desplaza a derecha o izquierda en la cantidad de bits que se indica cada operando empaquetado dentro del registro o dirección de memoria que se indica.
- **PSLLW**, **PSLLD**, **PSLLO**, y **PSSLDQ** desplazan en forma lógica a izquierda (Packed Shift Left Logical) los respectivos operandos empaquetados (Words, Doble words, Quad Words o DobleQuad- Words).
- **PSRLW**, **PSRLD**, **PSRLQ**, **PSRLDQ** realizan lo propio pero desplazando a la derecha (Packed Shift Right Logical).
- **PSRAW** y **PSRAD** realizan un desplazamiento a derecha aritmético (conservan el signo de cada operando empaquetado). Solo se dispone de instrucciones de Word y doble words empaquetadas.
- **PSSLDQ** y **PSRLDQ** trabajan solo con operandos de 128 bits como es lógico.
- El resto puede trabajar con operandos (registro o memoria) de 64 o 128 bits.

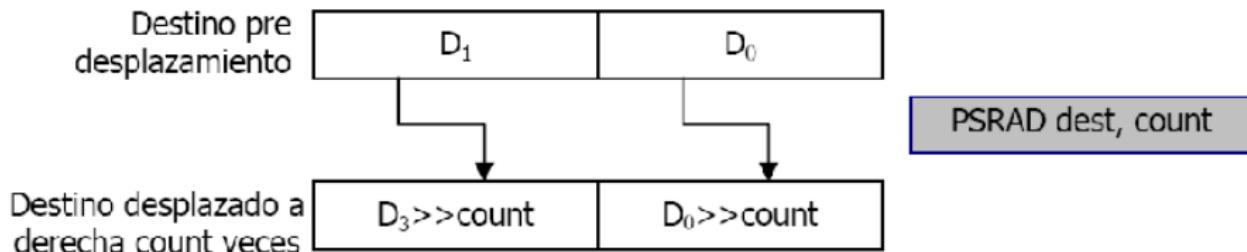
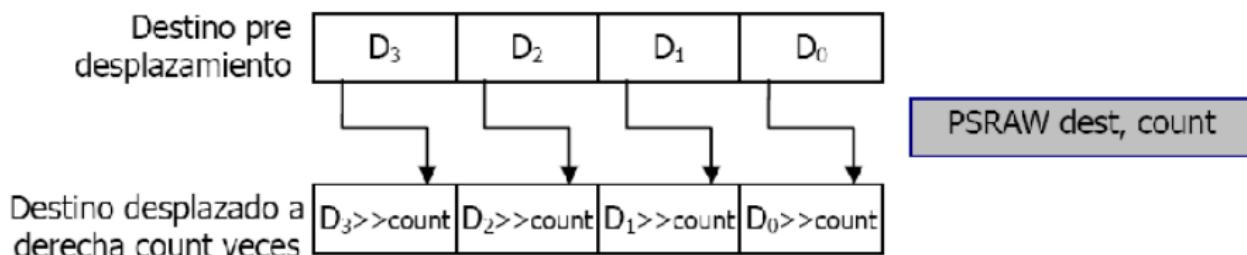
Desplazamiento empaquetado a izquierda en 64 bits



Desplazamiento lógico a derecha empaquetado 64 bits



Desplazamiento aritmético a derecha empaquetado 64 bits



Instrucciones de Comparación

Instrucciones de Comparación

- A diferencia de las instrucciones de comparación convencionales, estas instrucciones no afectan el contenido del registro **EFLAGS**.

Instrucciones de Comparación

- A diferencia de las instrucciones de comparación convencionales, estas instrucciones no afectan el contenido del registro **EFLAGS**.
- Lo que hacen en cambio es setear una máscara de bits en el operando destino, que pueden utilizarse como **MOV** condicionales sin saltos condicionales que demanden tiempo de verificación.

Instrucciones de Comparación

- A diferencia de las instrucciones de comparación convencionales, estas instrucciones no afectan el contenido del registro **EFLAGS**.
- Lo que hacen en cambio es setear una máscara de bits en el operando destino, que pueden utilizarse como **MOV** condicionales sin saltos condicionales que demanden tiempo de verificación.
- Hay dos tipos de comparaciones posibles: por igual (**PCMPEQ**) o por mayor (**PCMPGT**).

Instrucciones de Comparación

- A diferencia de las instrucciones de comparación convencionales, estas instrucciones no afectan el contenido del registro **EFLAGS**.
- Lo que hacen en cambio es setear una máscara de bits en el operando destino, que pueden utilizarse como **MOV** condicionales sin saltos condicionales que demanden tiempo de verificación.
- Hay dos tipos de comparaciones posibles: por igual (**PCMPEQ**) o por mayor (**PCMPGT**).
- **PCMPEQB**, **PCMPEQW**, **PCMPEQD**, **PCMPEQQ** y **PCMPGTB**, **PCMPGTW**, **PCMPGTD**, **PCMPGTQ** comparan los correspondientes elementos sinalados (bytes, words, double words, o quadwords) entre los operandos fuente y destino por igual o mayor que, respectivamente.

Instrucciones de Comparación

Instrucciones de Comparación

- En el caso de que la condición sea EQ (Equal) cada dato empaquetado del operando destino que cumpla la condición se pone en 0xFF, 0xFFFF, o 0xFFFFFFFF, según el tipo de dato empaquetado que se evalúe.

Instrucciones de Comparación

- En el caso de que la condición sea EQ (Equal) cada dato empaquetado del operando destino que cumpla la condición se pone en 0xFF, 0xFFFF, o 0xFFFFFFFF, según el tipo de dato empaquetado que se evalúe.
- Si la condición no se cumple se pone a 0.

Instrucciones de Comparación

- En el caso de que la condición sea EQ (Equal) cada dato empaquetado del operando destino que cumpla la condición se pone en 0xFF, 0xFFFF, o 0xFFFFFFFF, según el tipo de dato empaquetado que se evalúe.
- Si la condición no se cumple se pone a 0.
- Son instrucciones útiles para operaciones tan simples como inicializar en 1's un determinado registro, como por ejemplo:

```
PCMPEQB xmm0, xmm0 ;xmm0 = FF FF FF FF FF FF FF FF FF  
FF FF FF FF FF FF .
```

Instrucciones de Comparación

- En el caso de que la condición sea EQ (Equal) cada dato empaquetado del operando destino que cumpla la condición se pone en 0xFF, 0xFFFF, o 0xFFFFFFFF, según el tipo de dato empaquetado que se evalúe.
- Si la condición no se cumple se pone a 0.
- Son instrucciones útiles para operaciones tan simples como inicializar en 1's un determinado registro, como por ejemplo:

```
PCMPEQB xmm0, xmm0 ;xmm0 = FF FF FF FF FF FF FF FF FF  
FF FF FF FF FF FF.
```

- También para operaciones algo mas sofisticadas como generación de máscaras de modo de procesar bits de manera mas eficiente.

Ejemplo de uso de comparación empaquetada

Procesamiento de sprites o iconos en gráficos 2D.

Ejemplo de uso de comparación empaquetada

Procesamiento de sprites o iconos en gráficos 2D.

- Un sprite es un mapa de bits en el que se dibuja una figura pequeña, de modo tal que los bits que no se utilizan deben quedar “transparentes”, es decir, quedan del color del fondo.

Ejemplo de uso de comparación empaquetada

Procesamiento de sprites o iconos en gráficos 2D.

- Un sprite es un mapa de bits en el que se dibuja una figura pequeña, de modo tal que los bits que no se utilizan deben quedar “transparentes”, es decir, quedan del color del fondo.
- Supongamos un sprite de 8 pixels de ancho por otros 8 pixels de alto.

Ejemplo de uso de comparación empaquetada

Procesamiento de sprites o iconos en gráficos 2D.

- Un sprite es un mapa de bits en el que se dibuja una figura pequeña, de modo tal que los bits que no se utilizan deben quedar “transparentes”, es decir, quedan del color del fondo.
- Supongamos un sprite de 8 pixels de ancho por otros 8 pixels de alto.
- Para la primer línea de 8 pixels, el dibujo del sprite corresponde a los primeros cuatro mientras que los cuatro siguientes deben ser transparentes, es decir, deben ir del color del fondo.

Ejemplo de uso de comparación empaquetada

Procesamiento de sprites o iconos en gráficos 2D.

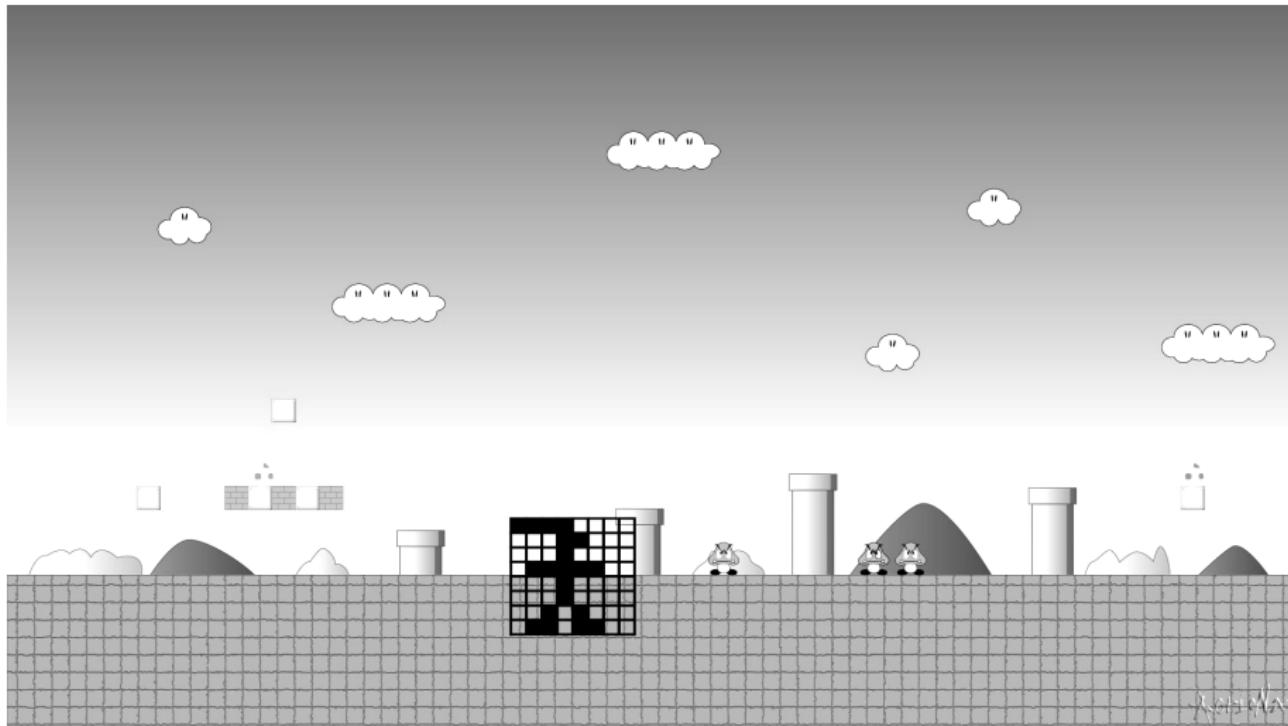
- Un sprite es un mapa de bits en el que se dibuja una figura pequeña, de modo tal que los bits que no se utilizan deben quedar “transparentes”, es decir, quedan del color del fondo.
- Supongamos un sprite de 8 pixels de ancho por otros 8 pixels de alto.
- Para la primer línea de 8 pixels, el dibujo del sprite corresponde a los primeros cuatro mientras que los cuatro siguientes deben ser transparentes, es decir, deben ir del color del fondo.
- Cargamos en mm0 la primer línea del sprite, (8 bytes empaquetados), y mm2 trabaja como auxiliar, está pre seteado en cero.

Ejemplo de uso de comparación empaquetada

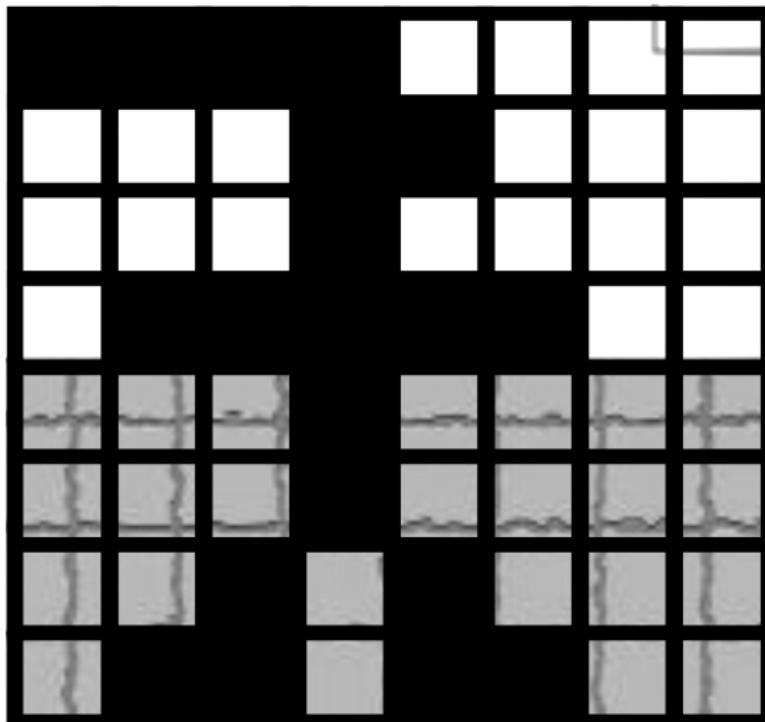
Procesamiento de sprites o iconos en gráficos 2D.

