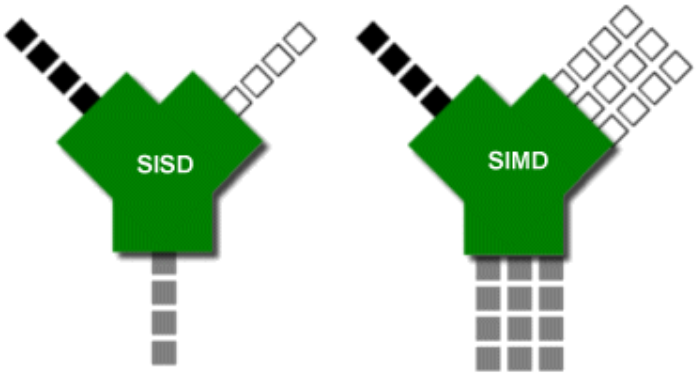


Modelo de Ejecución SIMD en ARMv7

Set de instrucciones NEON

Alejandro Furfaro

16 de agosto de 2021

Tabla de contenidos

Tabla de contenidos

Señal digitalizada

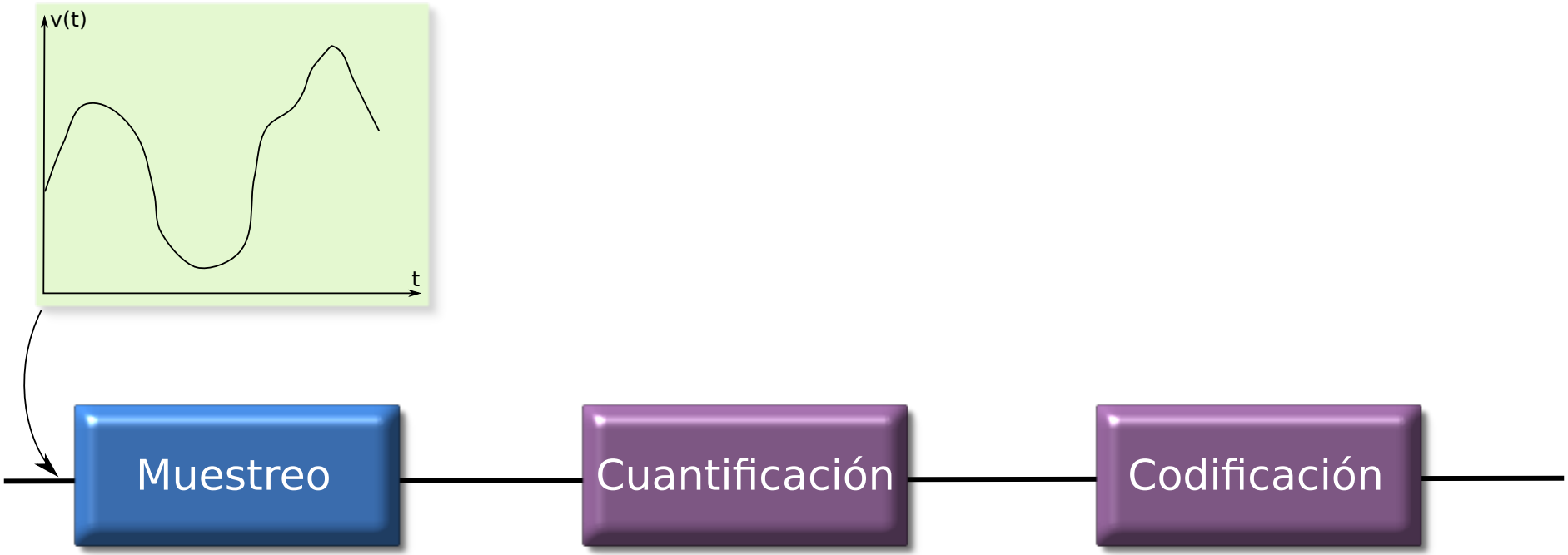
- El proceso de digitalización de una señal responde al siguiente modelo.

Señal digitalizada

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

Señal digitalizada

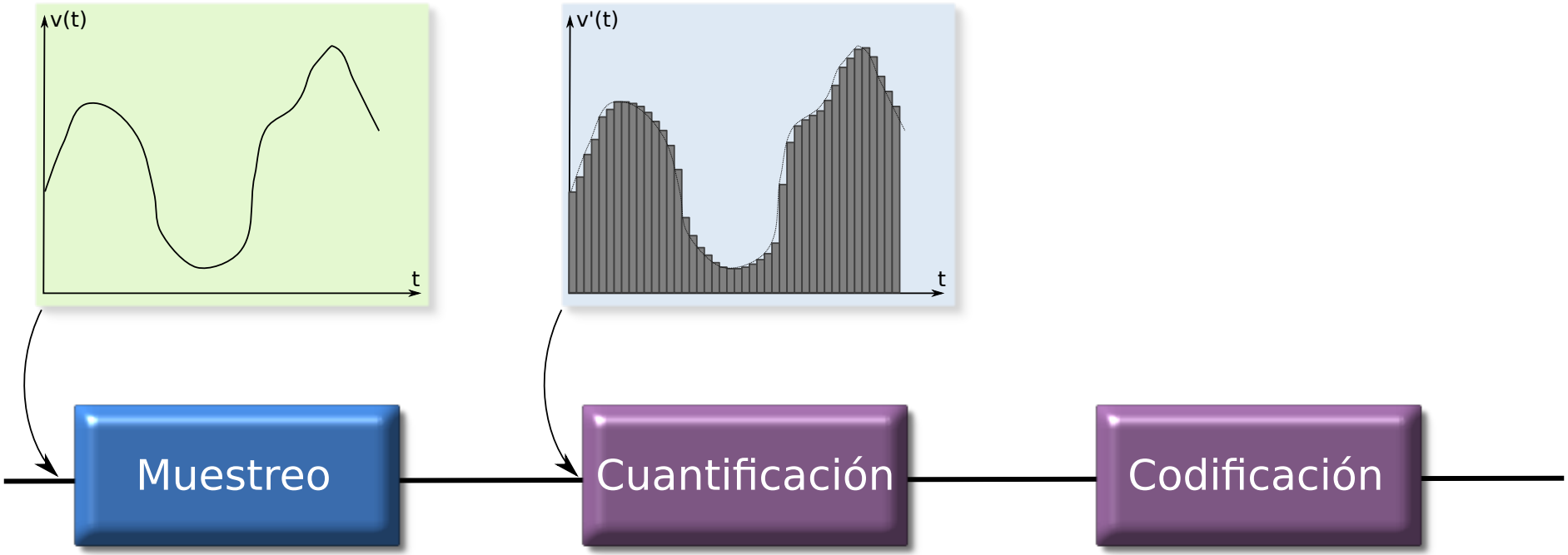
- El proceso de digitalización de una señal responde al siguiente modelo.



- La función *v(t)*, es una tensión de entrada analógica.

Señal digitalizada

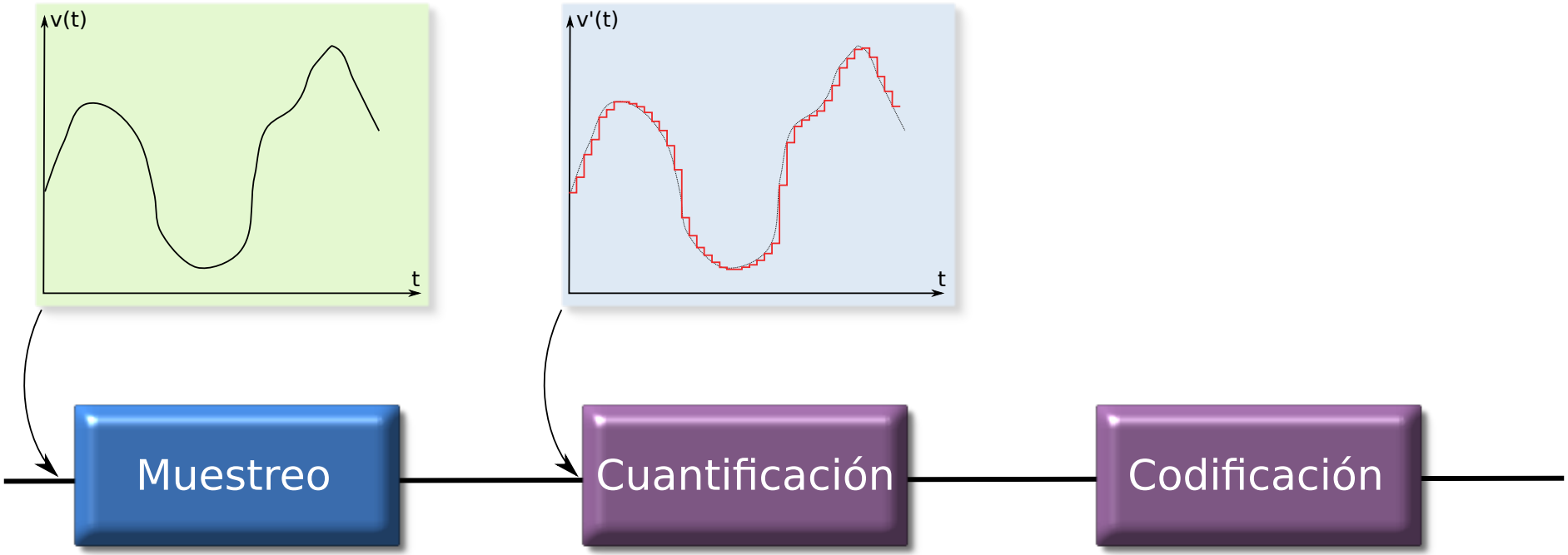
- El proceso de digitalización de una señal responde al siguiente modelo.



- La función *v(t)*, es una tensión de entrada analógica.
- La función *v'(t)*, es una función discreta, que solo cambia entre valores discretos (no contínua) a intervalos regulares, dados por la frecuencia de muestreo, y mantiene su valor durante el intervalo.

Señal digitalizada

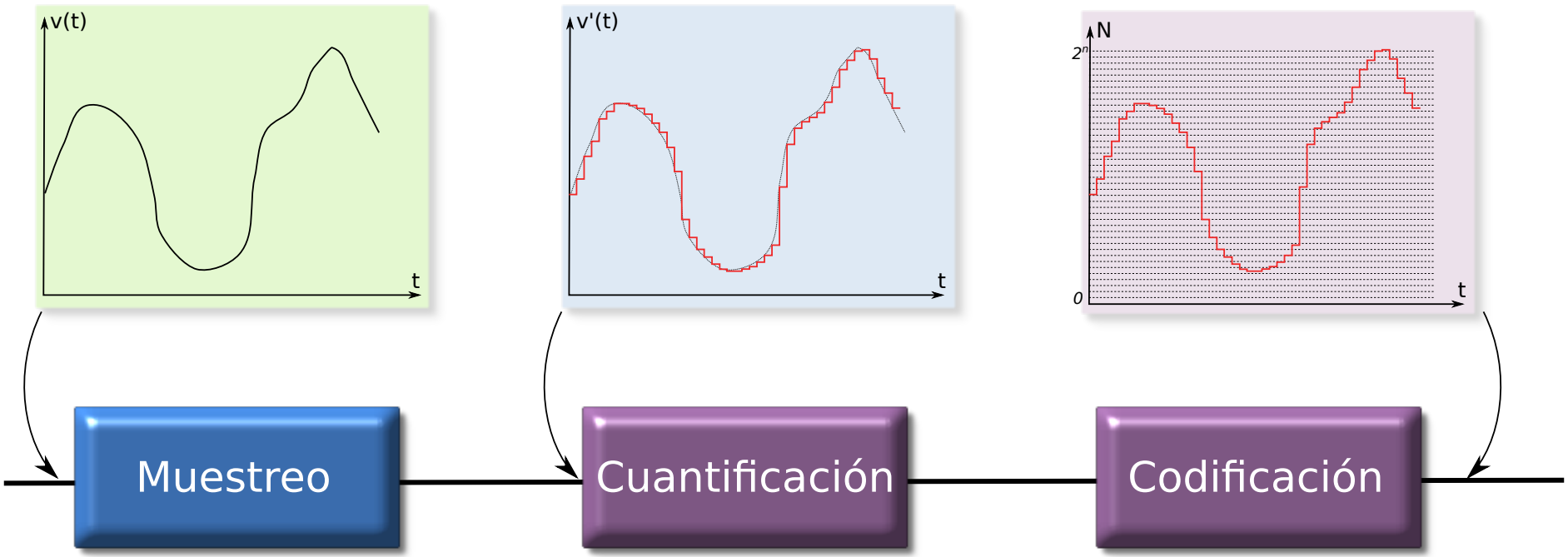
- El proceso de digitalización de una señal responde al siguiente modelo.



- La función *v(t)*, es una tensión de entrada analógica.
- La función *v'(t)*, es una función discreta, que solo cambia entre valores discretos (no contínua) a intervalos regulares, dados por la frecuencia de muestreo, y mantiene su valor durante el intervalo.

Señal digitalizada

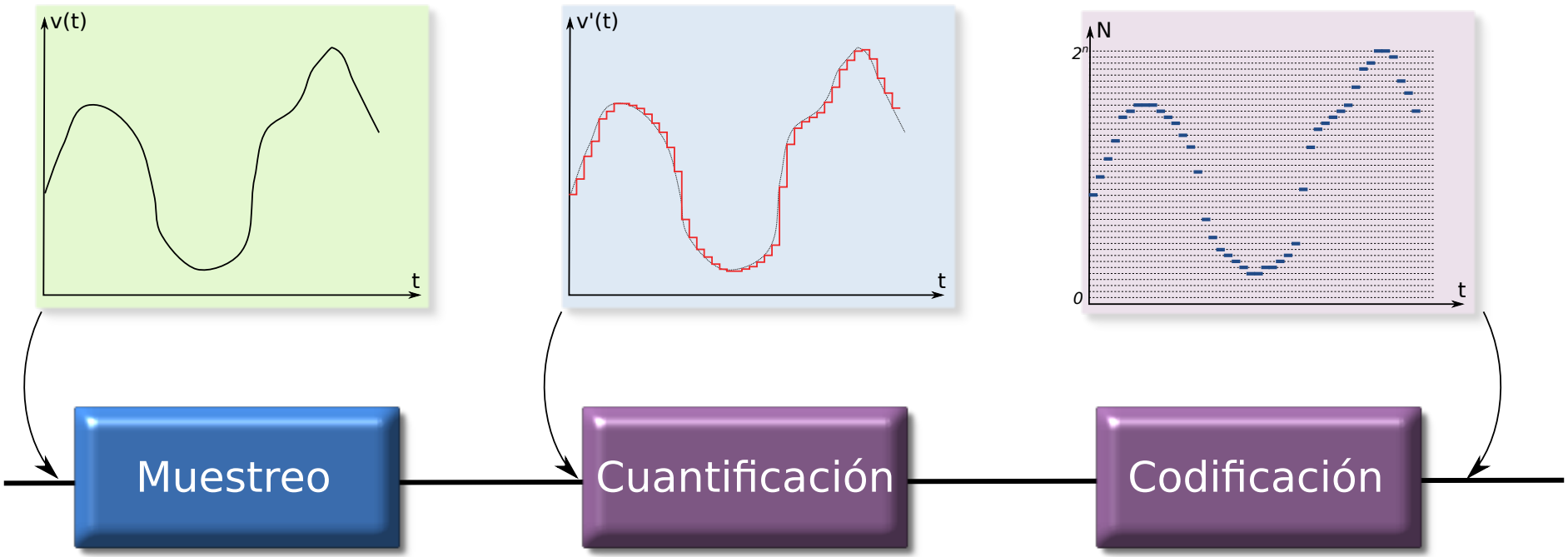
- El proceso de digitalización de una señal responde al siguiente modelo.



- La función *v(t)*, es una tensión de entrada analógica.
- La función *v'(t)*, es una función discreta, que solo cambia entre valores discretos (no contínua) a intervalos regulares, dados por la frecuencia de muestreo, y mantiene su valor durante el intervalo.

Señal digitalizada

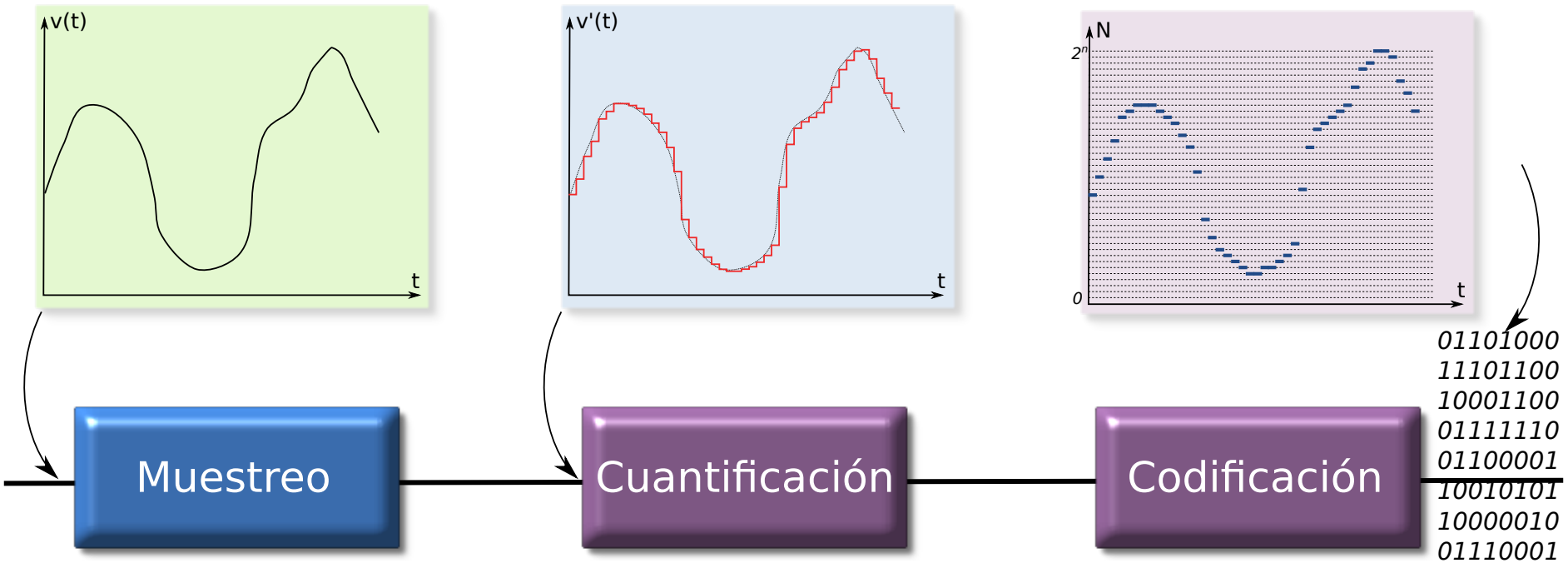
- El proceso de digitalización de una señal responde al siguiente modelo.



- La función *v(t)*, es una tensión de entrada analógica.
- La función *v'(t)*, es una función discreta, que solo cambia entre valores discretos (no contínua) a intervalos regulares, dados por la frecuencia de muestreo, y mantiene su valor durante el intervalo.

Señal digitalizada

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



- La función *v(t)*, es una tensión de entrada analógica.
- La función *v'(t)*, es una función discreta, que solo cambia entre valores discretos (no contínua) a intervalos regulares, dados por la frecuencia de muestreo, y mantiene su valor durante el intervalo.

Señal digitalizada

- El circuito de Muestreo y Retención sostiene durante todo el intervalo de muestreo el valor de la señal de entrada al inicio de dicho intervalo, ignorando los demás valores.

Señal digitalizada

- El circuito de Muestreo y Retención sostiene durante todo el intervalo de muestreo el valor de la señal de entrada al inicio de dicho intervalo, ignorando los demás valores.
- De este modo actúa sobre la variable independiente de la señal de entrada, en nuestro caso el tiempo, convirtiéndola de continua a discreta, ya que define un conjunto finito de valores de tiempo que resultan de interés.

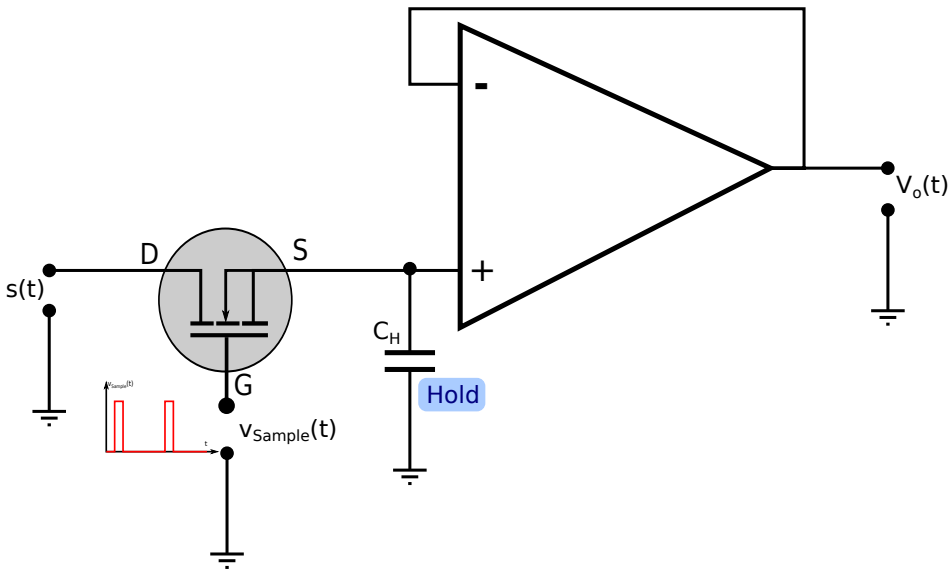
Señal digitalizada

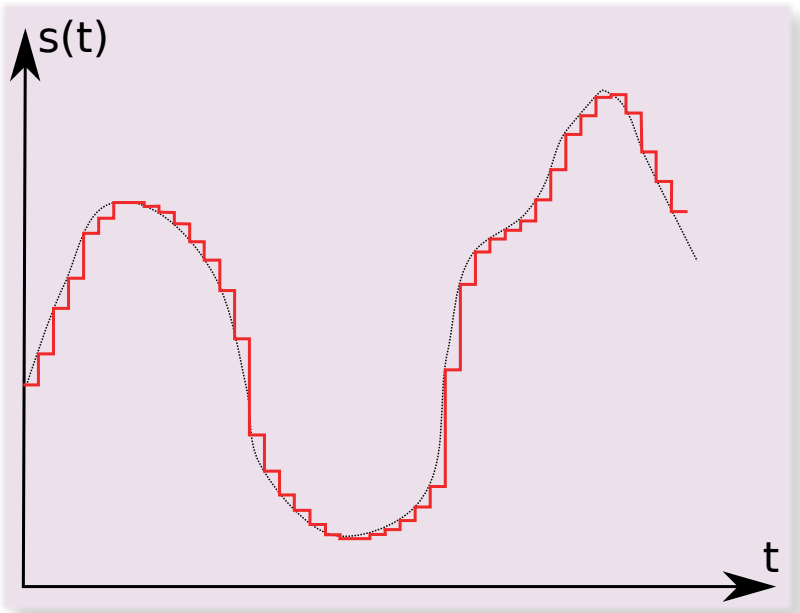
- El circuito de Muestreo y Retención sostiene durante todo el intervalo de muestreo el valor de la señal de entrada al inicio de dicho intervalo, ignorando los demás valores.
- De este modo actúa sobre la variable independiente de la señal de entrada, en nuestro caso el tiempo, convirtiéndola de continua a discreta, ya que define un conjunto finito de valores de tiempo que resultan de interés.
- El proceso de Cuantificación (o Cuantización), transforma los valores obtenidos de la variable dependiente aproximándolos al valor mas cercano de un conjunto finito de 2^n números.

Señal digitalizada

- El circuito de Muestreo y Retención sostiene durante todo el intervalo de muestreo el valor de la señal de entrada al inicio de dicho intervalo, ignorando los demás valores.
- De este modo actúa sobre la variable independiente de la señal de entrada, en nuestro caso el tiempo, convirtiéndola de continua a discreta, ya que define un conjunto finito de valores de tiempo que resultan de interés.
- El proceso de Cuantificación (o Cuantización), transforma los valores obtenidos de la variable dependiente aproximándolos al valor mas cercano de un conjunto finito de 2^n números.
- A continuación mas detalles...

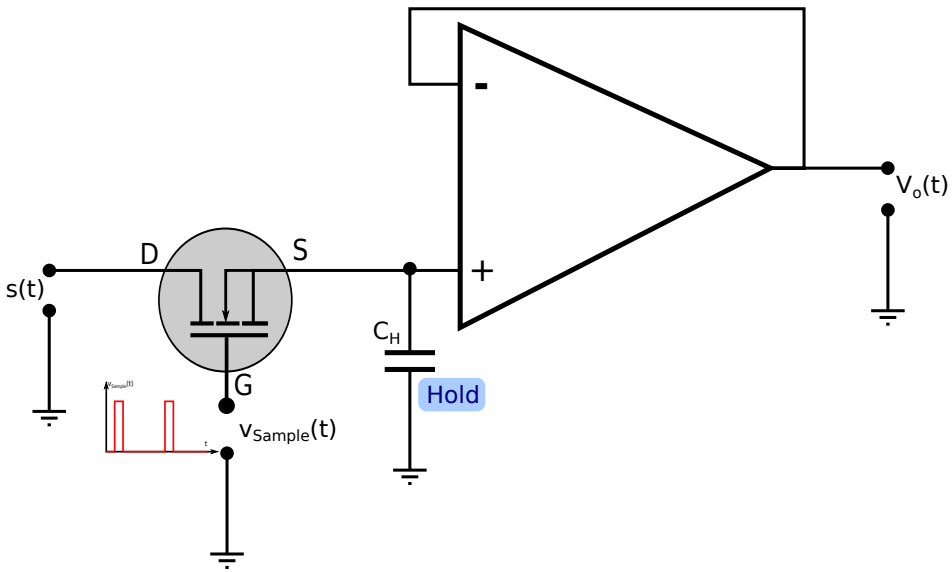
1er. Fase: Muestreo y retención

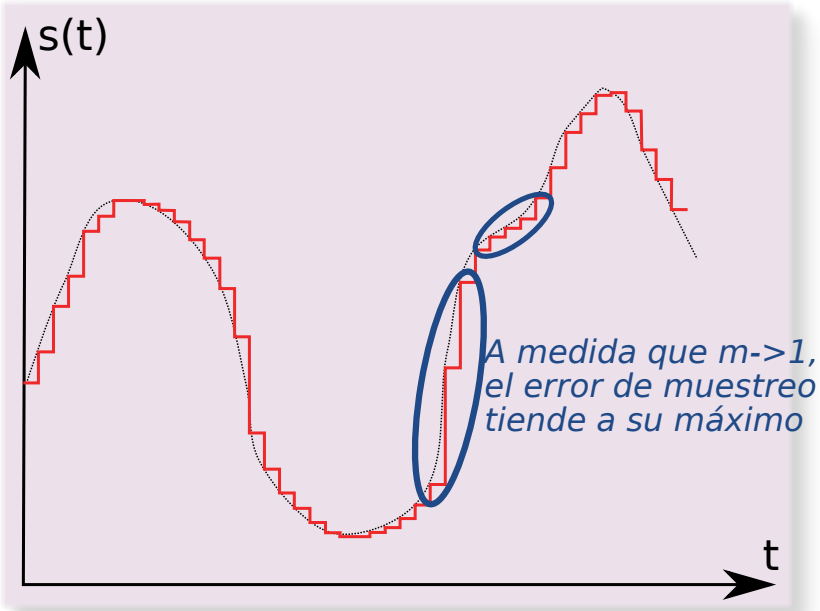




- Se toma un valor instantáneo de la señal y se “retiene” el valor de tensión en una capacidad, con un circuito resistivo de descarga de muy alta resistencia.

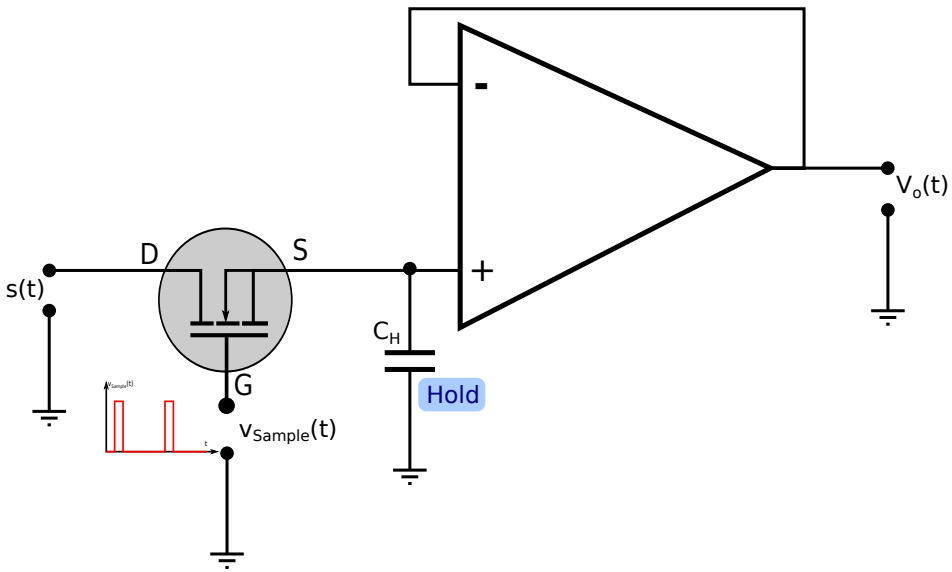
1er. Fase: Muestreo y retención

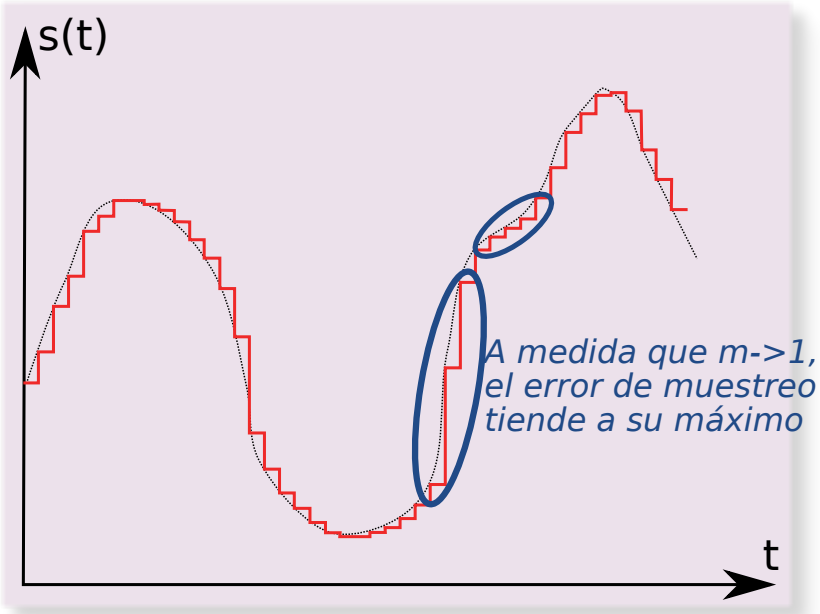




- Se toma un valor instantáneo de la señal y se “retiene” el valor de tensión en una capacidad, con un circuito resistivo de descarga de muy alta resistencia.
- *Comparando la señal saliente del circuito de Muestreo y Retención, con la señal original (en punteado suave), observar que el error aumenta en los cambios abruptos de señal.*

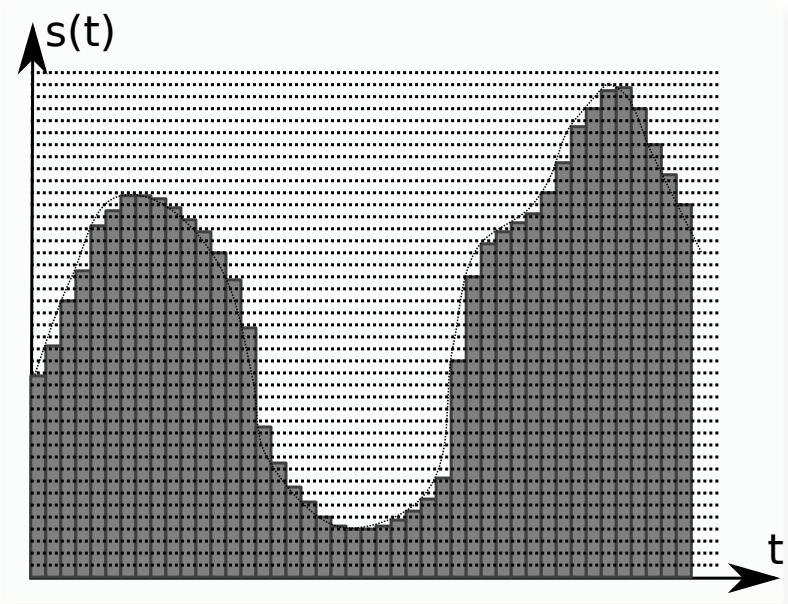
1er. Fase: Muestreo y retención





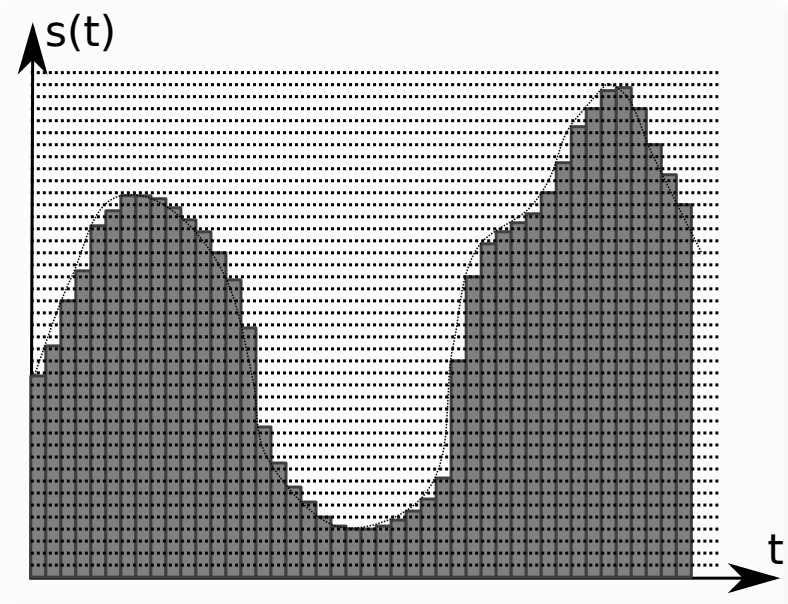
- Se toma un valor instantáneo de la señal y se “retiene” el valor de tensión en una capacidad, con un circuito resistivo de descarga de muy alta resistencia.
- *Comparando la señal saliente del circuito de Muestreo y Retención, con la señal original (en punteado suave), observar que el error aumenta en los cambios abruptos de señal.*
- Otra forma de observar la relación entre la frecuencia de muestreo y la máxima frecuencia de la señal original

Cuantificación y Codificación



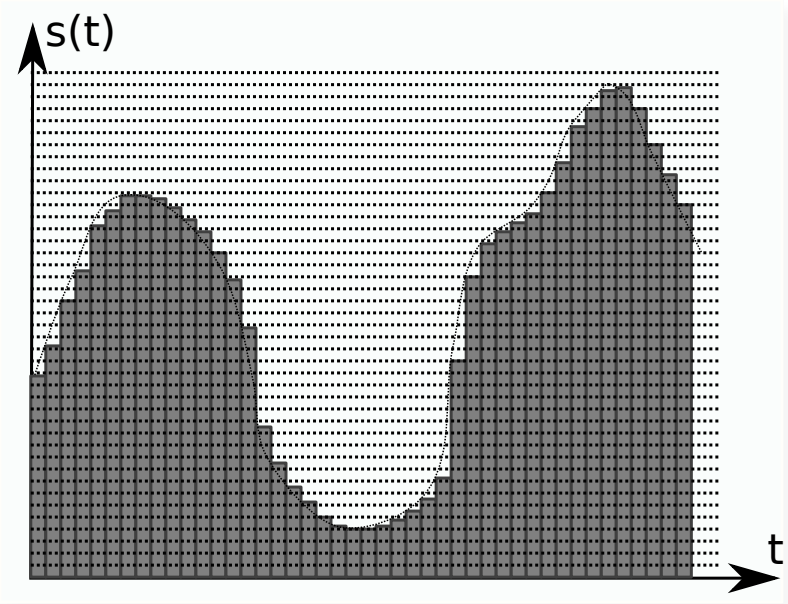
- Se llevan a cabo en un Conversor Analógico Digital.

