(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 527 937**

(51) Int. CI.:

G06F 12/00 (2006.01)
G06F 11/00 (2006.01)
G06F 9/30 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **12.07.2004 E 10191073 (5)**

(97) Fecha y número de publicación de la concesión europea: **26.11.2014 EP 2284709**

(54) Título: **Procesador programable y método con operaciones amplias**

(30) Prioridad:

10.07.2003 US 616303

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
02.02.2015

(73) Titular/es:

MICROUNITY SYSTEMS ENGINEERING, INC. (100.0%)
4 Main Street, Suite 100
Los Altos, CA 94022 , US

(72) Inventor/es:

HANSEN, CRAIG;
MOUSSOURIS, JOHN y
MASSALIN, ALEXIA

(74) Agente/Representante:

PONTI SALES, Adelaida

DESCRIPCIÓN

Procesador programable y método con operaciones amplias

Campo de la invención

[0001] La presente invención se refiere a arquitecturas de procesador de propósito general, y en particular se refiere a arquitecturas de operandos amplios.

ANTECEDENTES DE LA INVENCIÓN

[0002] Los productos de comunicaciones requieren un mayor rendimiento computacional para procesar señales digitales en software en tiempo real. Los incrementos del rendimiento han pasado por mejoras en la tecnología de procesos y por mejoras en el diseño de los microprocesadores. Un mayor paralelismo, mayores frecuencias de reloj, mayores densidades, acoplados con herramientas de diseño mejoradas y compiladores los han hecho más prácticos. Sin embargo, muchas de estas mejoras suponen una sobrecarga adicional en memoria y latencia, debido a la ausencia del ancho de banda necesario que está estrechamente acoplado con las unidades computacionales.

[0003] El nivel de rendimiento de un procesador, y en particular de un procesador de propósito general, se puede estimar a partir de varios de una serie de factores independientes: frecuencia de reloj, puertas por reloj, número de operandos, anchura del camino de operandos y de datos, y división del camino de operandos y de datos. La frecuencia de reloj está influida en gran parte por la elección de la tecnología lógica y de circuitos, pero está influida asimismo por el número de puertas por reloj. Las puertas por reloj consisten en cuántas puertas en una segmentación pueden cambiar de estado en un solo ciclo de reloj. Esto se puede reducir al introducir cerrojos en el camino de datos: cuando se reduce el número de puertas entre cerrojos, es posible un reloj más rápido. Sin embargo, los cerrojos adicionales producen una mayor longitud de segmentación, y por lo tanto tienen el coste de una mayor latencia de instrucciones. El número de operandos es sencillo; por ejemplo, sumando con técnicas de ahorro de acarreo, se pueden sumar juntos tres valores con poco más retardo del necesario para sumar dos valores. La anchura del camino de operandos y de datos define cuántos datos se pueden procesar a la vez; caminos de datos más amplios pueden realizar funciones más complejas, pero en general esto supone un mayor coste de implementación. La división de los caminos de operandos y de datos se refiere a la utilización eficiente del camino de datos cuando se aumenta la anchura, con el objetivo de mantener sustancialmente el uso máximo.

[0004] La última división del camino de factores, operandos y datos se trata extensivamente en las patentes U.S.A. asignadas en común con la presente, de números 5 742 840, 5 794 060, 5 794 061, 5 809 321 y 5 822 603, que describen sistemas y métodos para mejorar la utilización de un procesador de propósito general añadiendo clases de instrucciones. Estas clases de instrucciones utilizan los contenidos de registros de propósito general como fuentes de caminos de datos, dividen los operandos en símbolos de un tamaño especificado, realizan operaciones en paralelo, concatenan los resultados y colocan los resultados concatenados en un registro de propósito general. Éstas patentes, la totalidad de las cuales están asignadas al mismo cesionario de la presente invención, muestran un microprocesador de propósito general que se ha optimizado para procesar y transmitir flujos de datos multimedia mediante un paralelismo significativo.

[0005] Si bien las patentes anteriores han ofrecido mejoras significativas en la utilización y el rendimiento de un microprocesador de propósito general, en particular para manejar comunicaciones de banda ancha tales como flujos de datos multimedia, son posibles otras mejoras.