- Un sprite es un mapa de bits en el que se dibuja una figura pequeña, de modo tal que los bits que no se utilizan deben quedar “transparentes”, es decir, quedan del color del fondo.
- Supongamos un sprite de 8 pixels de ancho por otros 8 pixels de alto.
- Para la primer línea de 8 pixels, el dibujo del sprite corresponde a los primeros cuatro mientras que los cuatro siguientes deben ser transparentes, es decir, deben ir del color del fondo.
- Cargamos en mm0 la primer línea del sprite, (8 bytes empaquetados), y mm2 trabaja como auxiliar, está pre seteado en cero.
- Luego PCMPEQB entre ambos registros, generará una máscara con los bytes que contendrán información del sprite en cero y los que deben resultar transparentes en 0xFF.

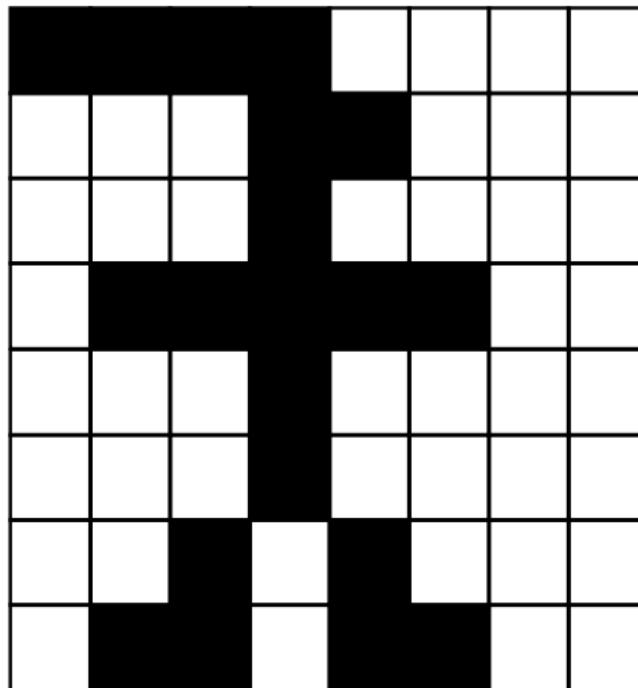
Ejemplo de uso de comparación empaquetada



Ejemplo de uso de comparación empaquetada



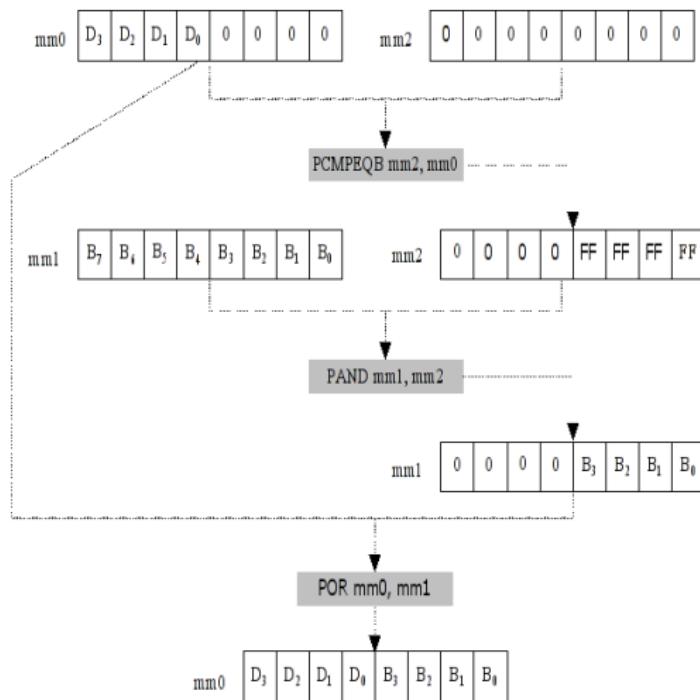
Ejemplo de uso de comparación empaquetada



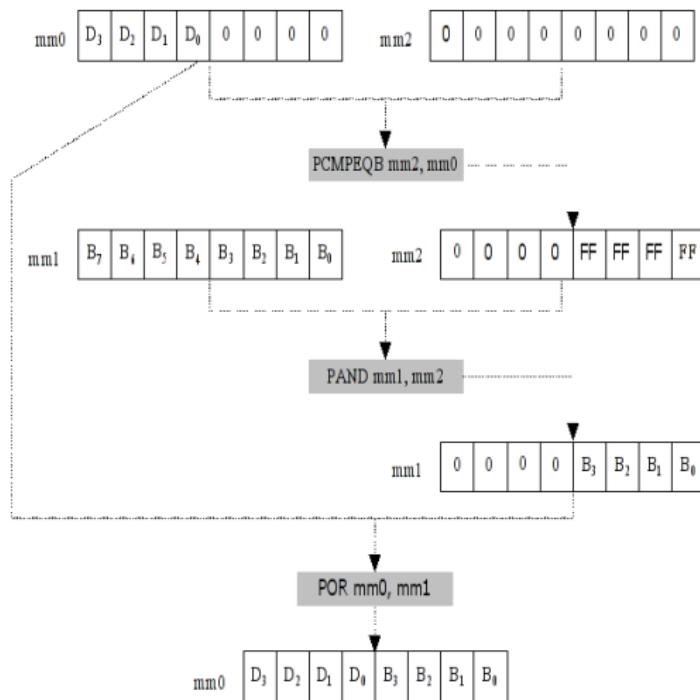
Ejemplo de uso de comparación empaquetada

X	X	X	X	0	0	0	0
0	0	0	X	X	0	0	0
0	0	0	X	0	0	0	0
0	X	X	X	X	X	0	0
0	0	0	X	0	0	0	0
0	0	0	X	0	0	0	0
0	0	X	0	X	0	0	0
0	X	X	0	X	X	0	0

Ejemplo de uso de comparación empaquetada

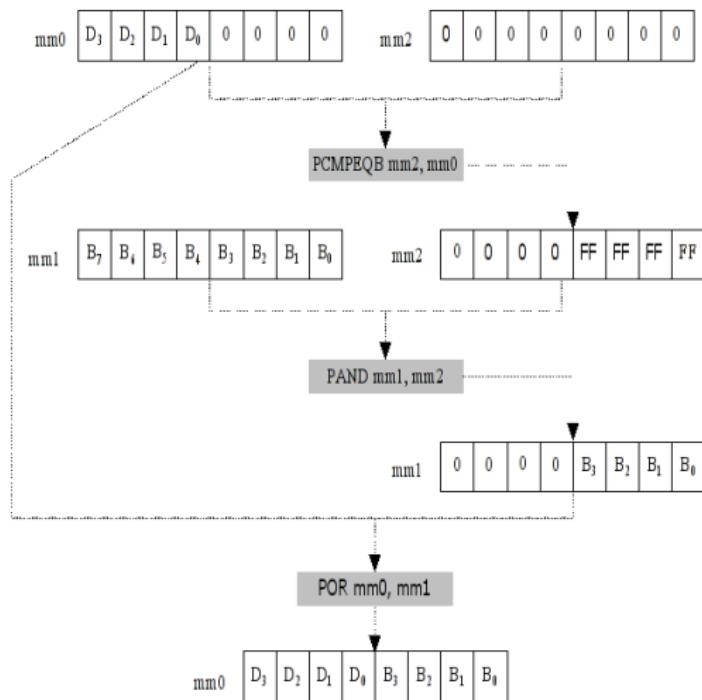


Ejemplo de uso de comparación empaquetada



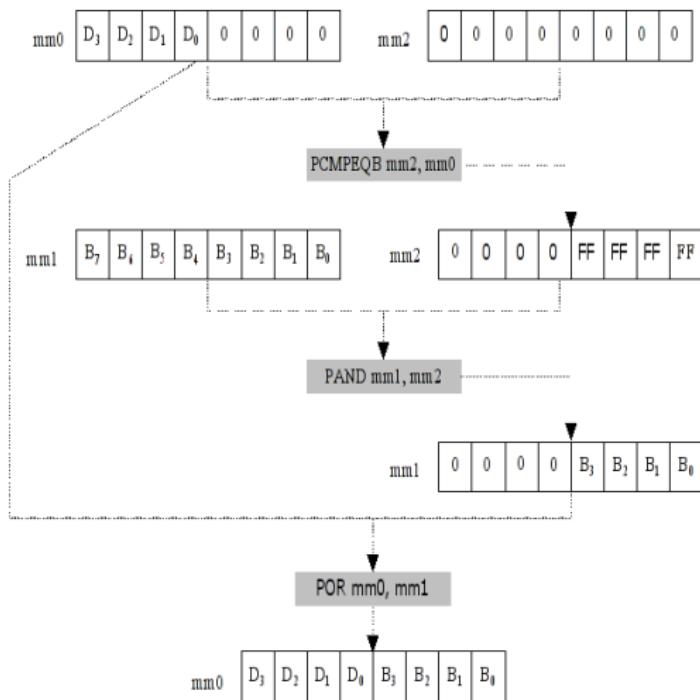
- El resultado se procesa con la información de los 8 pixels del fondo de la imagen (pre almacenado en el registro mm1) a través de un operador AND. Al resultado le quedan solo los cuatro pixels del fondo que se deben mostrar.

Ejemplo de uso de comparación empaquetada



- El resultado se procesa con la información de los 8 pixels del fondo de la imagen (pre almacenado en el registro mm1) a través de un operador AND. Al resultado le quedan solo los cuatro pixels del fondo que se deben mostrar.
- Nos queda imprimir la línea del sprite sobre este fondo, del cual hemos eliminado los cuatro pixels que serán sobre impresos por la línea del sprite.

Ejemplo de uso de comparación empaquetada



- El resultado se procesa con la información de los 8 pixels del fondo de la imagen (pre almacenado en el registro mm1) a través de un operador AND. Al resultado le quedan solo los cuatro pixels del fondo que se deben mostrar.
- Nos queda imprimir la línea del sprite sobre este fondo, del cual hemos eliminado los cuatro pixels que serán sobre impresos por la línea del sprite.
- Tratando a este resultado mediante una operación OR con los 8 pixels originales del sprite se compone el dibujo buscado sobre el fondo existente.

Conversiones de datos empaquetados enteros

Conversiones de datos empaquetados enteros

- Se trata de un conjunto de instrucciones auxiliares para convertir datos empaquetados a formatos de datos empaquetados de menor tamaño.

Conversiones de datos empaquetados enteros

- Se trata de un conjunto de instrucciones auxiliares para convertir datos empaquetados a formatos de datos empaquetados de menor tamaño.
- **PACKSSWB**, se utiliza para convertir words signadas empaquetadas en bytes signados empaquetados.

Conversiones de datos empaquetados enteros

- Se trata de un conjunto de instrucciones auxiliares para convertir datos empaquetados a formatos de datos empaquetados de menor tamaño.
- **PACKSSWB**, se utiliza para convertir words signadas empaquetadas en bytes signados empaquetados.
- **PACKSSDW**, se utiliza para convertir doble words signadas empaquetadas en words signadas empaquetadas.