Cuantificación y Codificación



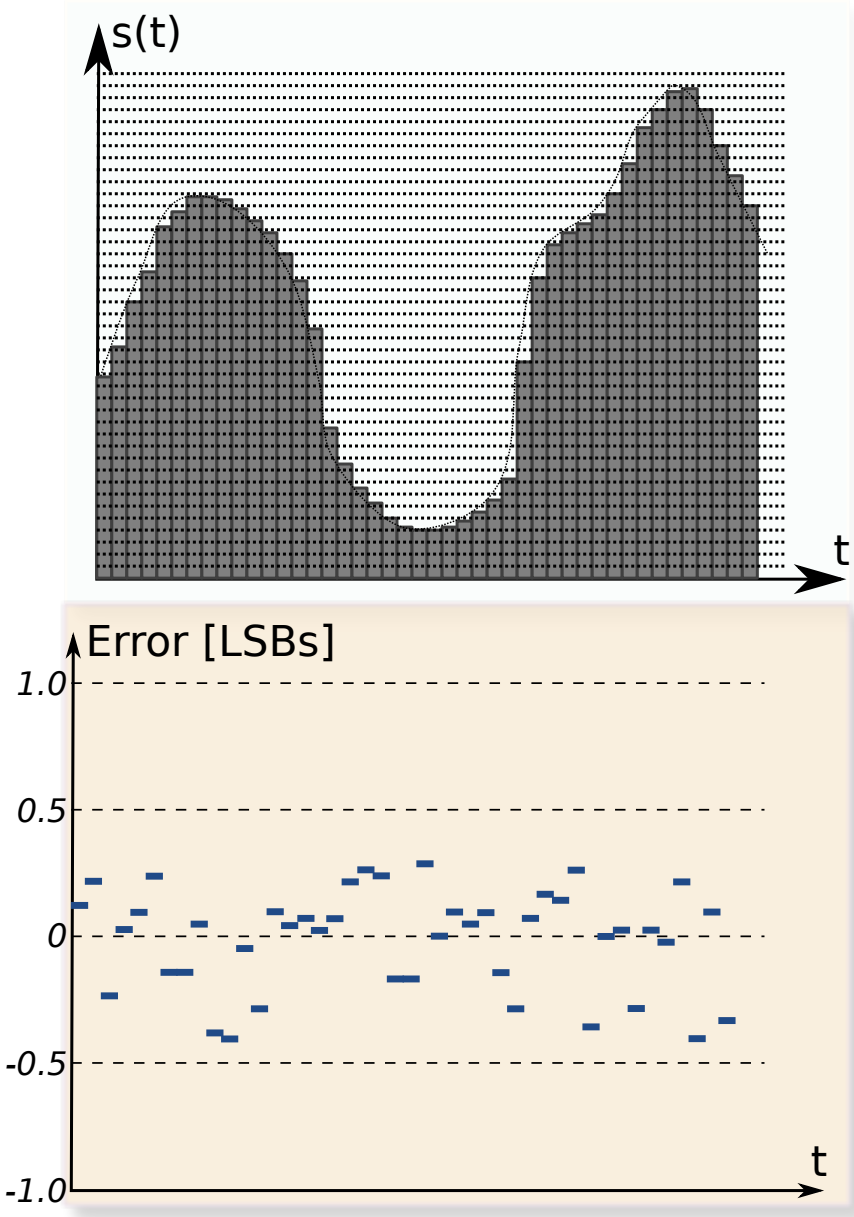
- Se llevan a cabo en un Conversor Analógico Digital.
- Se puede aproximar por redondeo o truncamiento

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 mas alta.

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 mas alta.
- Si tomamos la diferencia entre la señal muestreada y la cuantificada y operamos una diferencia se obtiene el error de Cuantificación o Cuantización)

Tabla de contenidos

Procesador de Señales Digitales

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.

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 bis). Se trata de pixeles de una imagen, o de valores instantáneos de audio, o de señales médicas o de mapas térmicos, etc.

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 bis). 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 N8)

Procesador de Señales Digitales

Procesador de Señales Digitales

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

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

Procesador de Señales Digitales

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

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

- En general se deben resolver sumas de productos y acumular su resultado.

Procesador de Señales Digitales

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

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

- En general se deben resolver sumas de productos y acumular su resultado.
- Para optimizar se deberían leer y procesar varios datos en paralelo

Procesador de Señales Digitales

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

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

- 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 Paralelism***.

Arquitectura de un Procesador de Señales Digitales

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:

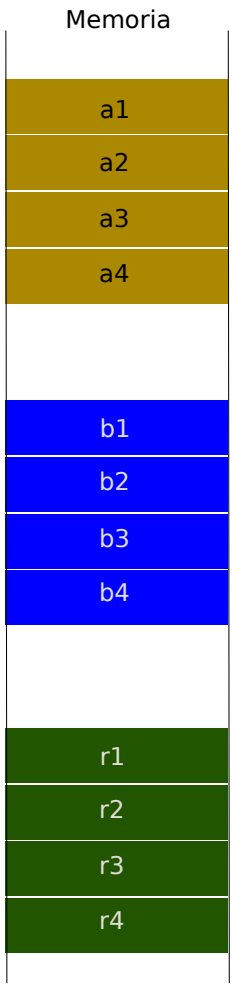
  El acceso a los operandos Para este fin se implementan buses paralelos con una cantidad de líneas de datos superior al ancho de palabra de la CPU

  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)

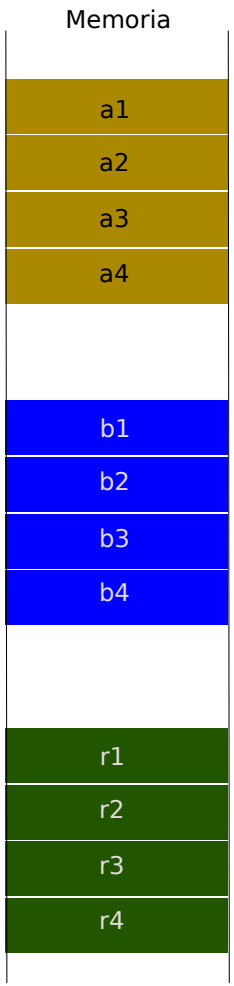
  El procesamiento de la mayor cantidad de datos posible Los primeros pasos se conocen como VLIW (Very Large Instruction Word), que derivó en el modelo de ejecución SIMD (Single Instruction Multiple Data)

Tabla de contenidos

Single Instruction Multiple Data

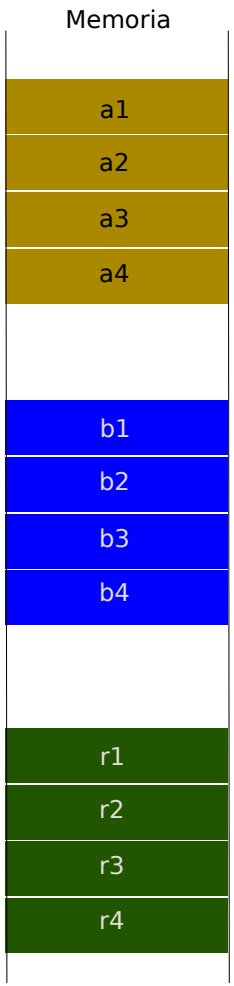


Single Instruction Multiple Data



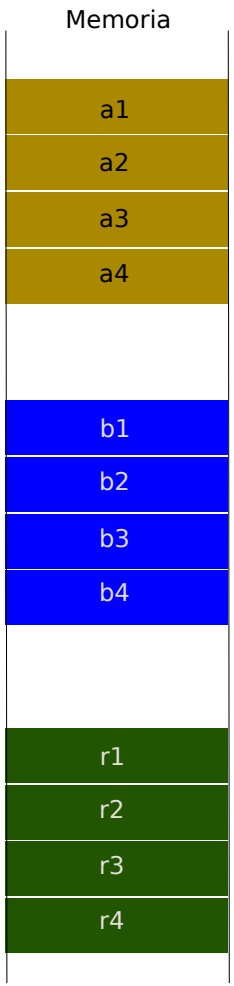
- Se trata de un modelo de ejecución capaz de computar una sola operación sobre un conjunto de múltiples datos.

Single Instruction Multiple Data



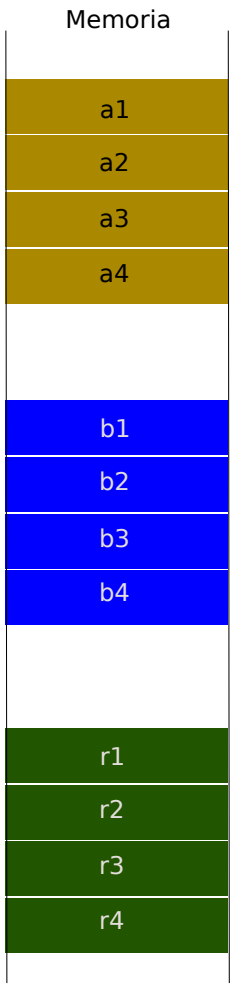
- Se trata de un modelo de ejecución capaz de computar una sola operación sobre un conjunto de múltiples datos.
- Se refiere a esta técnica como paralelismo a nivel de datos.

Single Instruction Multiple Data



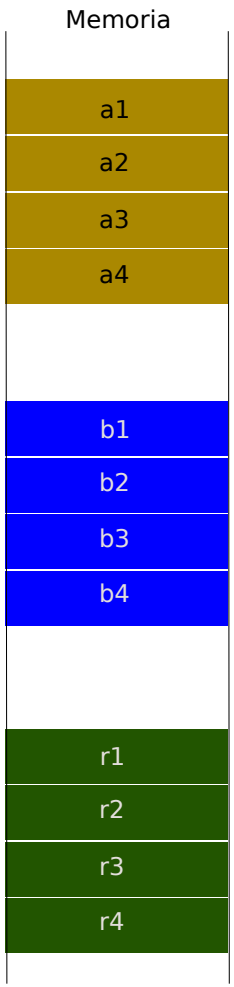
- Se trata de un modelo de ejecución capaz de computar una sola operación sobre un conjunto de múltiples datos.
- Se refiere a esta técnica como paralelismo a nivel de datos.
- Es particularmente útil para procesar audio, video, o imágenes en donde se aplican algoritmos repetitivos sobre sets de datos 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.

Single Instruction Multiple Data

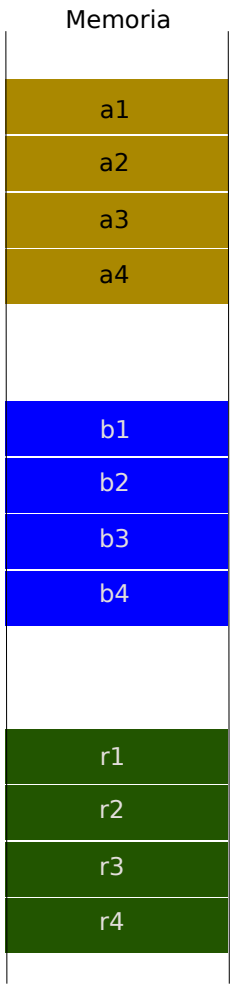


- Se trata de un modelo de ejecución capaz de computar una sola operación sobre un conjunto de múltiples datos.
- Se refiere a esta técnica como paralelismo a nivel de datos.
- Es particularmente útil para procesar audio, video, o imágenes en donde se aplican algoritmos repetitivos sobre sets de datos 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?

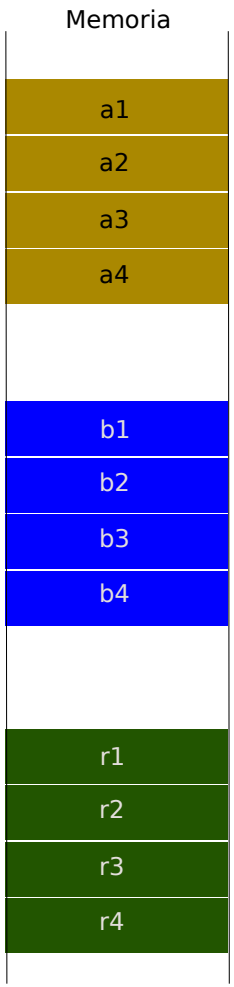


Como sería la vida sin SIMD?



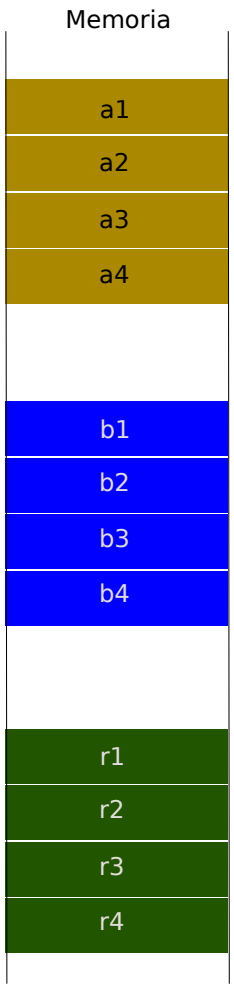
- En contraposición a SIMD el modelo previo es SISD (**S**ingle **I**nstruction **S**ingle **D**ata)

Como sería la vida sin SIMD?



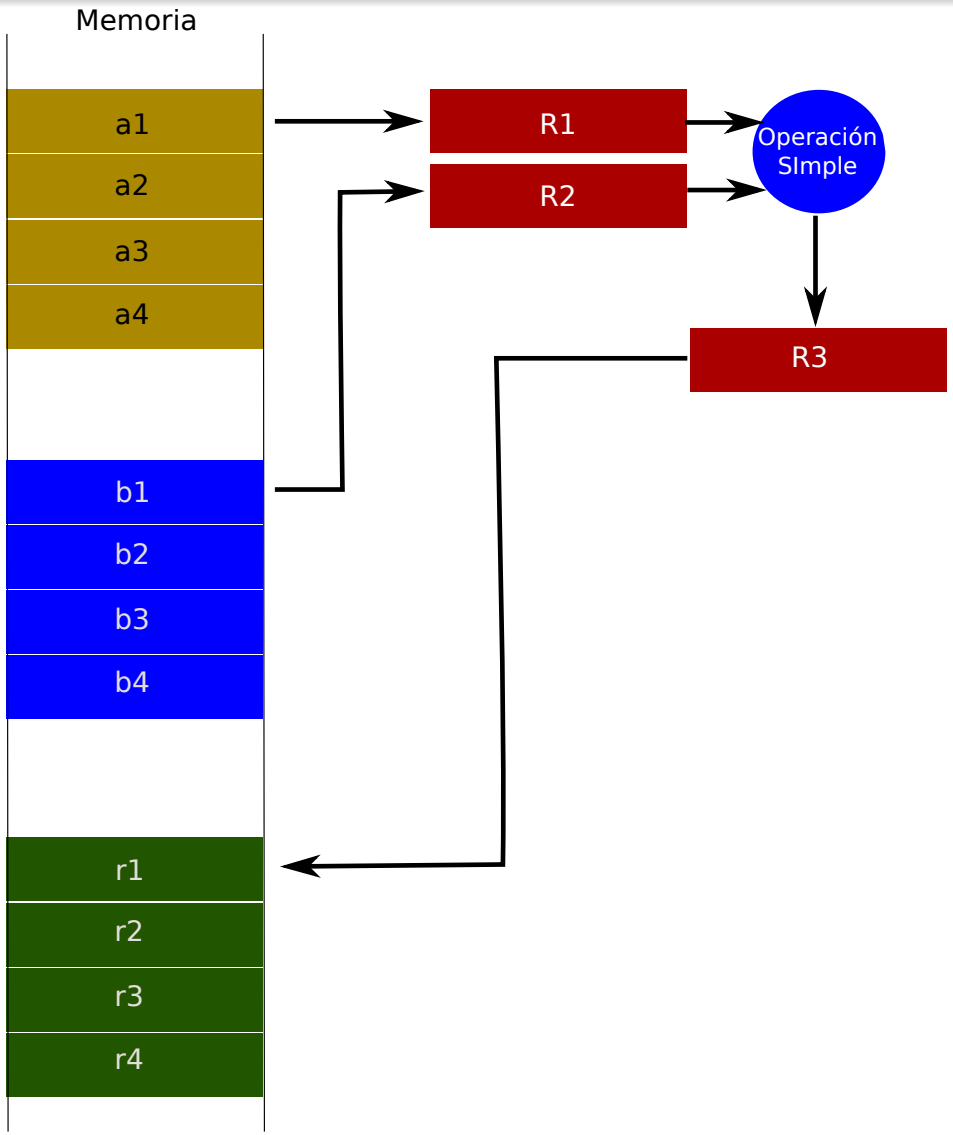
- En contraposición a SIMD el modelo previo es SISD (**S**ingle **I**nstruction **S**ingle **D**ata)
- Considerando el sector de memoria de la figura, nos proponemos realizar una operación aritmética o lógica sobre las cadenas de datos a_n y b_n, almacenando el resultado en r_n

Como sería la vida sin SIMD?

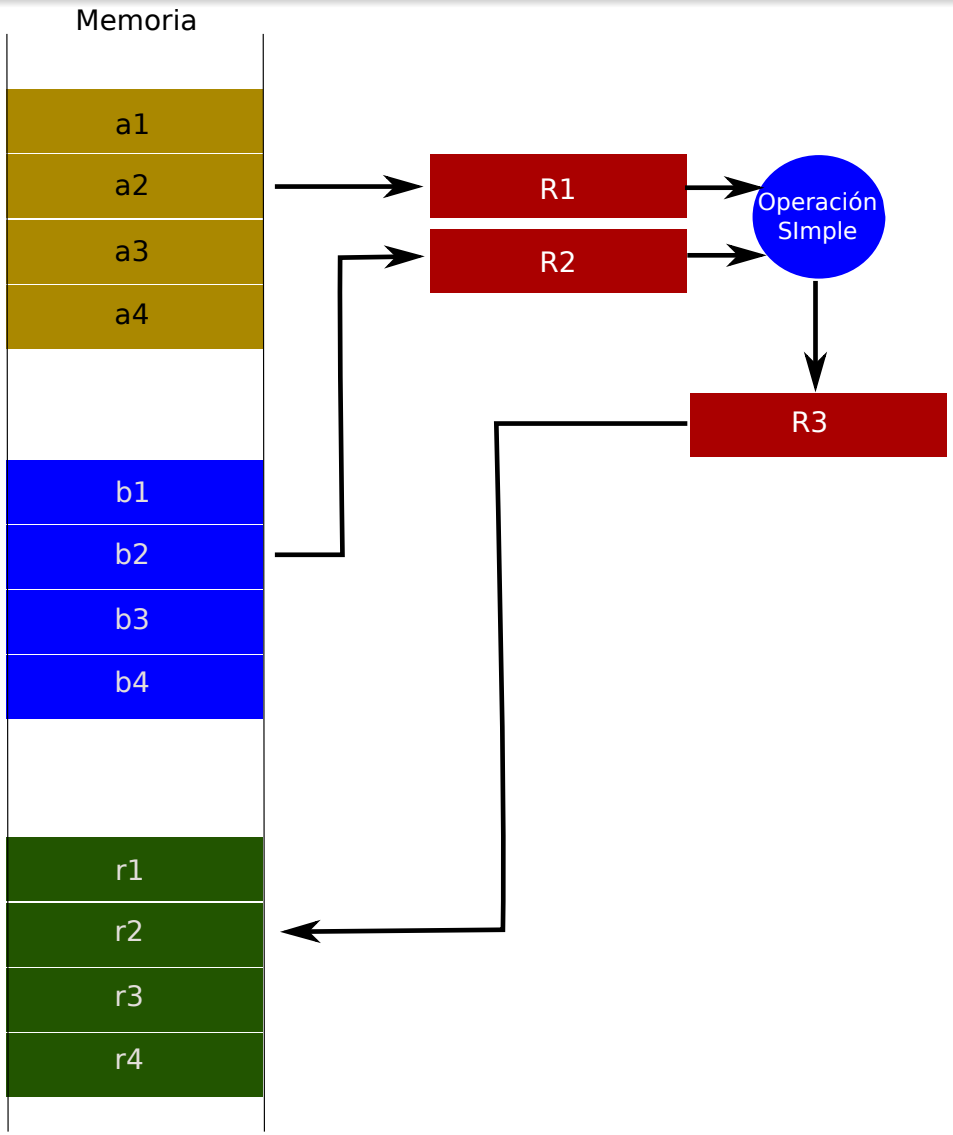


- En contraposición a SIMD el modelo previo es SISD (**S**ingle **I**nstruction **S**ingle **D**ata)
- Considerando el sector de memoria de la figura, nos proponemos realizar una operación aritmética o lógica sobre las cadenas de datos a_n y b_n, almacenando el resultado en r_n
- Si un procesador no 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).

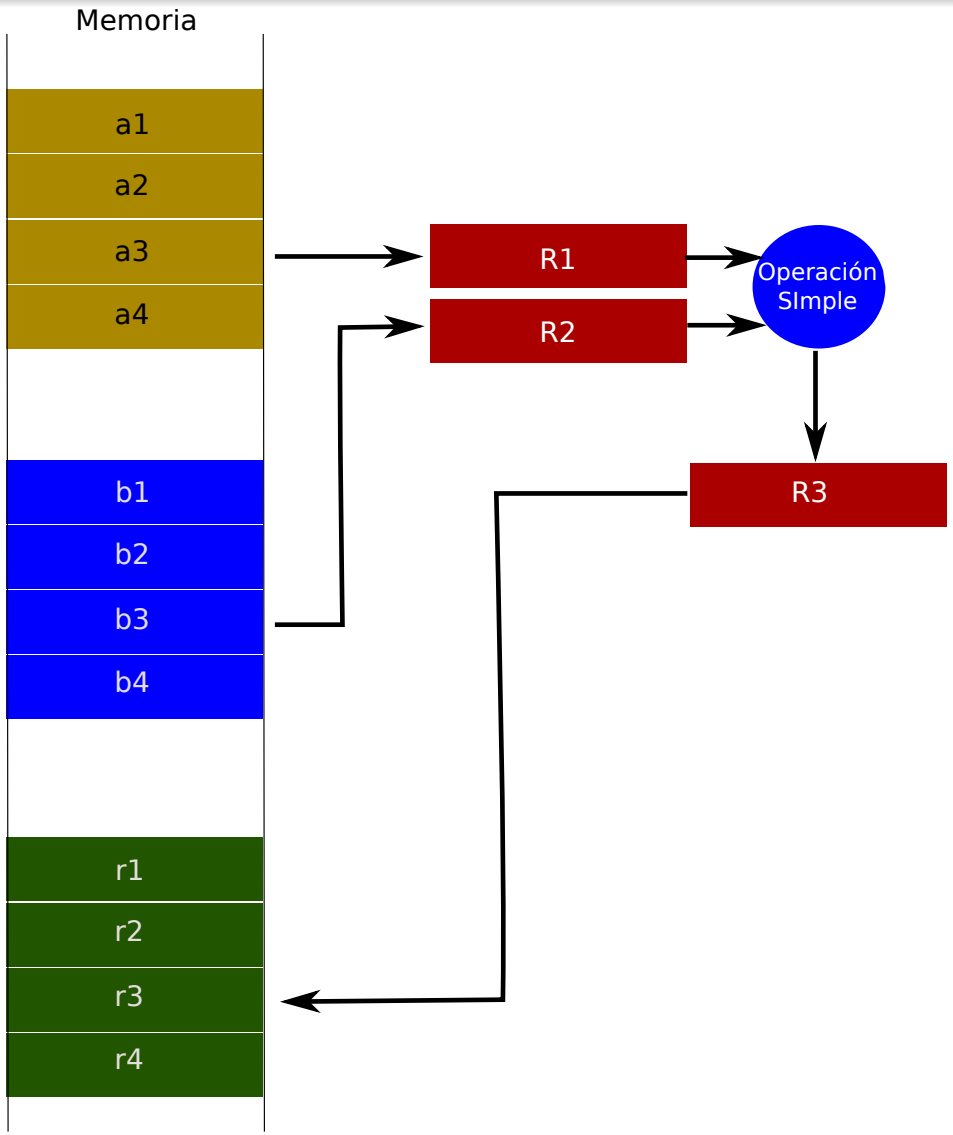
Single Instruction Single Data



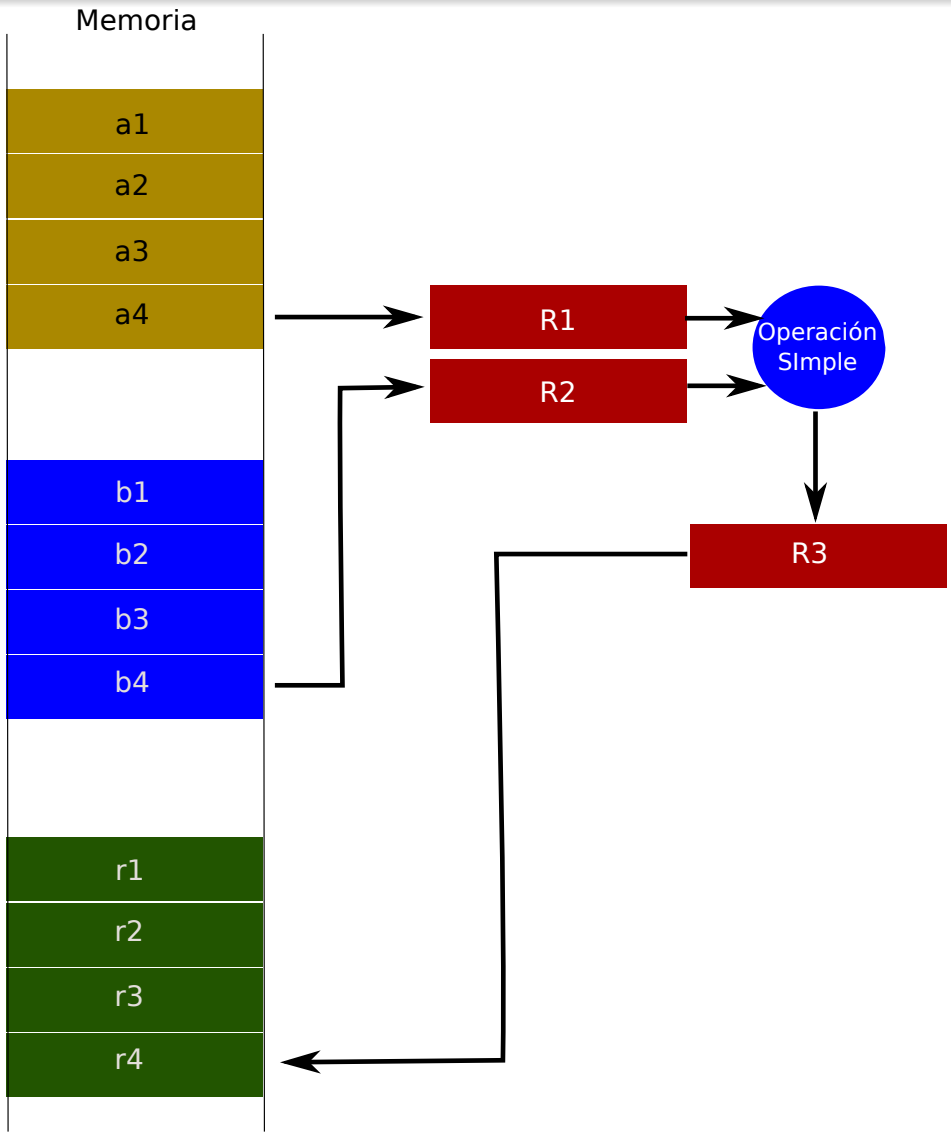
Single Instruction Single Data



Single Instruction Single Data

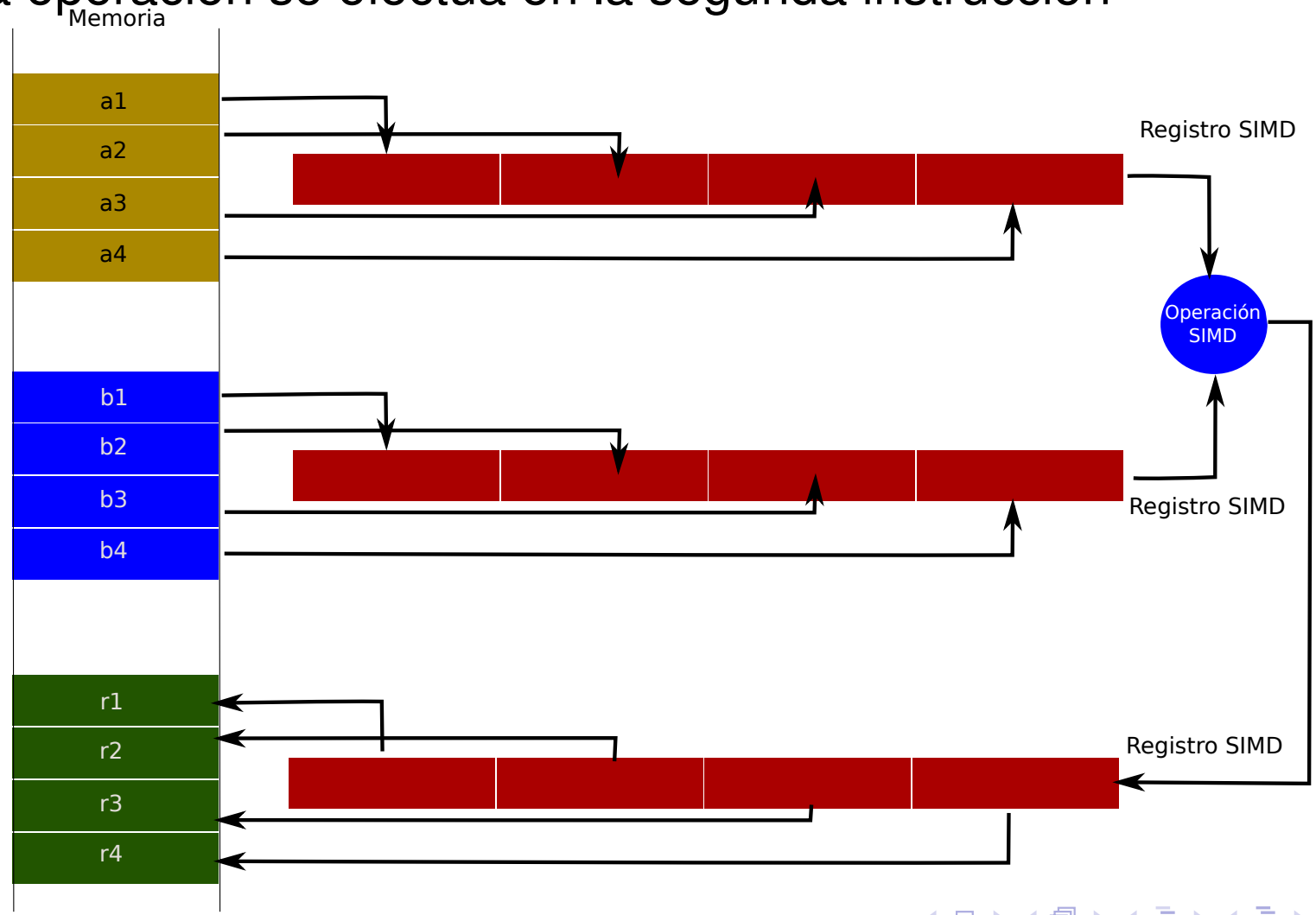


Single Instruction Single Data



Single Instruction Multiple Data

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



Resumiendo, SIMD vs. SISD

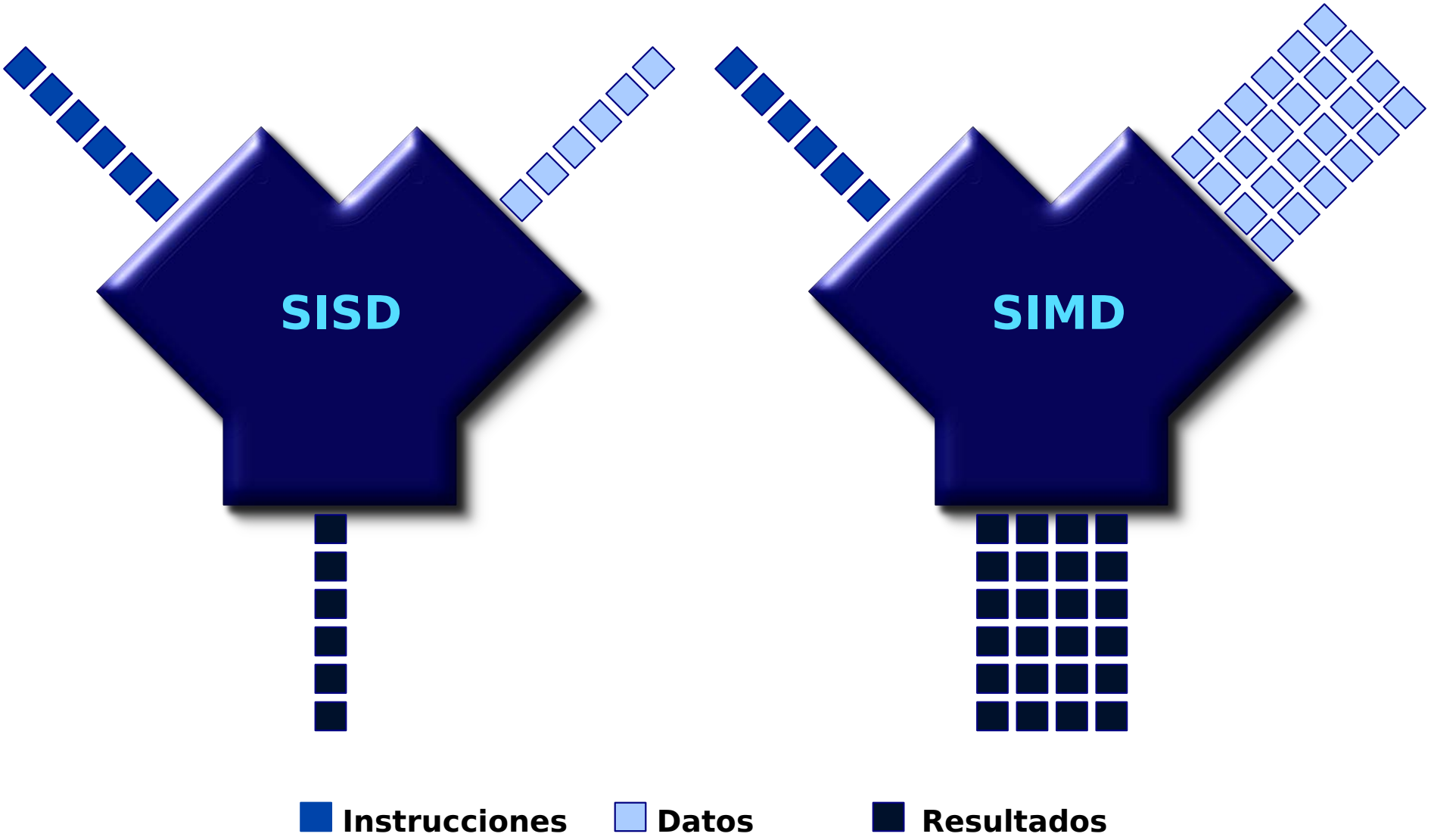
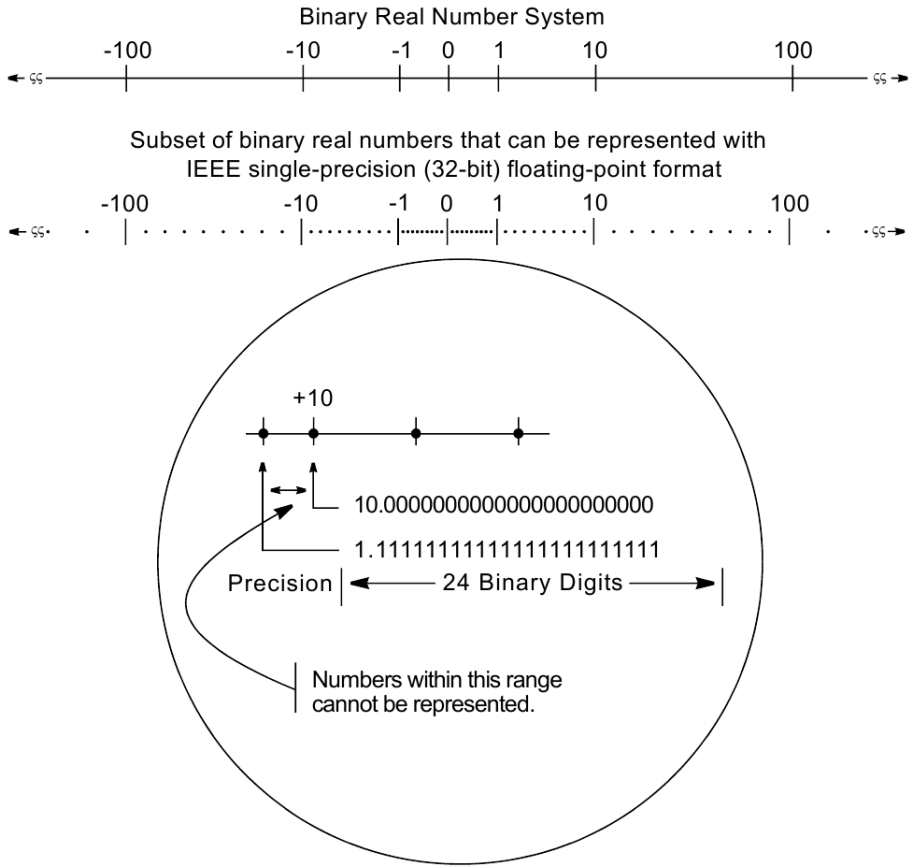