[0006] Muchos procesadores de propósito general tienen registros generales para almacenar operandos para instrucciones, haciéndose coincidir la anchura del registro con el tamaño del camino de datos. Los diseños de los procesadores limitan, en general, el número de registros accesibles por instrucción, debido a que el hardware para acceder a estos registros es relativamente costoso en términos de energía y superficie. Mientras que el número de registros accesibles varía entre los diseños de procesadores, a menudo está limitado a dos, tres o cuatro registros por instrucción cuando dichas instrucciones están diseñadas para funcionar en un solo ciclo de reloj del procesador o en un solo flujo de segmentación. Algunos procesadores, tales como el Motorola 68000, tienen instrucciones para guardar y reestablecer un número ilimitado de registros, pero requieren múltiples ciclos para realizar dicha instrucción.

[0007] El Motorola 68000 intenta asimismo superar un camino de datos estrecho combinado con un archivo de registros estrecho tomando múltiples ciclos o flujos de segmentación para realizar una instrucción, y emulando de ese modo un camino de datos más amplio. Sin embargo, dichas técnicas de precisión múltiple ofrecen solamente una mejora marginal, en vista de los ciclos de reloj adicionales requeridos. Por lo tanto, la anchura y el número accesible de los registros de propósito general limita fundamentalmente la cantidad de procesamiento que puede ser realizado mediante una única instrucción en una máquina basada en registros.

[0008] Los procesadores existentes pueden proporcionar instrucciones que aceptan operandos, para las que se leen uno o varios operandos desde un sistema de memoria de un procesador de propósito general. Sin embargo,

dado que estos operandos de memoria están especificados generalmente mediante operandos de registro, y el camino de datos del sistema de memoria no es más amplio que el camino de datos del procesador, no se mejora la anchura y el número accesible de operandos de propósito general por ciclo de instrucción o flujo de segmentación.

[0009] El número de operandos de registro de propósito general accesibles por instrucción está limitado generalmente por la complejidad lógica y el tamaño de las instrucciones. Por ejemplo, se podrían implementar ciertas funciones deseables pero complejas, especificando un número mayor de registros de propósito general, pero habría que haber añadido una cantidad sustancial de lógica adicional a un diseño convencional, para permitir la lectura y omisión simultáneas de los valores de registro. Si bien en algunos diseños de la técnica anterior se han utilizado registros dedicados para aumentar el número o el tamaño de resultados u operandos fuente, instrucciones explícitas cargan o almacenan valores en estos registros dedicados, y se requieren instrucciones adicionales para guardar y reestablecer estos registros tras un cambio de contexto del procesador.

[0010] El tamaño de un resultado de la unidad de ejecución puede estar limitado al de un registro general, de manera que no se requiere un almacenamiento dedicado u otro especial para el resultado. Especificar un gran número de registros de propósito general como un resultado, requeriría análogamente añadir una cantidad sustancial de lógica adicional a un diseño convencional, para permitir la escritura y omisión simultáneas de los valores de registro.

[0011] Cuando el tamaño de un resultado de la unidad de ejecución está limitado, éste puede limitar la cantidad de computación que puede ser gestionada razonablemente por una sola instrucción. Como consecuencia, es necesario implementar los algoritmos como una serie de etapas de una sola instrucción, en las que todos los resultados intermedios se pueden representar dentro de los límites. Eliminando este límite, se pueden desarrollar conjuntos de instrucciones en los que un componente mayor de un algoritmo se implementa como una sola instrucción, y la representación de los resultados intermedios deja de estar limitada en tamaño. Además, no es necesario que algunos de estos resultados intermedios se retengan tras la finalización del componente mayor de un algoritmo, de manera que un procesador liberado de estas limitaciones puede mejorar el rendimiento y reducir la potencia de funcionamiento al no almacenar y recuperar estos resultados desde el archivo de registros generales. Cuando los resultados intermedios no se retienen en el archivo de registros generales, los conjuntos de instrucciones del procesador y los algoritmos implementados tampoco están limitados por el tamaño del archivo de registros generales.

[0012] Por lo tanto, ha existido la necesidad de un sistema de procesador que pueda manejar eficientemente operandos y resultados de mayor anchura que el sistema de memoria o cualquier registro de propósito general accesible. Existe asimismo la necesidad de un sistema de procesador que pueda manejar eficientemente operandos y resultados de mayor tamaño global que todo el archivo de registros generales.