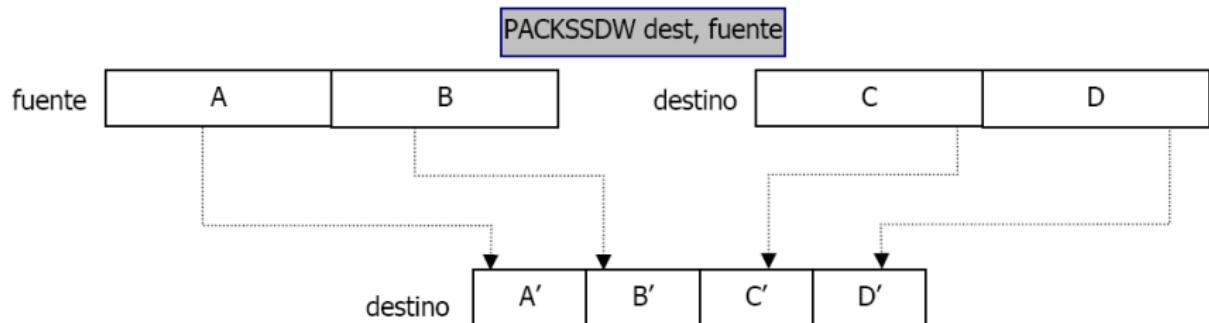
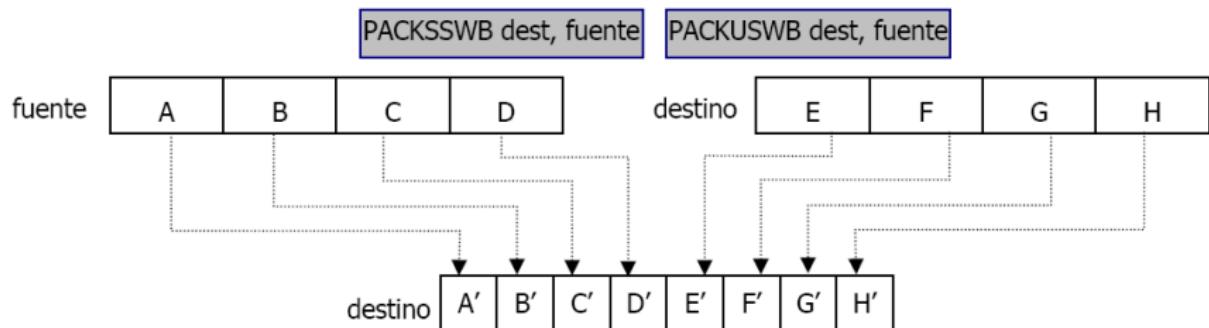
Conversiones de datos empaquetados enteros

- Se trata de un conjunto de instrucciones auxiliares para convertir datos empaquetados a formatos de datos empaquetados de menor tamaño.
- **PACKSSWB**, se utiliza para convertir words signadas empaquetadas en bytes signados empaquetados.
- **PACKSSDW**, se utiliza para convertir doble words signadas empaquetadas en words signadas empaquetadas.
- **PACKUSWB**, se utiliza para convertir words no signadas en bytes no signados

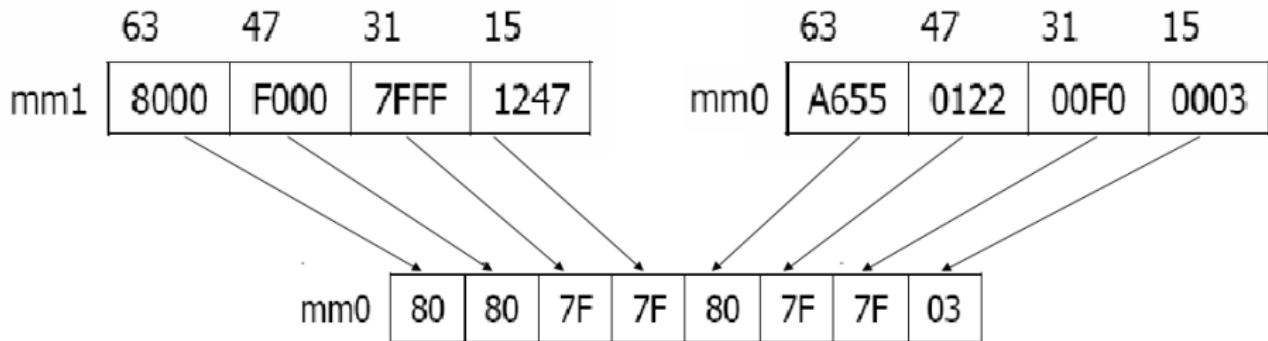
Conversiones de datos empaquetados enteros

- Se trata de un conjunto de instrucciones auxiliares para convertir datos empaquetados a formatos de datos empaquetados de menor tamaño.
- **PACKSSWB**, se utiliza para convertir words signadas empaquetadas en bytes signados empaquetados.
- **PACKSSDW**, se utiliza para convertir doble words signadas empaquetadas en words signadas empaquetadas.
- **PACKUSWB**, se utiliza para convertir words no signadas en bytes no signados
- Los datos se arman a partir de dos registros **MMX** o de un registro **MMX** (operando destino) y una dirección de memoria de 64 bits (operando origen), y el resultado se almacena en el operando destino.

Conversiones de datos empaquetados enteros



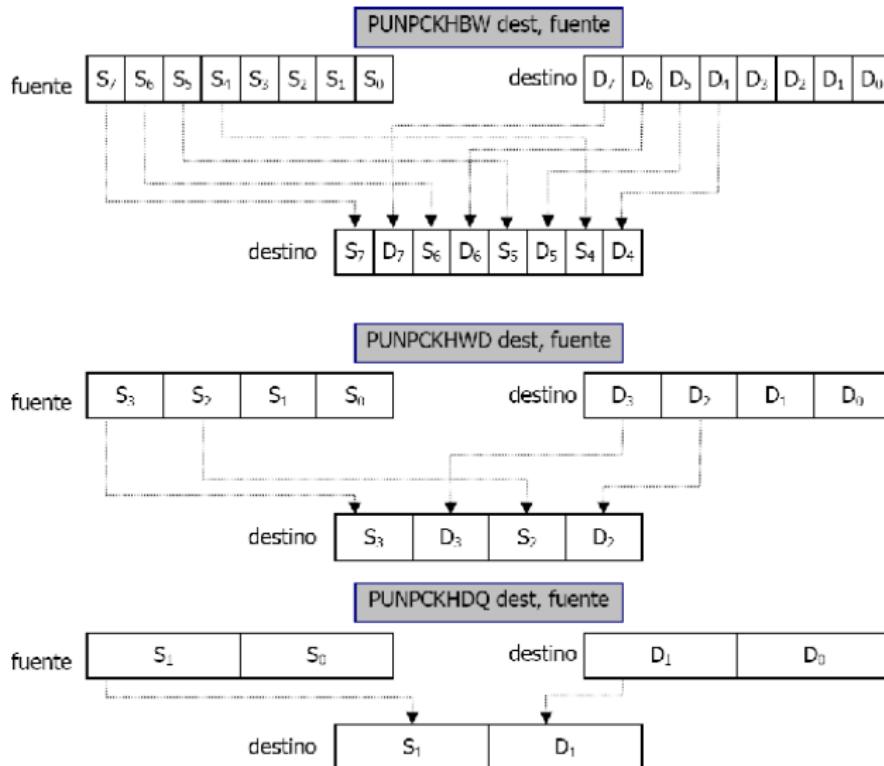
Un ejemplo



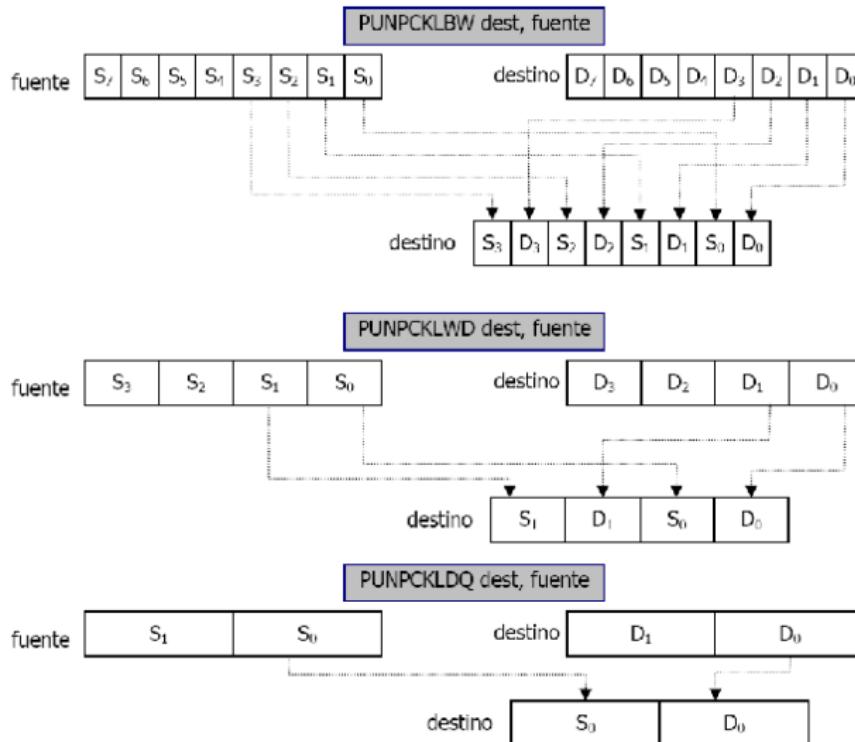
Conversiones de datos empaquetados enteros

- **PUNPCKHBW**, **PUNPCKHWD**, y **PUNPCKHDO**, desempaquetan (**UNPCK**), la parte alta de los operandos fuente y destino (32 bits en cada caso) almacenando los 64 bits del resultado en el operando destino. **PUNPCKHBW** desempaquetá los bytes empaquetados en la mitad alta de cada operando en el operando destino

Conversiones de datos empaquetados enteros



Conversiones de datos empaquetados enteros



Ejemplo de procesamiento de audio

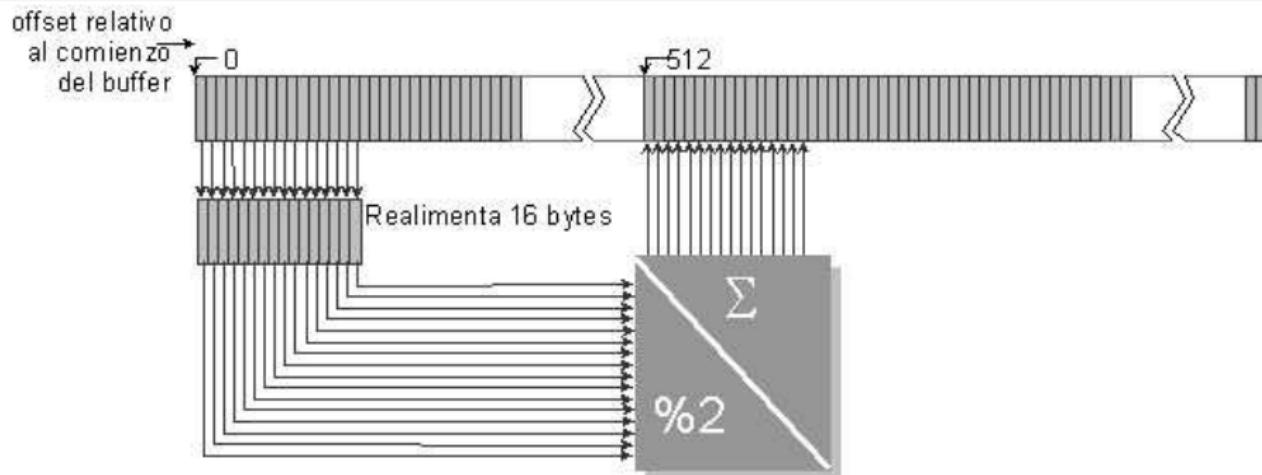
Eco simple

El algoritmo está pensado con de modo tal de comenzar a realimentar las muestras hacia la entrada con una demora equivalente al tiempo de 512 muestras de modo que el efecto resulte apreciable.

Ejemplo de procesamiento de audio

Eco simple

El algoritmo está pensado con de modo tal de comenzar a realimentar las muestras hacia la entrada con una demora equivalente al tiempo de 512 muestras de modo que el efecto resulte apreciable.