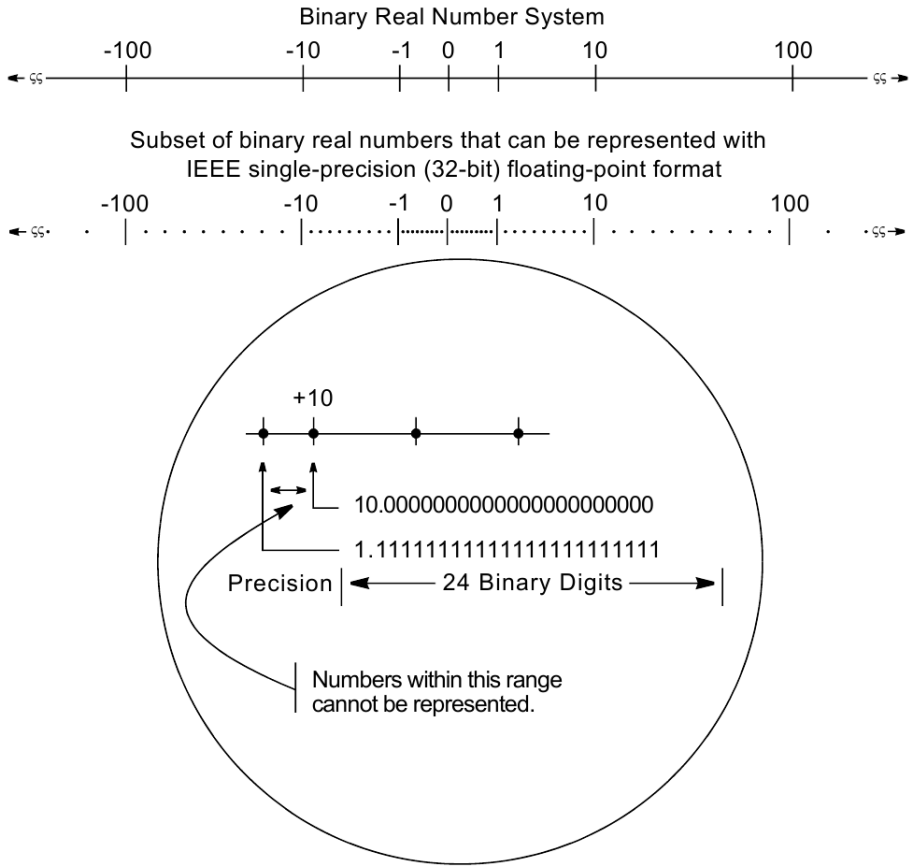
Tabla de contenidos

Números Reales



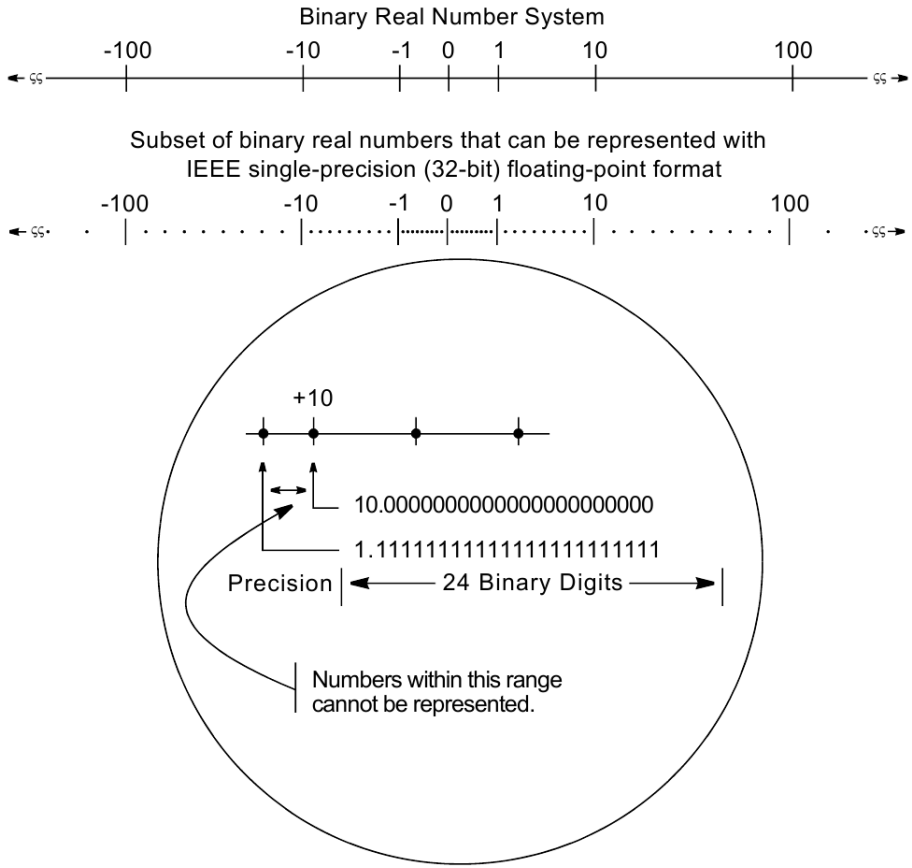
Números Reales

- El rango de los números reales comprende desde $-\infty$ hasta $+\infty$.



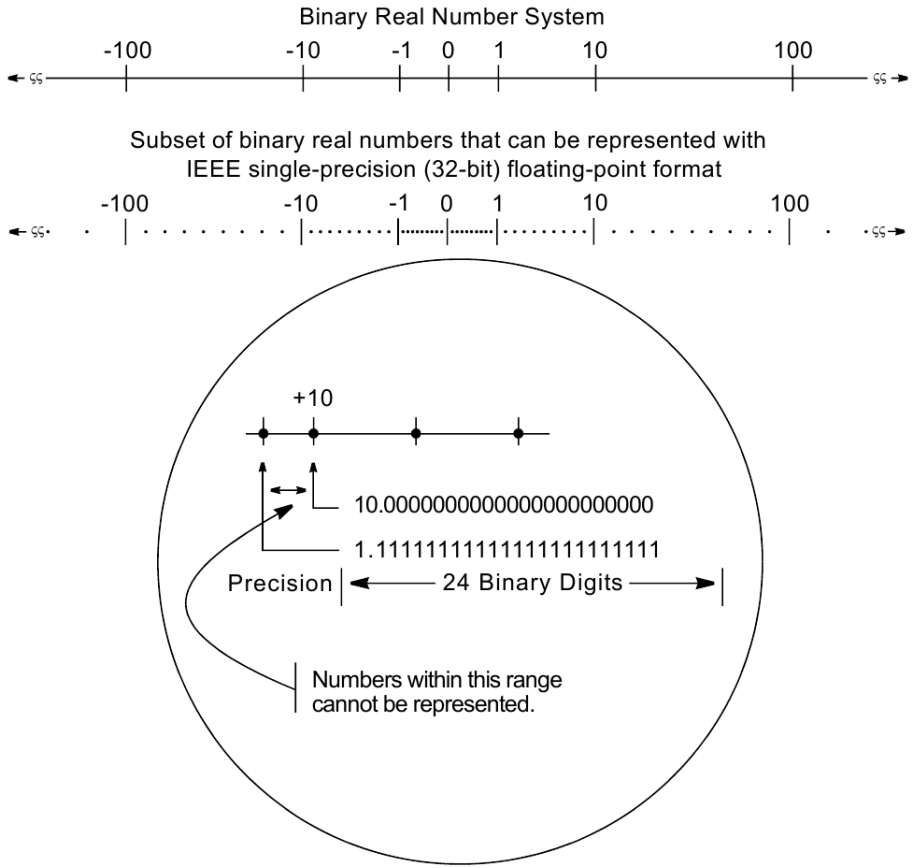
Números Reales

- El rango de los números reales comprende desde $-\infty$ hasta $+\infty$.
- Los registros de un procesador tienen resolución finita.



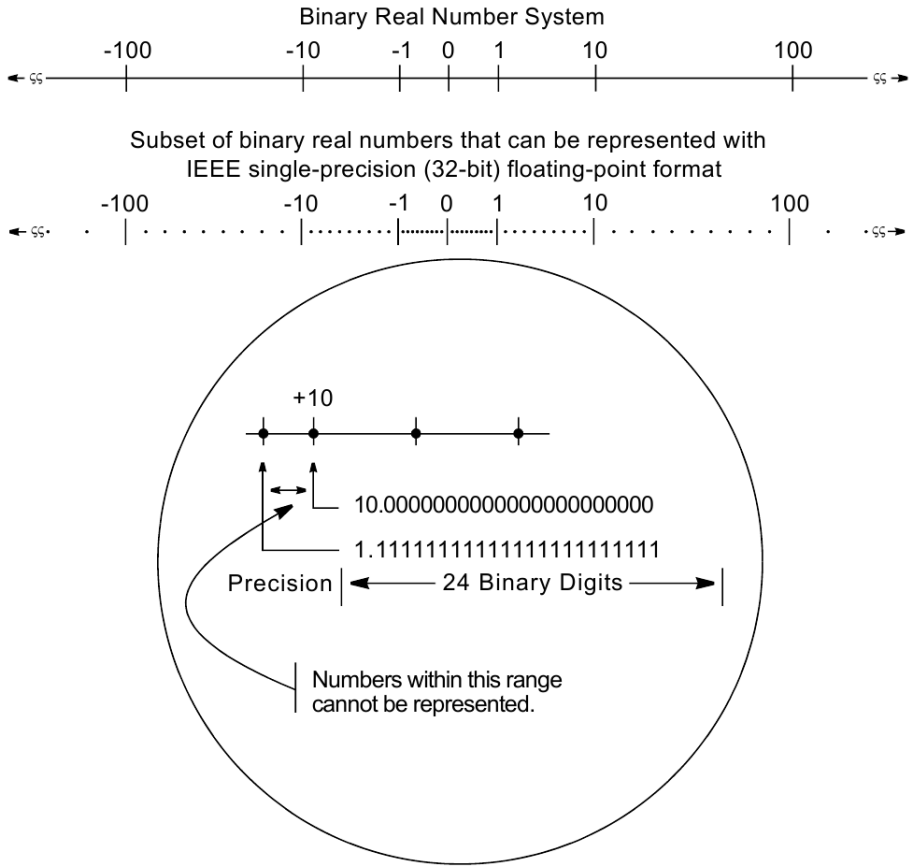
Números Reales

- 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 sub conjunto de $\Re$.



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 sub conjunto de $\Re$.
- Además, no es solo un tema de magnitud sino de resolución.



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:

$0{,}1 \leq \boldsymbol{f} < 1$,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:

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

Tabla de contenidos

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

Representación binaria en Punto Fijo con signo

Representación binaria en Punto Fijo con signo

- Se representan mediante una expresión del tipo:
  $(a_n a_{n-1} \ldots a_0 . a_{-1} a_{-2} \ldots a_{-m})_2 = (-1)^s * (a_n * 2^n + \cdots + a_0 * 2^0 + a_{-1} * 2^{-1} + a_{-2} * 2^{-2} + \cdots + a_{-m} * 2^{-m})$

Representación binaria en Punto Fijo con signo

- Se representan mediante una expresión del tipo:
  $(a_n a_{n-1} \ldots a_0 . a_{-1} a_{-2} \ldots a_{-m})_2 = (-1)^s * (a_n * 2^n + \cdots + a_0 * 2^0 + a_{-1} * 2^{-1} + a_{-2} * 2^{-2} + \cdots + a_{-m} * 2^{-m})$
- Donde:

Representación binaria en Punto Fijo con signo

- Se representan mediante una expresión del tipo:
  $(a_n a_{n-1} \ldots a_0 . a_{-1} a_{-2} \ldots a_{-m})_2 = (-1)^s * (a_n * 2^n + \cdots + a_0 * 2^0 + a_{-1} * 2^{-1} + a_{-2} * 2^{-2} + \cdots + a_{-m} * 2^{-m})$
- Donde:
  - ***s*** es el signo: **0** si el número es positivo y **1** si es negativo

Representación binaria en Punto Fijo con signo

- Se representan mediante una expresión del tipo:
  $(a_n a_{n-1} \ldots a_0 . a_{-1} a_{-2} \ldots a_{-m})_2 = (-1)^s * (a_n * 2^n + \cdots + a_0 * 2^0 + a_{-1} * 2^{-1} + a_{-2} * 2^{-2} + \cdots + a_{-m} * 2^{-m})$
- 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,. . . n*

Representación binaria en Punto Fijo con signo

- Se representan mediante una expresión del tipo:
  $(a_n a_{n-1} \ldots a_0 . a_{-1} a_{-2} \ldots a_{-m})_2 = (-1)^s * (a_n * 2^n + \cdots + a_0 * 2^0 + a_{-1} * 2^{-1} + a_{-2} * 2^{-2} + \cdots + a_{-m} * 2^{-m})$
- 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,. . . n*
- Distancia entre dos números consecutivos es 2^{-m}.

Representación binaria en Punto Fijo con signo

- Se representan mediante una expresión del tipo:
  $(a_n a_{n-1} \ldots a_0 . a_{-1} a_{-2} \ldots a_{-m})_2 = (-1)^s * (a_n * 2^n + \cdots + a_0 * 2^0 + a_{-1} * 2^{-1} + a_{-2} * 2^{-2} + \cdots + a_{-m} * 2^{-m})$
- 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,. . . n*
- Distancia entre dos números consecutivos es 2^{-m}.
- *Deja de ser un rango continuo de números para transformarse en un rango discreto.*

Representación en Punto Flotante

Representación en Punto Flotante

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

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

Representación en Punto Flotante

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

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

- En donde:

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.

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.

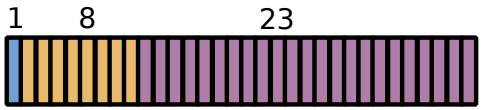
Representación en Punto Flotante

- 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).

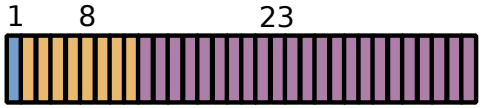
Formatos IEEE 754



Single Precision

	Rango	
	Binario	**Decimal**
	2^{-126} **a** 2^{127}	1.18×10^{-38} **a** 3.40×10^{38}

Formatos IEEE 754



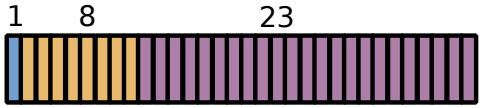
Single Precision

	Rango	
	Binario	**Decimal**
	2^{-126} **a** 2^{127}	1.18×10^{-38} **a** 3.40×10^{38}

Double Precision

	Rango	
	Binario	**Decimal**
	2^{-1022} **a** 2^{1023}	2.23×10^{-308} **a** 1.79×10^{308}

Formatos IEEE 754



Single Precision

	Rango Binario	Rango Decimal
	2^{-126} a 2^{127}	$1.18x10^{-38}$ a $3.40x10^{38}$

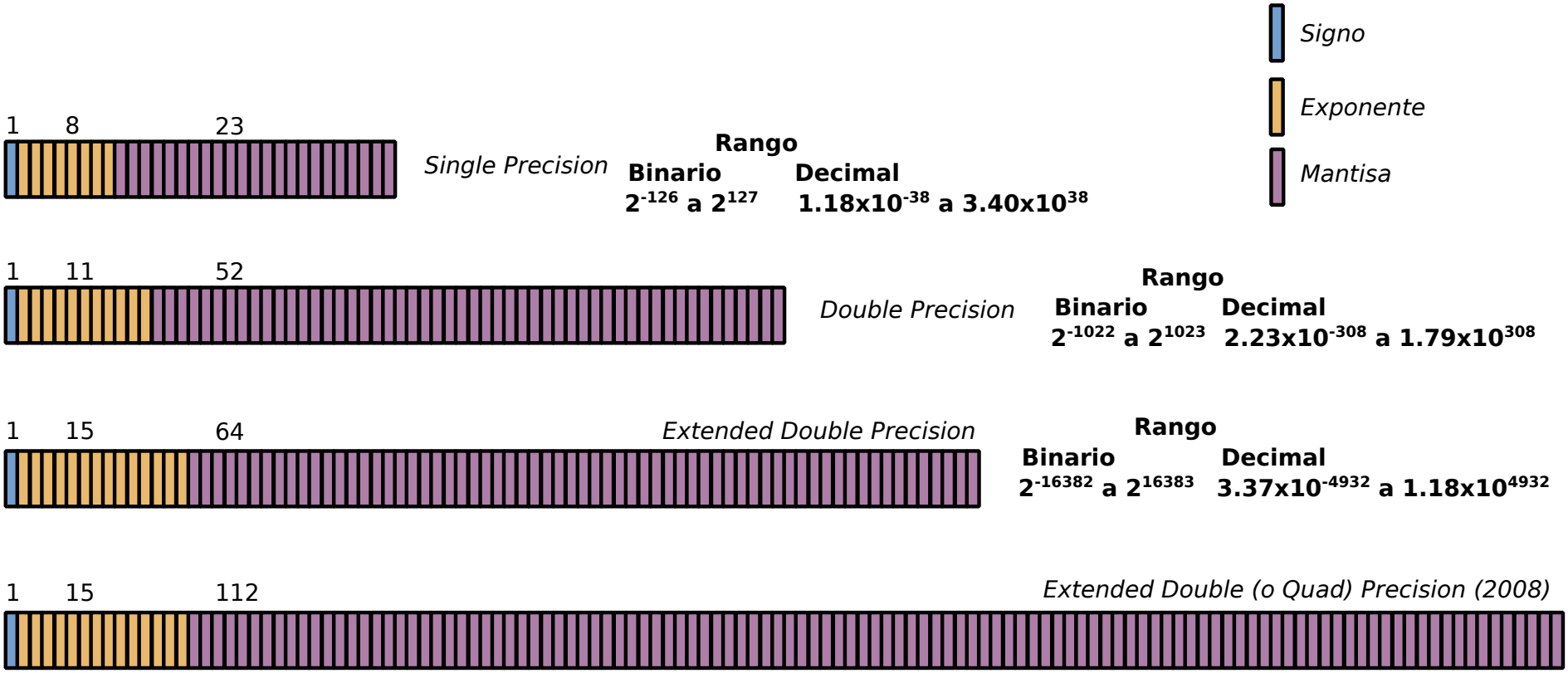
Double Precision

	Rango Binario	Rango Decimal
	2^{-1022} a 2^{1023}	$2.23x10^{-308}$ a $1.79x10^{308}$

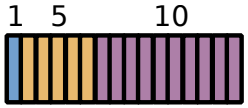
Extended Double Precision

	Rango Binario	Rango Decimal
	2^{-16382} a 2^{16383}	$3.37x10^{-4932}$ a $1.18x10^{4932}$

Formatos IEEE 754

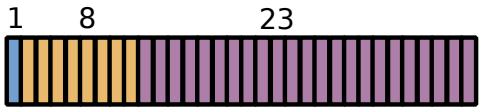


Formatos IEEE 754



Half Precision (2008)

	Rango Binario	Rango Decimal
	2^{-15} a 2^{16}	$3.05x10^{-5}$ a $6.55x10^{4}$



Single Precision

	Rango Binario	Rango Decimal
	2^{-126} a 2^{127}	$1.18x10^{-38}$ a $3.40x10^{38}$

Double Precision

	Rango Binario	Rango Decimal
	2^{-1022} a 2^{1023}	$2.23x10^{-308}$ a $1.79x10^{308}$

Extended Double Precision

	Rango Binario	Rango Decimal
	2^{-16382} a 2^{16383}	$3.37x10^{-4932}$ a $1.18x10^{4932}$

Extended Double (o Quad) Precision (2008)

Signo

Exponente

Mantisa

Tabla de contenidos

Introducción

Introducción

Los gráficos anteriores en instrucciones de procesadores ARM se traducen muy fácilmente en código.

Introducción

Los gráficos anteriores en instrucciones de procesadores ARM se traducen muy fácilmente en código.

SISD

```
add    r0,r5
add    r1,r6
add    r2,r7
add    r3,r8
```

Introducción

Los gráficos anteriores en instrucciones de procesadores ARM se traducen muy fácilmente en código.

SISD

```
add   r0,r5
add   r1,r6
add   r2,r7
add   r3,r8
```

SIMD (Modo empaquetado)

```
VADD.I32 Q10, Q8, Q9
// Una suma de dos registros de 128bit,
// Cada vía de 32bit de los registros se suma
    por separado.
// No hay carry entre las vías.
```

Introducción

Los gráficos anteriores en instrucciones de procesadores ARM se traducen muy fácilmente en código.

SISD

```
add   r0,r5
add   r1,r6
add   r2,r7
add   r3,r8
```

SIMD (Modo empaquetado)

```
VADD.I32 Q10, Q8, Q9
// Una suma de dos registros de 128bit,
// Cada vía de 32bit de los registros se suma
     por separado.
// No hay carry entre las vías.
```

- ARMv5 introduce las extensiones denominadas **VFP** (Por**V**irtual **F**loating **P**oint)

Introducción

Los gráficos anteriores en instrucciones de procesadores ARM se traducen muy fácilmente en código.

SISD

```
add   r0,r5
add   r1,r6
add   r2,r7
add   r3,r8
```

SIMD (Modo empaquetado)

```
VADD.I32 Q10, Q8, Q9
// Una suma de dos registros de 128bit,
// Cada vía de 32bit de los registros se suma
     por separado.
// No hay carry entre las vías.
```

- ARMv5 introduce las extensiones denominadas **VFP** (Por**V**irtual **F**loating **P**oint)
- Incluían operaciones vectoriales con los registros de punto flotante, utilizando para el ejemplo anterior la misma instrucción con la única diferencia en el *DataType* (`F32` en lugar de `I32`).

Introducción

Los gráficos anteriores en instrucciones de procesadores ARM se traducen muy fácilmente en código.

SISD

```
add   r0,r5
add   r1,r6
add   r2,r7
add   r3,r8
```

SIMD (Modo empaquetado)

```
VADD.I32 Q10, Q8, Q9
// Una suma de dos registros de 128bit,
// Cada vía de 32bit de los registros se suma
     por separado.
// No hay carry entre las vías.
```

- ARMv5 introduce las extensiones denominadas **VFP** (Por**V**irtual **F**loating **P**oint)
- Incluían operaciones vectoriales con los registros de punto flotante, utilizando para el ejemplo anterior la misma instrucción con la única diferencia en el *DataType* (`F32` en lugar de `I32`).
- La implementación SIMD en ARMv7 perfiles A y R, se denomina NEON. Y comparte los mismos registros que las **VFP**.

Introducción

- Las instrucciones vectoriales de punto flotante ejecutan una secuencia de operaciones que involucran un grupo de registros (cantidad definida en un registro de configuración).

Introducción

- Las instrucciones vectoriales de punto flotante ejecutan una secuencia de operaciones que involucran un grupo de registros (cantidad definida en un registro de configuración).

SIMD (Modo Vector) (Obsoleto)

```
VADD.F32 S24, S8, S16
// Se ejecutan cuatro operaciones:
// S24 = S8 + S16
// S25 = S9 + S17
// S26 = S10 + S18
// S27 = S11 + S19
```

Introducción

- Las instrucciones vectoriales de punto flotante ejecutan una secuencia de operaciones que involucran un grupo de registros (cantidad definida en un registro de configuración).

SIMD (Modo Vector) (Obsoleto)

```
VADD.F32 S24, S8, S16
// Se ejecutan cuatro operaciones:
// S24 = S8 + S16
// S25 = S9 + S17
// S26 = S10 + S18
// S27 = S11 + S19
```

- SIMD no es una secuencia, sino una única operación. Por eso en ARMv7 las operaciones vectoriales de punto flotante se reemplazan por SIMD empaquetadas del slide anterior. Es decir NEON.

Introducción

- Las instrucciones vectoriales de punto flotante ejecutan una secuencia de operaciones que involucran un grupo de registros (cantidad definida en un registro de configuración).

SIMD (Modo Vector) (Obsoleto)

```
VADD.F32 S24, S8, S16
// Se ejecutan cuatro operaciones:
// S24 = S8 + S16
// S25 = S9 + S17
// S26 = S10 + S18
// S27 = S11 + S19
```

- SIMD no es una secuencia, sino una única operación. Por eso en ARMv7 las operaciones vectoriales de punto flotante se reemplazan por SIMD empaquetadas del slide anterior. Es decir NEON.
- ARMv8, extiende NEON y lo mantiene compatible con ARMv7 A y R.

Tabla de contenidos

Cuestiones de Implementación

- Mas bien una colección de malas noticias:

Cuestiones de Implementación

- Mas bien una colección de malas noticias:
- ARM no garantiza que todos los cores incluyan NEON y VFP. Podemos encontrarnos con procesadores ARM que incluyan una de las dos o las dos.

Cuestiones de Implementación

- Mas bien una colección de malas noticias:
- ARM no garantiza que todos los cores incluyan NEON y VFP. Podemos encontrarnos con procesadores ARM que incluyan una de las dos o las dos.
- Si un core incluye NEON, pero no incluye VFP, no puede ejecutar instrucciones de punto flotante en hardware.

Cuestiones de Implementación

- Mas bien una colección de malas noticias:
- ARM no garantiza que todos los cores incluyan NEON y VFP. Podemos encontrarnos con procesadores ARM que incluyan una de las dos o las dos.
- Si un core incluye NEON, pero no incluye VFP, no puede ejecutar instrucciones de punto flotante en hardware.
- Al ser NEON mas eficiente en el calculo vectorial, las instrucciones vectoriales de las VFP se discontinúan en los cores v7 con NEON incluido. Por tal razón la documentación técnica de ARM a menudo se refiere a la VFP como FPU (Por Floating Point Unit).

Cuestiones de Implementación

- Mas bien una colección de malas noticias:
- ARM no garantiza que todos los cores incluyan NEON y VFP. Podemos encontrarnos con procesadores ARM que incluyan una de las dos o las dos.
- Si un core incluye NEON, pero no incluye VFP, no puede ejecutar instrucciones de punto flotante en hardware.
- Al ser NEON mas eficiente en el calculo vectorial, las instrucciones vectoriales de las VFP se discontinúan en los cores v7 con NEON incluido. Por tal razón la documentación técnica de ARM a menudo se refiere a la VFP como FPU (Por Floating Point Unit).
- El soporte a Half-Presicion y Multiplicación-Suma Fusionada depende de cada implementación de core.

Cuestiones de Implementación

- Mas bien una colección de malas noticias:
- ARM no garantiza que todos los cores incluyan NEON y VFP. Podemos encontrarnos con procesadores ARM que incluyan una de las dos o las dos.
- Si un core incluye NEON, pero no incluye VFP, no puede ejecutar instrucciones de punto flotante en hardware.
- Al ser NEON mas eficiente en el calculo vectorial, las instrucciones vectoriales de las VFP se discontinúan en los cores v7 con NEON incluido. Por tal razón la documentación técnica de ARM a menudo se refiere a la VFP como FPU (Por Floating Point Unit).
- El soporte a Half-Presicion y Multiplicación-Suma Fusionada depende de cada implementación de core.
- Una misma instrucción varía su tiempo de ejecución dependiendo del core en el que se implementa, y aun en el mismo core depende de la cacheabilidad lograda con los datos con que se opera.

Cuestiones de Implementación

- NEON y VFP comparten algunas instrucciones. Se las llama Shared Instructions.

Cuestiones de Implementación

- NEON y VFP comparten algunas instrucciones. Se las llama Shared Instructions.
- Si el core soporta ambas extensiones, los registros de ambas se solapan, las instrucciones VFP se ejecutan en la unidad de hardware de la VFP, pero las Shared, se ejecutan en el pipeline de Punto Flotante de NEON cuya eficiencia es mayor.

Cuestiones de Implementación

- NEON y VFP comparten algunas instrucciones. Se las llama Shared Instructions.
- Si el core soporta ambas extensiones, los registros de ambas se solapan, las instrucciones VFP se ejecutan en la unidad de hardware de la VFP, pero las Shared, se ejecutan en el pipeline de Punto Flotante de NEON cuya eficiencia es mayor.
- VFP y NEON ejecutan en el espacio de instrucciones de los coprocesadores CP10 y CP11.

Cuestiones de Implementación

- NEON y VFP comparten algunas instrucciones. Se las llama Shared Instructions.
- Si el core soporta ambas extensiones, los registros de ambas se solapan, las instrucciones VFP se ejecutan en la unidad de hardware de la VFP, pero las Shared, se ejecutan en el pipeline de Punto Flotante de NEON cuya eficiencia es mayor.
- VFP y NEON ejecutan en el espacio de instrucciones de los coprocesadores CP10 y CP11.
- VFP implementa operaciones de punto flotante 100 % compatibles con IEEE-754, precisión simple. Dependiendo de la implementación puede soportar doble precisión.

Cuestiones de Implementación

- NEON y VFP comparten algunas instrucciones. Se las llama Shared Instructions.
- Si el core soporta ambas extensiones, los registros de ambas se solapan, las instrucciones VFP se ejecutan en la unidad de hardware de la VFP, pero las Shared, se ejecutan en el pipeline de Punto Flotante de NEON cuya eficiencia es mayor.
- VFP y NEON ejecutan en el espacio de instrucciones de los coprocesadores CP10 y CP11.
- VFP implementa operaciones de punto flotante 100 % compatibles con IEEE-754, precisión simple. Dependiendo de la implementación puede soportar doble precisión.
- NEON procesa en una sola instrucción datos enteros o punto flotante precisión simple empaquetados. En este último caso no se asegura compatibilidad completa con IEEE-754. No opera en doble precisión.

Cuestiones de Implementación

- NEON no reemplaza a VFP. De hecho VFP incluye operaciones que no poseen su correspondiente empaquetada en NEON.

Cuestiones de Implementación

- NEON no reemplaza a VFP. De hecho VFP incluye operaciones que no poseen su correspondiente empaquetada en NEON.
- Half-Precision solo esta soportado por una extensión específica core dependiente.

Cuestiones de Implementación

- NEON no reemplaza a VFP. De hecho VFP incluye operaciones que no poseen su correspondiente empaquetada en NEON.
- Half-Precision solo esta soportado por una extensión específica core dependiente.
- Si están las extensiones Half-Precision presentes en el core, entonces tanto VFP como NEON manejarán ese tipo de datos y disponen de instrucciones de conversión de Single a Half Precision.

Cuestiones de Implementación

- NEON no reemplaza a VFP. De hecho VFP incluye operaciones que no poseen su correspondiente empaquetada en NEON.
- Half-Precision solo esta soportado por una extensión específica core dependiente.
- Si están las extensiones Half-Precision presentes en el core, entonces tanto VFP como NEON manejarán ese tipo de datos y disponen de instrucciones de conversión de Single a Half Precision.
- NEON soporta operaciones de Suma de productos (Fused Multiply-Add) se soportan mediante una extensión core dependiente.

Cuestiones de Implementación

- NEON no reemplaza a VFP. De hecho VFP incluye operaciones que no poseen su correspondiente empaquetada en NEON.
- Half-Precision solo esta soportado por una extensión específica core dependiente.
- Si están las extensiones Half-Precision presentes en el core, entonces tanto VFP como NEON manejarán ese tipo de datos y disponen de instrucciones de conversión de Single a Half Precision.
- NEON soporta operaciones de Suma de productos (Fused Multiply-Add) se soportan mediante una extensión core dependiente.
- Estas operaciones implican en una misma instrucción productos y acumulaciones con un solo paso de redondeo. Es de esperar una pérdida de precisión.

Cuestiones de Implementación

- NEON no reemplaza a VFP. De hecho VFP incluye operaciones que no poseen su correspondiente empaquetada en NEON.
- Half-Precision solo esta soportado por una extensión específica core dependiente.
- Si están las extensiones Half-Precision presentes en el core, entonces tanto VFP como NEON manejarán ese tipo de datos y disponen de instrucciones de conversión de Single a Half Precision.
- NEON soporta operaciones de Suma de productos (Fused Multiply-Add) se soportan mediante una extensión core dependiente.
- Estas operaciones implican en una misma instrucción productos y acumulaciones con un solo paso de redondeo. Es de esperar una pérdida de precisión.
- Si NEON y VFP no se habilitan en el Registro CPACR del coprocesador CP15, cualquier instrucción generará una excepción Undefined Instruction.

Tabla de contenidos

Cuestiones de Implementación

- ARMv7 es una arquitectura de 32 bits, sin embargo NEON implementa registros de 64 bit y 128 bit

Cuestiones de Implementación

- ARMv7 es una arquitectura de 32 bits, sin embargo NEON implementa registros de 64 bit y 128 bit
- Los componentes de hardware de una unidad NEON son:

Cuestiones de Implementación

- ARMv7 es una arquitectura de 32 bits, sin embargo NEON implementa registros de 64 bit y 128 bit
- Los componentes de hardware de una unidad NEON son:
  - Registros NEON

Cuestiones de Implementación

- ARMv7 es una arquitectura de 32 bits, sin embargo NEON implementa registros de 64 bit y 128 bit
- Los componentes de hardware de una unidad NEON son:
  - Registros NEON
  - Un pipeline dedicado para operaciones de datos enteros empaquetados.