[0013] El documento WO 00/23875 describe un procesador de propósito general que puede ejecutar una instrucción de matriz de multiplicación amplia que toma de un registro general una dirección para recuperar de la memoria un operando grande (amplio), recuperar de un registro general un segundo operando, realizar un grupo de operaciones sobre bits divididos en los operandos, y concatenar juntos los resultados, colocando el resultado en un registro general.

[0014] Un documento de Pitsianis NP y otros, titulado "High-performance FFT implementation on the BOPS ManArray parallel DSP", publicado en los documentos de Advanced Signal Processing Algorithms, Architectures, and Implementations 1X, 19 al 21 de julio de 1999, conferencia SPIE, volumen 3807, páginas 164 a 171, describe una implementación de alto rendimiento de un algoritmo de FFT que, para poner de manifiesto el paralelismo inherente a un algoritmo FFT, utiliza una factorización de la matriz DFT en productos de Kronecker, permutación y matrices diagonales.

[0015] La patente U.S.A. asignada en común y relacionada, número 6 295 599, describe en detalle un método y un sistema para mejorar el rendimiento de procesadores de propósito general expandiendo por lo menos un operando fuente de hasta una anchura mayor que cualquiera de la anchura del registro de propósito general o la anchura del camino de datos. Se pueden conseguir mejoras adicionales en el rendimiento mediante

RESUMEN DE LA INVENCIÓN

[0016] Un aspecto de la invención da a conocer un procesador tal como el definido en la reivindicación 1. Un segundo aspecto de la invención da a conocer un método tal como el definido en la reivindicación 25. Las realizaciones de la invención se exponen en las reivindicaciones dependientes.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

[0017]

La figura 1 es un diagrama a nivel de sistema que muestra los bloques funcionales de un procesador de propósito

general.

La figura 2 es una representación matricial de una multiplicación de matriz amplia.

La figura 3 es otra representación de una multiplicación de matriz amplia.

La figura 4 es un diagrama a nivel de sistema que muestra los bloques funcionales de un sistema que incorpora un procesador de multihilos simultáneos y acceso desacoplado de la ejecución.

La figura 5 muestra un operando amplio.

La figura 6 muestra un enfoque para la descodificación del especificador.

La figura 7 muestra un bloque operacional a partir de una unidad de función amplia.

La figura 8 muestra, en forma de diagrama de flujo, la función de control de microcaché amplia.

La figura 9 nuestra estructuras de datos de microcaché amplia.

Las figuras 10 y 11 muestran un control de microcaché amplia.

Las figuras 12A a 12D muestran una instrucción de Galois de matriz de multiplicación amplia.

Las figuras 13A a 13G muestran una instrucción de extracción de conjunto in situ.

Las figuras 14A a 14J muestran una instrucción de extracción de conjunto.

Las figuras 15A a 15C muestran una instrucción booleana de grupo.

Las figuras 16A a 16C muestran instrucciones de suma de grupo.

Las figuras 17A a 17C muestran instrucciones de establecimiento de grupo e instrucciones de resta del grupo.

Las figuras 18A a 18C muestran instrucciones de convolución de conjunto, división de conjunto, multiplicación de conjunto y suma de multiplicaciones de conjunto.

La figura 19 muestra funciones a modo de ejemplo que se definen para su utilización dentro de las definiciones detalladas de instrucciones, en otras secciones.

Las figuras 20A a 20C muestran instrucciones suma con coma flotante de conjunto, división con coma flotante de conjunto y multiplicación con coma flotante de conjunto.

Las figuras 21A a 21C muestran instrucciones de resta con coma flotante de conjunto.

Las figuras 22A a 22D muestran instrucciones de compresión, expansión, rotación y desplazamiento de barras cruzadas.

Las figuras 23A a 23D muestran instrucciones de extracción.

Las figuras 24A a 24B muestran instrucciones de Galois de solución amplia.

Las figuras 25A a 25B muestran instrucciones de rebanada de transformada amplia, de acuerdo con una realización a modo de ejemplo de la presente invención.

Las figuras 26A a 26K muestran instrucciones de extracción convolución amplia.

La figura 27 muestra transferencias entre memorias de operandos amplios.