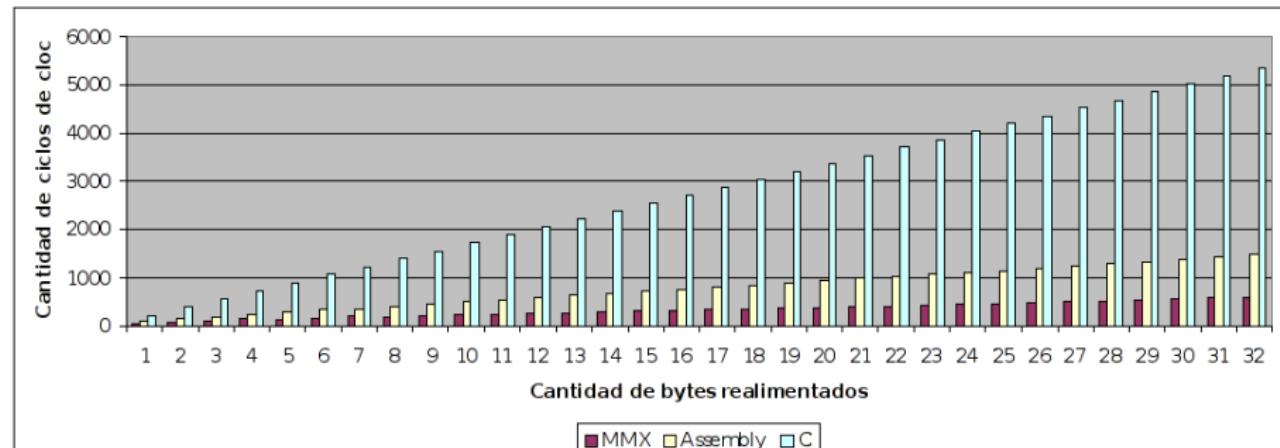


Resultados

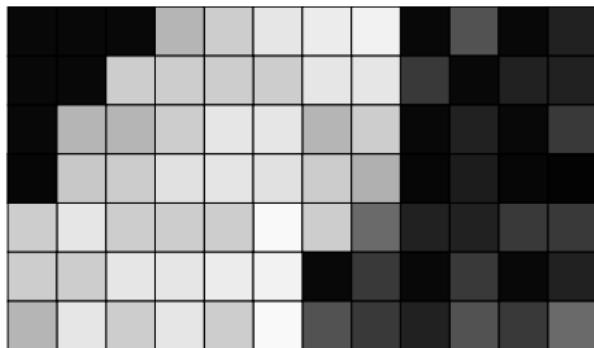
Cantidad de bytes Realimentados	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	256	
Cantidad de clocks por Algoritmo *	MMX	54	69	101	146	130	144	192	178	198	219	235	246	266	282	302	314
	Assembly	91	144	187	231	276	320	364	407	451	496	540	584	627	671	716	760
	C	211	395	559	727	893	1063	1230	1391	1557	1717	1884	2055	2217	2381	2536	2716
		25,6%	17,5%	18,1%	20,1%	14,6%	13,5%	15,6%	12,8%	12,7%	12,8%	12,5%	12,0%	12,0%	11,8%	11,9%	11,6%

Cantidad de bytes Realimentados	272	288	304	320	336	352	368	384	400	416	432	448	464	480	496	512	
Cantidad de clocks por Algoritmo *	MMX	334	349	372	382	404	416	439	450	470	486	510	519	542	554	578	586
	Assembly	804	847	891	936	980	1024	1067	1111	1156	1200	1244	1287	1331	1376	1420	1464
	C	2878	3035	3213	3358	3541	3712	3856	4034	4188	4336	4536	4671	4849	5035	5192	5350
		11,6%	11,5%	11,6%	11,4%	11,4%	11,2%	11,4%	11,2%	11,2%	11,2%	11,2%	11,1%	11,2%	11,0%	11,1%	11,0%

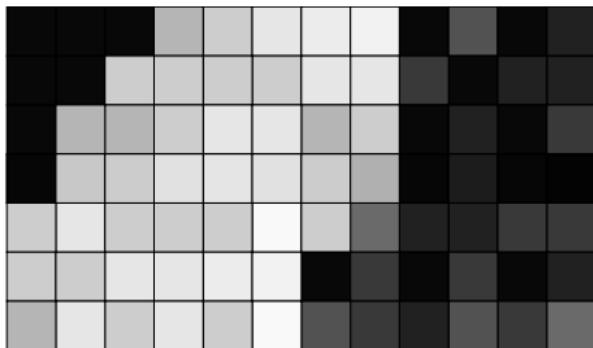
* Para sistemas no real time es el menor valor de 500 iteraciones del mismo ciclo de acuerdo con lo explicado



Ejemplo de procesamiento de imágenes



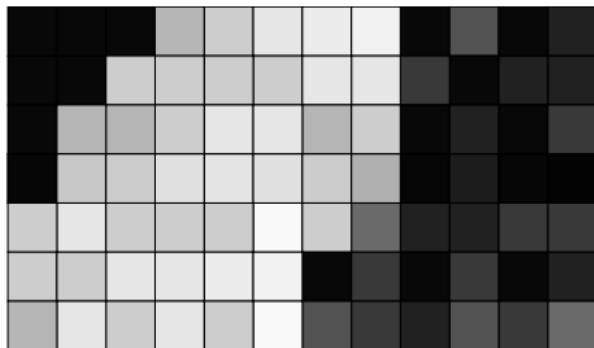
Ejemplo de procesamiento de imágenes



- Un borde es una transición brusca de iluminación entre dos pixeles vecinos.



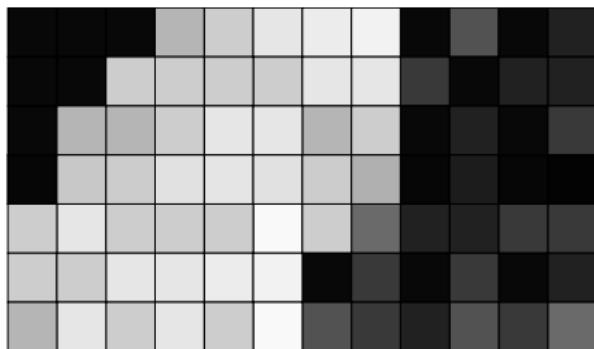
Ejemplo de procesamiento de imágenes



- Un borde es una transición brusca de iluminación entre dos pixeles vecinos.
- En el mapa de bits de la izquierda se muestra como queda una imagen resultante de la detección de bordes



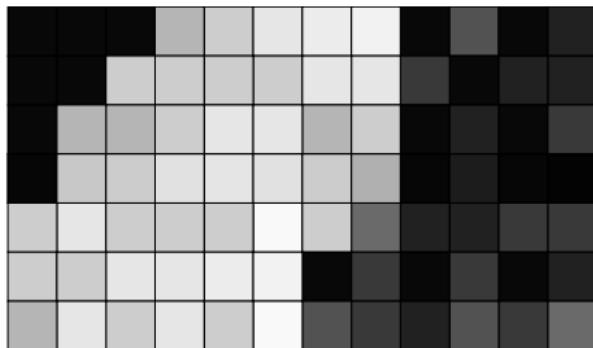
Ejemplo de procesamiento de imágenes



- Un borde es una transición brusca de iluminación entre dos pixeles vecinos.
- En el mapa de bits de la izquierda se muestra como queda una imagen resultante de la detección de bordes
- Un método práctico es por cada pixel tomar los ocho vecinos (N_8), y evaluar su diferencia con el mínimo del N_8

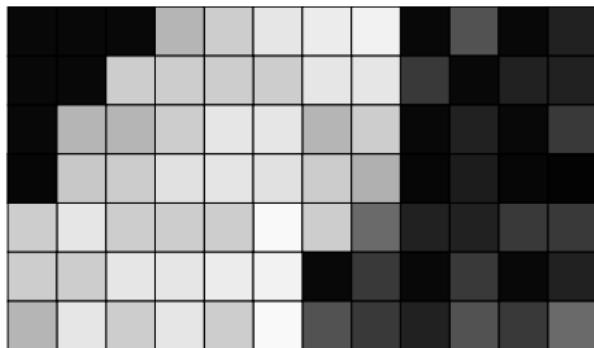


Ejemplo de procesamiento de imágenes



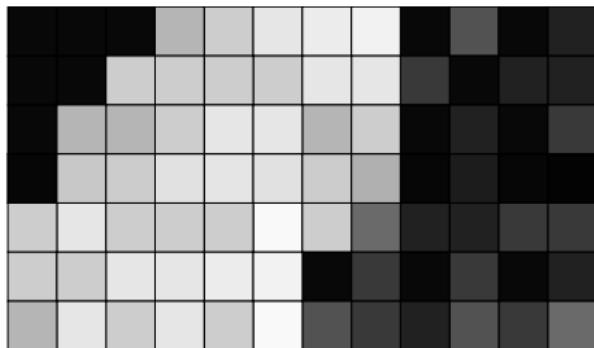
- Un borde es una transición brusca de iluminación entre dos pixeles vecinos.
- En el mapa de bits de la izquierda se muestra como queda una imagen resultante de la detección de bordes
- Un método práctico es por cada pixel tomar los ocho vecinos (N_8), y evaluar su diferencia con el mínimo del N_8
- Si la diferencia es muy grande el valor resultante está próximo al nivel de blanco.

Ejemplo de procesamiento de imágenes



- Un borde es una transición brusca de iluminación entre dos pixeles vecinos.
- En el mapa de bits de la izquierda se muestra como queda una imagen resultante de la detección de bordes
- Un método práctico es por cada pixel tomar los ocho vecinos (N_8), y evaluar su diferencia con el mínimo del N_8
- Si la diferencia es muy grande el valor resultante está próximo al nivel de blanco.
- Si ambos valores tienen una diferencia muy pequeña, esta estará próxima al nivel de negro.

Ejemplo de procesamiento de imágenes



- Un borde es una transición brusca de iluminación entre dos pixeles vecinos.
- En el mapa de bits de la izquierda se muestra como queda una imagen resultante de la detección de bordes
- Un método práctico es por cada pixel tomar los ocho vecinos (N_8), y evaluar su diferencia con el mínimo del N_8
- Si la diferencia es muy grande el valor resultante está próximo al nivel de blanco.
- Si ambos valores tienen una diferencia muy pequeña, esta estará próxima al nivel de negro.
- Reemplazando cada pixel con la diferencia entre su valor y el $\text{MIN}(N_8)$, se puede obtener una buena aproximación de los bordes de una imagen

Ejemplo

```
1 lazo1:  
2     movdqu xmm0,[ esi+(2*ebx)+2] ; Método cálculo por N8  
3     movdqu xmm1,[ esi+(2*ebx)+1]  
4     movdqu xmm2,[ esi+(2*ebx)]  
5     movdqu xmm3,[ esi+ebx+2]  
6     movdqu xmm4,[ esi+ebx]  
7     movdqu xmm5,[ esi+2]  
8     movdqu xmm6,[ esi+1]  
9     movdqu xmm7,[ esi+0]  
10; Calcula el mínimo de los bytes empaquetados en cada registro  
11     pminub xmm0,xmm1  
12     pminub xmm0,xmm2  
13     pminub xmm0,xmm3  
14     pminub xmm0,xmm4  
15     pminub xmm0,xmm5  
16     pminub xmm0,xmm6  
17     pminub xmm0,xmm7;Máximo de los N8 en xmm0.  
18     movdqu xmm1,[ esi+ebx+1]; Leo Pixel central  
19     pminub xmm0,xmm1 ;Lo computo  
20     psbusw xmm1,xmm0 ;Restamos para hallar los bordes  
21     movdqu [ edi ],xmm1  
22     add esi,16          ;Siguiente tira de 16 N8's  
23     add edi,16  
24     loop lazo1
```



Resultados con Imágenes: Detección de Bordes

Bordes		
C	ASM	SSE
14269038	9461102	182518
14269584	9461564	182518
14271390	9461494	182518
14271138	9461326	182532
14270690	9461746	182490
14269864	9461438	182504
14270284	9461445	182513,333
1,51	1,00	0,02

Medición de performance

Medición de performance

- El recurso que se utiliza es el Registro Model Specific TSC (*Time Stamp Counter* introducido a partir del procesador *Pentium*).

Medición de performance

- El recurso que se utiliza es el Registro Model Specific **TSC** (*Time Stamp Counter* introducido a partir del procesador *Pentium*).
- Ancho: 64 bits.

Medición de performance

- El recurso que se utiliza es el Registro Model Specific TSC (*Time Stamp Counter* introducido a partir del procesador *Pentium*).
- Ancho: 64 bits.
- Se incrementa con cada ciclo de clock del procesador. No hay medida mas “fina” del tiempo de procesamiento que este registro.

Medición de performance

- El recurso que se utiliza es el Registro Model Specific **TSC** (*Time Stamp Counter* introducido a partir del procesador *Pentium*).
- Ancho: 64 bits.
- Se incrementa con cada ciclo de clock del procesador. No hay medida mas “fina” del tiempo de procesamiento que este registro.
- Para leerlo utilizamos la instrucción `rdtsc`: `[edx:eax] ← [tsc]`

1 Fundamentos

2 Procesamiento de Señales digitales

3 Modelo de ejecución SIMD

4 Implementaciones SIMD en x86

5 Instrucciones

- Transferencias (las mas comunes)
- Aritmética en algoritmos DSP
- **Instrucciones de punto flotante**
- Instrucciones para manejo de enteros para SSEn
- Instrucciones para manejo de cacheabilidad

Formatos

1 Nro. en Punto Flotante simple precisión escalar

4 Nros. en Punto Flotante simple precisión empaquetados

1 Nro. en Punto Flotante doble precisión escalar

2 Nros. en Punto Flotante doble precisión empaquetados

- Las mismas operaciones en cada formato se manejan con diferentes instrucciones en cada caso.

Grupos de Instrucciones

- Transferencia de Datos.
- Aritmética Empaquetada.
- Comparación.
- Lógicas.
- Shuffle y Desempaquetado.
- Conversión.

Instrucciones de Transferencia de Datos

Instrucciones de Transferencia de Datos

- **MOVAPS** Mueve cuatro valores alineados empaquetados de punto flotante single-precision entre dos registros XMM o entre un registro XMM y memoria. En el caso del operando de memoria, éste debe estar alineado a 16 bytes, o la instrucción generará una excepción #GP.

Instrucciones de Transferencia de Datos

- **MOVAPS** Mueve cuatro valores alineados empaquetados de punto flotante single-precision entre dos registros XMM o entre un registro XMM y memoria. En el caso del operando de memoria, éste debe estar alineado a 16 bytes, o la instrucción generará una excepción #GP.
- **MOVUPS** Mueve cuatro valores no alineados empaquetados de punto flotante single-precision entre dos registros XMM o entre un registro XMM y memoria. En el caso del operando en memoria no se requiere alineación alguna.