Cuestiones de Implementación

- ARMv7 es una arquitectura de 32 bits, sin embargo NEON implementa registros de 64 bit y 128 bit
- Los componentes de hardware de una unidad NEON son:
  - Registros NEON
  - Un pipeline dedicado para operaciones de datos enteros empaquetados.
  - Un pipeline dedicado para operaciones de datos en Punto Flotante Precisión Simple empaquetados.

Cuestiones de Implementación

- ARMv7 es una arquitectura de 32 bits, sin embargo NEON implementa registros de 64 bit y 128 bit
- Los componentes de hardware de una unidad NEON son:
  - Registros NEON
  - Un pipeline dedicado para operaciones de datos enteros empaquetados.
  - Un pipeline dedicado para operaciones de datos en Punto Flotante Precisión Simple empaquetados.
  - Un pipeline dedicado para operaciones load/store y permutación de datos empaquetados.

Cuestiones de Implementación

- ARMv7 es una arquitectura de 32 bits, sin embargo NEON implementa registros de 64 bit y 128 bit
- Los componentes de hardware de una unidad NEON son:
  - Registros NEON
  - Un pipeline dedicado para operaciones de datos enteros empaquetados.
  - Un pipeline dedicado para operaciones de datos en Punto Flotante Precisión Simple empaquetados.
  - Un pipeline dedicado para operaciones load/store y permutación de datos empaquetados.
- Las instrucciones NEON y las Floating Point comparten el mismo pack de registros (slides siguientes).

Cuestiones de Implementación

- ARMv7 es una arquitectura de 32 bits, sin embargo NEON implementa registros de 64 bit y 128 bit
- Los componentes de hardware de una unidad NEON son:
  - Registros NEON
  - Un pipeline dedicado para operaciones de datos enteros empaquetados.
  - Un pipeline dedicado para operaciones de datos en Punto Flotante Precisión Simple empaquetados.
  - Un pipeline dedicado para operaciones load/store y permutación de datos empaquetados.
- Las instrucciones NEON y las Floating Point comparten el mismo pack de registros (slides siguientes).
- De acuerdo con el tipo de instrucción la visión de esos registros es como registros de 32 bit, 64 bit, o 128 bit.

Cuestiones de Implementación

- ARMv7 es una arquitectura de 32 bits, sin embargo NEON implementa registros de 64 bit y 128 bit
- Los componentes de hardware de una unidad NEON son:
  - Registros NEON
  - Un pipeline dedicado para operaciones de datos enteros empaquetados.
  - Un pipeline dedicado para operaciones de datos en Punto Flotante Precisión Simple empaquetados.
  - Un pipeline dedicado para operaciones load/store y permutación de datos empaquetados.
- Las instrucciones NEON y las Floating Point comparten el mismo pack de registros (slides siguientes).
- De acuerdo con el tipo de instrucción la visión de esos registros es como registros de 32 bit, 64 bit, o 128 bit.
- Un registro NEON contiene un *vector de elementos*, del mismo *tipo de datos*. Cada contenedor de dato individual se llama *lane*.

Cuestiones de Implementación

- Cada instrucción NEON consiste de *n* operaciones que ocurren en paralelo, donde *n*, es el número de *lanes* en que está dividido cada vector de entrada.

Cuestiones de Implementación

- Cada instrucción NEON consiste de *n* operaciones que ocurren en paralelo, donde *n*, es el número de *lanes* en que está dividido cada vector de entrada.
- Cada una de las *n* operaciones ocurren dentro del *lane*, es decir no hay posibilidad de carry ni overflow de un *lane* al siguiente.

Cuestiones de Implementación

- Cada instrucción NEON consiste de *n* operaciones que ocurren en paralelo, donde *n*, es el número de *lanes* en que está dividido cada vector de entrada.
- Cada una de las *n* operaciones ocurren dentro del *lane*, es decir no hay posibilidad de carry ni overflow de un *lane* al siguiente.

Vector NEON de 64 bit
- Ocho Elementos de 8 bit
- Cuatro Elementos de 16 bit
- Dos Elementos de 32 bit
- Un Elementos de 64 bit

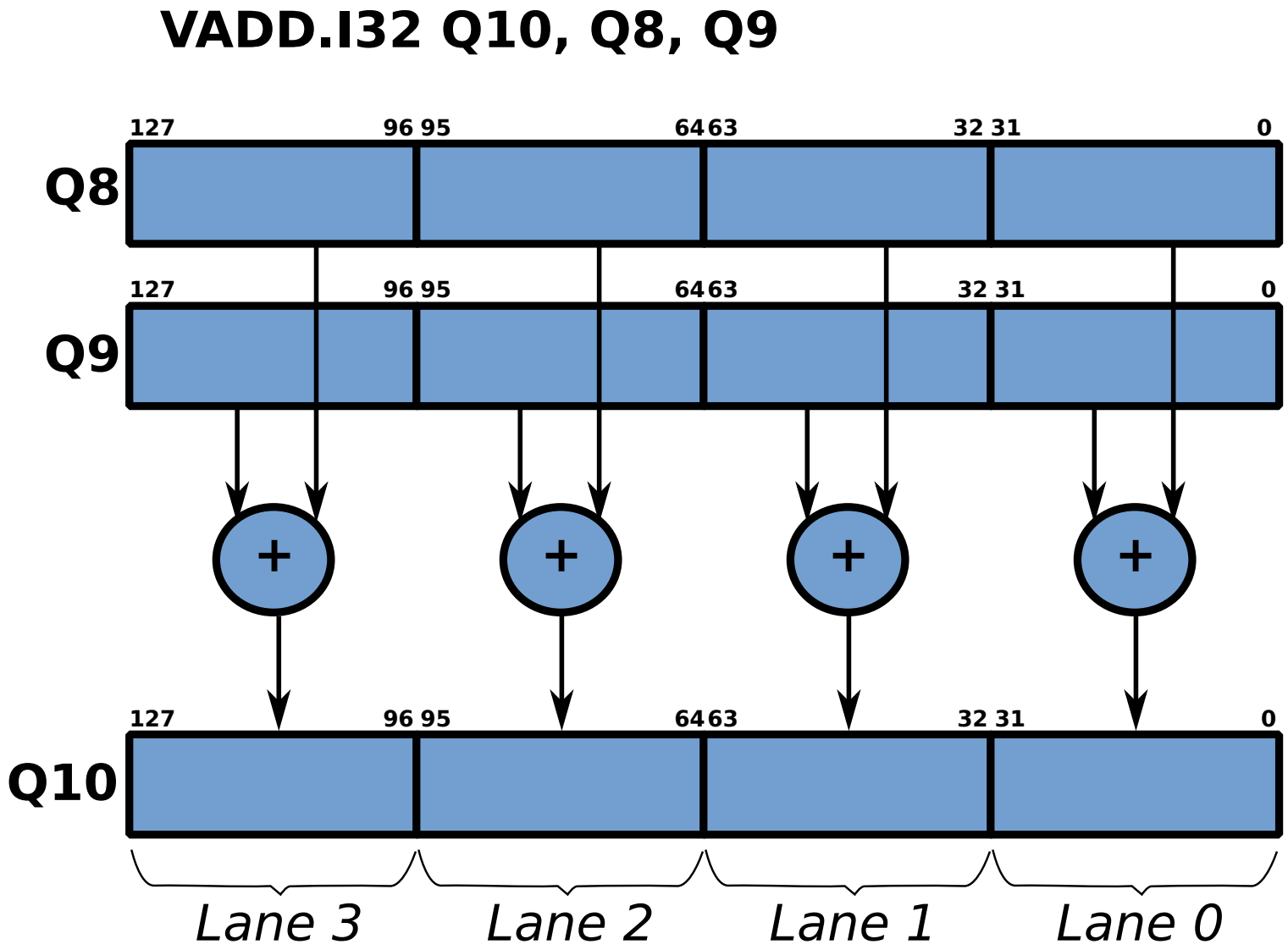
Cuestiones de Implementación

- Cada instrucción NEON consiste de *n* operaciones que ocurren en paralelo, donde *n*, es el número de *lanes* en que está dividido cada vector de entrada.
- Cada una de las *n* operaciones ocurren dentro del *lane*, es decir no hay posibilidad de carry ni overflow de un *lane* al siguiente.

Vector NEON de 64 bit
- Ocho Elementos de 8 bit
- Cuatro Elementos de 16 bit
- Dos Elementos de 32 bit
- Un Elementos de 64 bit

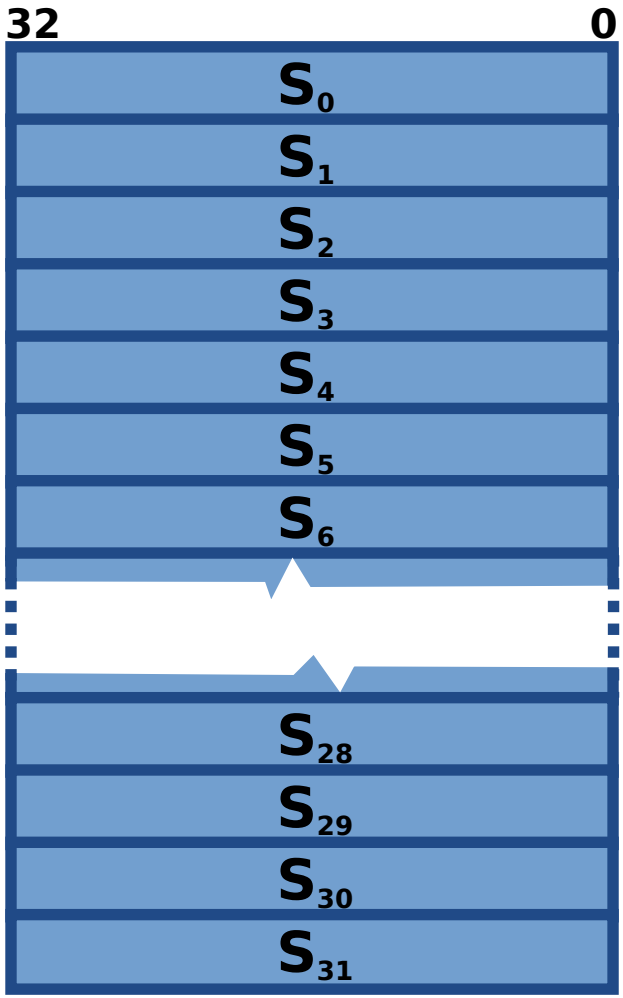
Vector NEON de 128 bit
- Dieciséis Elementos de 8 bit
- Ocho Elementos de 16 bit
- Cuatro Elementos de 32 bit
- Dos Elementos de 64 bit

Cuestiones de Implementación



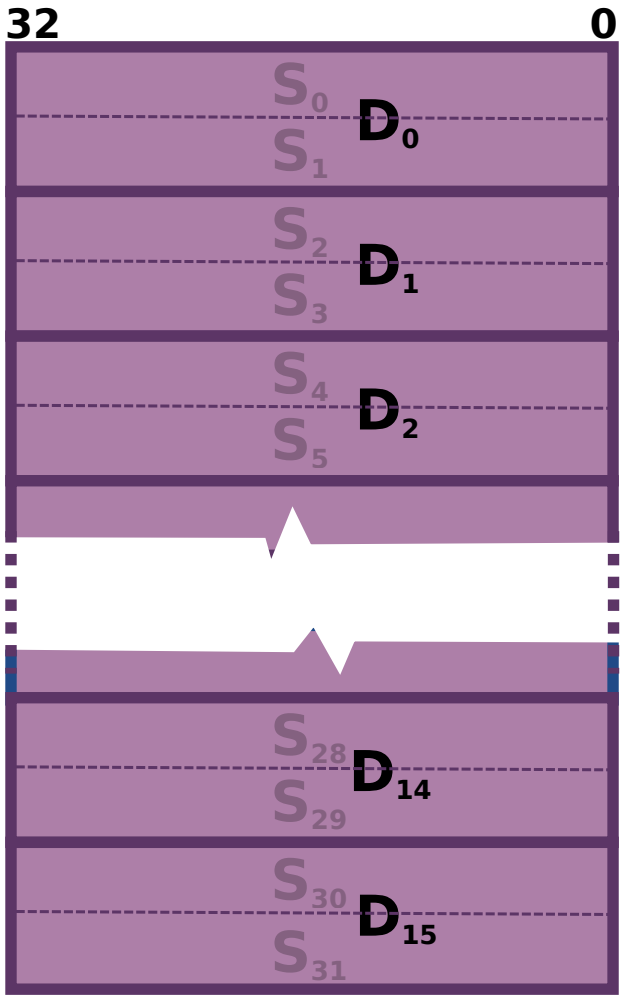
Solapamiento de Registros

La extensión VFP-v2, introduce un set de 32 registros de 32 bit para almacenar números en Punto Flotante simple precisión en formato IEEE-754. Solo accesibles con VFP.



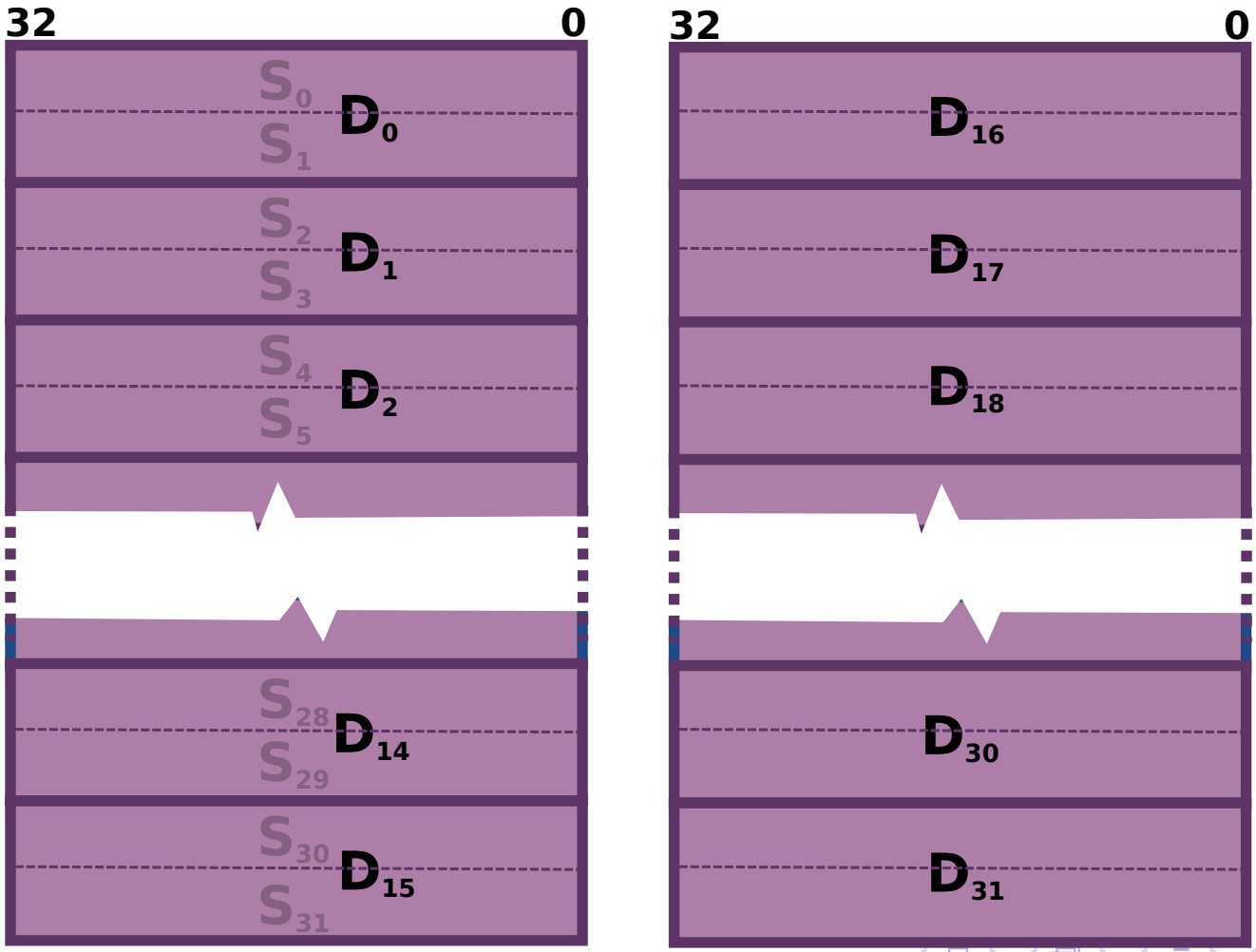
Solapamiento de Registros

VFP-v2 permite usar estos 32 registros de 32 bit como 16 registros de 64 bit, para almacenar números en Punto Flotante doble precisión IEEE-754. Solo accesibles con VFP, si es VFP-v2



Solapamiento de Registros

La VFPv-3 agrega otros 16 registros de 64 bit, para Punto Flotante doble precisión y para datos empaquetados en las Extensiones avanzadas SIMD (NEON).



Solapamiento de Registros

Las Extensiones avanzadas SIMD (NEON) pueden trabajar con datos empaquetados de 128 bit agrupando los 32 registros de 64 bit en 16 registros de 128 bit. Solo accesible con NEON.

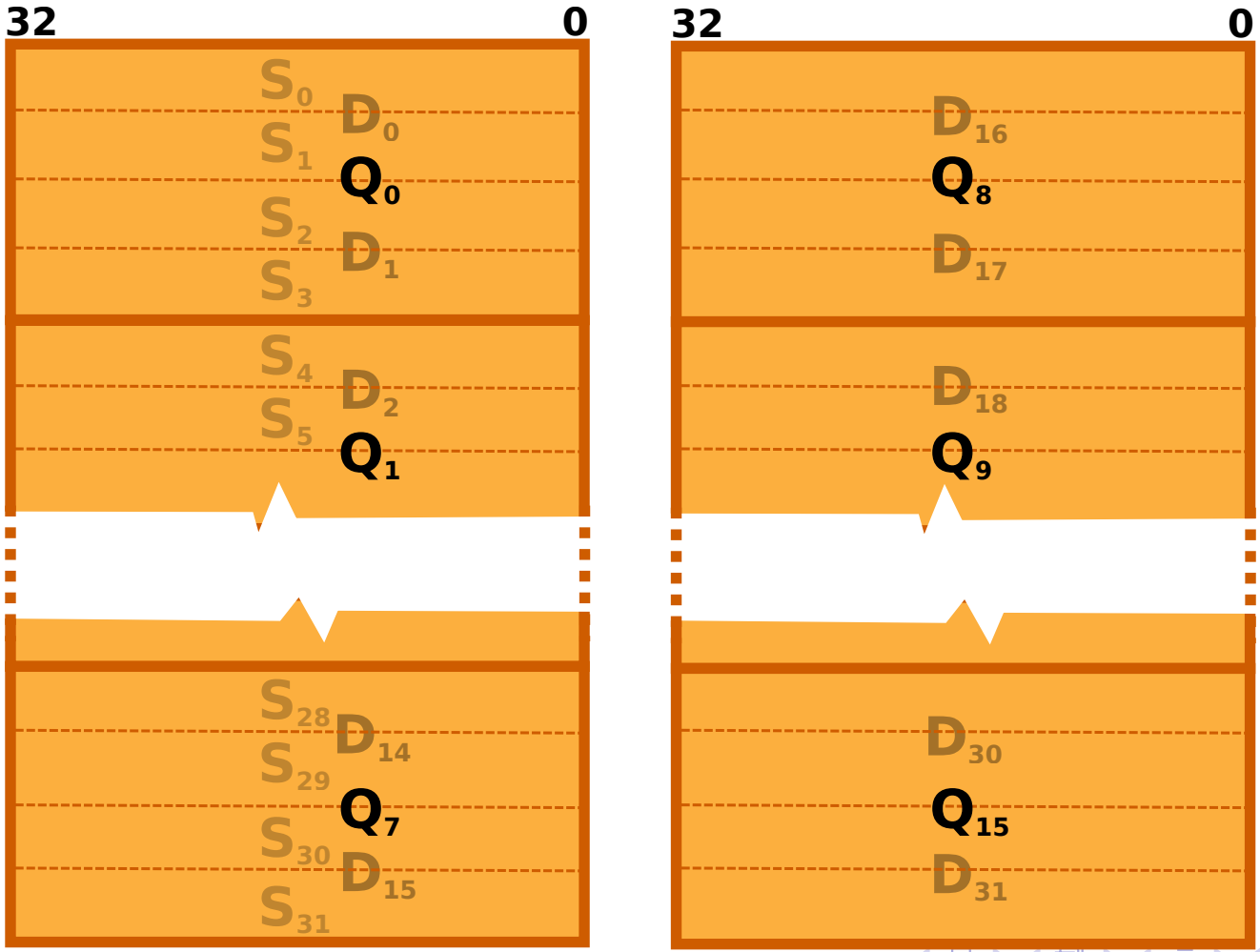


Tabla de contenidos

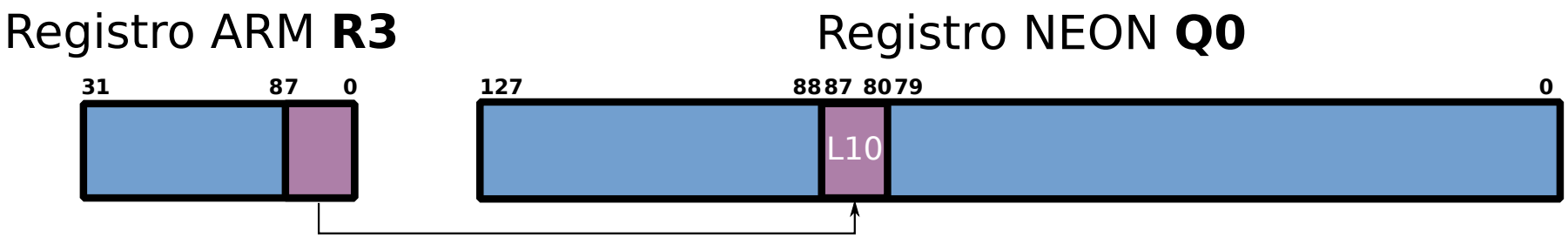
Escalares

Un escalar se refiere a un valor simple de 8, 16, 32, o 64 bit en lugar de un vector con múltiples valores de cualquiera de estos tipos empaquetados en lanes.
Se lo accede utilizando el número de lane como índice dentro del vector.

Escalares

Un escalar se refiere a un valor simple de 8, 16, 32, o 64 bit en lugar de un vector con múltiples valores de cualquiera de estos tipos empaquetados en lanes.
Se lo accede utilizando el número de lane como índice dentro del vector.

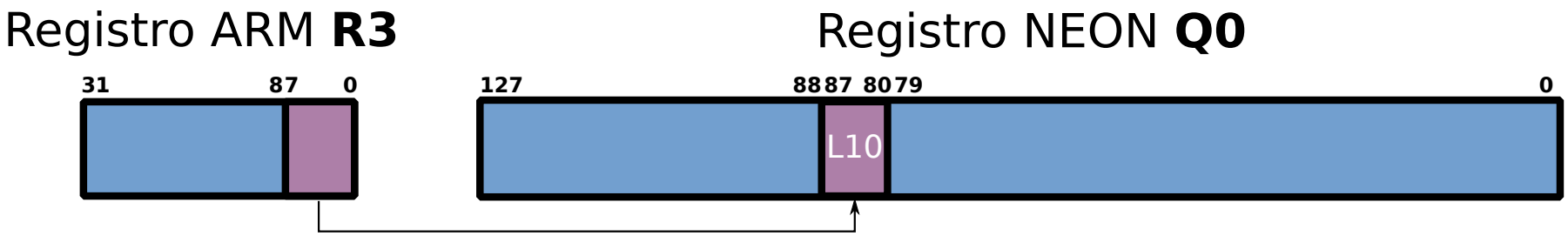
VMOV.8 Q0[10], R3



Escalares

Un escalar se refiere a un valor simple de 8, 16, 32, o 64 bit en lugar de un vector con múltiples valores de cualquiera de estos tipos empaquetados en lanes.
Se lo accede utilizando el número de lane como índice dentro del vector.

VMOV.8 Q0[10], R3



Para instrucciones de multiplicación solo trabaja con escalares de 16 o 32 bit.
- Escalares de 16 bit solo para D0[x]-D7[x], con $0 \leq x \leq 3$
- Escalares de 32 bit solo para D0[x]-D15[x], para x 0 ó 1.

Tipos de Datos

	8 bit	16 bit	32 bit	64 bit
Unsigned integer	U8	U16	U32	U64
Signed integer	S8	S16	S32	S64
Integer (Tipo no especificado)	I8	I16	I32	I64
Floating-point	NA	F16	F32 or F	NA
Polynomial over {0,1}	P8	P16	NA	NA

Tipos de Datos NEON

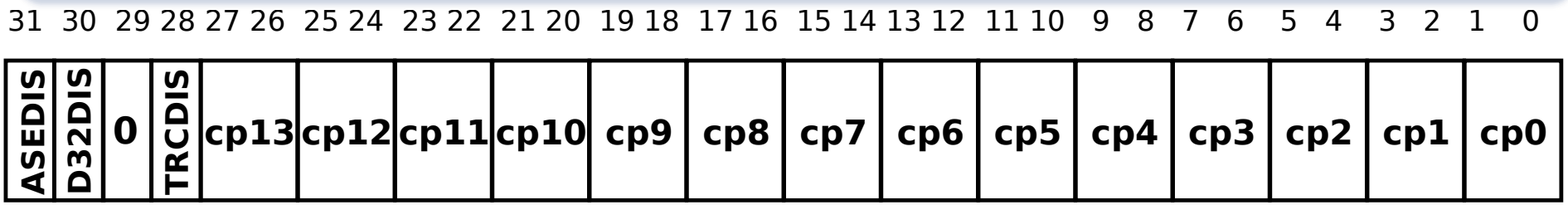
	16 bit	32 bit	64 bit
Unsigned integer	U16	U32	NA
Signed integer	S16	S32	NA
Floating-point	F16	F32 o F	F64 o D

Tipos de Datos VFP

Tabla de contenidos

Habilitación de NEON y VFP

El Registro de Control de Acceso del CP15, es donde se habilitan los coprocesadores CP0 a CP13. En particular necesitamos activar CP10 y CP11, para tener disponibles los recursos SIMD y VFP.



Habilitación de NEON y VFP

El Registro de Control de Acceso del CP15, es donde se habilitan los coprocesadores CP0 a CP13. En particular necesitamos activar CP10 y CP11, para tener disponibles los recursos SIMD y VFP.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASEDIS	D32DIS	0	TRCDIS	cp13		cp12		cp11		cp10		cp9		cp8		cp7		cp6		cp5		cp4		cp3		cp2		cp1		cp0	

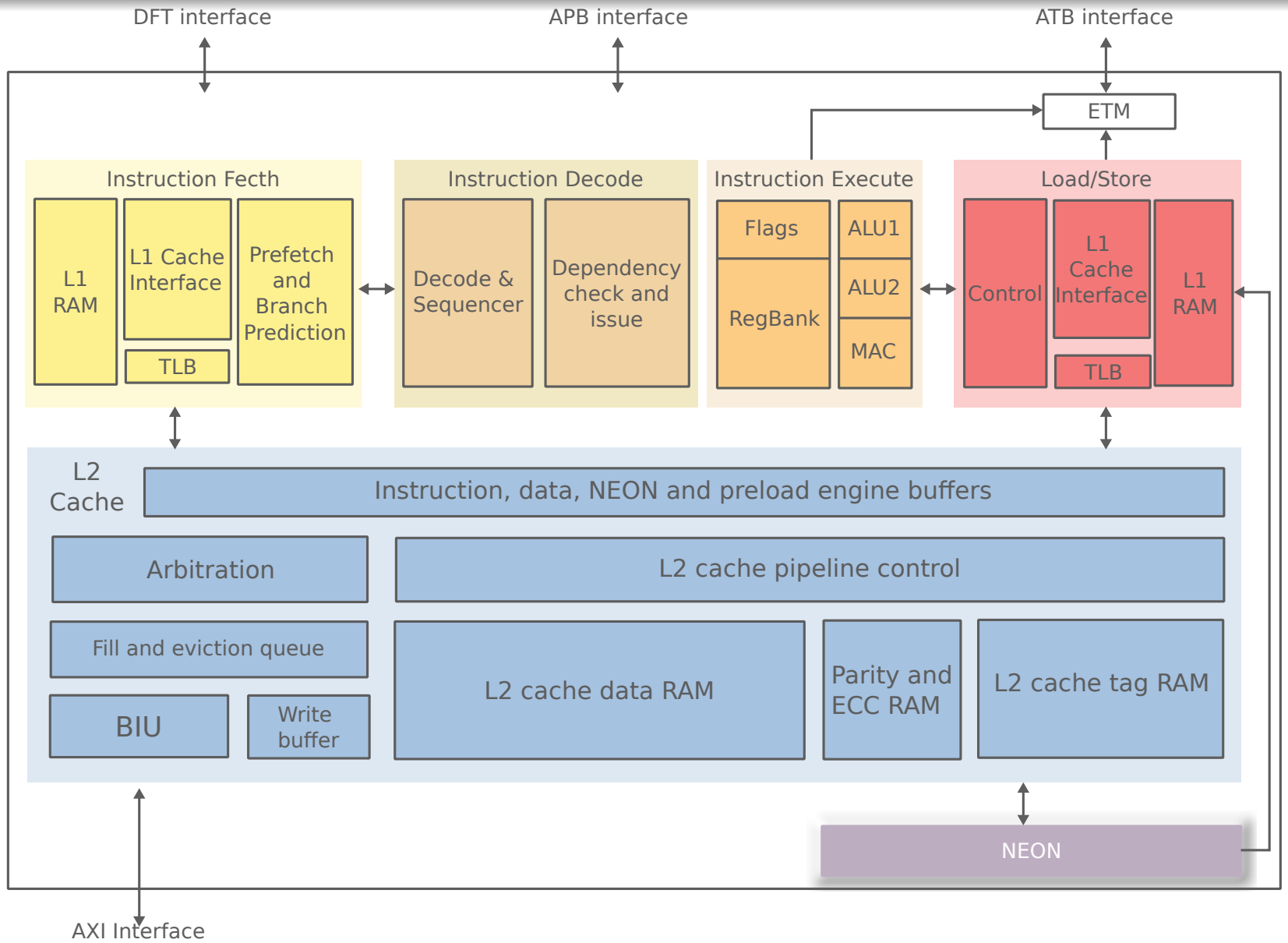
```
MRC p15,0,r0,c1,c0,2      // Lee CPACR de cp15 (Access Control Register).
ORR r0,r0,#0x00f00000     // Prepara acceso a NEON/VFP habilitando
                          // acceso a Coprocessores 10 y 11 CPACR[23-20]
MCR p15,0,r0,c1,c0,2      // Escribe CPACR (Habilita NEON).
ISB                       // Instruction Syncronization Barrier
                          // flush pipeline, y vuelve a fetchear todo
MOV r0,#0x40000000        // Enciende hardware VFP y NEON.
MSR FPEXC,r0              // Setea bit EN en FPEXC
```

Tabla de contenidos

NEON en el procesador Cortex A8

El Cortex-A8 es el primer miembro de la familia ARMv7A. Es mas complejo que los procesadores ARM presentados previamente.
Incluye dos pipelines simétricos para instrucciones enteras de 13 etapas, con ejecución de instrucciones en orden, y un pipeline de 10 etapas para instrucciones NEON y de Punto Flotante (VFP), sobre regstros de 64 y 128 bits.
Soporta VFPv3 y Jazzele RCT.

NEON en el procesador Cortex A8



NEON en el procesador Cortex A8: El cache

NEON en el procesador Cortex A8: El cache

El sistema cache del Cortex A8, se compone de un cache L1 para instrucciones de 16 KiB, un cache L1 de Datos de 32 KiB, ambos unificados en un cache L2 de 1 MiB.

NEON en el procesador Cortex A8: El cache

El sistema cache del Cortex A8, se compone de un cache L1 para instrucciones de 16 KiB, un cache L1 de Datos de 32 KiB, ambos unificados en un cache L2 de 1 MiB.

Ambos caches L1 y L2 se vinculan al core con un Bus de Datos de 128 bit. El L1 está indexado virtualmente y tageado físicamente, mientras que el L2 utiliza direcciones físicas tanto para índices como para tags.

NEON en el procesador Cortex A8: El cache

El sistema cache del Cortex A8, se compone de un cache L1 para instrucciones de 16 KiB, un cache L1 de Datos de 32 KiB, ambos unificados en un cache L2 de 1 MiB.

Ambos caches L1 y L2 se vinculan al core con un Bus de Datos de 128 bit. El L1 está indexado virtualmente y tageado físicamente, mientras que el L2 utiliza direcciones físicas tanto para índices como para tags.

Los datos utilizados por la Unidad NEON por default no se almacenan en el L1. Sin embargo NEON puede leer y escribir en L1 si lo necesita.

NEON en el procesador Cortex A8: MPE

NEON en el procesador Cortex A8: MPE

En el MPE (Media Processing Engine) el inicio de la Unidad de Ejecución Neon, se conecta a la salida del pipeline de enteros principal. Como resultado, excepciones o mal predicciones de salto pueden resolverse antes de que la instrucción los alcance.

NEON en el procesador Cortex A8: MPE

En el MPE (Media Processing Engine) el inicio de la Unidad de Ejecución Neon, se conecta a la salida del pipeline de enteros principal. Como resultado, excepciones o mal predicciones de salto pueden resolverse antes de que la instrucción los alcance.

La Unidad Entera de un Cortex A8, calcula la dirección de las instrucciones NEON de load y store a medida que pasan por el pipeline, permitiendo buscar los datos en la cache L1 antes de ser requeridos para su procesamiento por la instrucción NEON.

NEON en el procesador Cortex A8: MPE

En el MPE (Media Processing Engine) el inicio de la Unidad de Ejecución Neon, se conecta a la salida del pipeline de enteros principal. Como resultado, excepciones o mal predicciones de salto pueden resolverse antes de que la instrucción los alcance.

La Unidad Entera de un Cortex A8, calcula la dirección de las instrucciones NEON de load y store a medida que pasan por el pipeline, permitiendo buscar los datos en la cache L1 antes de ser requeridos para su procesamiento por la instrucción NEON.

La profundidad adicional del pipeline NEON, junto con el buffering en la carga de datos entre la Unidad NEON, el pipeline ARM de enteros y el sistema de memoria, permite neutralizar la demora en el acceso al cache L2 (latency).

NEON en el procesador Cortex A8: MPE

En las operaciones de store se emplea un buffer de escrituras que previene de bloqueo al pipeline para instrucciones NEON de store, y detecta colisiones de direcciones entre la Unidad entera de ARM, e instrucciones NEON de load.

NEON en el procesador Cortex A8: MPE

En las operaciones de store se emplea un buffer de escrituras que previene de bloqueo al pipeline para instrucciones NEON de store, y detecta colisiones de direcciones entre la Unidad entera de ARM, e instrucciones NEON de load.

NIQ: **N**EON **I**nstruction **Q**ueue, es el desacople entre el pipeline NEON de 10 etapas, y el final del pipeline de Enteros de ARM.

NEON en el procesador Cortex A8: MPE

En las operaciones de store se emplea un buffer de escrituras que previene de bloqueo al pipeline para instrucciones NEON de store, y detecta colisiones de direcciones entre la Unidad entera de ARM, e instrucciones NEON de load.

NIQ: **N**EON **I**nstruction **Q**ueue, es el desacople entre el pipeline NEON de 10 etapas, y el final del pipeline de Enteros de ARM.

La Unidad de Ejecución de Instrucciones general del pipeline le puede enviar a la Unidad NEON hasta dos instrucciones en cada ciclo de clock. La Unidad NEON envía y ejecuta una instrucción (entra o de punto flotante) por clock y la retira en orden. Excepciones y miss predictions se resuelven en la Unidad de Enteros, por lo tanto las instrucciones NEON no generan excepciones.