DESCRIPCIÓN DETALLADA DE LA INVENCIÓN

Distribución del procesador

[0018] Haciendo referencia en primer lugar a la figura 1, se muestra en la misma un procesador de propósito general en forma de diagrama de bloques. En la figura 1, se muestran cuatro copias de una unidad de acceso, cada una con una cola de recuperación de instrucciones de acceso cola-A 101 a 104. Cada cola de recuperación de instrucciones de acceso, cola-A 101 a 104 está acoplada a un archivo de registros de acceso AR 105 a 108, estando

acoplados cada uno de estos a dos unidades funcionales de acceso A 109 a 116. En una realización habitual, cada hilo del procesador puede tener del orden de sesenta y cuatro registros de propósito general (por ejemplo, los AR 105 a 108 y ER 125 a 128). Las unidades de acceso funcionan independientemente para cuatro hilos simultáneos de ejecución, y cada una calcula el flujo de control del programa realizando instrucciones aritméticas y de ramificación, y accede a la memoria realizando instrucciones de carga y almacenamiento. Estas unidades de acceso proporcionan asimismo especificadores de operandos amplios para instrucciones de operandos amplios. Estas ocho unidades funcionales de acceso A 109 a 116 producen resultados para archivos de registros de acceso AR 105 a 108, y direcciones de memoria para un sistema de memoria compartido 117 a 120.

[0019] En una realización, la jerarquía de memoria incluye memoria de datos e instrucciones en chip, cachés de instrucciones y datos, una instalación de memoria virtual e interfaces a dispositivos externos. En la figura 1, el sistema de memoria se compone de una memoria combinada caché y de nicho 117, una interfaz de bus externo 118, y, externamente al dispositivo, una caché secundaria 119 y un sistema de memoria principal con dispositivos E/S 120. Los contenidos de memoria recuperados del sistema de memoria 117 a 120 se combinan con instrucciones de ejecución no realizadas por la unidad de acceso, y se introducen en las cuatro colas de instrucción de ejecución colas-E 121 a 124. Para instrucciones amplias, los contenidos de memoria recuperados del sistema de memoria 117 a 120 se proporcionan asimismo a microcachés de operandos amplios 132 a 136 mediante el bus 137. Los datos de memoria e instrucciones de la cola-E 121 a 124 se presentan a archivos de registros de ejecución 125 a 128, que recuperan operandos fuente de archivo de registros de ejecución. Las instrucciones se acoplan a la unidad de arbitraje de las unidades de ejecución Arbitraje 131, que selecciona qué instrucciones de los cuatro hilos se tienen que encaminar a las unidades funcionales de ejecución disponibles E 141 y 149, X 142 y 148, G 143 a 144 y 146 a 147 y T 145. Cada una de las unidades funcionales de ejecución E 141 y 149, las unidades funcionales de ejecución X 142 y 148 y la unidades funcionales de ejecución T 145 contiene una microcaché de operandos amplios 132 a 136, cada una de las cuales está acoplada al sistema de memoria 117 mediante el bus 137.

[0020] Las unidades funcionales de ejecución G 143 a 144 y 146 a 147 son unidades aritméticas y lógicas de grupos que realizan instrucciones aritméticas y lógicas simples, incluyendo operaciones de grupos en las que los operandos de fuentes y de resultado representan un grupo de valores de un tamaño de símbolo especificado, que se dividen y se manejan por separado, concatenándose juntos los resultados. En una realización preferida actualmente, el camino de datos tiene 128 bits de amplitud, aunque la presente invención no desea limitarse a ningún tamaño específico del camino de datos.

[0021] Las unidades funcionales de ejecución X 142 y 148 son unidades de conmutador de barras cruzadas que realizan instrucciones de conmutación de barras cruzadas. Las unidades de conmutación de barras cruzadas 142 y 148 realizan operaciones de manipulación de datos en el flujo de datos proporcionado sobre los buses de operandos fuente del camino de datos 151 a 158, incluyendo transacciones, aleatorizaciones, desplazamientos, expansiones, compresiones, mezclas, permutaciones e inversiones, más las operaciones amplias descritas en adelante. En un elemento clave de un primer aspecto de la invención, por lo menos una de dichas operaciones será expandida a una anchura mayor que la anchura del camino de registros generales y de datos.