Instrucciones de Transferencia de Datos

- **MOVAPS** Mueve cuatro valores alineados empaquetados de punto flotante single-precision entre dos registros XMM o entre un registro XMM y memoria. En el caso del operando de memoria, éste debe estar alineado a 16 bytes, o la instrucción generará una excepción #GP.
- **MOVUPS** Mueve cuatro valores no alineados empaquetados de punto flotante single-precision entre dos registros XMM o entre un registro XMM y memoria. En el caso del operando en memoria no se requiere alineación alguna.
- **MOVSS** Mueve un valor escalar de punto flotante single-precision entre registros XMM o entre un registro XMM y memoria.

Instrucciones de Transferencia de Datos

Instrucciones de Transferencia de Datos

- **MOVAPD** Mueve dos valores alineados empaquetados de punto flotante doble precisión entre dos registros XMM o entre un registro XMM y memoria. En el caso del operando de memoria, éste debe estar alineado a 16 bytes, o la instrucción generará una excepción #GP.

Instrucciones de Transferencia de Datos

- **MOVAPD** Mueve dos valores alineados empaquetados de punto flotante doble precisión entre dos registros XMM o entre un registro XMM y memoria. En el caso del operando de memoria, éste debe estar alineado a 16 bytes, o la instrucción generará una excepción #GP.
- **MOVUPD** Mueve dos valores no alineados empaquetados de punto flotante doble precisión entre dos registros XMM o entre un registro XMM y memoria. En el caso del operando en memoria no se requiere alineación alguna.

Instrucciones de Transferencia de Datos

- **MOVAPD** Mueve dos valores alineados empaquetados de punto flotante doble precisión entre dos registros XMM o entre un registro XMM y memoria. En el caso del operando de memoria, éste debe estar alineado a 16 bytes, o la instrucción generará una excepción #GP.
- **MOVUPD** Mueve dos valores no alineados empaquetados de punto flotante doble precisión entre dos registros XMM o entre un registro XMM y memoria. En el caso del operando en memoria no se requiere alineación alguna.
- **MOVSD** Mueve un valor escalar de punto flotante doble precisión entre registros XMM o entre un registro XMM y memoria.

Instrucciones de Transferencia de Datos

Instrucciones de Transferencia de Datos

- **MOVMSKPS** Extrae la máscara de signo de cuatro valores empaquetados de punto flotante single-precision.

Instrucciones de Transferencia de Datos

- **MOVMSKPS** Extrae la máscara de signo de cuatro valores empaquetados de punto flotante single-precision.
- **MOVHPS** Mueve dos valores de punto flotante single-precision empaquetados entre la quadword alta de un registro XMM y una dirección de memoria de 64 bits. La doble quadword baja del destino permanece sin cambio.

Instrucciones de Transferencia de Datos

- **MOVMSKPS** Extrae la máscara de signo de cuatro valores empaquetados de punto flotante single-precision.
- **MOVHPS** Mueve dos valores de punto flotante single-precision empaquetados entre la quadword alta de un registro XMM y una dirección de memoria de 64 bits. La doble quadword baja del destino permanece sin cambio.
- **MOVLPS** Mueve dos valores de punto flotante single-precision empaquetados entre la quadword baja de un registro XMM y una dirección de memoria de 64 bits. La doble quadword alta del destino permanece sin cambio.

Instrucciones de Transferencia de Datos

Instrucciones de Transferencia de Datos

- **MOVMSKPD** Extrae la máscara de signo de dos valores empaquetados de punto flotante doble precisión.

Instrucciones de Transferencia de Datos

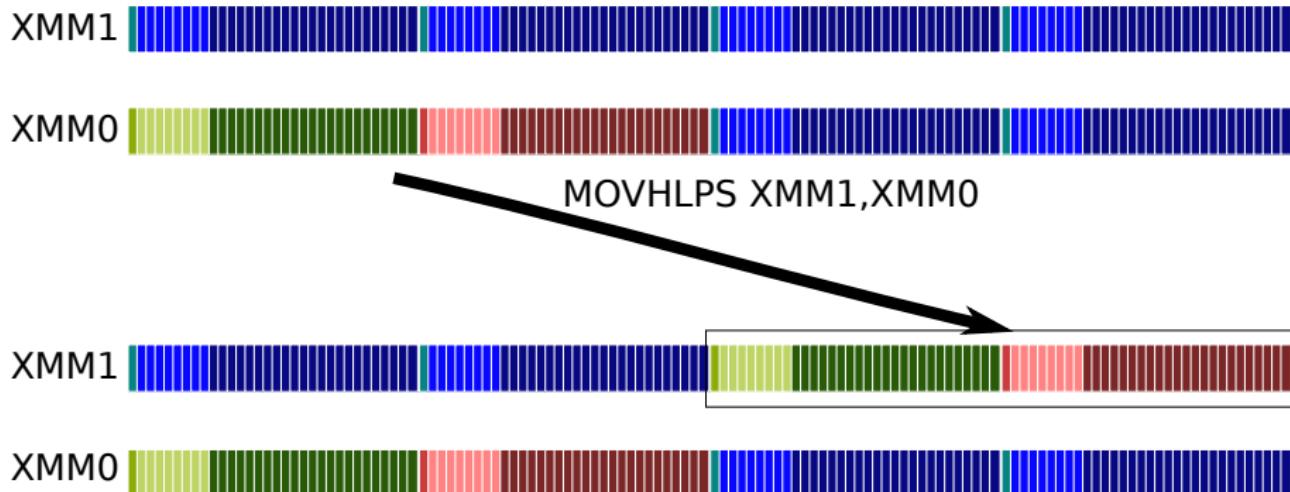
- **MOVMSKPD** Extrae la máscara de signo de dos valores empaquetados de punto flotante doble precisión.
- **MOVHPD** Mueve un valor de punto flotante doble precisión entre la quadword alta de un registro XMM y una dirección de memoria de 64 bits. La doble quadword baja del destino permanece sin cambio.

Instrucciones de Transferencia de Datos

- **MOVMSKPD** Extrae la máscara de signo de dos valores empaquetados de punto flotante doble precisión.
- **MOVHPD** Mueve un valor de punto flotante doble precisión entre la quadword alta de un registro XMM y una dirección de memoria de 64 bits. La doble quadword baja del destino permanece sin cambio.
- **MOVLPD** Mueve un valor de punto flotante doble precisión entre la quadword baja de un registro XMM y una dirección de memoria de 64 bits. La doble quadword alta del destino permanece sin cambio.

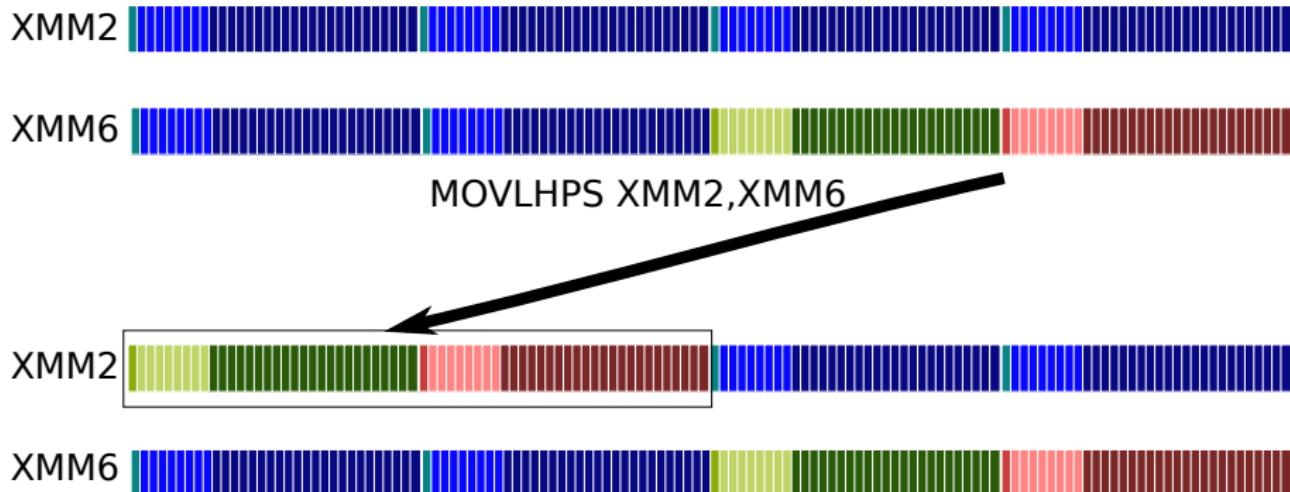
Instrucciones de Transferencia de Datos

- **MOVHLPS** Mueve dos valores de punto flotante single-precision empaquetados desde la quadword alta de un registro XMM a la quadword baja de otro registro XMM. La doble quadword alta del destino permanece sin cambio.



Instrucciones de Transferencia de Datos

- **MOVLHPS** Mueve dos valores empaquetados de punto flotante single-precision desde la quadword baja de un registro XMM a la quadword alta de otro registro XMM. La doble quadword baja del destino permanece sin cambio.



Instrucciones de Aritmética Empaquetada

Instrucciones de Aritmética Empaquetada

- **ADDPS** Suma valores de punto flotante empaquetados single-precision

Instrucciones de Aritmética Empaquetada

- **ADDPS** Suma valores de punto flotante empaquetados single-precision.
- **ADDSS** Suma valores escalares de punto flotante single-precision.

Instrucciones de Aritmética Empaquetada

- **ADDPS** Suma valores de punto flotante empaquetados single-precision.
- **ADDSS** Suma valores escalares de punto flotante single-precision.
- **SUBPS** Resta valores de punto flotante empaquetados single-precision.

Instrucciones de Aritmética Empaquetada

- **ADDPS** Suma valores de punto flotante empaquetados single-precision.
- **ADDSS** Suma valores escalares de punto flotante single-precision.
- **SUBPS** Resta valores de punto flotante empaquetados single-precision.
- **SUBSS** Resta valores escalares de punto flotante single-precision.

Instrucciones de Aritmética Empaquetada

- **ADDPS** Suma valores de punto flotante empaquetados single-precision.
- **ADDSS** Suma valores escalares de punto flotante single-precision.
- **SUBPS** Resta valores de punto flotante empaquetados single-precision.
- **SUBSS** Resta valores escalares de punto flotante single-precision.
- **MULPS** Multiplica valores de punto flotante empaquetados single-precision.

Instrucciones de Aritmética Empaquetada

- **ADDPS** Suma valores de punto flotante empaquetados single-precision.
- **ADDSS** Suma valores escalares de punto flotante single-precision.
- **SUBPS** Resta valores de punto flotante empaquetados single-precision.
- **SUBSS** Resta valores escalares de punto flotante single-precision.
- **MULPS** Multiplica valores de punto flotante empaquetados single-precision.
- **MULSS** Multiplica valores escalares de punto flotante single-precision.

Instrucciones de Aritmética Empaquetada

Instrucciones de Aritmética Empaquetada

- **DIVPS** Divide valores de punto flotante empaquetados single-precision

Instrucciones de Aritmética Empaquetada

- **DIVPS** Divide valores de punto flotante empaquetados single-precision.
- **DIVSS** Divide valores escalares de punto flotante single-precision.

Instrucciones de Aritmética Empaquetada

- **DIVPS** Divide valores de punto flotante empaquetados single-precision.
- **DIVSS** Divide valores escalares de punto flotante single-precision.
- **RCPSS** Computa el inverso de valores de punto flotante empaquetado single-precision.

Instrucciones de Aritmética Empaquetada

- **DIVPS** Divide valores de punto flotante empaquetados single-precision.
- **DIVSS** Divide valores escalares de punto flotante single-precision.
- **RCPPS** Computa el inverso de valores de punto flotante empaquetado single-precision.
- **RCPSS** Computa el inverso de un valor escalar de punto flotante single-precision.

Instrucciones de Aritmética Empaquetada

- **DIVPS** Divide valores de punto flotante empaquetados single-precision.
- **DIVSS** Divide valores escalares de punto flotante single-precision.
- **RCPPS** Computa el inverso de valores de punto flotante empaquetado single-precision.
- **RCPSS** Computa el inverso de un valor escalar de punto flotante single-precision.
- **SQRTPS** Computa la raíz cuadrada de valores de punto flotante empaquetados single-precision.

Instrucciones de Aritmética Empaquetada

- **DIVPS** Divide valores de punto flotante empaquetados single-precision.
- **DIVSS** Divide valores escalares de punto flotante single-precision.
- **RCPPS** Computa el inverso de valores de punto flotante empaquetado single-precision.
- **RCPSS** Computa el inverso de un valor escalar de punto flotante single-precision.
- **SQRTPS** Computa la raíz cuadrada de valores de punto flotante empaquetados single-precision.
- **SQRTSS** Computa la raíz cuadrada de valores escalares de punto flotante single-precision.