NEON en el procesador Cortex A8: MPE

La Unidad NEON de un Cortex A8 tiene:

- Vías de 128 bit para load y store con las cache L1 y L2, como capacidad de streaming de datos.

NEON en el procesador Cortex A8: MPE

La Unidad NEON de un Cortex A8 tiene:

- Vías de 128 bit para load y store con las cache L1 y L2, como capacidad de streaming de datos.
- Tres Pipelines SIMD enteros:

NEON en el procesador Cortex A8: MPE

La Unidad NEON de un Cortex A8 tiene:

- Vías de 128 bit para load y store con las cache L1 y L2, como capacidad de streaming de datos.
- Tres Pipelines SIMD enteros:
  - ✓ Multiplicador Acumulador entero (MAC)

NEON en el procesador Cortex A8: MPE

La Unidad NEON de un Cortex A8 tiene:

- Vías de 128 bit para load y store con las cache L1 y L2, como capacidad de streaming de datos.
- Tres Pipelines SIMD enteros:
  - ✓ Multiplicador Acumulador entero (MAC)
  - ✓ Barrel Shifter de Enteros

NEON en el procesador Cortex A8: MPE

La Unidad NEON de un Cortex A8 tiene:

- Vías de 128 bit para load y store con las cache L1 y L2, como capacidad de streaming de datos.
- Tres Pipelines SIMD enteros:
  - ✓ Multiplicador Acumulador entero (MAC)
  - ✓ Barrel Shifter de Enteros
  - ✓ ALU entera

NEON en el procesador Cortex A8: MPE

La Unidad NEON de un Cortex A8 tiene:

- Vías de 128 bit para load y store con las cache L1 y L2, como capacidad de streaming de datos.
- Tres Pipelines SIMD enteros:
  - ✓ Multiplicador Acumulador entero (MAC)
  - ✓ Barrel Shifter de Enteros
  - ✓ ALU entera
- Un Pipeline de permutación de operaciones Load/Store

NEON en el procesador Cortex A8: MPE

La Unidad NEON de un Cortex A8 tiene:

- Vías de 128 bit para load y store con las cache L1 y L2, como capacidad de streaming de datos.
- Tres Pipelines SIMD enteros:
  - ✓ Multiplicador Acumulador entero (MAC)
  - ✓ Barrel Shifter de Enteros
  - ✓ ALU entera
- Un Pipeline de permutación de operaciones Load/Store
  - ✓ Load/Store de datos NEON

NEON en el procesador Cortex A8: MPE

La Unidad NEON de un Cortex A8 tiene:

- Vías de 128 bit para load y store con las cache L1 y L2, como capacidad de streaming de datos.
- Tres Pipelines SIMD enteros:
  - ✓ Multiplicador Acumulador entero (MAC)
  - ✓ Barrel Shifter de Enteros
  - ✓ ALU entera
- Un Pipeline de permutación de operaciones Load/Store
  - ✓ Load/Store de datos NEON
  - ✓ Transferencias hacia/desde la Unidad de enteros.

NEON en el procesador Cortex A8: MPE

La Unidad NEON de un Cortex A8 tiene:

- Vías de 128 bit para load y store con las cache L1 y L2, como capacidad de streaming de datos.
- Tres Pipelines SIMD enteros:
  - ✓ Multiplicador Acumulador entero (MAC)
  - ✓ Barrel Shifter de Enteros
  - ✓ ALU entera
- Un Pipeline de permutación de operaciones Load/Store
  - ✓ Load/Store de datos NEON
  - ✓ Transferencias hacia/desde la Unidad de enteros.
  - ✓ Permutación de datos tales como entrelazado y desentrelazado.

NEON en el procesador Cortex A8: MPE

La Unidad NEON de un Cortex A8 tiene:

- Vías de 128 bit para load y store con las cache L1 y L2, como capacidad de streaming de datos.
- Tres Pipelines SIMD enteros:
  - ✓ Multiplicador Acumulador entero (MAC)
  - ✓ Barrel Shifter de Enteros
  - ✓ ALU entera
- Un Pipeline de permutación de operaciones Load/Store
  - ✓ Load/Store de datos NEON
  - ✓ Transferencias hacia/desde la Unidad de enteros.
  - ✓ Permutación de datos tales como entrelazado y desentrelazado.
- Dos pipelines SIMD Punto flotante Precisión Simple

NEON en el procesador Cortex A8: MPE

La Unidad NEON de un Cortex A8 tiene:

- Vías de 128 bit para load y store con las cache L1 y L2, como capacidad de streaming de datos.
- Tres Pipelines SIMD enteros:
  - ✓ Multiplicador Acumulador entero (MAC)
  - ✓ Barrel Shifter de Enteros
  - ✓ ALU entera
- Un Pipeline de permutación de operaciones Load/Store
  - ✓ Load/Store de datos NEON
  - ✓ Transferencias hacia/desde la Unidad de enteros.
  - ✓ Permutación de datos tales como entrelazado y desentrelazado.
- Dos pipelines SIMD Punto flotante Precisión Simple
  - ✓ Multiplicación

NEON en el procesador Cortex A8: MPE

La Unidad NEON de un Cortex A8 tiene:

- Vías de 128 bit para load y store con las cache L1 y L2, como capacidad de streaming de datos.
- Tres Pipelines SIMD enteros:
  - ✓ Multiplicador Acumulador entero (MAC)
  - ✓ Barrel Shifter de Enteros
  - ✓ ALU entera
- Un Pipeline de permutación de operaciones Load/Store
  - ✓ Load/Store de datos NEON
  - ✓ Transferencias hacia/desde la Unidad de enteros.
  - ✓ Permutación de datos tales como entrelazado y desentrelazado.
- Dos pipelines SIMD Punto flotante Precisión Simple
  - ✓ Multiplicación
  - ✓ Suma.

NEON en el procesador Cortex A8: MPE

La Unidad NEON de un Cortex A8 tiene:

- Vías de 128 bit para load y store con las cache L1 y L2, como capacidad de streaming de datos.
- Tres Pipelines SIMD enteros:
  - ✓ Multiplicador Acumulador entero (MAC)
  - ✓ Barrel Shifter de Enteros
  - ✓ ALU entera
- Un Pipeline de permutación de operaciones Load/Store
  - ✓ Load/Store de datos NEON
  - ✓ Transferencias hacia/desde la Unidad de enteros.
  - ✓ Permutación de datos tales como entrelazado y desentrelazado.
- Dos pipelines SIMD Punto flotante Precisión Simple
  - ✓ Multiplicación
  - ✓ Suma.
- Una Unidad de punto flotante para instrucciones VFPv3 por fuera del pipeline NEON.

Procesador Cortex A8: Acceso a memoria de Datos

- El bit L1NEON en cp15, habilita el cache L1 para NEON. No obstante aún con este bit deshabilitado puede tenerse un hit en L1 si el dato fue accedido mediante instrucciones ARM (No NEON).

Procesador Cortex A8: Acceso a memoria de Datos

- El bit L1NEON en cp15, habilita el cache L1 para NEON. No obstante aún con este bit deshabilitado puede tenerse un hit en L1 si el dato fue accedido mediante instrucciones ARM (No NEON).
- Los hazards entre datos NEON y datos del pipeline ARM se resuelven con la granularidad de las líneas de cache.

Procesador Cortex A8: Acceso a memoria de Datos

- El bit L1NEON en cp15, habilita el cache L1 para NEON. No obstante aún con este bit deshabilitado puede tenerse un hit en L1 si el dato fue accedido mediante instrucciones ARM (No NEON).
- Los hazards entre datos NEON y datos del pipeline ARM se resuelven con la granularidad de las líneas de cache.
- Para maximizar performance en acceso a datos:

Procesador Cortex A8: Acceso a memoria de Datos

- El bit L1NEON en cp15, habilita el cache L1 para NEON. No obstante aún con este bit deshabilitado puede tenerse un hit en L1 si el dato fue accedido mediante instrucciones ARM (No NEON).
- Los hazards entre datos NEON y datos del pipeline ARM se resuelven con la granularidad de las líneas de cache.
- Para maximizar performance en acceso a datos:
  - ✓ Evitar mezclar accesos de datos NEON y ARM dentro de la misma línea de cache.

Procesador Cortex A8: Acceso a memoria de Datos

- El bit L1NEON en cp15, habilita el cache L1 para NEON. No obstante aún con este bit deshabilitado puede tenerse un hit en L1 si el dato fue accedido mediante instrucciones ARM (No NEON).
- Los hazards entre datos NEON y datos del pipeline ARM se resuelven con la granularidad de las líneas de cache.
- Para maximizar performance en acceso a datos:
  - ✓ Evitar mezclar accesos de datos NEON y ARM dentro de la misma línea de cache.
  - ✓ Utilizar para rutinas de copia de memoria instrucciones NEON. Se obtiene mayor performace y se evita cache pollution en el L1.

Procesador Cortex A8: Acceso a memoria de Datos

- El bit L1NEON en cp15, habilita el cache L1 para NEON. No obstante aún con este bit deshabilitado puede tenerse un hit en L1 si el dato fue accedido mediante instrucciones ARM (No NEON).
- Los hazards entre datos NEON y datos del pipeline ARM se resuelven con la granularidad de las líneas de cache.
- Para maximizar performance en acceso a datos:
  - ✓ Evitar mezclar accesos de datos NEON y ARM dentro de la misma línea de cache.
  - ✓ Utilizar para rutinas de copia de memoria instrucciones NEON. Se obtiene mayor performace y se evita cache pollution en el L1.
- Ambos niveles de cache tienen 64 B de tamaño, y en virtud de la profundidad del pipeline para las instrucciones NEON, por default los datos NEON se mantienen en L2.

Procesador Cortex A8: Acceso a memoria de Datos

Solo se puede admitir el envío de dos instrucciones NEON en el mismo ciclo de clock, si:

- Instrucciones sin dependencias de datos

Procesador Cortex A8: Acceso a memoria de Datos

Solo se puede admitir el envío de dos instrucciones NEON en el mismo ciclo de clock, si:

- Instrucciones sin dependencias de datos
- Una de ellas es de procesamiento y la otra se ajusta a uno de los siguientes casos

Procesador Cortex A8: Acceso a memoria de Datos

Solo se puede admitir el envío de dos instrucciones NEON en el mismo ciclo de clock, si:

- Instrucciones sin dependencias de datos
- Una de ellas es de procesamiento y la otra se ajusta a uno de los siguientes casos
  - ✓ Es una instrucción Load/Store

Procesador Cortex A8: Acceso a memoria de Datos

Solo se puede admitir el envío de dos instrucciones NEON en el mismo ciclo de clock, si:

- Instrucciones sin dependencias de datos
- Una de ellas es de procesamiento y la otra se ajusta a uno de los siguientes casos
  - ✓ Es una instrucción Load/Store
  - ✓ Es una instrucción de transferencia ARM NEON o viceversa

Procesador Cortex A8: Acceso a memoria de Datos

Solo se puede admitir el envío de dos instrucciones NEON en el mismo ciclo de clock, si:

- Instrucciones sin dependencias de datos
- Una de ellas es de procesamiento y la otra se ajusta a uno de los siguientes casos
  - ✓ Es una instrucción Load/Store
  - ✓ Es una instrucción de transferencia ARM NEON o viceversa
  - ✓ Es una instrucción de permutación.

Procesador Cortex A8: Pipeline Hazards

- El pipeline NEON ejecuta varias etapas luego del pipeline ARM, recibiendo de éste sus instrucciones a través de una FIFO. Las transferencias entre registros NEON y ARM es muy rápida pero el acceso a registros de coprocesadores mediante MRC desde NEON insume 20 ciclos de clock.

Procesador Cortex A8: Pipeline Hazards

- El pipeline NEON ejecuta varias etapas luego del pipeline ARM, recibiendo de éste sus instrucciones a través de una FIFO. Las transferencias entre registros NEON y ARM es muy rápida pero el acceso a registros de coprocesadores mediante MRC desde NEON insume 20 ciclos de clock.
- NEON tiene su propia Unidad Load/Store que opera en principio sin considerar el L1. Dispone de hardware para resolver accesos no ordenados a memoria. Si ambos accesos se dan en la misma línea de cache, se requieren 20 ciclos de reloj para resolverlo.

Procesador Cortex A8: Pipeline Hazards

- El pipeline NEON ejecuta varias etapas luego del pipeline ARM, recibiendo de éste sus instrucciones a través de una FIFO. Las transferencias entre registros NEON y ARM es muy rápida pero el acceso a registros de coprocesadores mediante MRC desde NEON insume 20 ciclos de clock.
- NEON tiene su propia Unidad Load/Store que opera en principio sin considerar el L1. Dispone de hardware para resolver accesos no ordenados a memoria. Si ambos accesos se dan en la misma línea de cache, se requieren 20 ciclos de reloj para resolverlo.
- Para evitarlo se debe escribir código tratando de reducir la combinación de accesos a memoria desde ARM y NEON. En caso de ser necesario levantar desde registros ARM el resultado de un procesamiento NEON almacenado en memoria, se requieren 20 ciclos de clock.

Procesador Cortex A8: Pipeline Hazards

- Una alternativa a considerar es antes del acceso resolver alguna tarea con código ARM de modo de no entrelazar los acceso a memoria. Esto generalmente funciona mejor que transferencias NEON ARM, ya que éstas pueden congestionar el pipeline.

Procesador Cortex A8: Pipeline Hazards

- Una alternativa a considerar es antes del acceso resolver alguna tarea con código ARM de modo de no entrelazar los acceso a memoria. Esto generalmente funciona mejor que transferencias NEON ARM, ya que éstas pueden congestionar el pipeline.

- Los compiladores utilizan una función propia para retornar un valor de punto flotante para evitar los 20 ciclos de clock de delay.

Procesador Cortex A8: Pipeline Hazards

- Una alternativa a considerar es antes del acceso resolver alguna tarea con código ARM de modo de no entrelazar los acceso a memoria. Esto generalmente funciona mejor que transferencias NEON ARM, ya que éstas pueden congestionar el pipeline.

- Los compiladores utilizan una función propia para retornar un valor de punto flotante para evitar los 20 ciclos de clock de delay.

- Los saltos condicionales que evalúan datos en punto flotante incurren también en esta penalidad. A pesar que la Unidad de punto flotante tiene todos los elementos para evaluar correctamente la condición. Pero retornar el resultado de la evaluación al pipeline ARM consume 20 ciclos de clock.

Tabla de contenidos

Sintaxis de instrucciones NEON

Una misma instrucción puede operar sobre diferentes tipos de datos.
El tipo de dato se especifica como sufijo de la instrucción.
La cantidad de datos (lanes) empaquetados, depende del tipo de dato y de los registros empleados.
Hemos visto ejemplos de instrucciones que muestran esta definición.
Sin embargo hay casos en los que los tamaños de registro no son homogéneos

```
VMULL.S16 Q2, D8, D9      //Vector Multiply Long
                          // Q2[3]=D8[3]*D9[3], Q2[2]=D8[2]*D9[2]
                          // Q2[1]=D8[1]*D9[1], Q2[0]=D8[0]*D9[0]
```

Sintaxis de instrucciones NEON

Formato General:

```
V{<mod>}<op>{<shape>}{<cond>}{.<dt>} <dest1>{, <dest2>}, <src1>{, <
    src2>}
```

mod Modificador: **Q** Saturado, **H** Halfing, **D** Doubling, **R** Redondeado

op Operación: Es la operación en sí (por ejemplo ADD, MUL, etc)

shape Modificador: **L** Long, **W** Wide, o **N** Narrow.

cond Condition: Se utiliza como ya sabemos con la instrucción IT.

dt Data Type

$dest_n$ Registro o registros destino

src_n Registro o registros fuente

Modificadores de Instrucción

Modif.	Acción	Ejemplo	Descripción
Nada	Op. Básica	VADD.I16 Q0, Q1, Q2	Resultado sin modificación
Q	Saturation	VQADD.S16 D0, D2, D3	Si el resultado satura el rango queda en el máximo o mínimo valor según corresponda.
H	Halved	VHADD.S16 Q0, Q1, Q4	Cada operando se shiftea antes un lugar a la derecha (división por dos con truncado). En el ejemplo se calcula un promedio con truncado
D	Doubled	VQDMULL.S16 Q0, D1, D3	Se duplica el resultado luego de la saturación, cuando se trabaja en punto fijo Q15. Siempre se requiere adaptar el rango del resultado en los productos.
R	Rounded	VRSUBHN.I16 D0, Q1, Q3	Redondea el resultado para corregir el error de truncado. Equivale a sumar 0.5 antes de truncar.

Formas de Instrucción

No especificado:
Operandos y resultados son del mismo tamaño.
Ej:

```
VADD.I16 Q0, Q1, Q2
```

Formas de Instrucción

No especificado:
Operandos y resultados son del mismo tamaño.
Ej:

```
VADD.I16 Q0, Q1, Q2
```

Long-L: Operandos del mismo tamaño en bits, y resultados del doble de tamaño.
Ej:

```
VADDL.S16 Q0, D2, D3
```

Formas de Instrucción

No especificado: Operandos y resultados son del mismo tamaño.
Ej:

```
VADD.I16 Q0, Q1, Q2
```

Long-L: Operandos del mismo tamaño en bits, y resultados del doble de tamaño.
Ej:

```
VADDL.S16 Q0, D2, D3
```

Narrow-N: Operandos del mismo tamaño en bits, y resultados de la mitad de tamaño.
Ej:

```
VADDL.S16 D0, Q2, Q3
```

Formas de Instrucción

No especificado: Operandos y resultados son del mismo tamaño.
Ej:

```
VADD.I16 Q0, Q1, Q2
```

Long-L: Operandos del mismo tamaño en bits, y resultados del doble de tamaño.
Ej:

```
VADDL.S16 Q0, D2, D3
```

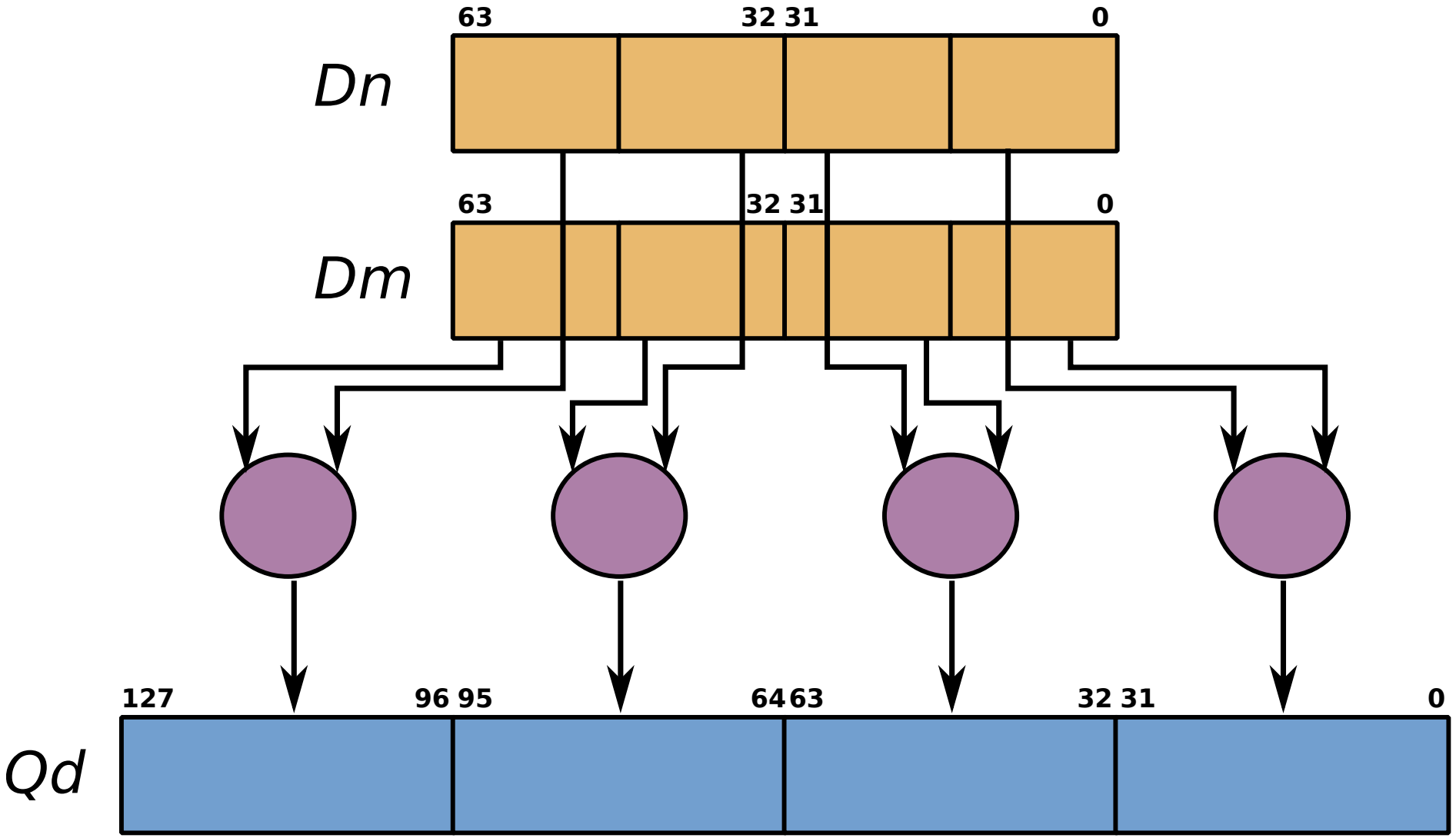
Narrow-N: Operandos del mismo tamaño en bits, y resultados de la mitad de tamaño.
Ej:

```
VADDL.S16 D0, Q2, Q3
```

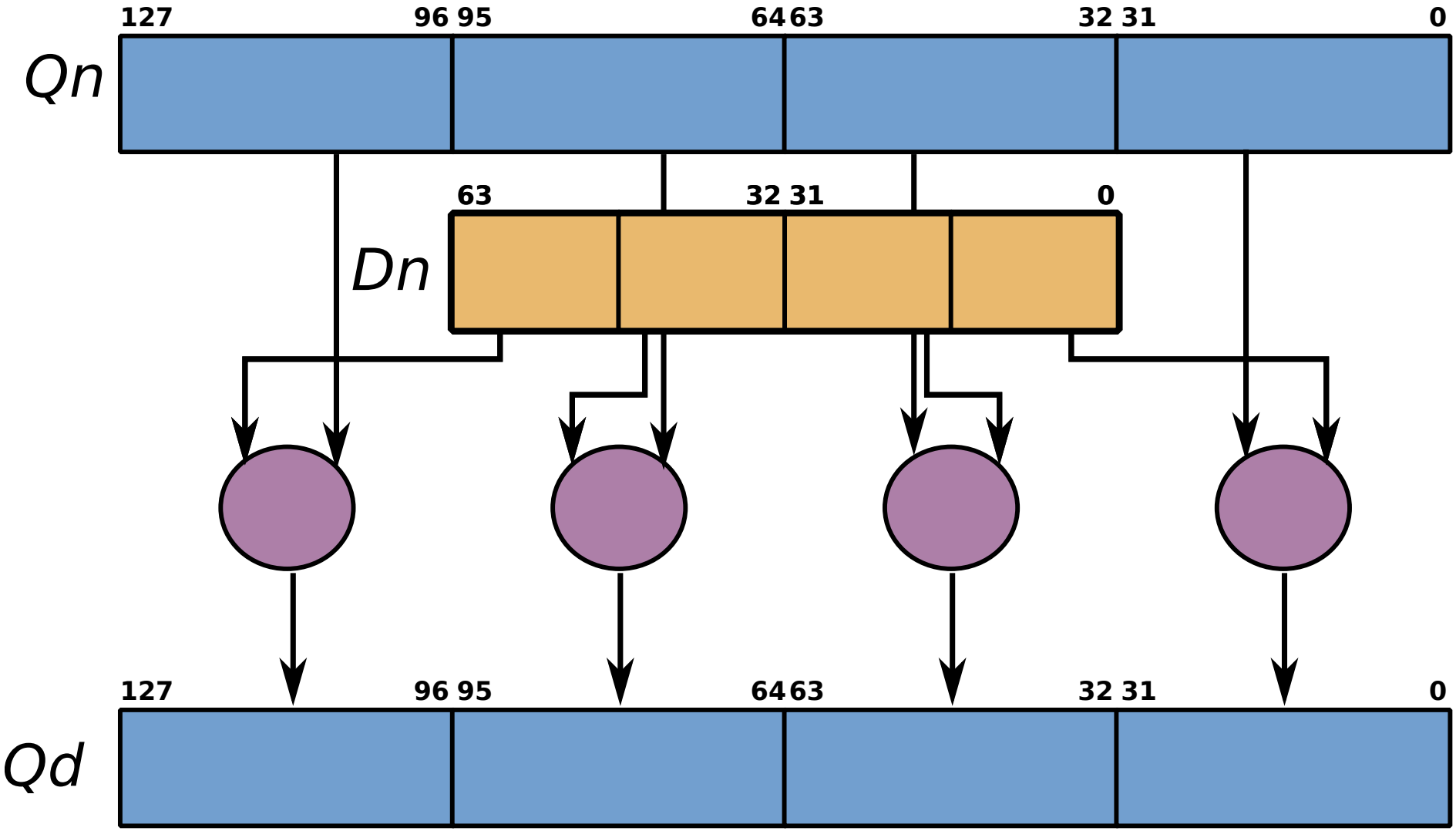
Wide-W: El resultado y el primer operando tienen el doble de tamaño en bits del segundo operando.
Ej:

```
VADDW.I16 Q0, Q1, D4
```

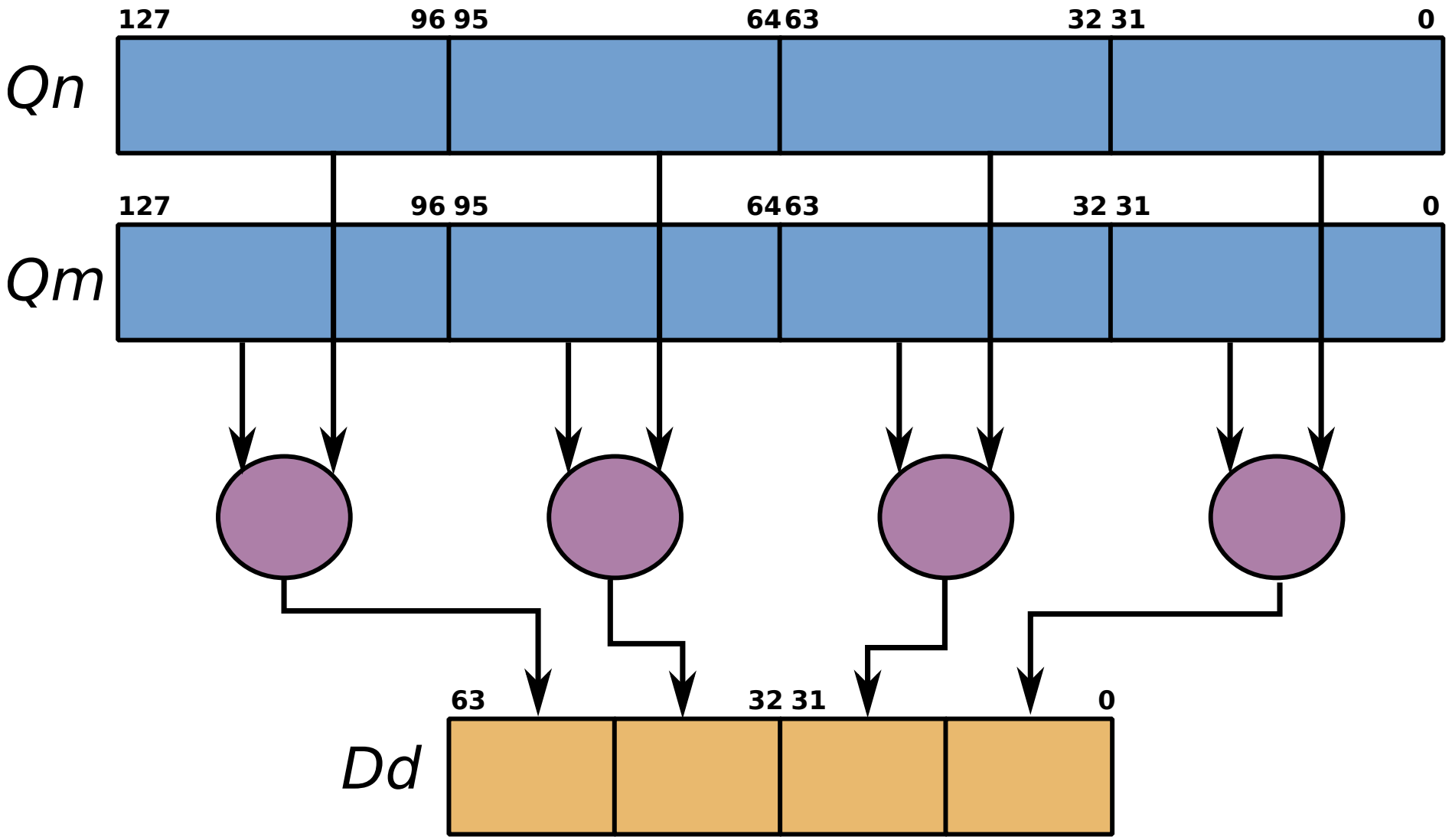
Formas: NEON Long Instruction



Formas: NEON Wide Instruction



Formas: NEON Narrow Instruction



Empaquetado y desempaquetado de datos

Empaquetado y desempaquetado de datos

- NEON trabaja con datos empaquetados en un registro para procesarlos en una única instrucción.

Empaquetado y desempaquetado de datos

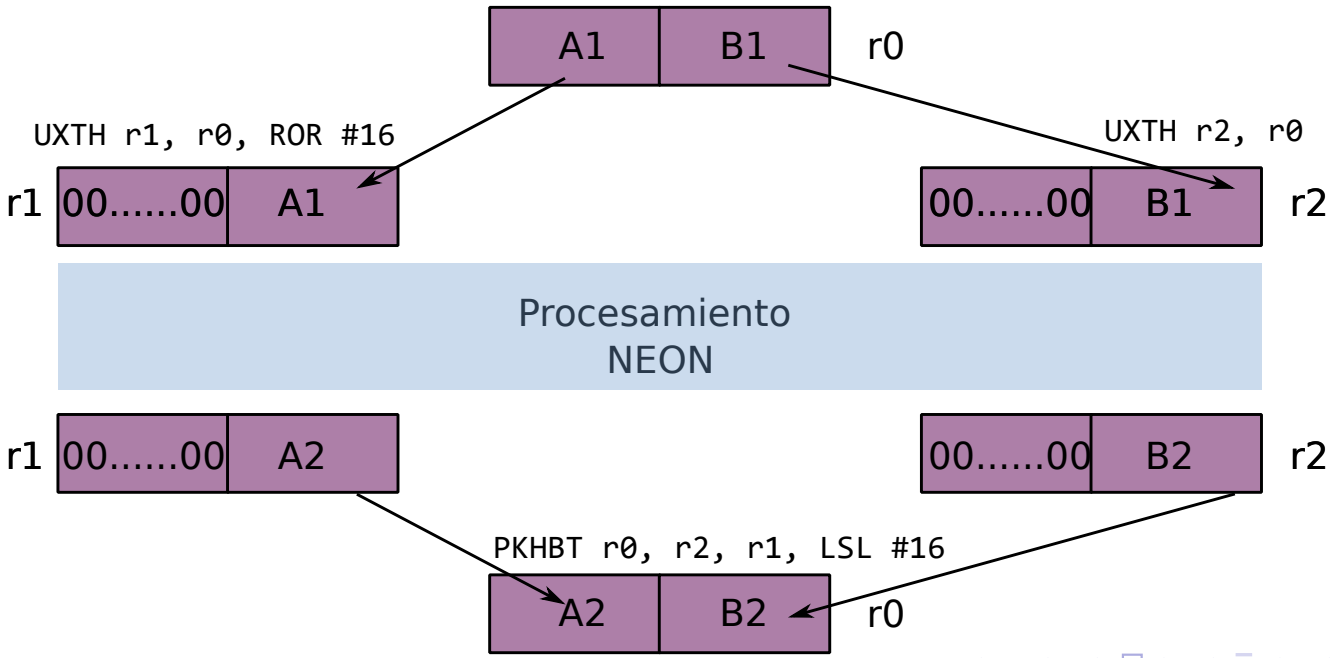
- NEON trabaja con datos empaquetados en un registro para procesarlos en una única instrucción.
- La preparación de esos operandos, lo mismo que la extracción de los resultados empaquetados no siempre pueden llevarse a cabo con instrucciones NEON

Empaquetado y desempaquetado de datos

- NEON trabaja con datos empaquetados en un registro para procesarlos en una única instrucción.
- La preparación de esos operandos, lo mismo que la extracción de los resultados empaquetados no siempre pueden llevarse a cabo con instrucciones NEON
- Existen para ayudar a esta tarea algunas instrucciones ARMv7 comunes para manipular los datos pre y post procesamiento NEON.

Empaquetado y desempaquetado de datos

- NEON trabaja con datos empaquetados en un registro para procesarlos en una única instrucción.
- La preparación de esos operandos, lo mismo que la extracción de los resultados empaquetados no siempre pueden llevarse a cabo con instrucciones NEON
- Existen para ayudar a esta tarea algunas instrucciones ARMv7 comunes para manipular los datos pre y post procesamiento NEON.



Empaquetado y desempaquetado de datos

UXTH

Extensión de ceros a un halfword. Extiende un valor de 16 bit a 32 bit.

```
UXTH{cond} {Rd}, Rm {,rotation}
```

`rotation` es la cantidad de bits que se rota a derecha. Puede ser `#ROR 8, #ROR 16, #ROR 24`

Empaquetado y desempaquetado de datos

PKHBT y **PKHTB**

Instrucciones de empaquetado de Halfword. Combinan las halfwords de un registro con una halfword de otro regitro. Uno de los registros puede ser desplazado n bits a derecha o izquierda antes de la extracción del dato.

```
PKHBT{cond} {Rd}, Rn, Rm{, LSL #leftshift}
PKHTB{cond} {Rd}, Rn, Rm{, ASR #rightshift}
```

PKHBT Combina Rn[15:0] con Rm[31:16] previamente desplazado.

PKHTB Combina Rn[31:16] con Rm[15:0] previamente desplazado.

$0 \leq leftshift \leq 31, y 1 \leq rightshift \leq 32$

Alineación de datos

Alineación de datos

- Si bien hay un soporte a la alineación de datos a nivel de arquitectura ARMv7, además se incluye en las instrucciones NEON que operan con memoria, la capacidad de especificar la alineación de datos deseada.

Alineación de datos

- Si bien hay un soporte a la alineación de datos a nivel de arquitectura ARMv7, además se incluye en las instrucciones NEON que operan con memoria, la capacidad de especificar la alineación de datos deseada.
- De mas está decir que ésta especificación debe ser consistente con el tipo de operandos de la instrucción.

Alineación de datos

- Si bien hay un soporte a la alineación de datos a nivel de arquitectura ARMv7, además se incluye en las instrucciones NEON que operan con memoria, la capacidad de especificar la alineación de datos deseada.
- De mas está decir que ésta especificación debe ser consistente con el tipo de operandos de la instrucción.
- Si la dirección de memoria no resulta consistente con el alineamiento especificado, se genera una Data Abort exception.