[0022] Las unidades funcionales de ejecución E 141 y 149 son unidades de conjunto que realizan instrucciones de conjunto utilizando un multiplicador matricial grande, que incluye multiplicación de grupos o de vectores y multiplicación de matrices, de operandos divididos desde los buses 151 a 158 de operandos fuente del camino de datos, y tratados como valores enteros, de coma flotante, polinómicos o de campos de Galois. Las instrucciones de multiplicación de matrices y otras operaciones utilizan un operando amplio cargado en la microcaché de operandos amplios 132 y 136.

[0023] La unidad funcional de ejecución T 145 es una unidad de traducción que realiza operaciones de búsqueda en tablas, sobre un grupo de operandos divididos desde un operando de registro, y concatena el resultado. La instrucción de traducción amplia utiliza un operando amplio cargado en la microcaché de operandos amplios 134.

[0024] Las unidades funcionales de ejecución E 141, 149, las unidades funcionales de ejecución X 142, 148 y la unidad funcional de ejecución T contienen, cada una, un almacenamiento dedicado para permitir el almacenamiento de operandos fuente que incluyen operandos amplios, tal como se describe más adelante. El almacenamiento dedicado 132 a 136, que se puede considerar como una microcaché amplia, tiene habitualmente una anchura que es un múltiplo de la anchura de los operandos del camino de datos relacionados con los buses 151 a 158 de operandos fuente del camino de datos. De este modo, si la anchura del camino de datos 151 a 158 es de 128 bits, el almacenamiento dedicado 132 a 136 puede tener una anchura de 256, 512, 1024 ó 2048 bits. Los operandos que utilizan toda la anchura del almacenamiento dedicado se denominan en el presente documento operandos amplios, aunque no es necesario en todos los casos que un operando amplio utilice íntegramente la anchura del almacenamiento dedicado; es suficiente que el operando amplio utilice una parte mayor que la anchura del camino de datos de memoria de la salida del sistema de memoria a 117 a 120 y del camino de datos de la unidad funcional de la entrada de las unidades funcionales de ejecución 141 a 149, aunque no necesariamente mayor que la anchura de los dos combinados. Debido a que la anchura del almacenamiento dedicado 132 a 136 es mayor que la anchura del bus 137 de operandos de memoria, se cargan secuencialmente partes de los operandos amplios en el almacenamiento dedicado 132 a 136. Sin embargo, una vez cargados, a continuación los operandos amplios

pueden ser utilizados sustancialmente al mismo tiempo. Se puede observar que las unidades funcionales 141 a 149 y los registros de ejecución asociados 125 a 128 forman una unidad funcional de datos, cuyos elementos exactos pueden variar con la implementación.

[0025] Los operandos fuente del archivo de registros de ejecución 125 a 128 se acoplan a las unidades de ejecución 141 a 145 utilizando buses 151 a 154 de operandos fuente, y a las unidades de ejecución 145 a 149 utilizando buses 155 a 158 de operandos fuente. Los operandos resultado de unidades de función, procedentes de las unidades de ejecución 141 a 145, se acoplan al archivo de registros de ejecución ER 125 a 128 utilizando el bus de resultados 161, y los operandos de resultado de unidades de función, procedentes de las unidades de ejecución 145 a 149, se acoplan al archivo de registros de ejecución utilizando el bus de resultados 162.

Matriz de multiplicación amplia

[0026] Los operandos amplios proporcionan la capacidad de ejecutar instrucciones complejas, tales como la instrucción de matriz de multiplicación amplia mostrada en la figura 2, que se puede apreciar asimismo en una forma alternativa en la figura 3. Tal como se puede apreciar por las figuras 2 y 3, un operando amplio permite, por ejemplo, la multiplicación matricial de varias formas y tamaños que exceden la anchura del camino de datos. El ejemplo de la figura 2 involucra una matriz especificada por el registro rc que tiene 128*64/tamaño bits (512 bits en este ejemplo), multiplicada por un vector contenido en el registro rb que tiene 128 bits, para proporcionar un resultado, colocado en el registro rd, de 128 bits.

[0027] La notación utilizada en la figura 2 y siguientes figuras similares muestra una multiplicación como un área sombreada en la intersección de dos operandos proyectados en las dimensiones horizontal y vertical. Un nodo de sumación se muestra como un segmento de línea que conecta puntos en negrita en la posición de los productos de multiplicador que se suman. Los productos que se restan en el nodo de sumación se indican con un símbolo menos en el interior del área sombreada.