Instrucciones de Aritmética Empaquetada

Instrucciones de Aritmética Empaquetada

- **RSQRTPS** Computa la inversa de la raíz cuadrada de valores empaquetados de punto flotante single-precision.

Instrucciones de Aritmética Empaquetada

- **RSQRTPS** Computa la inversa de la raíz cuadrada de valores empaquetados de punto flotante single-precision.
- **RSQRTSS** Computa la inversa de la raíz cuadrada de valores escalares de punto flotante single-precision.

Instrucciones de Aritmética Empaquetada

- **RSQRTPS** Computa la inversa de la raíz cuadrada de valores empaquetados de punto flotante single-precision.
- **RSQRTSS** Computa la inversa de la raíz cuadrada de valores escalares de punto flotante single-precision.
- **MAXPS** Retorna el máximo de valores empaquetados de punto flotante single-precision.

Instrucciones de Aritmética Empaquetada

- **RSQRTPS** Computa la inversa de la raíz cuadrada de valores empaquetados de punto flotante single-precision.
- **RSQRTSS** Computa la inversa de la raíz cuadrada de valores escalares de punto flotante single-precision.
- **MAXPS** Retorna el máximo de valores empaquetados de punto flotante single-precision.
- **MAXSS** Retorna el máximo de un valor escalar de punto flotante single-precision.

Instrucciones de Aritmética Empaquetada

- **RSQRTPS** Computa la inversa de la raíz cuadrada de valores empaquetados de punto flotante single-precision.
- **RSQRTSS** Computa la inversa de la raíz cuadrada de valores escalares de punto flotante single-precision.
- **MAXPS** Retorna el máximo de valores empaquetados de punto flotante single-precision.
- **MAXSS** Retorna el máximo de un valor escalar de punto flotante single-precision.
- **MINPS** Retorna el mínimo de valores empaquetados de punto flotante single-precision.

Instrucciones de Aritmética Empaquetada

- **RSQRTPS** Computa la inversa de la raíz cuadrada de valores empaquetados de punto flotante single-precision.
- **RSQRTSS** Computa la inversa de la raíz cuadrada de valores escalares de punto flotante single-precision.
- **MAXPS** Retorna el máximo de valores empaquetados de punto flotante single-precision.
- **MAXSS** Retorna el máximo de un valor escalar de punto flotante single-precision.
- **MINPS** Retorna el mínimo de valores empaquetados de punto flotante single-precision.
- **MINSS** Retorna el mínimo de un valor escalar de punto flotante single-precision.

Instrucciones de Aritmética Empaquetada

Instrucciones de Aritmética Empaquetada

- **ADDPD** Suma valores de punto flotante empaquetados doble precisión.

Instrucciones de Aritmética Empaquetada

- **ADDPD** Suma valores de punto flotante empaquetados doble precisión.
- **ADDSD** Suma valores escalares de punto flotante doble precisión.

Instrucciones de Aritmética Empaquetada

- **ADDPD** Suma valores de punto flotante empaquetados doble precisión.
- **ADDSD** Suma valores escalares de punto flotante doble precisión.
- **SUBPD** Resta valores de punto flotante empaquetados doble precisión.

Instrucciones de Aritmética Empaquetada

- **ADDPD** Suma valores de punto flotante empaquetados doble precisión.
- **ADDSD** Suma valores escalares de punto flotante doble precisión.
- **SUBPD** Resta valores de punto flotante empaquetados doble precisión.
- **SUBSD** Resta valores escalares de punto flotante doble precisión.

Instrucciones de Aritmética Empaquetada

- **ADDPD** Suma valores de punto flotante empaquetados doble precisión.
- **ADDSD** Suma valores escalares de punto flotante doble precisión.
- **SUBPD** Resta valores de punto flotante empaquetados doble precisión.
- **SUBSD** Resta valores escalares de punto flotante doble precisión.
- **MULPD** Multiplica valores de punto flotante empaquetados doble precisión.

Instrucciones de Aritmética Empaquetada

- **ADDPD** Suma valores de punto flotante empaquetados doble precisión.
- **ADDSD** Suma valores escalares de punto flotante doble precisión.
- **SUBPD** Resta valores de punto flotante empaquetados doble precisión.
- **SUBSD** Resta valores escalares de punto flotante doble precisión.
- **MULPD** Multiplica valores de punto flotante empaquetados doble precisión.
- **MULSD** Multiplica valores escalares de punto flotante doble precisión.

Instrucciones de Aritmética Empaquetada

Instrucciones de Aritmética Empaquetada

- **DIVPD** Divide valores de punto flotante empaquetados doble precisión.

Instrucciones de Aritmética Empaquetada

- **DIVPD** Divide valores de punto flotante empaquetados doble precisión.
- **DIVSD** Divide valores escalares de punto flotante doble precisión.

Instrucciones de Aritmética Empaquetada

- **DIVPD** Divide valores de punto flotante empaquetados doble precisión.
- **DIVSD** Divide valores escalares de punto flotante doble precisión.
- **RCPD** Computa el inverso de valores de punto flotante empaquetado doble precisión.

Instrucciones de Aritmética Empaquetada

- **DIVPD** Divide valores de punto flotante empaquetados doble precisión.
- **DIVSD** Divide valores escalares de punto flotante doble precisión.
- **RCPD** Computa el inverso de valores de punto flotante empaquetado doble precisión.
- **RCPD** Computa el inverso de un valor escalar de punto flotante doble precisión.

Instrucciones de Aritmética Empaquetada

- **DIVPD** Divide valores de punto flotante empaquetados doble precisión.
- **DIVSD** Divide valores escalares de punto flotante doble precisión.
- **RCPD** Computa el inverso de valores de punto flotante empaquetado doble precisión.
- **RCPD** Computa el inverso de un valor escalar de punto flotante doble precisión.
- **SQRTPD** Computa la raíz cuadrada de valores de punto flotante empaquetados doble precisión.

Instrucciones de Aritmética Empaquetada

- **DIVPD** Divide valores de punto flotante empaquetados doble precisión.
- **DIVSD** Divide valores escalares de punto flotante doble precisión.
- **RCPD** Computa el inverso de valores de punto flotante empaquetado doble precisión.
- **RCPD** Computa el inverso de un valor escalar de punto flotante doble precisión.
- **SQRTPD** Computa la raíz cuadrada de valores de punto flotante empaquetados doble precisión.
- **SQRTSD** Computa la raíz cuadrada de valores escalares de punto flotante doble precisión.

Instrucciones de Aritmética Empaquetada

Instrucciones de Aritmética Empaquetada

- **RSQRTPD** Computa la inversa de la raíz cuadrada de valores empaquetados de punto flotante doble precisión.

Instrucciones de Aritmética Empaquetada

- **RSQRTPD** Computa la inversa de la raíz cuadrada de valores empaquetados de punto flotante doble precisión.
- **RSQRTSD** Computa la inversa de la raíz cuadrada de valores escalares de punto flotante doble precisión.

Instrucciones de Aritmética Empaquetada

- **RSQRTPD** Computa la inversa de la raíz cuadrada de valores empaquetados de punto flotante doble precisión.
- **RSQRTSD** Computa la inversa de la raíz cuadrada de valores escalares de punto flotante doble precisión.
- **MAXPD** Retorna el máximo de valores empaquetados de punto flotante doble precisión.

Instrucciones de Aritmética Empaquetada

- **RSQRTPD** Computa la inversa de la raíz cuadrada de valores empaquetados de punto flotante doble precisión.
- **RSQRTSD** Computa la inversa de la raíz cuadrada de valores escalares de punto flotante doble precisión.
- **MAXPD** Retorna el máximo de valores empaquetados de punto flotante doble precisión.
- **MAXSD** Retorna el máximo de un valor escalar de punto flotante doble precisión.

Instrucciones de Aritmética Empaquetada

- **RSQRTPD** Computa la inversa de la raíz cuadrada de valores empaquetados de punto flotante doble precisión.
- **RSQRTSD** Computa la inversa de la raíz cuadrada de valores escalares de punto flotante doble precisión.
- **MAXPD** Retorna el máximo de valores empaquetados de punto flotante doble precisión.
- **MAXSD** Retorna el máximo de un valor escalar de punto flotante doble precisión.
- **MINPD** Retorna el mínimo de valores empaquetados de punto flotante doble precisión.

Instrucciones de Aritmética Empaquetada

- **RSQRTPD** Computa la inversa de la raíz cuadrada de valores empaquetados de punto flotante doble precisión.
- **RSQRTSD** Computa la inversa de la raíz cuadrada de valores escalares de punto flotante doble precisión.
- **MAXPD** Retorna el máximo de valores empaquetados de punto flotante doble precisión.
- **MAXSD** Retorna el máximo de un valor escalar de punto flotante doble precisión.
- **MINPD** Retorna el mínimo de valores empaquetados de punto flotante doble precisión.
- **MINSD** Retorna el mínimo de un valor escalar de punto flotante doble precisión.

Instrucciones Lógicas

Todas consideran a los operandos como un vector de bits, de modo que no tienen diferencia con las instrucciones lógicas convencionales. Se las denomina operaciones “bitwise”.

Instrucciones Lógicas

Todas consideran a los operandos como un vector de bits, de modo que no tienen diferencia con las instrucciones lógicas convencionales. Se las denomina operaciones “bitwise”.

- **ANDPS** Realiza una AND lógica entre dos valores de punto flotante single-precision empaquetados.

Instrucciones Lógicas

Todas consideran a los operandos como un vector de bits, de modo que no tienen diferencia con las instrucciones lógicas convencionales. Se las denomina operaciones “bitwise”.

- **ANDPS** Realiza una AND lógica entre dos valores de punto flotante single-precision empaquetados.
- **ANDNPS** Realiza una NAND lógica entre dos valores de punto flotante single-precision empaquetados.

Instrucciones Lógicas

Todas consideran a los operandos como un vector de bits, de modo que no tienen diferencia con las instrucciones lógicas convencionales. Se las denomina operaciones “bitwise”.

- **ANDPS** Realiza una AND lógica entre dos valores de punto flotante single-precision empaquetados.
- **ANDNPS** Realiza una NAND lógica entre dos valores de punto flotante single-precision empaquetados.
- **ORPS** Realiza una OR lógica entre dos valores de punto flotante single-precision empaquetados.

Instrucciones Lógicas

Todas consideran a los operandos como un vector de bits, de modo que no tienen diferencia con las instrucciones lógicas convencionales. Se las denomina operaciones “bitwise”.

- **ANDPS** Realiza una AND lógica entre dos valores de punto flotante single-precision empaquetados.
- **ANDNPS** Realiza una NAND lógica entre dos valores de punto flotante single-precision empaquetados.
- **ORPS** Realiza una OR lógica entre dos valores de punto flotante single-precision empaquetados.
- **XORPS** Realiza una XOR lógica entre dos valores de punto flotante single-precision empaquetados.

Instrucciones Lógicas

Instrucciones Lógicas

- **ANDPD** Realiza una AND lógica entre dos valores de punto flotante doble precisión empaquetados.

Instrucciones Lógicas

- **ANDPD** Realiza una AND lógica entre dos valores de punto flotante doble precisión empaquetados.
- **ANDNPD** Realiza una NAND lógica entre dos valores de punto flotante doble precisión empaquetados.

Instrucciones Lógicas

- **ANDPD** Realiza una AND lógica entre dos valores de punto flotante doble precisión empaquetados.
- **ANDNPD** Realiza una NAND lógica entre dos valores de punto flotante doble precisión empaquetados.
- **ORPD** Realiza una OR lógica entre dos valores de punto flotante doble precisión empaquetados.

Instrucciones Lógicas

- **ANDPD** Realiza una AND lógica entre dos valores de punto flotante doble precisión empaquetados.
- **ANDNPD** Realiza una NAND lógica entre dos valores de punto flotante doble precisión empaquetados.
- **ORPD** Realiza una OR lógica entre dos valores de punto flotante doble precisión empaquetados.
- **XORPD** Realiza una XOR lógica entre dos valores de punto flotante doble precisión empaquetados.

Instrucciones de Comparación

Realizan comparaciones entre operandos y afectan al operando destino o al registro EFLAGS.

Instrucciones de Comparación

Realizan comparaciones entre operandos y afectan al operando destino o al registro EFLAGS.

- **CMPPS** Compara valores empaquetados de punto flotante de simple precisión.

Instrucciones de Comparación

Realizan comparaciones entre operandos y afectan al operando destino o al registro EFLAGS.

- **CMPPS** Compara valores empaquetados de punto flotante de simple precisión.
- **CMPPD** Compara valores empaquetados de punto flotante doble precisión.

Instrucciones de Comparación

Realizan comparaciones entre operandos y afectan al operando destino o al registro EFLAGS.

- **CMPPS** Compara valores empaquetados de punto flotante de simple precisión.
- **CMPPD** Compara valores empaquetados de punto flotante doble precisión.
- **CMPSS** Compara escalares de punto flotante simple precisión.

Instrucciones de Comparación

Realizan comparaciones entre operandos y afectan al operando destino o al registro EFLAGS.