Alineación de datos

- Si bien hay un soporte a la alineación de datos a nivel de arquitectura ARMv7, además se incluye en las instrucciones NEON que operan con memoria, la capacidad de especificar la alineación de datos deseada.
- De mas está decir que ésta especificación debe ser consistente con el tipo de operandos de la instrucción.
- Si la dirección de memoria no resulta consistente con el alineamiento especificado, se genera una Data Abort exception.

```
VLD1.8 {D0},          [R1:64]   //D0[0]=[R1], D0[1]=[R1+1]...D0[7]=[R1+7]
VLD1.8 {D0,D1},       [R4:128]!     // ! Finalizada, R4 se incrementa.
VLD1.8 {D0,D1,D2,D3},[R7:256], R2 // R2 offset que se suma a R7.
```

Alineación de datos

- Si bien hay un soporte a la alineación de datos a nivel de arquitectura ARMv7, además se incluye en las instrucciones NEON que operan con memoria, la capacidad de especificar la alineación de datos deseada.
- De mas está decir que ésta especificación debe ser consistente con el tipo de operandos de la instrucción.
- Si la dirección de memoria no resulta consistente con el alineamiento especificado, se genera una Data Abort exception.

```
VLD1.8 {D0},         [R1:64]   //D0[0]=[R1], D0[1]=[R1+1]...D0[7]=[R1+7]
VLD1.8 {D0,D1},      [R4:128]!     // ! Finalizada, R4 se incrementa.
VLD1.8 {D0,D1,D2,D3},[R7:256], R2 // R2 offset que se suma a R7.
```

- La alineación se expresa en la instrucción como `<Reg>:<align>`.

Alineación de datos

- Si bien hay un soporte a la alineación de datos a nivel de arquitectura ARMv7, además se incluye en las instrucciones NEON que operan con memoria, la capacidad de especificar la alineación de datos deseada.
- De mas está decir que ésta especificación debe ser consistente con el tipo de operandos de la instrucción.
- Si la dirección de memoria no resulta consistente con el alineamiento especificado, se genera una Data Abort exception.

```
VLD1.8 {D0},         [R1:64]   //D0[0]=[R1], D0[1]=[R1+1]...D0[7]=[R1+7]
VLD1.8 {D0,D1},      [R4:128]!     // ! Finalizada, R4 se incrementa.
VLD1.8 {D0,D1,D2,D3},[R7:256], R2 // R2 offset que se suma a R7.
```

- La alineación se expresa en la instrucción como `<Reg>:<align>`.
- En el manual de instrucciones ARMv7 la alineación se expresa como un número precedido de '@'. No es necesaria en el código fuente.

Tabla de contenidos

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

Resumen del comportamiento planteado en el slide anterior: Al llegar al extremo superior del rango, los procesadores convencionales realizan una operación de que se denomina wraparound, (reset del contador al valor inicial y activar flag de overflow). Si se decrementa un registro, por debajo de cero se pasa al valor máximo y se sigue desde allí.
Esto se conoce como *Aritmética de Desborde*.
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

Resumen del comportamiento planteado en el slide anterior: Al llegar al extremo superior del rango, los procesadores convencionales realizan una operación de que se denomina wraparound, (reset del contador al valor inicial y activar flag de overflow). Si se decrementa un registro, por debajo de cero se pasa al valor máximo y se sigue desde allí.
Esto se conoce como *Aritmética de Desborde*.
El costo es comprobar en cada ciclo de un lazo el valor del flag para decidir si se continúa en el lazo o no.

Cuando se implementa *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

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

Aritmética Saturada Signada

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

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

Aritmética Saturada No Signada

Aritmética Saturada vs. Aritmética de Desborde

Aritmética Saturada vs. Aritmética de Desborde

- Consideremos la siguiente suma empaquetada de 8 bytes.

Aritmética Saturada vs. Aritmética de Desborde

- Consideremos la siguiente suma empaquetada de 8 bytes.

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

Suma Empaquetada saturada vs. suma empaquetada de desborde

Aritmética Saturada vs. Aritmética de Desborde

- Consideremos la siguiente suma empaquetada de 8 bytes.

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

Suma Empaquetada saturada vs. suma empaquetada de desborde

- En el byte 6 se observa la diferencia entre ambas operaciones

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.

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.
- En ARM v7, el modificador Q que llevan eventualmente las instrucciones NEON, sirve para indicar que la operación implementa aritmética saturada.

Operaciones de punto flotante

Operaciones de punto flotante

- NEON no es full compatible con IEEE-754

Operaciones de punto flotante

- NEON no es full compatible con IEEE-754
- Cuando un número se desnormaliza se lo trunca a cero. (Denormalized are zero). En aplicaciones no críticas mejora la performance.

Operaciones de punto flotante

- NEON no es full compatible con IEEE-754
- Cuando un número se desnormaliza se lo trunca a cero. (Denormalized are zero). En aplicaciones no críticas mejora la performance.
- Si el resultado de una operación single precision es del rango $\pm 2^{-126}$ se reemplaza por 0. (Para double precision es del rango $\pm 2^{-1022}$)

Operaciones de punto flotante

- NEON no es full compatible con IEEE-754
- Cuando un número se desnormaliza se lo trunca a cero. (Denormalized are zero). En aplicaciones no críticas mejora la performance.
- Si el resultado de una operación single precision es del rango $\pm 2^{-126}$ se reemplaza por 0. (Para double precision es del rango $\pm 2^{-1022}$)
- No hay excepción si se denormaliza. Pero si se utiliza un número denormalizado, se genera una excepción inexacta.

Operaciones de punto flotante

- NEON no es full compatible con IEEE-754
- Cuando un número se desnormaliza se lo trunca a cero. (Denormalized are zero). En aplicaciones no críticas mejora la performance.
- Si el resultado de una operación single precision es del rango $\pm 2^{-126}$ se reemplaza por 0. (Para double precision es del rango $\pm 2^{-1022}$)
- No hay excepción si se denormaliza. Pero si se utiliza un número denormalizado, se genera una excepción inexacta.
- El redondeo es siempre al mas cercano excepto en operaciones de coversión

Operaciones de punto flotante

- NEON no es full compatible con IEEE-754
- Cuando un número se desnormaliza se lo trunca a cero. (Denormalized are zero). En aplicaciones no críticas mejora la performance.
- Si el resultado de una operación single precision es del rango $\pm 2^{-126}$ se reemplaza por 0. (Para double precision es del rango $\pm 2^{-1022}$)
- No hay excepción si se denormaliza. Pero si se utiliza un número denormalizado, se genera una excepción inexacta.
- El redondeo es siempre al mas cercano excepto en operaciones de coversión
- Solo trabaja con Single Precision

Operaciones de punto flotante

- NEON no es full compatible con IEEE-754
- Cuando un número se desnormaliza se lo trunca a cero. (Denormalized are zero). En aplicaciones no críticas mejora la performance.
- Si el resultado de una operación single precision es del rango $\pm 2^{-126}$ se reemplaza por 0. (Para double precision es del rango $\pm 2^{-1022}$)
- No hay excepción si se denormaliza. Pero si se utiliza un número denormalizado, se genera una excepción inexacta.
- El redondeo es siempre al mas cercano excepto en operaciones de coversión
- Solo trabaja con Single Precision
- Instrucciones separadas para escalares.

Operaciones de punto flotante

- NEON no es full compatible con IEEE-754
- Cuando un número se desnormaliza se lo trunca a cero. (Denormalized are zero). En aplicaciones no críticas mejora la performance.
- Si el resultado de una operación single precision es del rango $\pm 2^{-126}$ se reemplaza por 0. (Para double precision es del rango $\pm 2^{-1022}$)
- No hay excepción si se denormaliza. Pero si se utiliza un número denormalizado, se genera una excepción inexacta.
- El redondeo es siempre al mas cercano excepto en operaciones de coversión
- Solo trabaja con Single Precision
- Instrucciones separadas para escalares.
- Cada instrucción especifica si produce excepción de punto flotante y provee la información para su tratamiento.

Tabla de contenidos

Grupos de Instrucciones

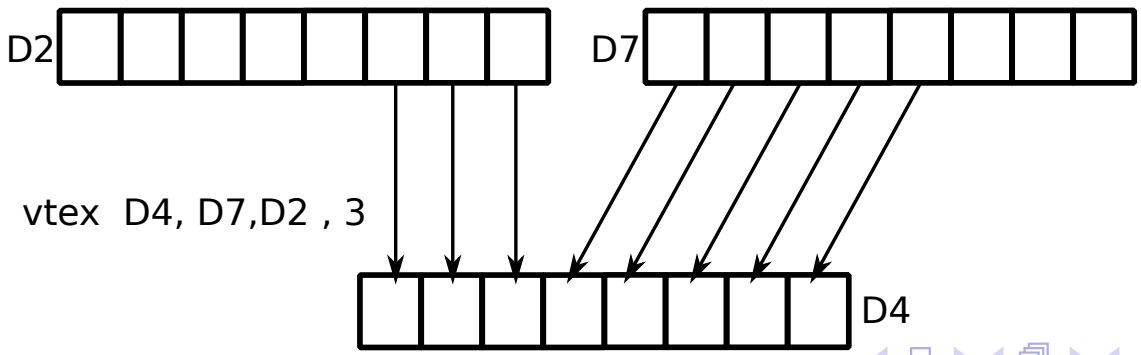
- General Data Processing
- Desplazamiento
- Lógicas y Comparación
- Aritméticas
- Multiplicación
- Load/Store

NEON General Data Processing Instructions

VCVT Convierte enteros de 32 bits o punto fijo de 32 bits a Floating Point Single Precisión, o visceversa. Si tine extensiones Half Precisión (el A8 no las tiene), acepta un operando de mitad de tamaño para colocar allí el vector de F16. En este caso la conversión es solo F32 y F16. No incluye enteros, ni punto fijo.

VDUP Toma el escalar definido en el registro origen (NEON o ARM), lo duplica, y lo escribe a lo largo del vector de destino.

VEXT Extrae #imm elementos de 8 bit del extremo menos significativo del primer operando y los lleva al extremo mas significativo del registro destino, que se completa con los restantes elementos de 8 bit del extremo mas significativo del segundo operando.

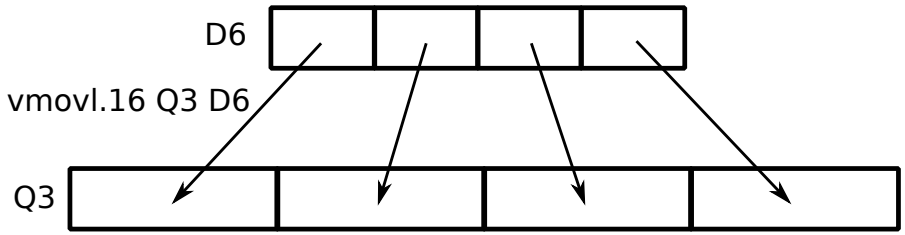


NEON General Data Processing Instructions

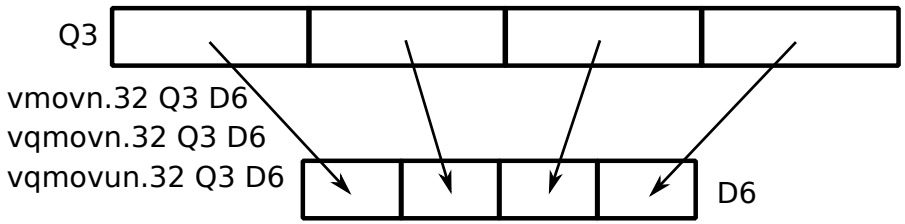
VMOV Rellena un vector con un dato inmediato: I8, I16, I32, I64, F32.

VMVN Rellena un vector con la inversa de un dato inmediato: I8, I16, I32, I64, F32.

VMOVL Extiende rango.



VMOVN Q Saturation, U Unsigned result.



VREV Vector Reverse Halfword. Permuta las halfwords de cada word de un registro origen y almacena el resultado en un registro destino.

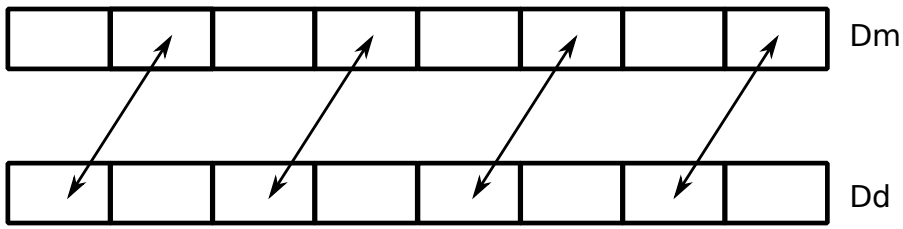
NEON General Data Processing Instructions

VSWP Intercambia los lanes de cada registro destino con los del registro origen

VTBL Busca un vector de índices en una tabla y genera un vector de resultados. Los índices fuera de rango retornan 0.

VTBX Igual que VTBL excepto que un índice fuera de rango no modifica el elemento del correspondiente vector destino.

VTRN Asume sus dos operandos como conformadores de un vector de elementos donde cada elemento es una matriz de 2x2, y simplemente las transpone.



NEON General Data Processing Instructions

VUZP Vector Unzip. Desentrelaza elementos de dos vectores.

Operandos antes de VUZP.8

A7	A6	A5	A4	A3	A2	A1	A0	Dm
B7	B6	B5	B4	B3	B2	B1	B0	Dd

Resultado de VUZP.8

B6	B4	B2	B0	A6	A4	A2	A0
B7	B5	B3	B1	A7	A5	A3	A1

VZIP Vector Zip. Entrelaza los elementos de dos vectores.

Operandos antes de VZIP.8

A7	A6	A5	A4	A3	A2	A1	A0	Dm
B7	B6	B5	B4	B3	B2	B1	B0	Dd

Resultado de VZIP.8

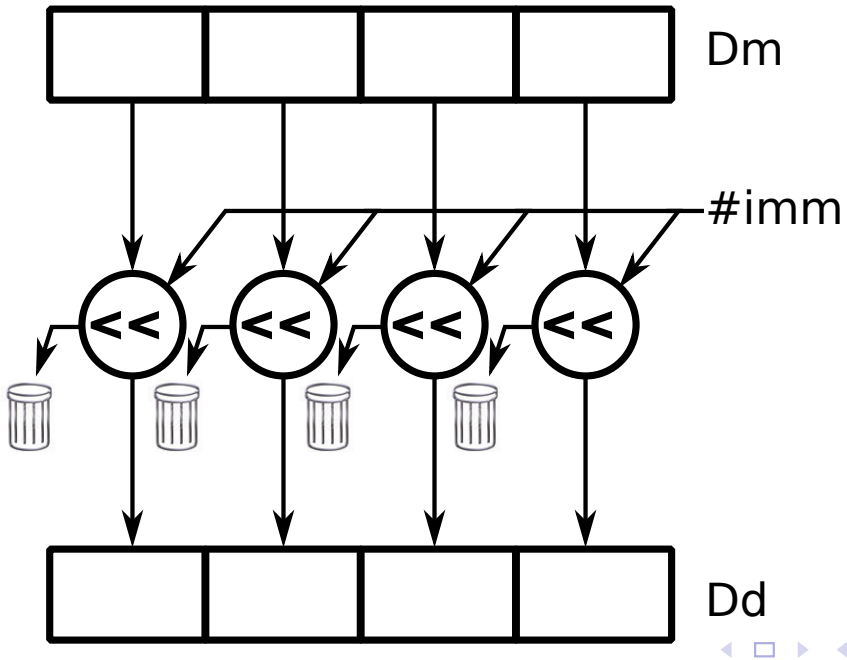
B3	A3	B2	A2	B1	A1	B0	A0
B7	A7	B6	A6	B5	A5	B4	A4

NEON Shift Instructions

VSHL Vector Shift Left. Desplaza a izquierda cada elemento del vector, la cantidad de bits `#imm` descartando los bits que salen del elemento por derecha.

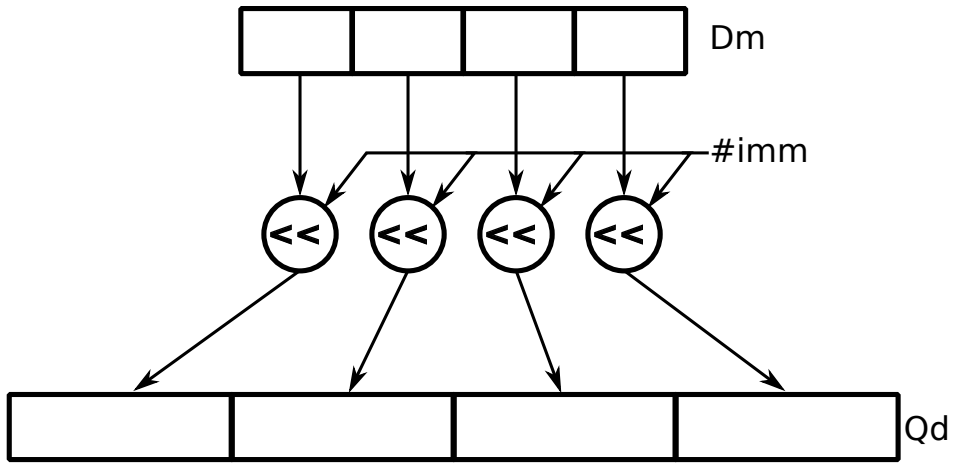
VQSHL Vector Saturating Shift Left. Setea el bit QC (FPSCR[27]), para indicar que ocurrió una saturación.

VQSHLU Vector Saturating Shift Left Unsigned.Setea el bit QC (FPSCR [27]), para indicar que ocurrió una saturación.



NEON Shift Instructions

VSHLL Vector Shift Left Long. Toma un vector de tamaño doble word, desplaza a izquierda cada elemento del vector, la cantidad de bits `#imm` y almacena los resultados en un vector de tamaño Quad Word de la misma cantidad de elementos.



Tipos de datos: I8 , I16 , I32 , I64 para VSHL, S8 , S16 , S32 para VSHLL , VQSHL , o VQSHLU, U8 , U16 , U32 para VSHLL o VQSHL, S64 para VQSHL o VQSHLU, U64 para VQSHL .

NEON Shift Instructions

V{Q}{R}SHL Vector Shift Left por variable signada. Desplaza los elementos de un vector en función del número almacenado en los elementos de otro vector. El número puede ser signado (Positivo desplaza a la izquierda, y negativo a la derecha). El resultado se almacena en un tercer registro. Si se emplea la instrución con saturación, se setea el bit QC (FPSCR[27]), para indicar que ocurrió una saturación.

Sintaxis:

```
V{Q}{R}SHL{cond}.datatype {Qd,} Qm, Qn
V{Q}{R}SHL{cond}.datatype {Dd,} Dm, Dn
```

Q, indica saturación,
R Redondeo (si no se especifica, se truncan los resultados)
Qd/Dd Registro destino
Qm/Dm Registro Operando 1
Qn/Dn Registro Operando 2
datatype: S8, S16, S32, S64, U8, U16, U32, o U64.

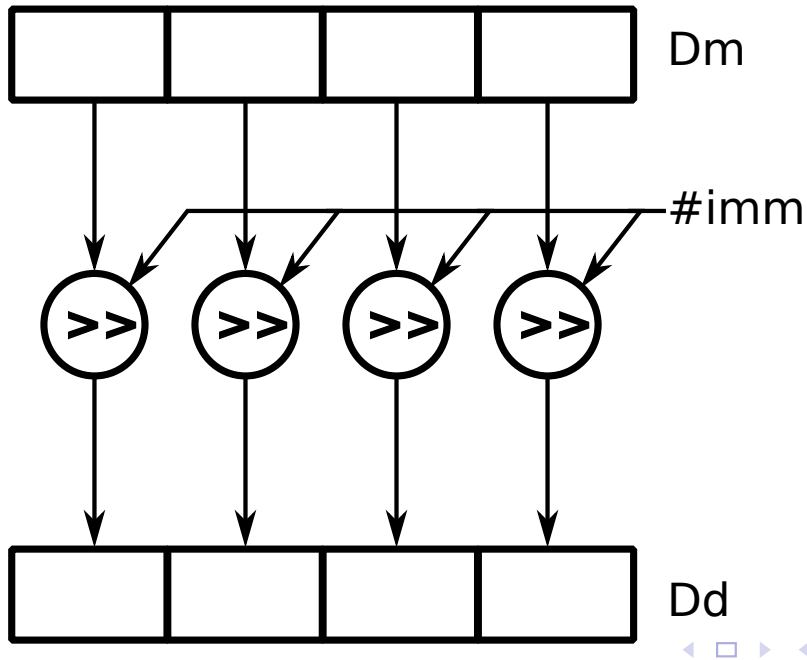
NEON Shift Instructions

VSHR Vector Shift Right. Desplaza a derecha cada elemento del vector, la cantidad de bits `#imm`.

VRSHR Idem con redondeo del resultado (Sin la primer R, trunca).

VSHRN Idem con Narroewd del resultado (en este caso el primer operando es un registro Q).

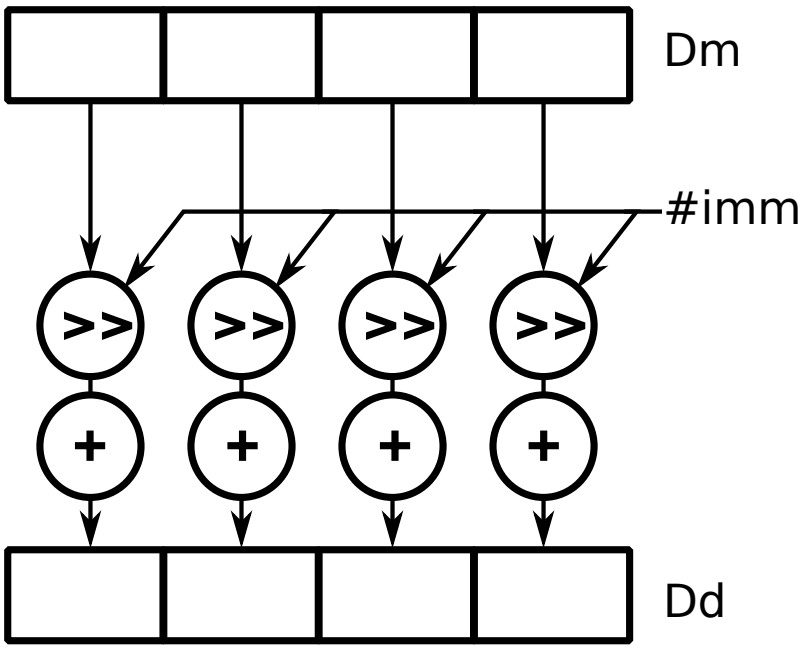
VRSHRN Idem con redondeo y Narrowed del resultado.



NEON Shift Instructions

VSRA Vector Shift Right and Accumulate. Desplaza a derecha cada elemento del vector, la cantidad de bits `#imm` y en lugar de copiarlo al destino se lo acumula.

VRSRA Idem con redondeo del resultado (Sin la primer R, trunca).

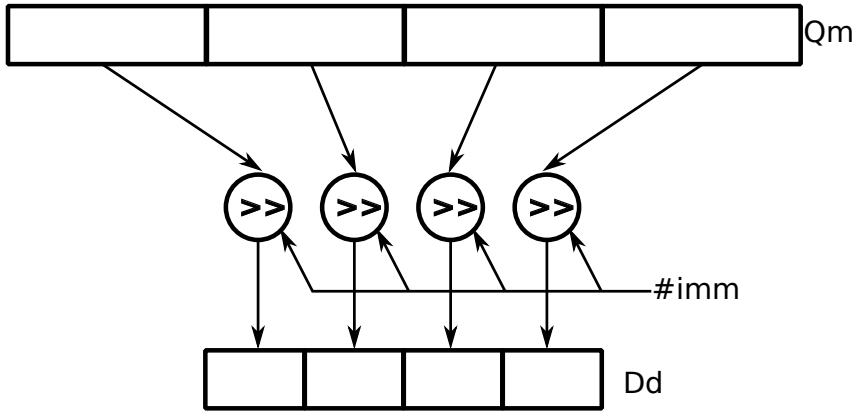


NEON Shift Instructions

VQSHRN Vector Saturating Shift Right Narrow.

VRQSHRN Idem con redondeo del resultado.

VRQSHRUN Idem con números no signados.

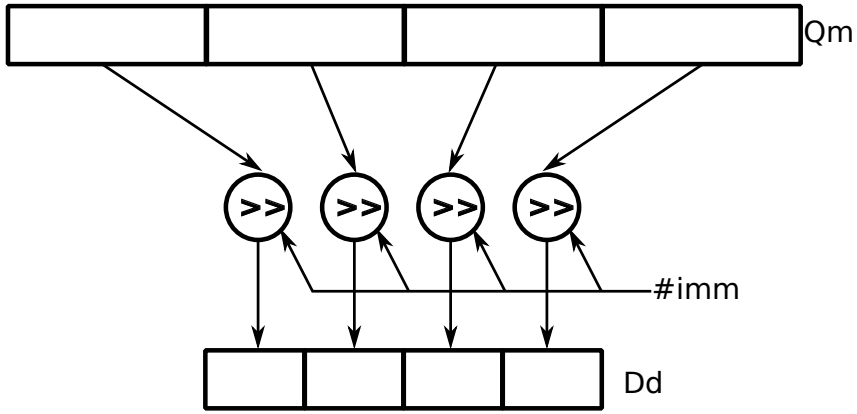


NEON Shift Instructions

VQSHRN Vector Saturating Shift Right Narrow.

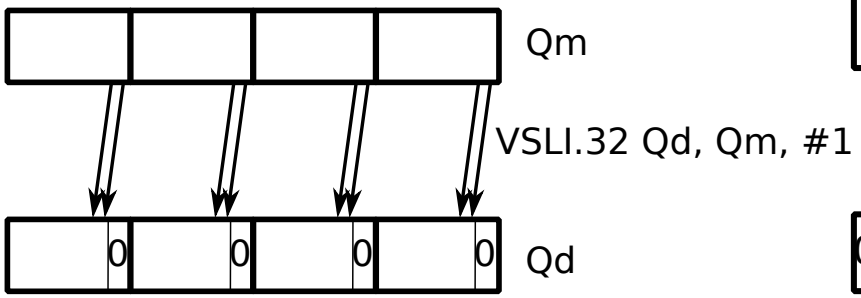
VRQSHRN Idem con redondeo del resultado.

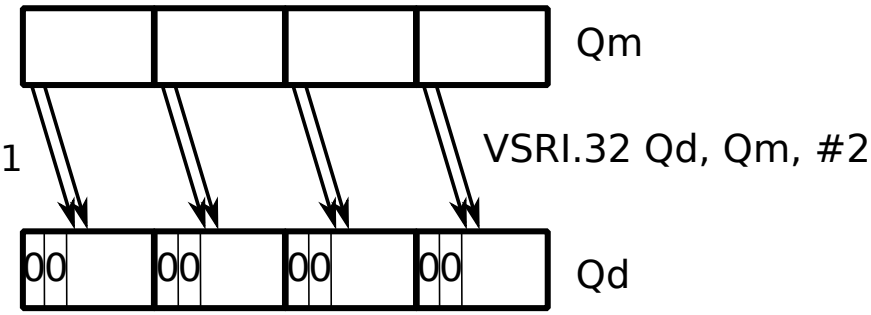
VRQSHRUN Idem con números no signados.



VSLI Vector Shift Left and Insert.

VSRI Vector Shift Left and Insert.





NEON logical and compare operations

VAND AND Bitwise entre dos registros. Resultado en el operando destino. Datatype es ignorado.

VEOR XOR Bitwise entre dos registros. Resultado en el operando destino. Datatype es ignorado.

VORR OR Bitwise entre dos registros. Resultado en el operando destino. O inmediato a registro (`#imm`). Datatype es ignorado.

VORN OR Bitwise entre dos registros (uno Negado previamente). Resultado en el operando destino. Datatype es ignorado.

VMVN Vector Bitwise NOT. Invierte bit a bit el valor del registro Operando en el registro Destino.

VMOV Vector Mov. Copia un registro en otro.

VTST Vector Test Bits. Ejecuta un AND Bitwise por cada elemento en dos registros. Si el resultado es cero, el elemento correspondiente del operando destino se rellena de '0's. De otro modo se rellena de '1's.

NEON logical and compare operations

VBIC Vector Bitwise Clear. Dos modos: Inmediato o registro. Inmediato toma el valor inmediato como una máscara. Aquellos bit que están en ’1’en la máscara llevan a limpiar (setear ’0’) en el bit correspondiente del registro resultado. Modo Registro. Operando 1 valos a limpiar, Operando 2 Máscara. Ambos operandos no cambian, ya que se aplica en el registro resultado.

VBIF Vector Bitwise Insert if False. Operando 2 trabaja como máscara. Por cada ’0’ en el Operando 2 el bit correspondiente del Operando 1 reemplaza (se inserta en) al bit correspondiente del resultado, que de otro modo no se altera.

VBIF Vector Bitwise Insert if True. Idem anterior pero por cada ’1’ en Operando 2.

VBSL Vector Bitwise Select. El Operando Destino es una máscara. Por cada ’1’ toma el bit del operando 1 y por cada ’0’ lo toma del operando 2. El valor compuesto reemplaza la máscara.

NEON logical and compare operations

VACop Vector Absolute Compare. `op` puede ser GE = Greater or Equal, o GT = Greater Than. Compara dos vectores Datatype F32 elemento a elemento en valor absoluto. Cada elemento del operando destino se llena de '1's o '0's, si el par de elementos de los operandos verifica o no la condición. El tipo de dato del resultado es I32.

VCop Vector Compare. Compara dos vectores elemento a elemento. La condición `op`, puede resultar True, en cuyo caso, el elemento destino se completa con '1's, o False, en cuyo caso se completa con '0's.
`op`: EQ=Equal, GE=Greater or Equal, GT=Greater Than, LE=Lowe
or Equal, LT=Lower Than. LE y LT valen solo si el segundo operando es #0.
Trabajan con diferentes tipos de datos tanto los operandos como el resultado de acuerdo con la condición.

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.

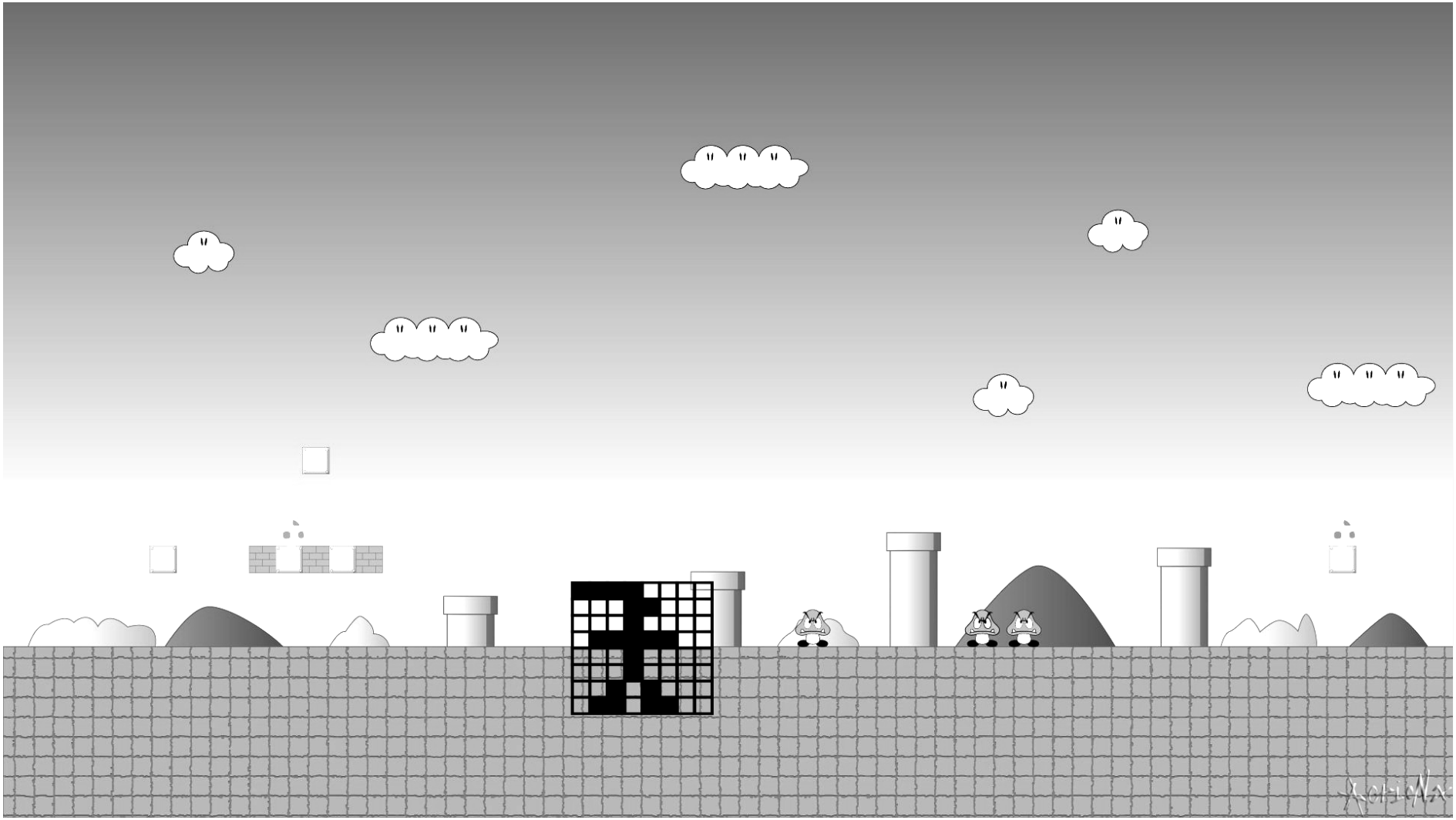
- Un sprite es un mapa de bits en el que se dibuja una figura pequeña, de modo tal que los bits que no se utilizan deben quedar “transparentes”, es decir, quedan del color del fondo.
- Supongamos un sprite de 8 pixels de ancho por otros 8 pixels de alto.
- Para la primer línea de 8 pixels, el dibujo 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 **`D0`** la primer línea del sprite, (8 bytes empaquetados), y **`D2`** trabaja como auxiliar, está pre seteado en cero.

Ejemplo de uso de comparación empaquetada

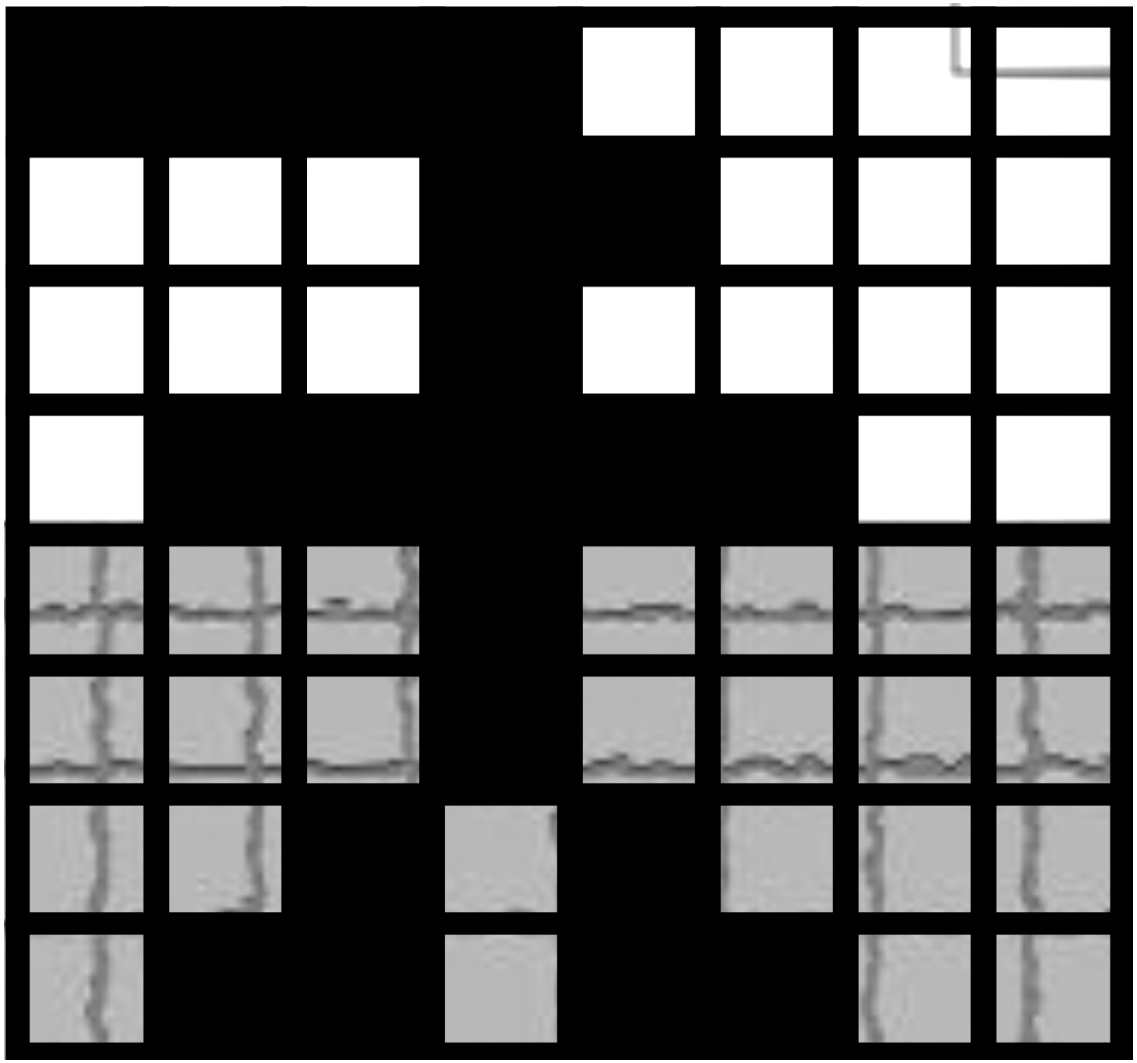
Procesamiento de sprites o iconos en gráficos 2D.