- **CMPPS** Compara valores empaquetados de punto flotante de simple precisión.
- **CMPPD** Compara valores empaquetados de punto flotante doble precisión.
- **CMPSS** Compara escalares de punto flotante simple precisión.
- **CMPSD** Compara escalares de punto flotante doble precisión.

Instrucciones de Comparación

Realizan comparaciones entre operandos y afectan al operando destino o al registro EFLAGS.

- **CMPPS** Compara valores empaquetados de punto flotante de simple precisión.
- **CMPPD** Compara valores empaquetados de punto flotante doble precisión.
- **CMPSS** Compara escalares de punto flotante simple precisión.
- **CMPSD** Compara escalares de punto flotante doble precisión.
- **COMISS** Compara los valores escalares de punto flotante simple precisión y modifica flags en el registro EFLAGS. Genera excepción #IU (Invalid Operation), tanto si uno de los operandos es SNaN como QNaN.

Instrucciones de Comparación

Realizan comparaciones entre operandos y afectan al operando destino o al registro EFLAGS.

- **CMPPS** Compara valores empaquetados de punto flotante de simple precisión.
- **CMPPD** Compara valores empaquetados de punto flotante doble precisión.
- **CMPSS** Compara escalares de punto flotante simple precisión.
- **CMPSD** Compara escalares de punto flotante doble precisión.
- **COMISS** Compara los valores escalares de punto flotante simple precisión y modifica flags en el registro EFLAGS. Genera excepción #IU (Invalid Operation), tanto si uno de los operandos es SNaN como QNaN.
- **UCOMISS** Es igual a **COMISS** pero solo genera excepción #IU si uno de los operandos es SNaN.

Instrucciones de Comparación

Realizan comparaciones entre operandos y afectan al operando destino o al registro EFLAGS.

- **CMPPS** Compara valores empaquetados de punto flotante de simple precisión.
- **CMPPD** Compara valores empaquetados de punto flotante doble precisión.
- **CMPSS** Compara escalares de punto flotante simple precisión.
- **CMPSD** Compara escalares de punto flotante doble precisión.
- **COMISS** Compara los valores escalares de punto flotante simple precisión y modifica flags en el registro EFLAGS. Genera excepción #IU (Invalid Operation), tanto si uno de los operandos es SNaN como QNaN.
- **UCOMISS** Es igual a **COMISS** pero solo genera excepción #IU si uno de los operandos es SNaN.

Instrucciones de Comparación

La comparación en detalle lleva tres operandos: Dos registros **XMM** o un registro **XMM** y una dirección de memoria, y un tercer operando que es un byte, cuyo valor define hasta 8 alternativas.

Cada dword del operando destino será **0xFFFFFFFF** si el resultado de la comparación es **TRUE** y **0x00000000** si es **FALSE**.

- **CMPEQPS** **xmm1, xmm2**; compara si es igual a...;
- **CMPLTPS** **xmm1, xmm2**; compara si es menor que...;
- **CMPLEPS** **xmm1, xmm2**; compara si es menor o igual a...;
- **CMPUNORDPS** **xmm1, xmm2**; compara si está desordenado (cada operando == NaN);
- **CMPNEQPS** **xmm1, xmm2**; compara si no es igual a...;
- **CMPNLTPS** **xmm1, xmm2**; compara si no es menor que...;
- **CMPNLEPS** **xmm1, xmm2**; compara si no es menor o igual a...;

Instrucciones de Comparación

Para escalares Simple precisión.

- **CMPEQSS** **xmm1**, **xmm2**; compara si es igual a...;
- **CMPLTSS** **xmm1**, **xmm2**; compara si es menor que...;
- **CMPLESS** **xmm1**, **xmm2**; compara si es menor o igual a...;
- **CMPUNORDSS** **xmm1**, **xmm2**; compara si está desordenado (dword baja == NaN);
- **CMPNEQSS** **xmm1**, **xmm2**; compara si no es igual a...;
- **CMPNLTSS** **xmm1**, **xmm2**; compara si no es menor que...;
- **CMPNLESS** **xmm1**, **xmm2**; compara si no es menor o igual a...;
- **CMPORDSS** **xmm1**, **xmm2**; compara si está ordenado (dword baja != NaN);

Instrucciones de Comparación

Para Punto flotante empaquetado de doble precisión.

- **CMPEQPD** **xmm1**, **xmm2**; compara si es igual a...;
- **CMPLTBD** **xmm1**, **xmm2**; compara si es menor que...;
- **CMPLTBD** **xmm1**, **xmm2**; compara si es menor o igual a...;
- **CMPUNORDPD** **xmm1**, **xmm2**; compara si está desordenado (cada operando == NaN);
- **CMPNEQPD** **xmm1**, **xmm2**; compara si no es igual a...;
- **CMPNLTPD** **xmm1**, **xmm2**; compara si no es menor que...;
- **CMPNLEPD** **xmm1**, **xmm2**; compara si no es menor o igual a...;
- **CMPORDPD** **xmm1**, **xmm2**; compara si está ordenado (cada operando != NaN);

Instrucciones de Comparación

Para escalares Doble precisión.

- **CMPEQSD** **xmm1**, **xmm2**; compara si es igual a...
- **CMPLTSD** **xmm1**, **xmm2**; compara si es menor que...
- **CMPLESD** **xmm1**, **xmm2**; compara si es menor o igual a...
- **CMPUNORDSD** **xmm1**, **xmm2**; compara si está desordenado (qword baja == NaN)
- **CMPNEQSD** **xmm1**, **xmm2**; compara si no es igual a...
- **CMPNLTSD** **xmm1**, **xmm2**; compara si no es menor que...
- **CMPNLESD** **xmm1**, **xmm2**; compara si no es menor o igual a...
- **CMPORDSD** **xmm1**, **xmm2**; compara si está ordenado (qword baja != NaN)

Instrucciones de Shuffle y Comparación

Instrucciones de Shuffle y Comparación

- **SHUFPS** Copia dos de las cuatro dwords del destino en la qword baja del operando destino, y dos de las cuatro dwords del operando origen en la qword alta del operando destino.

Instrucciones de Shuffle y Comparación

- **SHUFPS** Copia dos de las cuatro dwords del destino en la qword baja del operando destino, y dos de las cuatro dwords del operando origen en la qword alta del operando destino.
- Sintaxis:

SHUFPS xmm1, xmm2. 00110001b;

Utiliza un tercer operando que establece cual de los dword de cada operando al destino. La lógica es un mapa de bits de a pares dd:cc:bb:aa. Cada par de bits señala la dword a copiar: aa y bb seleccionan las dos dword del operando destino que se copiarán en la qword baja del destino, y cc y dd hacen lo propio con las dworts del operando origen que se copiarán a la qword alta del operando destino.

Instrucciones de Shuffle y Comparación

- **SHUFPS** Copia dos de las cuatro dwords del destino en la qword baja del operando destino, y dos de las cuatro dwords del operando origen en la qword alta del operando destino.

- Sintaxis:

SHUFPS xmm1, xmm2. 00110001b;

Utiliza un tercer operando que establece cual de los dword de cada operando al destino. La lógica es un mapa de bits de a pares dd:cc:bb:aa. Cada par de bits señala la dword a copiar: aa y bb seleccionan las dos dword del operando destino que se copiarán en la qword baja del destino, y cc y dd hacen lo propio con las dwords del operando origen que se copiarán a la qword alta del operando destino.

- En el ejemplo anterior como resultado queda:

Instrucciones de Shuffle y Comparación

- **SHUFPS** Copia dos de las cuatro dwords del destino en la qword baja del operando destino, y dos de las cuatro dwords del operando origen en la qword alta del operando destino.

- Sintaxis:

SHUFPS xmm1, xmm2. 00110001b;

Utiliza un tercer operando que establece cual de los dword de cada operando al destino. La lógica es un mapa de bits de a pares dd:cc:bb:aa. Cada par de bits señala la dword a copiar: aa y bb seleccionan las dos dword del operando destino que se copiarán en la qword baja del destino, y cc y dd hacen lo propio con las dwords del operando origen que se copiarán a la qword alta del operando destino.

- En el ejemplo anterior como resultado queda:

- $XMM1_{[31-0]} = XMM1_{[63-32]}$, doble word de orden aa=01

Instrucciones de Shuffle y Comparación

- **SHUFPS** Copia dos de las cuatro dwords del destino en la qword baja del operando destino, y dos de las cuatro dwords del operando origen en la qword alta del operando destino.

- Sintaxis:

SHUFPS xmm1, xmm2. 00110001b;

Utiliza un tercer operando que establece cual de los dword de cada operando al destino. La lógica es un mapa de bits de a pares dd:cc:bb:aa. Cada par de bits señala la dword a copiar: aa y bb seleccionan las dos dword del operando destino que se copiarán en la qword baja del destino, y cc y dd hacen lo propio con las dwords del operando origen que se copiarán a la qword alta del operando destino.

- En el ejemplo anterior como resultado queda:

- $\text{XMM1}_{[31-0]} = \text{XMM1}_{[63-32]}$, doble word de orden aa=01
- $\text{XMM1}_{[63-32]} = \text{XMM1}_{[31-0]}$, doble word de orden bb=00

Instrucciones de Shuffle y Comparación

- **SHUFPS** Copia dos de las cuatro dwords del destino en la qword baja del operando destino, y dos de las cuatro dwords del operando origen en la qword alta del operando destino.

- Sintaxis:

SHUFPS xmm1, xmm2. 00110001b;

Utiliza un tercer operando que establece cual de los dword de cada operando al destino. La lógica es un mapa de bits de a pares dd:cc:bb:aa. Cada par de bits señala la dword a copiar: aa y bb seleccionan las dos dword del operando destino que se copiarán en la qword baja del destino, y cc y dd hacen lo propio con las dwords del operando origen que se copiarán a la qword alta del operando destino.

- En el ejemplo anterior como resultado queda:

- $\text{XMM1}_{[31-0]} = \text{XMM1}_{[63-32]}$, doble word de orden aa=01
- $\text{XMM1}_{[63-32]} = \text{XMM1}_{[31-0]}$, doble word de orden bb=00
- $\text{XMM1}_{[95-64]} = \text{XMM2}_{[127-96]}$, doble word de orden cc=11

Instrucciones de Shuffle y Comparación

- **SHUFPS** Copia dos de las cuatro dwords del destino en la qword baja del operando destino, y dos de las cuatro dwords del operando origen en la qword alta del operando destino.

- Sintaxis:

SHUFPS xmm1, xmm2. 00110001b;

Utiliza un tercer operando que establece cual de los dword de cada operando al destino. La lógica es un mapa de bits de a pares dd:cc:bb:aa. Cada par de bits señala la dword a copiar: aa y bb seleccionan las dos dword del operando destino que se copiarán en la qword baja del destino, y cc y dd hacen lo propio con las dwords del operando origen que se copiarán a la qword alta del operando destino.

- En el ejemplo anterior como resultado queda:

- $\text{XMM1}_{[31-0]} = \text{XMM1}_{[63-32]}$, doble word de orden aa=01
- $\text{XMM1}_{[63-32]} = \text{XMM1}_{[31-0]}$, doble word de orden bb=00
- $\text{XMM1}_{[95-64]} = \text{XMM2}_{[127-96]}$, doble word de orden cc=11
- $\text{XMM1}_{[127-96]} = \text{XMM2}_{[31-0]}$, doble word de orden dd=00

Instrucciones de Shuffle y Comparación

Instrucciones de Shuffle y Comparación

- **SHUFPD** Copia una de las dos qwords del destino en la qword baja del operando destino, y una de las dos qwords del operando origen en la qword alta del operando destino.

Instrucciones de Shuffle y Comparación

- **SHUFPD** Copia una de las dos qwords del destino en la qword baja del operando destino, y una de las dos qwords del operando origen en la qword alta del operando destino.
- Sintaxis:

SHUFPS xmm1, xmm2, 00000001b;

Utiliza un tercer operando que establece cual de los dword de cada operando al destino. La lógica es un mapa de bits del tipo 000000:b:a. Cada bit señala la qword a copiar: a selecciona la qword del operando destino que se copiará en la qword baja del destino, y b hace lo propio con la qword del operando origen que se copiará en la qword alta del operando destino.

Instrucciones de Shuffle y Comparación

- **SHUFPD** Copia una de las dos qwords del destino en la qword baja del operando destino, y una de las dos qwords del operando origen en la qword alta del operando destino.

- Sintaxis:

SHUFPS xmm1, xmm2, 00000001b;

Utiliza un tercer operando que establece cual de los dword de cada operando al destino. La lógica es un mapa de bits del tipo 000000:b:a. Cada bit señala la qword a copiar: a selecciona la qword del operando destino que se copiará en la qword baja del destino, y b hace lo propio con la qword del operando origen que se copiará en la qword alta del operando destino.