- 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 `D0` la primer línea del sprite, (8 bytes empaquetados), y `D2` trabaja como auxiliar, está pre seteado en cero.
- Luego `VCEQ.8` entre ambos registros, generará una máscara con los bytes que contendrán información del sprite en cero y los que deben resultar transparentes en 0xFF.

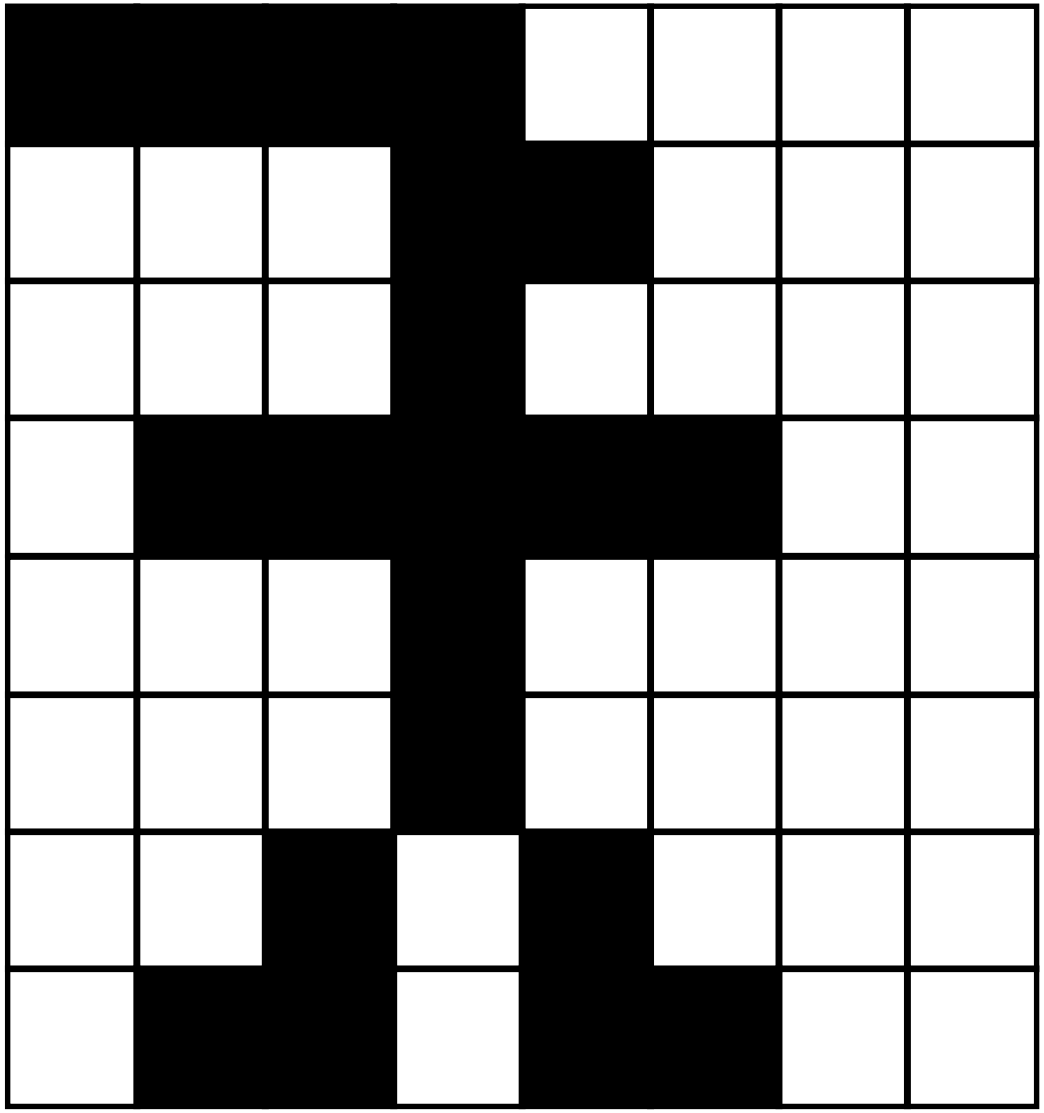
Ejemplo de uso de comparación empaquetada



Ejemplo de uso de comparación empaquetada



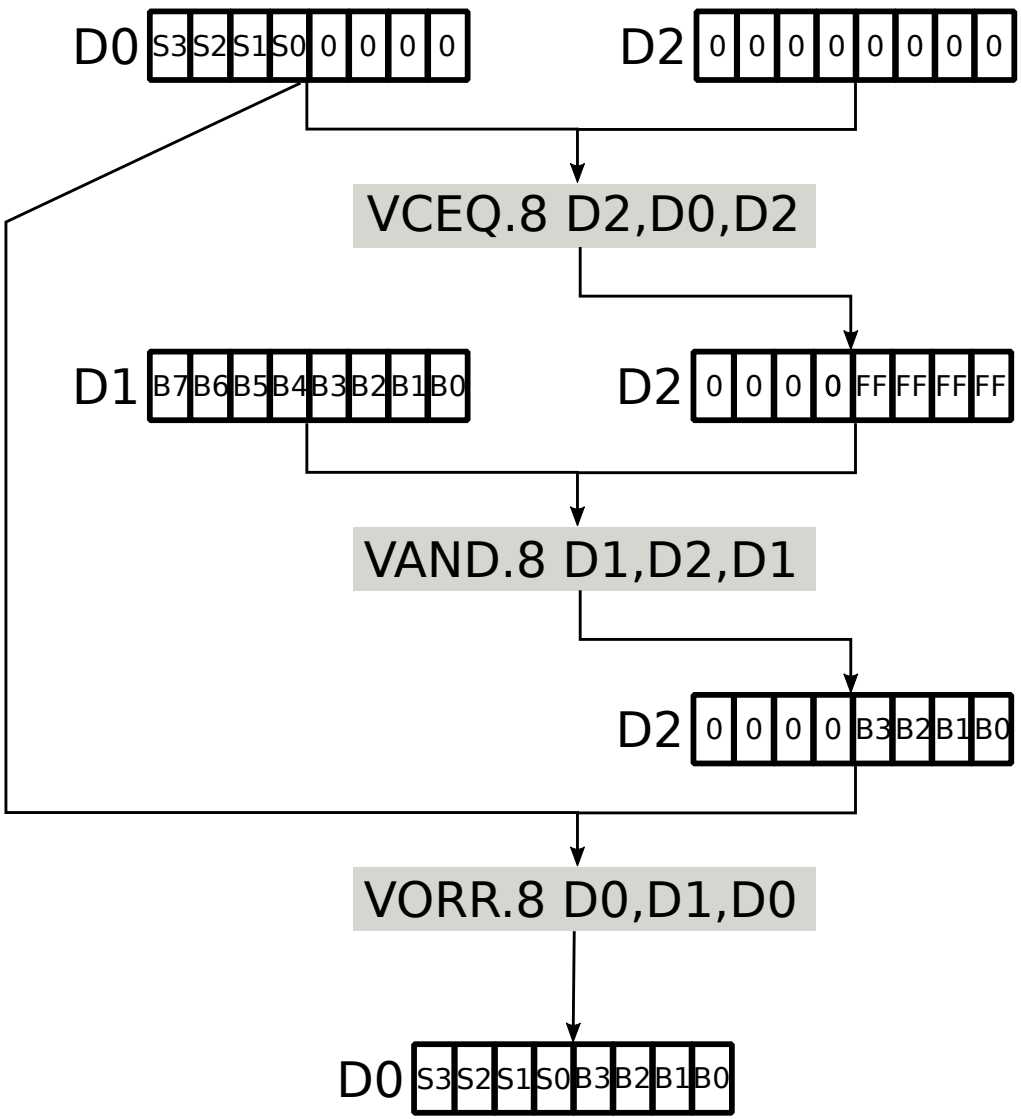
Ejemplo de uso de comparación empaquetada



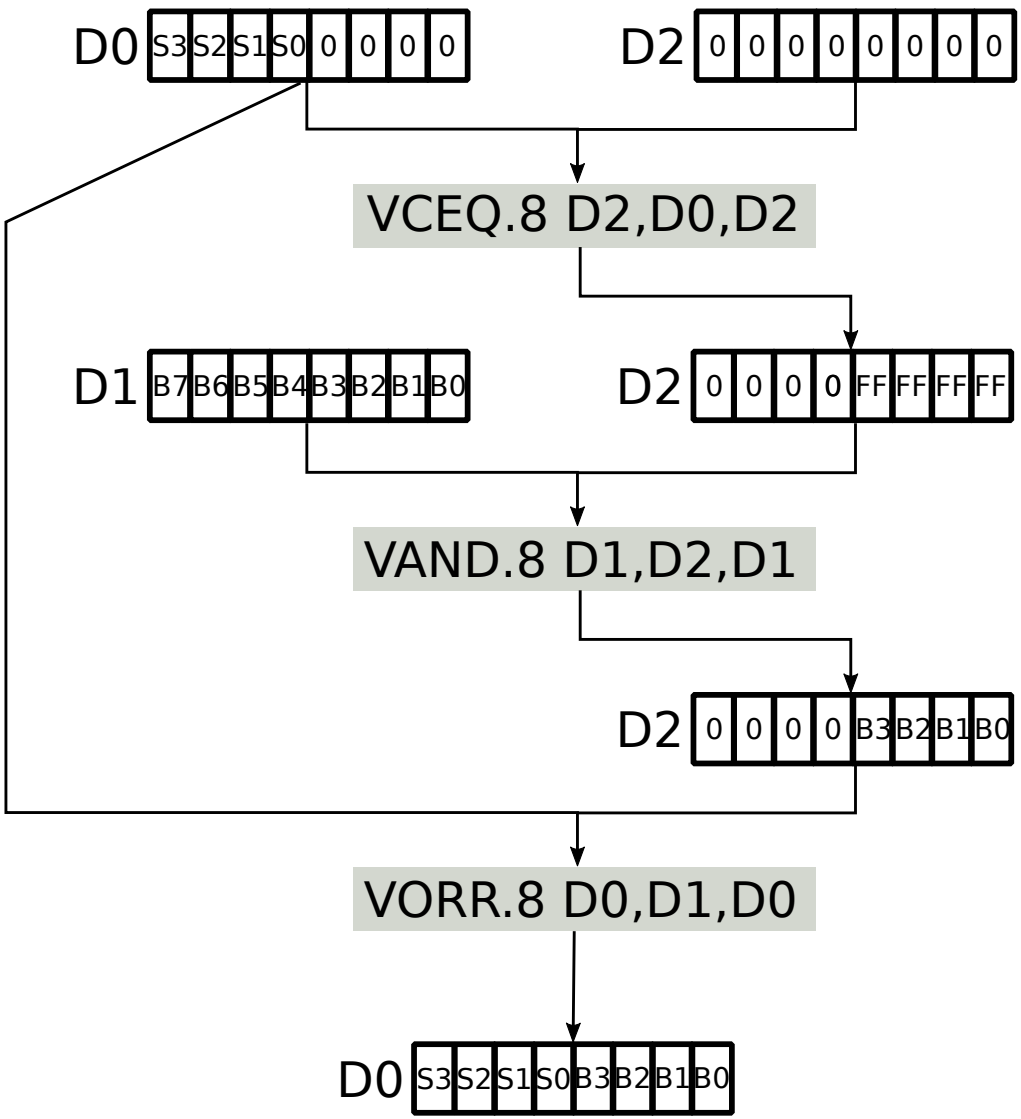
Ejemplo de uso de comparación empaquetada

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

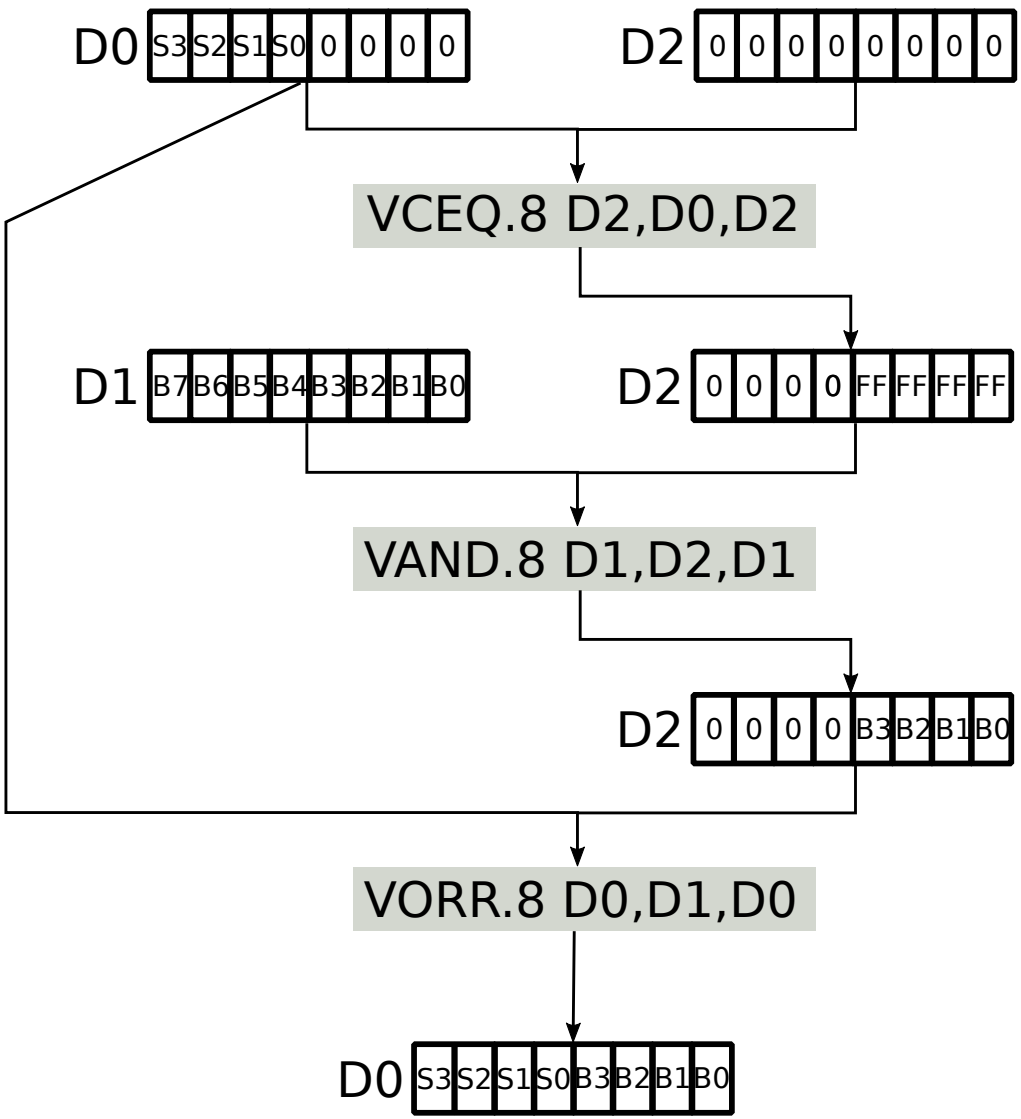


Ejemplo de uso de comparación empaquetada



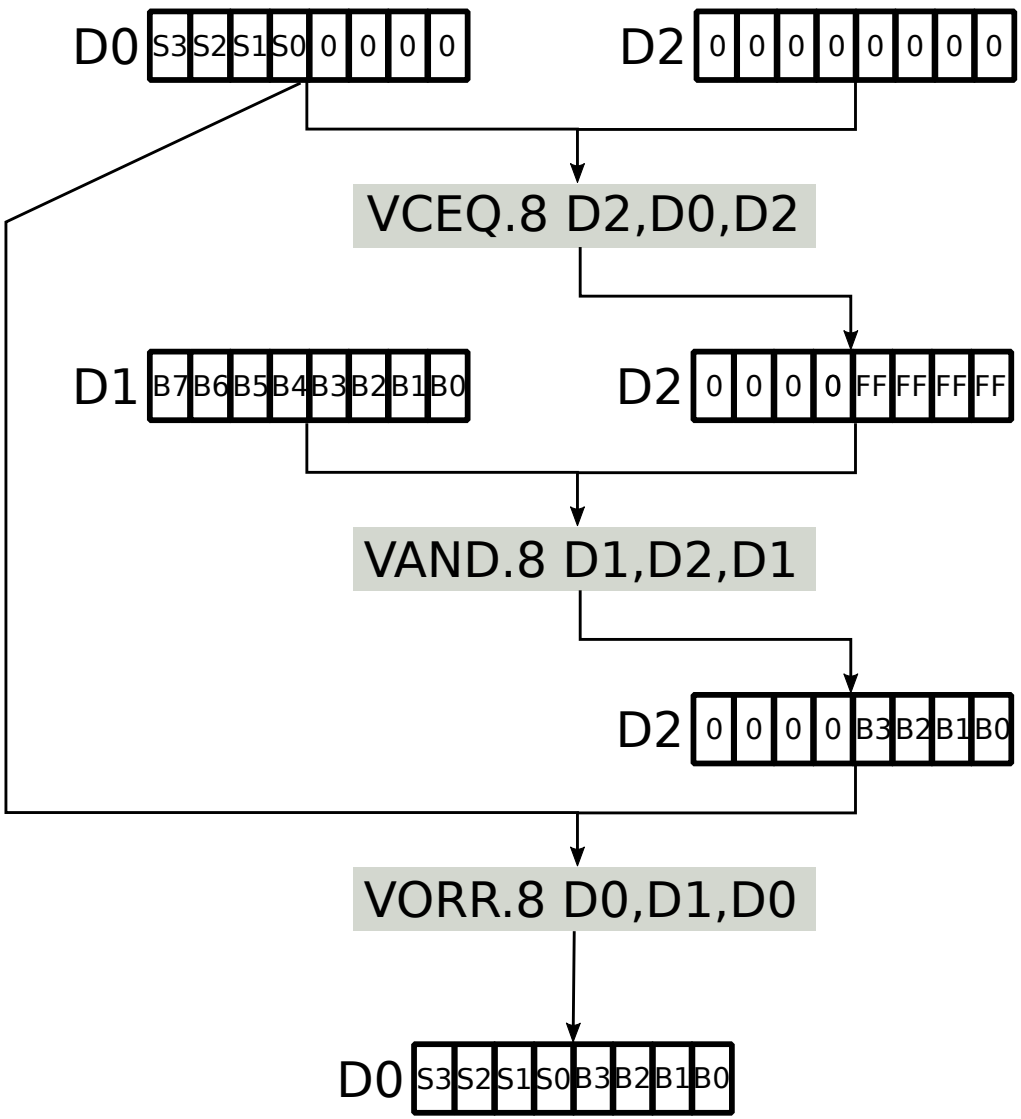
- El resultado se procesa con la información de los 8 pixels del fondo de la imagen (pre almacenado en el registro **`D1`**) a través de un operador AND. Al resultado le queden solo los cuatro pixels del fondo que se deben mostrar.

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 **`D1`**) a través de un operador AND. Al resultado le queden 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 **`D1`**) a través de un operador AND. Al resultado le queden 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.

NEON arithmetic instructions

VABA Vector Absolute Difference and Acumulate. Calcula la diferencia entre los elementos de los Vectores de operandos, y acumula el valor absoluto de las mismas en el operando destino. Hay una versión Long.

VABD Vector absolute Difference. Idem **VABA**, pero almacena el valor absoluto de las diferencias en el operando destino. Hay una versión Long.

VABS Vector Absolute. Calcula el valor absoluto de los elementos de un vector y almacena su resultado. En el caso de los datatypes de punto flotante simplemente pone '0' en el bit mas significativo. Hay una versión Saturada (VQABS), solo para enteros signados, en donde además se setea el flag QC.

VADD Vector Add. Suma los elementos de cada vector de operandos y almacena los resultados en lso elementos correspondientes del Registro destino. Tiene versión es Saturada, Long y Narrow.

NEON arithmetic instructions

VADDHN Vector Add and Narrow selecting High half. Suma los elementos de los vectores en los registros Operandos, selecciona la mitad alta de los resultados y los almacena en los elementos del vector destino.

VSUBHN Vector Subtract and Narrow, selecting High Half. Idem pero Resta.

VCLS Vector Count Leading Sign Bits. Cuenta la cantidad de bits consecutivos desde el MSB que coincidan en valor con este y almacena cada cuenta en el elemento correspondiente del registro Destino.

VCLZ Vector Count Leading Zeros. Cuenta la cantidad de ceros consecutivos iniciando en el MSB y almacena cada cuenta en el elemento correspondiente del registro Destino.

VCNT Vector Count Set Bits. Idem **VCLZ**, pero cuenta '1's.

NEON arithmetic instructions

VHADD Vector Halving Add. Suma los elementos de los dos vectores contenidos en los registros operandos, desplaza el resultado a la derecha 1 bit, y almacena cada resultado en el elemento del vector destino. El resultado puede ser redondeado (**VRHADD**), o truncado.

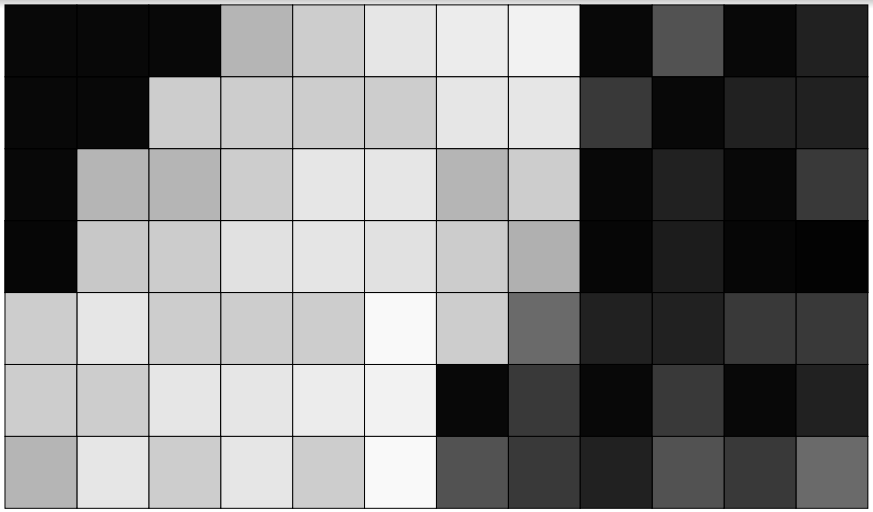
VHSUB Vector Halving Subtract. Idem anterior excepto que resta en lugar de sumar los operandos. No hay redondeo o truncado.

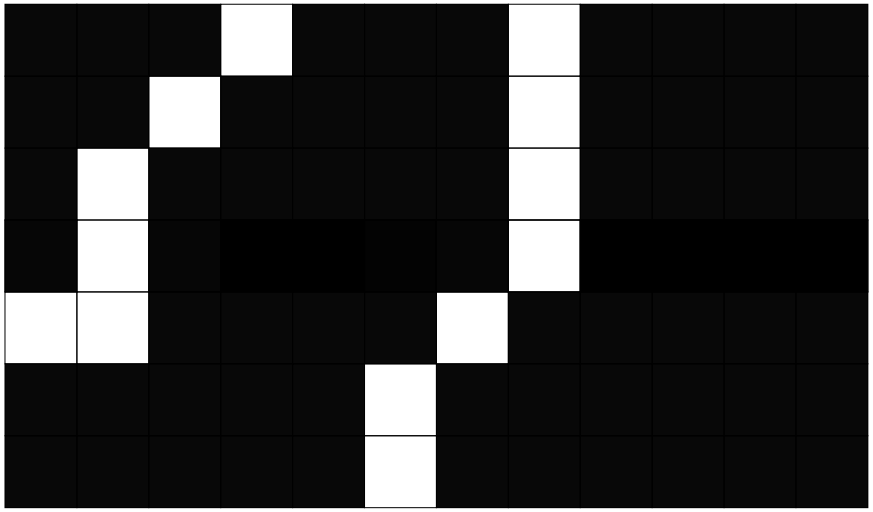
VMAX Compara los elementos de dos vectores y almacena en el vector destino el máximo en cada caso.

VMIN Compara los elementos de dos vectores y almacena en el vector destino el mínimo en cada caso.

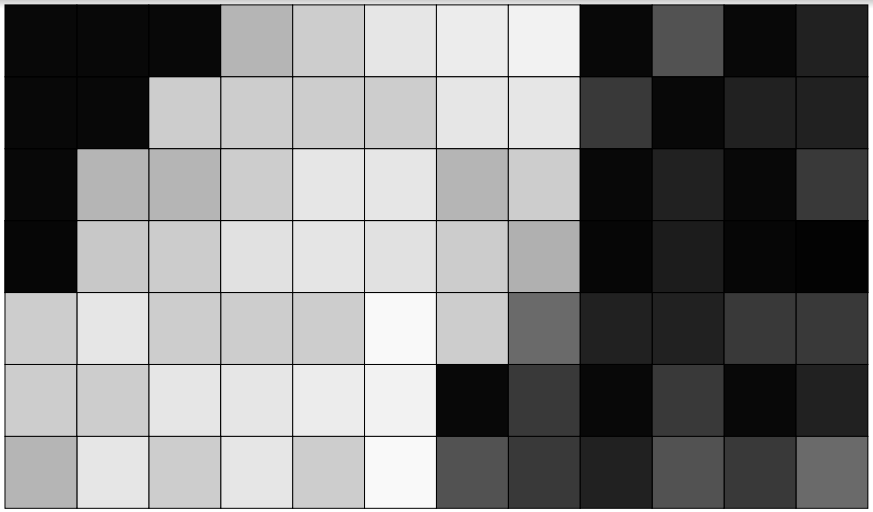
VNEG Niega cada elemento de un vector y escribe los resultados en el vector destino. Si el datatyp es Floating Point, invierte el signo. Si el datatype es Entero, dispone de **VQNEG**, como opción con saturación.

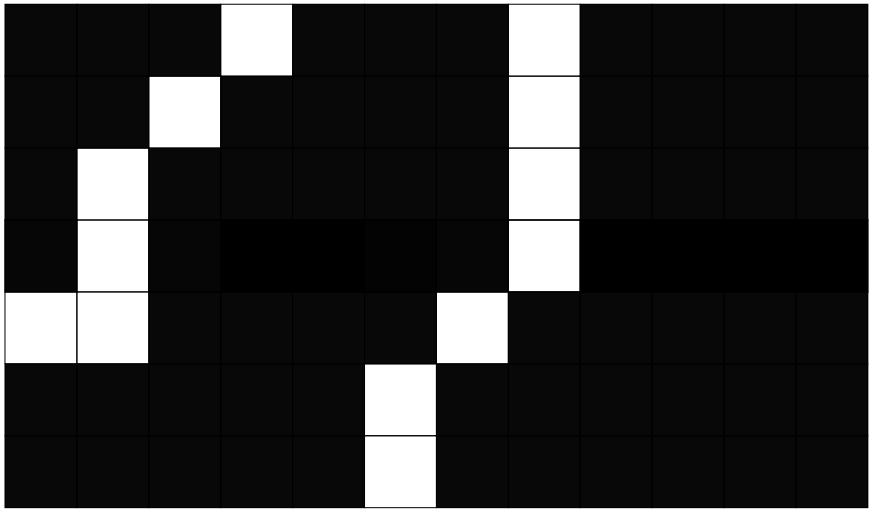
Ejemplo de procesamiento de imágenes





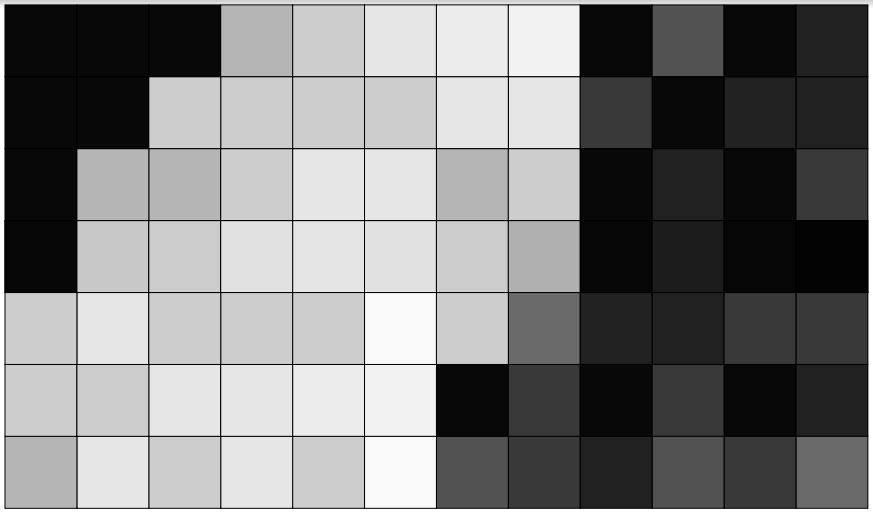
Ejemplo de procesamiento de imágenes

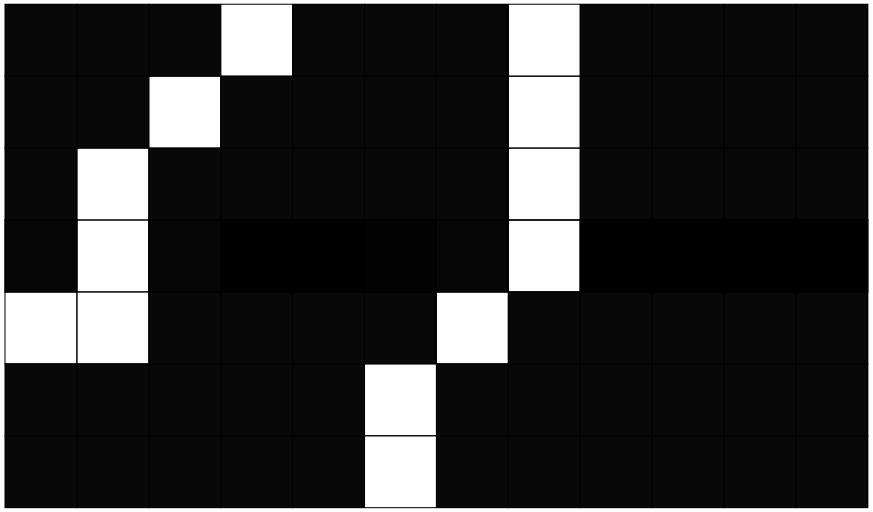




- Un borde es una transición brusca de iluminación entre dos pixles vecinos.