- En el ejemplo anterior como resultado queda:

Instrucciones de Shuffle y Comparación

- **SHUFPD** Copia una de las dos qwords del destino en la qword baja del operando destino, y una de las dos qwords del operando origen en la qword alta del operando destino.
- Sintaxis:

SHUFPS xmm1, xmm2, 00000001b;

Utiliza un tercer operando que establece cual de los dword de cada operando al destino. La lógica es un mapa de bits del tipo 000000:b:a. Cada bit señala la qword a copiar: a selecciona la qword del operando destino que se copiará en la qword baja del destino, y b hace lo propio con la qword del operando origen que se copiará en la qword alta del operando destino.

- En el ejemplo anterior como resultado queda:

- $XMM1_{[63-0]} = XMM1_{[127-64]}$, doble word de orden a=1

Instrucciones de Shuffle y Comparación

- **SHUFPD** Copia una de las dos qwords del destino en la qword baja del operando destino, y una de las dos qwords del operando origen en la qword alta del operando destino.
- Sintaxis:

SHUFPS xmm1, xmm2, 00000001b;

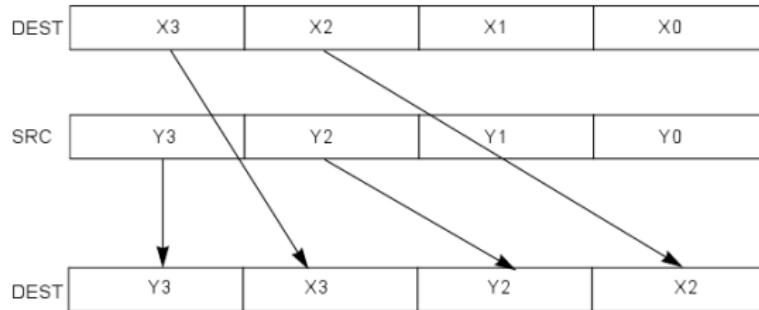
Utiliza un tercer operando que establece cual de los dword de cada operando al destino. La lógica es un mapa de bits del tipo 000000:b:a. Cada bit señala la qword a copiar: a selecciona la qword del operando destino que se copiará en la qword baja del destino, y b hace lo propio con la qword del operando origen que se copiará en la qword alta del operando destino.

- En el ejemplo anterior como resultado queda:

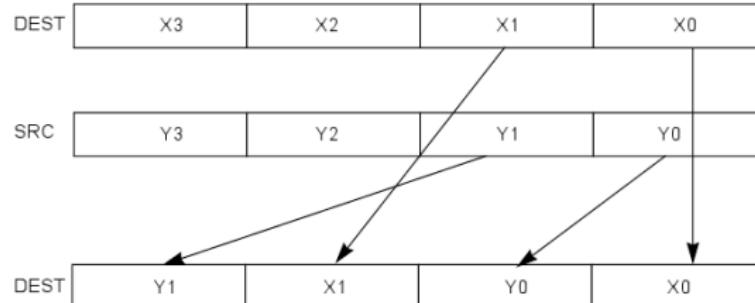
- $\text{XMM1}_{[63-0]} = \text{XMM1}_{[127-64]}$, doble word de orden a=1
- $\text{XMM1}_{[127-64]} = \text{XMM2}_{[63-0]}$, doble word de orden b=0

Instrucciones de Shuffle y Comparación

• UNPCKHPS

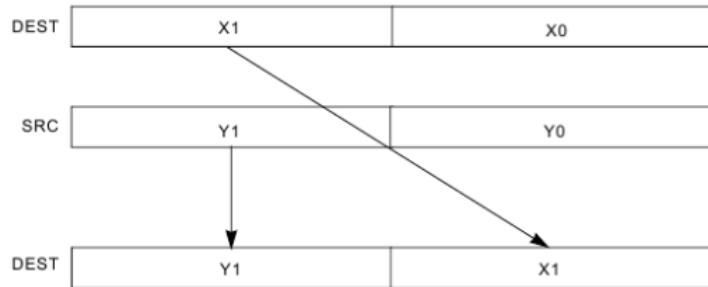


• UNPCKLPS

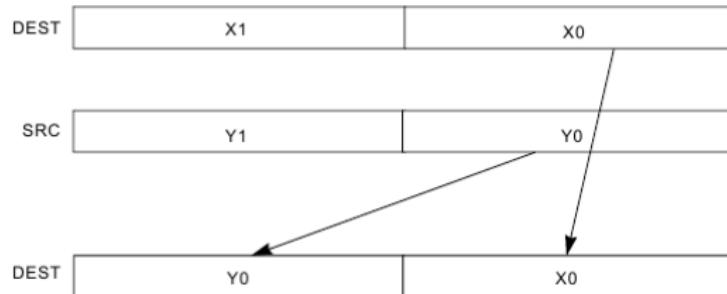


Instrucciones de Shuffle y Comparación

• UNPCKHPD



• UNPCKLPD



1 Fundamentos

2 Procesamiento de Señales digitales

3 Modelo de ejecución SIMD

4 Implementaciones SIMD en x86

5 Instrucciones

- Transferencias (las mas comunes)
- Aritmética en algoritmos DSP
- Instrucciones de punto flotante
- **Instrucciones para manejo de enteros para SSEn**
- Instrucciones para manejo de cacheabilidad

Instrucciones de manejo de enteros

Instrucciones de manejo de enteros

- **PAVGB** Calcula el promedio de bytes enteros sin signo empacados.

Instrucciones de manejo de enteros

- **PAVGB** Calcula el promedio de bytes enteros sin signo empaquetados.
- **PAVGW** Calcula el promedio de words enteras sin signo empaquetadas.

Instrucciones de manejo de enteros

- **PAVGB** Calcula el promedio de bytes enteros sin signo empaquetados.
- **PAVGW** Calcula el promedio de words enteras sin signo empaquetadas.
- **PEXTRW** Extrae word.

Instrucciones de manejo de enteros

- **PAVGB** Calcula el promedio de bytes enteros sin signo empaquetados.
- **PAVGW** Calcula el promedio de words enteras sin signo empaquetadas.
- **PEXTRW** Extrae word.
- **PINSRW** Inserta word.

Instrucciones de manejo de enteros

- **PAVGB** Calcula el promedio de bytes enteros sin signo empaquetados.
- **PAVGW** Calcula el promedio de words enteras sin signo empaquetadas.
- **PEXTRW** Extrae word.
- **PINSRW** Inserta word.
- **PMAXUB** Máximo de bytes enteros sin signo empaquetados.

Instrucciones de manejo de enteros

- **PAVGB** Calcula el promedio de bytes enteros sin signo empaquetados.
- **PAVGW** Calcula el promedio de words enteras sin signo empaquetadas.
- **PEXTRW** Extrae word.
- **PINSRW** Inserta word.
- **PMAXUB** Máximo de bytes enteros sin signo empaquetados.
- **PMAXSW** Máximo de words enteras signadas empaquetadas.

Instrucciones de manejo de enteros

- **PAVGB** Calcula el promedio de bytes enteros sin signo empaquetados.
- **PAVGW** Calcula el promedio de words enteras sin signo empaquetadas.
- **PEXTRW** Extrae word.
- **PINSRW** Inserta word.
- **PMAXUB** Máximo de bytes enteros sin signo empaquetados.
- **PMAXSW** Máximo de words enteras signadas empaquetadas.
- **PMINUB** Mínimo de bytes enteros sin signo empaquetados

Instrucciones de manejo de enteros

- **PAVGB** Calcula el promedio de bytes enteros sin signo empaquetados.
- **PAVGW** Calcula el promedio de words enteras sin signo empaquetadas.
- **PEXTRW** Extrae word.
- **PINSRW** Inserta word.
- **PMAXUB** Máximo de bytes enteros sin signo empaquetados.
- **PMAXSW** Máximo de words enteras signadas empaquetadas.
- **PMINUB** Mínimo de bytes enteros sin signo empaquetados
- **PMINSW** Mínimo de words enteras signadas empaquetadas

Instrucciones de manejo de enteros

- **PAVGB** Calcula el promedio de bytes enteros sin signo empaquetados.
- **PAVGW** Calcula el promedio de words enteras sin signo empaquetadas.
- **PEXTRW** Extrae word.
- **PINSRW** Inserta word.
- **PMAXUB** Máximo de bytes enteros sin signo empaquetados.
- **PMAXSW** Máximo de words enteras signadas empaquetadas.
- **PMINUB** Mínimo de bytes enteros sin signo empaquetados
- **PMINSW** Mínimo de words enteras signadas empaquetadas
- **PMOVMSKB** Mover byte de máscara

Instrucciones de manejo de enteros

- **PAVGB** Calcula el promedio de bytes enteros sin signo empaquetados.
- **PAVGW** Calcula el promedio de words enteras sin signo empaquetadas.
- **PEXTRW** Extrae word.
- **PINSRW** Inserta word.
- **PMAXUB** Máximo de bytes enteros sin signo empaquetados.
- **PMAXSW** Máximo de words enteras signadas empaquetadas.
- **PMINUB** Mínimo de bytes enteros sin signo empaquetados
- **PMINSW** Mínimo de words enteras signadas empaquetadas
- **PMOVMSKB** Mover byte de máscara
- **PMULHUW** Multiplica enteros empaquetados sin signo y almacena el resultado alto

Instrucciones de manejo de enteros

- **PAVGB** Calcula el promedio de bytes enteros sin signo empaquetados.
- **PAVGW** Calcula el promedio de words enteras sin signo empaquetadas.
- **PEXTRW** Extrae word.
- **PINSRW** Inserta word.
- **PMAXUB** Máximo de bytes enteros sin signo empaquetados.
- **PMAXSW** Máximo de words enteras signadas empaquetadas.
- **PMINUB** Mínimo de bytes enteros sin signo empaquetados
- **PMINSW** Mínimo de words enteras signadas empaquetadas
- **PMOVMSKB** Mover byte de máscara
- **PMULHUW** Multiplica enteros empaquetados sin signo y almacena el resultado alto
- **PSADBW** Calcula la suma de diferencias absolutas

Instrucciones de manejo de enteros

- **PAVGB** Calcula el promedio de bytes enteros sin signo empaquetados.
- **PAVGW** Calcula el promedio de words enteras sin signo empaquetadas.
- **PEXTRW** Extrae word.
- **PINSRW** Inserta word.
- **PMAXUB** Máximo de bytes enteros sin signo empaquetados.
- **PMAXSW** Máximo de words enteras signadas empaquetadas.
- **PMINUB** Mínimo de bytes enteros sin signo empaquetados
- **PMINSW** Mínimo de words enteras signadas empaquetadas
- **PMOVMSKB** Mover byte de máscara
- **PMULHUW** Multiplica enteros empaquetados sin signo y almacena el resultado alto
- **PSADBW** Calcula la suma de diferencias absolutas
- **PSHUFW** Cambia de orden words enteras empaquetadas en registros MMX

1 Fundamentos

2 Procesamiento de Señales digitales

3 Modelo de ejecución SIMD

4 Implementaciones SIMD en x86

5 Instrucciones

- Transferencias (las mas comunes)
- Aritmética en algoritmos DSP
- Instrucciones de punto flotante
- Instrucciones para manejo de enteros para SSEn
- **Instrucciones para manejo de cacheabilidad**

Manejo del cache

Manejo del cache

- **MASKMOVQ** Almacenamiento No-temporal de los bytes seleccionados de una quadword desde un registro XMM a memoria.

Manejo del cache

- **MASKMOVQ** Almacenamiento No-temporal de los bytes seleccionados de una quadword desde un registro XMM a memoria.
- **MOVNTQ** Almacenamiento No-temporal de una quadword desde un registro XMM a memoria.

Manejo del cache

- **MASKMOVQ** Almacenamiento No-temporal de los bytes seleccionados de una quadword desde un registro XMM a memoria.
- **MOVNTQ** Almacenamiento No-temporal de una quadword desde un registro XMM a memoria.
- **MOVNTPS** Almacenamiento No-temporal de cuatro valores de punto flotante single-precision desde un registro MMX a memoria.

Manejo del cache

- **MASKMOVQ** Almacenamiento No-temporal de los bytes seleccionados de una quadword desde un registro XMM a memoria.
- **MOVNTQ** Almacenamiento No-temporal de una quadword desde un registro XMM a memoria.
- **MOVNTPS** Almacenamiento No-temporal de cuatro valores de punto flotante single-precision desde un registro MMX a memoria.
- **PREFETCHh** Carga 32 bytes (una línea) en el nivel caché especificada a partir de una dirección especificada, hasta un nivel especificado del cache.

Manejo del cache

- **MASKMOVQ** Almacenamiento No-temporal de los bytes seleccionados de una quadword desde un registro XMM a memoria.
- **MOVNTQ** Almacenamiento No-temporal de una quadword desde un registro XMM a memoria.
- **MOVNTPS** Almacenamiento No-temporal de cuatro valores de punto flotante single-precision desde un registro MMX a memoria.
- **PREFETCHh** Carga 32 bytes (una línea) en el nivel caché especificada a partir de una dirección especificada, hasta un nivel especificado del cache.
- **SFENCE** Serializa operaciones de Store.