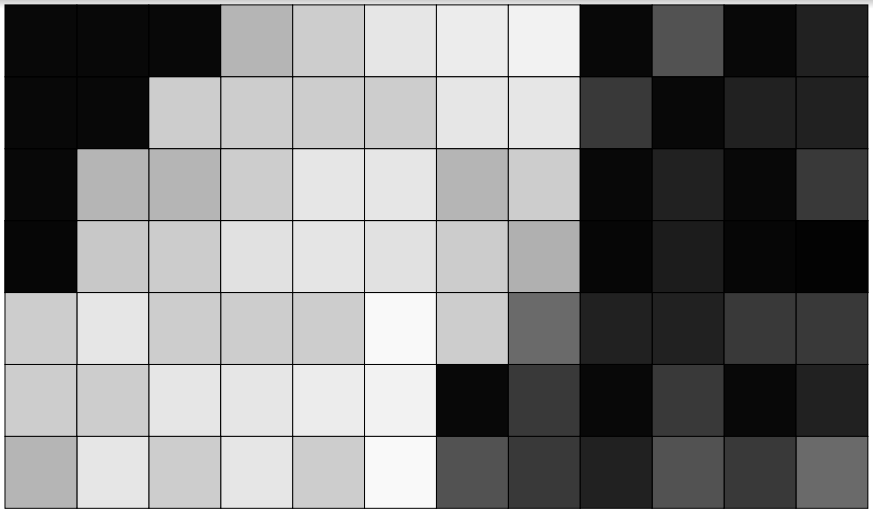
Ejemplo de procesamiento de imágenes

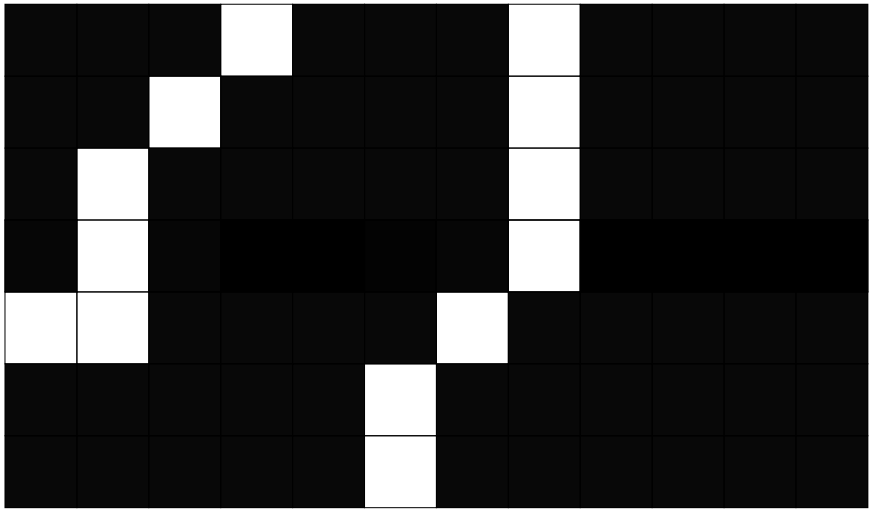




- Un borde es una transición brusca de iluminación entre dos pixles vecinos.
- En el mapa de bits de la izquierda se muestra como queda una imagen resultante de la detección de bordes

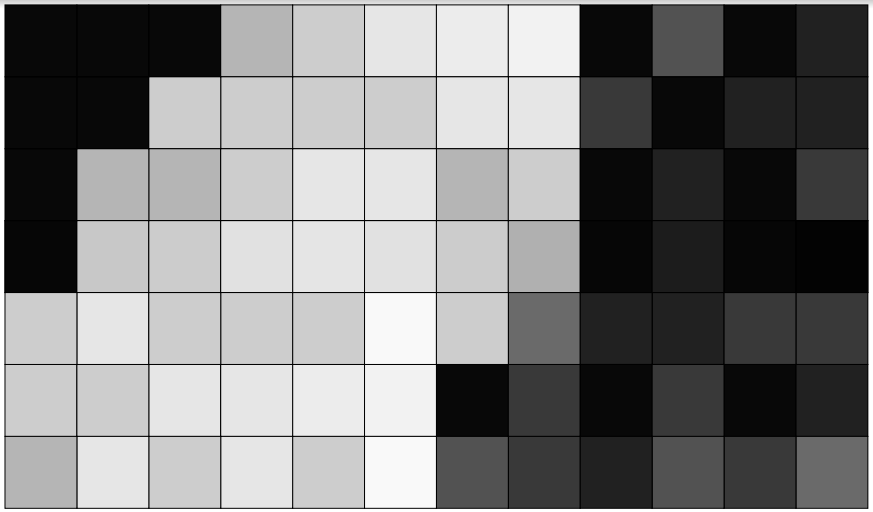
Ejemplo de procesamiento de imágenes

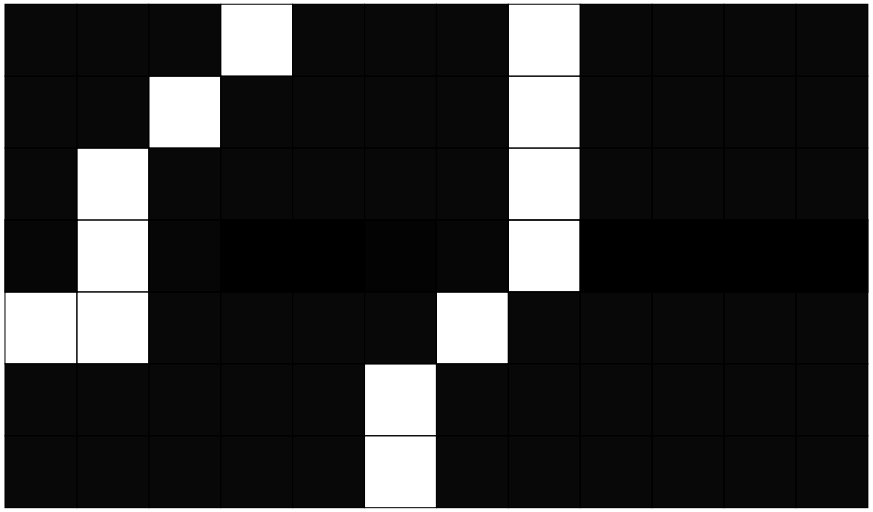




- Un borde es una transición brusca de iluminación entre dos pixles vecinos.
- En el mapa de bits de la izquierda se muestra como queda una imagen resultante de la detección de bordes
- Un método práctico es por cada pixel tomar los ocho vecinos (N8), y evaluar su diferencia con el mínimo del N8

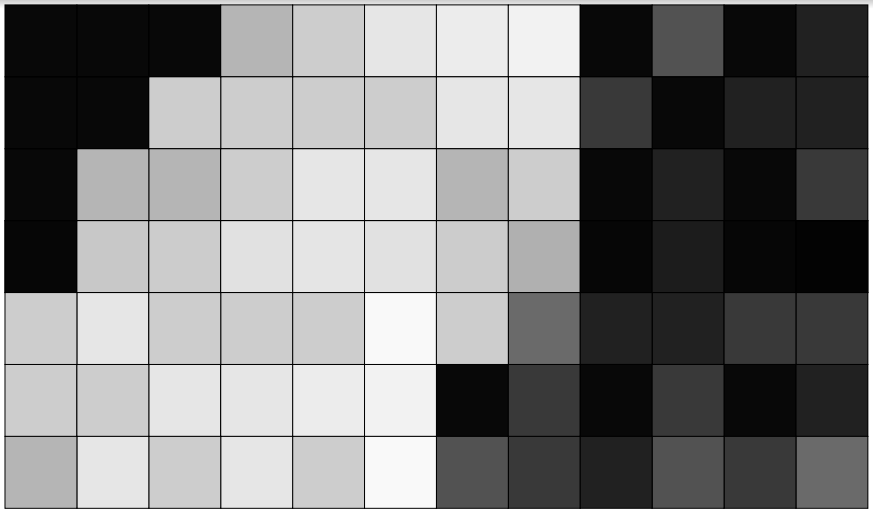
Ejemplo de procesamiento de imágenes

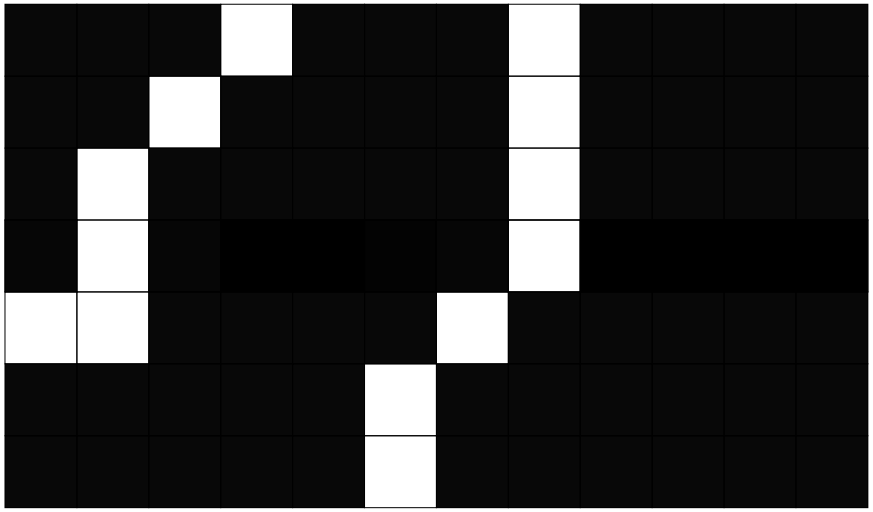




- Un borde es una transición brusca de iluminación entre dos pixles vecinos.
- En el mapa de bits de la izquierda se muestra como queda una imagen resultante de la detección de bordes
- Un método práctico es por cada pixel tomar los ocho vecinos (N8), y evaluar su diferencia con el mínimo del N8
- Si la diferencia es muy grande el valor resultante está próximo al nivel de blanco.

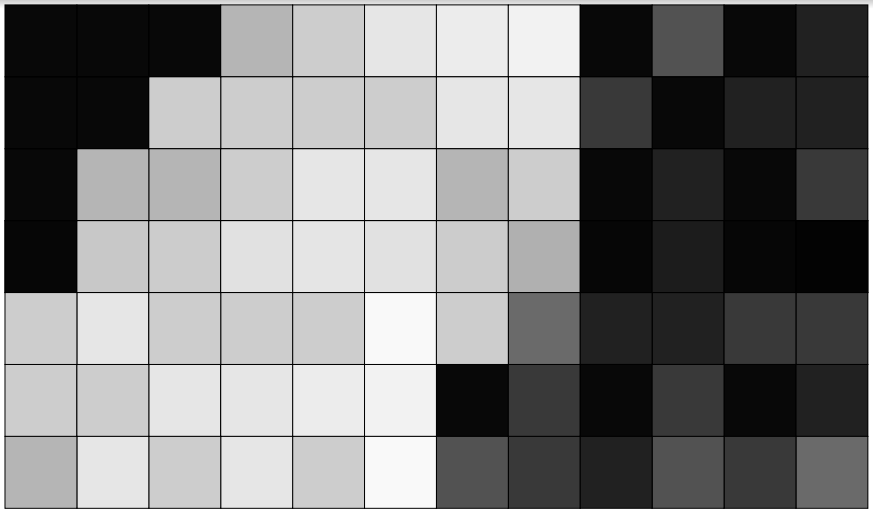
Ejemplo de procesamiento de imágenes

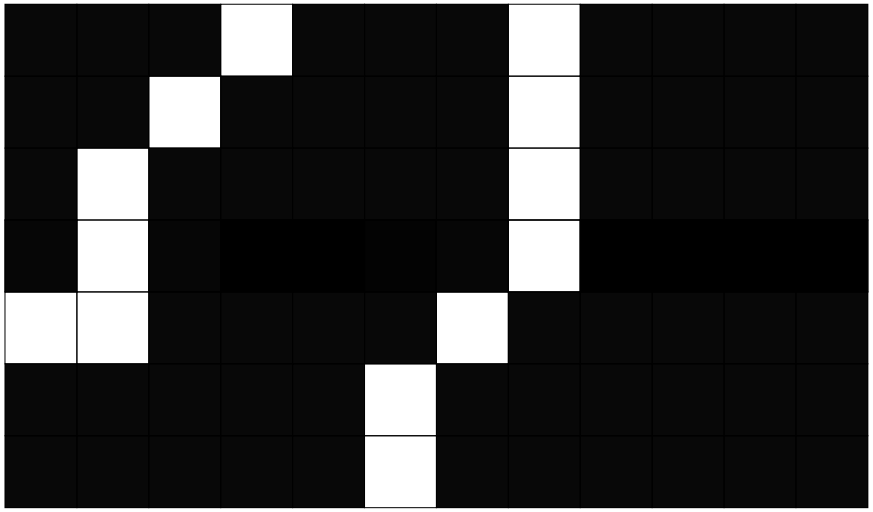




- Un borde es una transición brusca de iluminación entre dos pixles vecinos.
- En el mapa de bits de la izquierda se muestra como queda una imagen resultante de la detección de bordes
- Un método práctico es por cada pixel tomar los ocho vecinos (N8), y evaluar su diferencia con el mínimo del N8
- Si 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 pixles 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 (N8), y evaluar su diferencia con el mínimo del N8
- 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 MIN(N8), se puede obtener una buena aproximación de los bordes de una imagen

Ejemplo

Supongamos los registros R2, R3, y R4 como punteros a tres áreas de meoria cada una de las cuales contiene una fila de píxeles de una imagen monocromo. Las tres filas son por supuesto consecutivas.

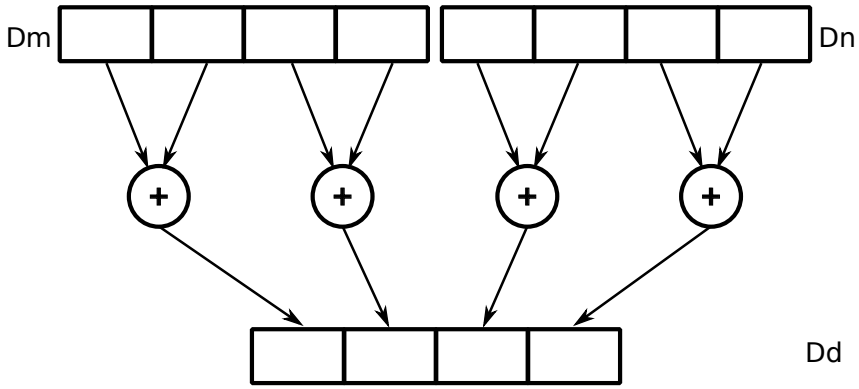
Ejemplo

Supongamos los registros R2, R3, y R4 como punteros a tres áreas de meoria cada una de las cuales contiene una fila de píxeles de una imagen monocromo. Las tres filas son por supuesto consecutivas.

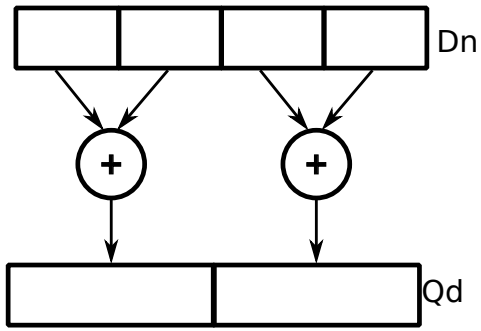
```
// Método cálculo por N8
      vldr.u64 {D0–D5},   r2! //q0,q1,q2 primer fila
      vldr.u64 {D6–D11},  r3! //q4,q5,q6 segunda fila
      //(q5 = lista de pivxeles de interés)
      vldr.u64 {D12–D17}, r4! //q7,q8,q9 tercer fila
// Calcula el mínimo de los bytes empaquetados en cada registro
      vmin.U8 Q0,Q0,Q1
      vmin.U8 Q0,Q0,Q2
      vmin.U8 Q0,Q0,Q3
      vmin.U8 Q0,Q0,Q4
      vmin.U8 Q0,Q0,Q6
      vmin.U8 Q0,Q0,Q7
      vmin.U8 Q0,Q0,Q8
      vmin.U8 Q0,Q0,Q9      //Mínimo de los N8 en Q0.
      vsub.U8 Q10,Q5,Q0 //Restamos para hallar los bordes
```

NEON arithmetic instructions

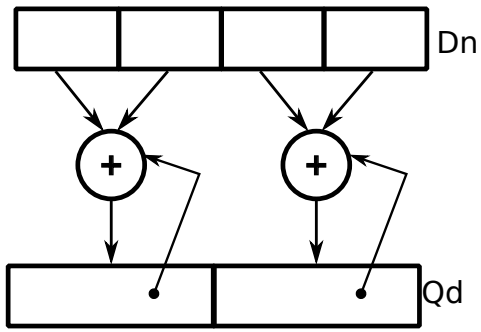
VPADD Vector Pairwise Add.



VPADDL Vector Pairwise Add Long .

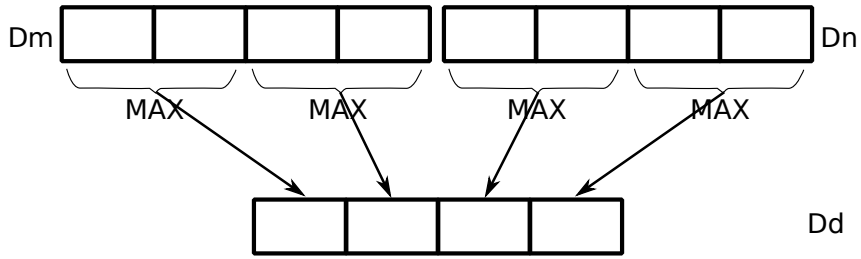


VPADAL Vector Pairwise Add and Accumulate Long .



NEON arithmetic instructions

VPMAX Vector Pairwise Maximum.



VPMIN Vector Pairwise Maximum.

VRECPE Vector Reciprocal Estimate. Calcula el recíproco estimado de cada elemento de un vector y lo guarda en el vector resultado.

VRECPS Vector Reciprocal Step. Multiplica dos vectores elemento a elemento, y resta cada producto de 2.0, y almacena el resultado final en el elemento del vector destino. Es parte de la iteración del algoritmo de Newton-Raphson.

VRSQRTE Vector Reciprocal Square Root Estimate. Calcula el recíproco estimado de la raíz cuadrada de cada elemento de un vector y lo guarda en el vector resultado.

NEON arithmetic instructions

VRSQRTS Vector Reciprocal Square Root Step. Multiplica dos vectores elemento a elemento, resta cada producto de 3.0, y divide cada resultado por 2.0, almacenando en el elemento del vector destino el resultado final. Es parte de la iteración del algoritmo de Newton-Raphson.

VSUB Vector Subtract. Resta dos vectores elemento a elemento, y almacena cada resultado en el elemento correspondiente del vector resultado. Tiene versiones saturada (**VQSUB**), Wide (**VSUBW**), y Long (**VSUBL**).

NEON Multiply instructions

VFMA Vector Fused Multiply Accumulate. Multiplica dos vectores elemento a elemento, y acumula cada resultado en el elemento correspondiente del vector resultado. No redondea antes de la suma, lo cual la convierte en la alternativa de menor pérdida de precisión.

VFMS Vector Fused Multiply Substract. Igual que la anterior, pero resta el resultado del producto de los elementos destino.

VMUL Vector Multiply. Multiplica dos vectors elemento a elemento y almacena cada resultado en el correspondiente elemento del vector resultado. Tiene versiones Long (VMULL), y escalar.

VMLA Vector Multiply Accumulate. Es como **VFMA** pero redondea antes de la suma.Tiene versiones Long (VMLAL) y escalar.

VMLS Vector Multiply Substract. Es como **VFMS** pero redeondea antes de la resta.Tiene versiones Long (VMLSL) y escalar.

NEON Multiply instructions

VQDMULH Vector Saturating Doubling Multiply Returning High Half. Multiplica sus operandos, dupilca los resultados y retorna la parte alta de los mismos. Si ocurrió saturación setea el bit QC. Acepta operar con opción de redondeo (**VQRDMULH**). Trabaja con vectors y con escalares.

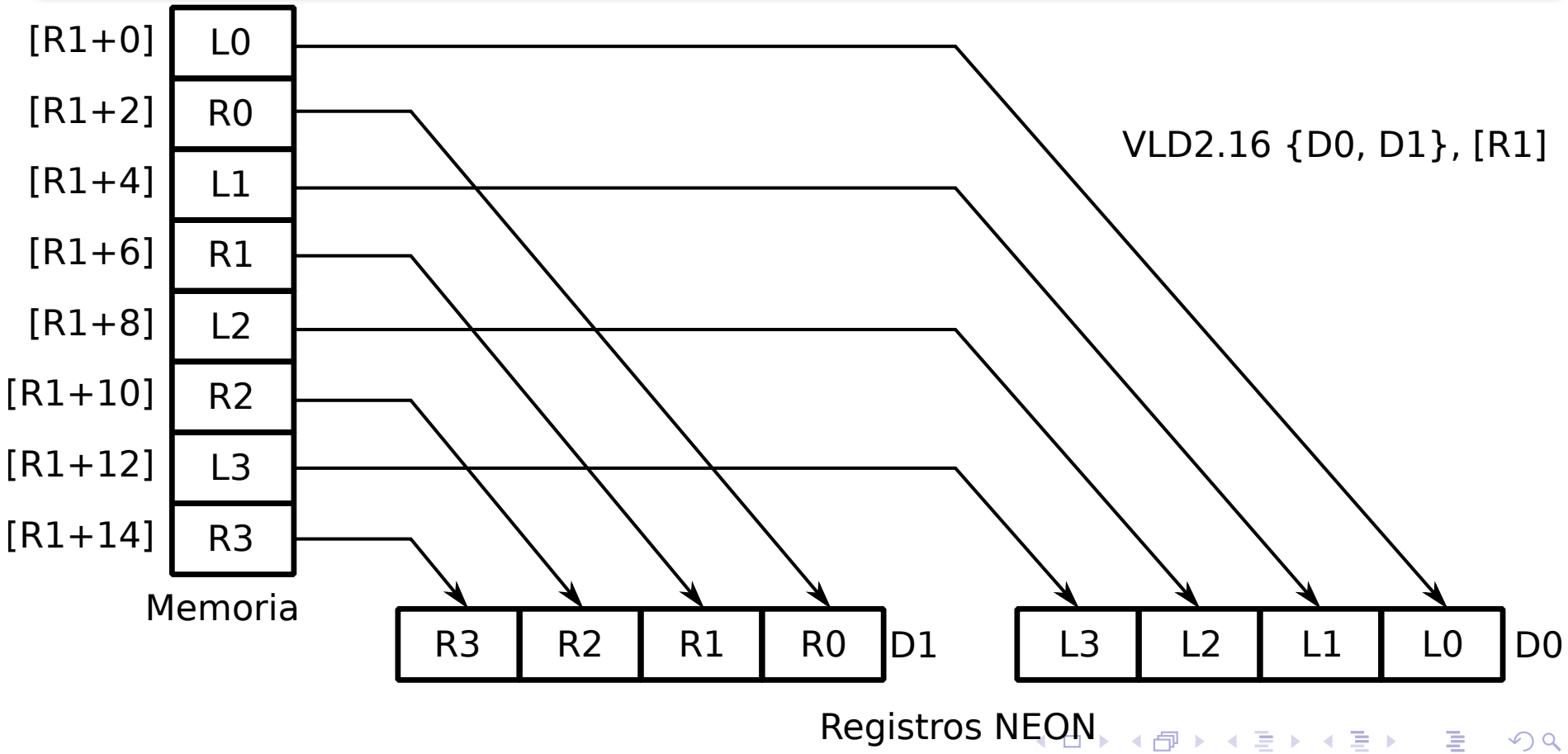
VQDMULL Vector Saturating Doubling Multiply Long. Multiplica los operandos y duplica el resultado. Si el resultado da overflow, satura, y setea el bit QC.

VQDMLAL Vector Saturating Doubling Multiply and Accumulate Long. Idem **VQDMULH**, con el agregado de acumular el resultado en el destino en lugar de almacenarlo.

VQDMLSL Vector Saturating Doubling Multiply and Substract Long. Idem **VQDMULH**, con el agregado de restar el resultado en el destino en lugar de almacenarlo.

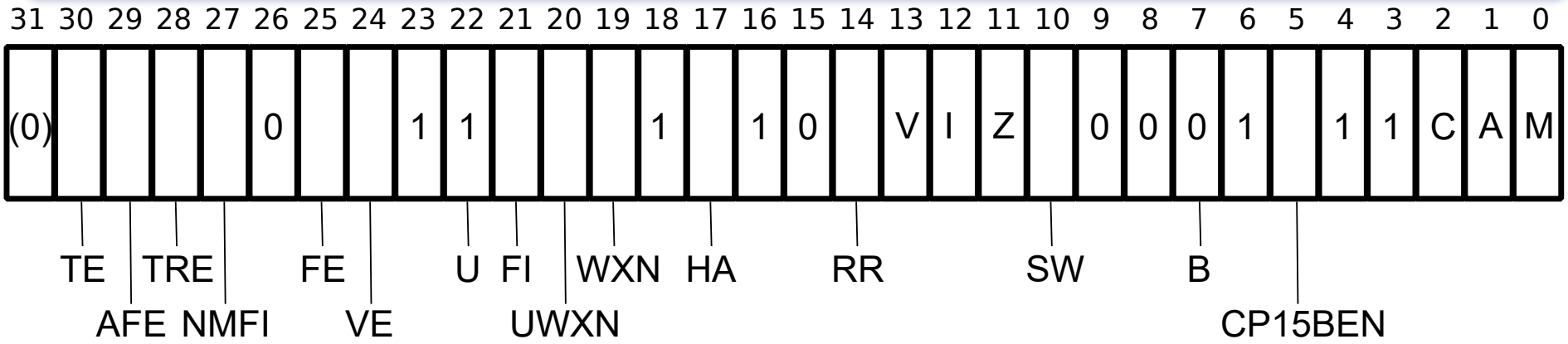
NEON Load and Store instructions

Muchas de las instrucciones Load Store proporcionan capacidad de entrelazado/desentrelazado de los datos cuando se almacenan/leen en/desde memoria, estructuras de datos.



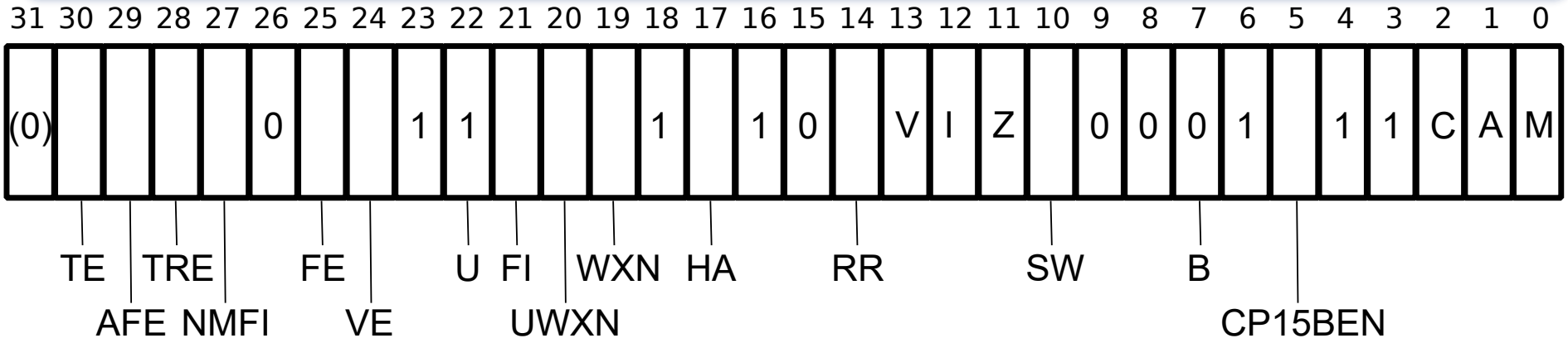
NEON Load and Store instructions

También una buena cantidad de las instrucciones Load/Store permiten establecer restricciones de alineamiento en memoria. Para habilitar este chequeo debe setearse el bit SCTLR.A (CP15 System Control Register, bit 1).



NEON Load and Store instructions

También una buena cantidad de las instrucciones Load/Store permiten establecer restricciones de alineamiento en memoria. Para habilitar este chequeo debe setearse el bit SCTLR.A (CP15 System Control Register, bit 1).




```
MRC p15, 0, R2, c1, c0, 0 # Lee registro SCTRL de CP15 en R2
ORR R2,R2,#0x00000002     # Setea el bit 1 (corresponde a A)
MCR p15, 0, R2, c1, c0, 0 # Activa alineacion escribiendo SCTRL
```

NEON Load and Store instructions

A partir de este código cualquier acceso no alineado a memoria genera una *alignment fault*.
Si A=0, no hay restricciones de alineación de memoria, excepto para accesos de ordenamiento estricto o para dispositivos mapeados en memoria (el resultado de un acceso no alineado puede resultar impredecible).

NEON Load and Store instructions

VLDn Vector Load. *n* puede ser 1, 2, 3, o 4. Carga una estructura de n elementos a uno o mas registros NEON. De acuerdo con la codificación tiene diferentes variantes.

- Cargar un a lane en particular de cada registro NEON, y el resto de los lanes no cambian. Utiliza Dd[x] para especificar el lane.
- Cargar a todos los lanes de cada registro el mismo dato de la estructura de memoria.

VSTn Vector Store. Almacena los lanes de uno o mas registros NEON en una estructura de memoria. Tiene las mismas dos variantes de **VLDn**.

VLDR Carga un Vector de memoria en un registro NEON destino.

VSTR Almacena un Registro Neon en n vector de memoria.

NEON Load and Store instructions

VLDM Carga desde un vector de memoria hacia una lista de uno múltiples registros NEON. Tiene opciones de auto incremento del registro puntero a memoria. Incluye un sufijo de modo:

- IA Increment After (default).
- DB Decrement Before.
- EA Empty Ascending. Se usa en stacks. Equivale a DB en Loads e IA en Stores.
- FD Full Descending. Se usa en stacks. Equivale a IA en Loads e DB en Stores.

VSTM Es la operación inversa de VLDM. Mismos sufijos y opciones.

VPOP Desapila un vector hacia una lista de registros NEON. Equivale a `VLDM sp!, {Registros}`.

VPUSH Guarda en la pila un conjunto de registros NEON. Equivale a `VSTMDB sp! {Registros}`.

VMOV Mueve datos entre Registros ARM y NEON de 64 bits. Mueve escalares si especificamos el lane en el registro NEON (Dm[x]).

NEON Load and Store instructions

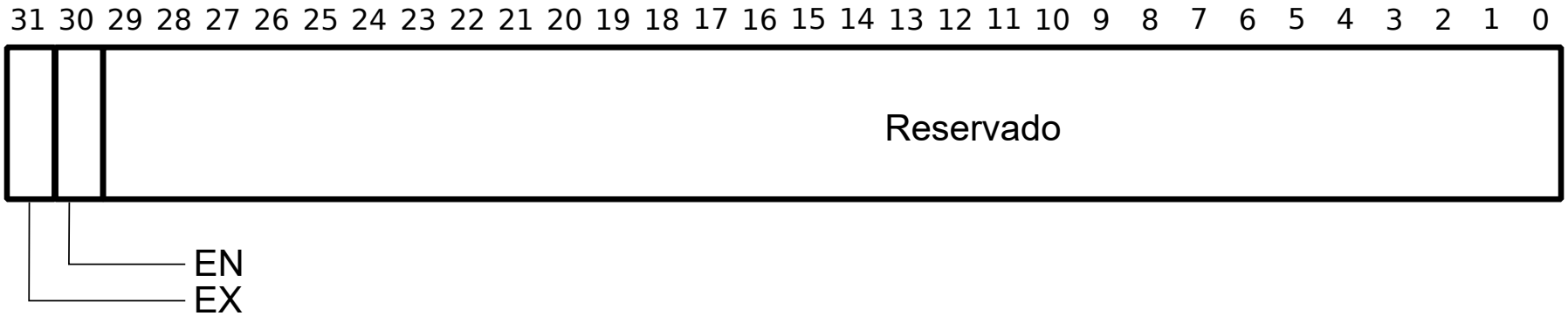
VMRS Transfiere el registro de sistema NEON o VFP a un registro core (Rn). El regstro de sistema se especifica en la instrución y puede ser: **FPSCR**, **FPSID** , o **FPEXC**. Detiene el pipeline NEON hasta que se complete su ejecución.

VMSR Operación de transferencia inversa a la anterior.

NEON Load and Store instructions

VMRS Transfiere el registro de sistema NEON o VFP a un registro core (Rn). El regstro de sistema se especifica en la instrución y puede ser: **FPSCR**, **FPSID** , o **FPEXC**. Detiene el pipeline NEON hasta que se complete su ejecución.

VMSR Operación de transferencia inversa a la anterior.



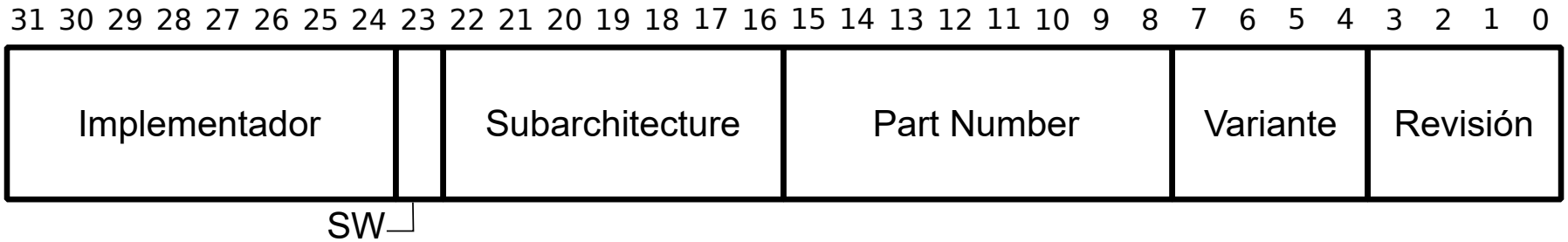
EX. Exception Bit: ’0’ Indica que el estado NEON y VFP a salvar se compone de los Registros NEON, FPSCR, y FPSID. ’1’ Indica que hay información adicional implementación específica.

EN. Enable bit. ’0’ (Default) indica que NEON y VFP no están habilitados. ’1’ los habilita.

NEON Load and Store instructions

VMRS Transfiere el registro de sistema NEON o VFP a un registro core (Rn). El regstro de sistema se especifica en la instrución y puede ser: **FPSCR**, **FPSID** , o **FPEXC**. Detiene el pipeline NEON hasta que se complete su ejecución.

VMSR Operación de transferencia inversa a la anterior.

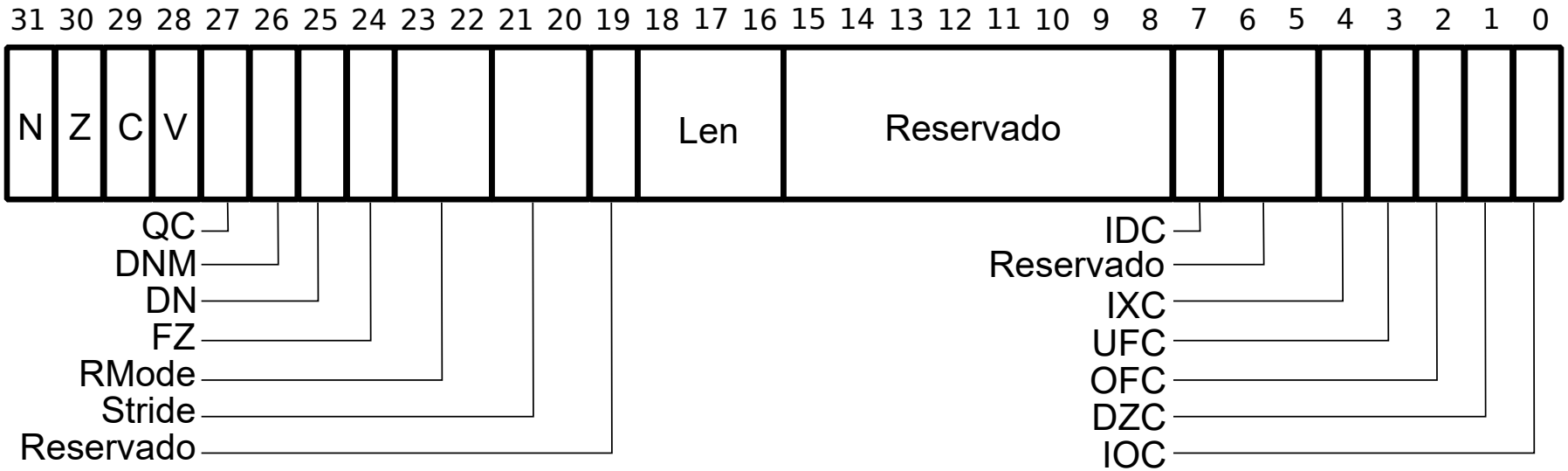


SW. '0' Indica que el coprocesador está implementado en hardware. '1' indica que está emulado por software.

NEON Load and Store instructions

VMRS Transfiere el registro de sistema NEON o VFP a un registro core (Rn). El regstro de sistema se especifica en la instrución y puede ser: **FPSCR**, **FPSID** , o **FPEXC**. Detiene el pipeline NEON hasta que se complete su ejecución.

VMSR Operación de transferencia inversa a la anterior.



N, **Z**, **C** , **V**, se setean si una comparación resulta *"Menor que"*, *"Igual a"*, *"Mayor a"* o genera un resultado no ordenado, o si produce un resultado no ordenado respectivamente. No actúan hasta que se copien al **CPSR**.

NEON Load and Store instructions

QC	Bit de Saturación
DNM	Do Not Modify (Reservado)
DN	Modo NaN. ’0’ deshabilitado (default). ’1’ Habilitado.
FZ	Modo Flush To Zero . ’0’ deshabilitado. ’1’ Habilitado.
RMode	Rounding Mode. ’00’ Nearest (RN), ’01’ Hacia $+\infty$ (RP), ’10’ Hacia $-\infty$ (RM), ’11’ Hacia Cero (RZ)
Stride	Ver tabla slide ....
Len	Ver tabla slide....
IDC	Flag de Entrada Denormalizada
IXC	Flag de Pérdida de precisión
UFC	Flag de Underflow
OFC	Flag de Overflow
DZC	Flag de División por cero
IOC	Flag de Operación Inválida

NEON Load and Store instructions

LEN	Vector Length	Stride	Vector Stride	SP Vector Instructions	DP Vector Instructions
b000	1	b00	-	All instructions are scalar	All instructions are scalar
b000	1	b11	-	Unpredictable	Unpredictable
b001	2	b00	1	Work normally	Work normally
b001	2	b11	2	Work normally	Work normally
b010	3	b00	1	Work normally	Work normally
b010	3	b11	2	Work normally	Unpredictable
b011	4	b00	1	Work normally	Work normally
b011	4	b11	2	Work normally	Unpredictable
b100	5	b00	1	Work normally	Unpredictable
b100	5	b11	2	Unpredictable	Unpredictable
b101	6	b00	1	Work normally	Unpredictable
b101	6	b11	2	Unpredictable	Unpredictable
b110	7	b00	1	Work normally	Unpredictable

Referencias

Una muy buena lectura introductoria (introductoria, o sea para iniciar):
Introducing NEON Development Article

Referencias Obligadas:
NEON Programmer's Guide.
Cortex-A8 Technical Reference Manual. Chapter 13. NEON and VFP Programmers Model