[0028] Cuando la instrucción aplica a valores de coma flotante, las multiplicaciones y las sumaciones mostradas son multiplicaciones y sumaciones de coma flotante. Una realización a modo de ejemplo puede realizar estas operaciones sin redondear los resultados intermedios, calculando por lo tanto el resultado final como si se hubiera calculado con precisión infinita, y redondeando a continuación solamente una vez.

[0029] Se puede apreciar que una realización a modo de ejemplo de los multiplicadores puede calcular el producto en modo de ahorro de acarreo, y puede codificar el multiplicador rb utilizando codificación de Booth para minimizar el área y el retardo del circuito. Se puede apreciar que una realización a modo de ejemplo de dichos nodos de sumación puede llevar a cabo la sumación de los productos en cualquier orden, prestando especial atención a minimizar el retardo de computación, tal como realizando las sumas en un árbol binario o de una base mayor, y puede utilizar sumadores con ahorro de acarreo para realizar la suma a efectos de minimizar el retardo de la sumación. Se puede apreciar asimismo que una realización a modo de ejemplo puede llevar a cabo la sumación utilizando la precisión intermedia suficiente para que no se produzcan desbordamientos de punto fijo o de coma flotante en los resultados intermedios.

[0030] Se puede utilizar una comparación de las figuras 2 y 3 para aclarar la relación entre la notación utilizada en la figura 2 y la notación esquemática más convencional de la figura 3, dado que se muestra la misma operación en estas dos figuras.

Operando amplio

[0031] Los operandos que son sustancialmente mayores que la anchura del camino de datos del procesador se proporcionan utilizando el registro de propósito general para especificar un especificador de memoria desde el que se puede leer, en el almacenamiento dedicado, datos de más de una, y en algunas realizaciones de varias, anchuras del camino de datos. El especificador de memoria incluye habitualmente la dirección de memoria junto con el tamaño y la forma de la matriz de datos sobre la que se está operando. El especificador de memoria o el especificador del operando amplio se pueden apreciar mejor en la figura 5, en la que se ve que un especificador 500 es una dirección, más un campo representativo del tamaño/2 y un campo adicional representativo de la anchura/2, donde el tamaño es el producto de la profundidad por la anchura de los datos. La dirección está alineada a un tamaño específico, por ejemplo sesenta y cuatro octetos, de manera que una serie de los bits de orden bajo (por ejemplo, seis bits) son cero. De este modo, se puede ver que el especificador 500 comprende un primer campo 505 para la dirección, más dos índices de campo 510 dentro de los seis bits de orden bajo, para indicar tamaño y anchura.

Descodificación del especificador

[0032] La descodificación del especificador 500 se puede apreciar mejor en la figura 6, para un especificador dado 600 que se compone de un campo de dirección 605 junto con un campo 610 que comprende varios bits de orden bajo. Mediante una serie de operaciones aritméticas mostradas en las etapas 615 y 620, se desarrolla la parte del

campo 610 representativa de anchura/2. En una serie similar de etapas mostrada en 625 y 630, se descodifica el valor de t, que se puede utilizar a continuación para descodificar tanto el tamaño como la dirección. La parte del campo 610 representativa de tamaño/2 se descodifica tal como se muestra en las etapas 635 y 640, mientras que la dirección se descodifica de manera similar a las etapas 645 y 650.

Unidad de función amplia

[0033] La unidad de función amplia se puede apreciar mejor en la figura 7, en la que se proporciona un número de registro 700 a un verificador de operandos 705. El especificador de operandos amplios 710 comunica con el verificador de operandos 705 y asimismo con la dirección de memoria 715 que tiene una anchura de memoria definida. La dirección de memoria incluye una serie de n operandos de registro 720A, que se acumulan en una parte de almacenamiento dedicado 714 de una unidad funcional de datos 725. En la realización a modo de ejemplo mostrada en la figura 7, se puede ver que el almacenamiento dedicado 714 tiene una anchura igual a ocho anchuras del camino de datos, de tal modo que ocho porciones 730A-H del operando amplio se cargan secuencialmente en el almacenamiento dedicado para formar el operando amplio. Aunque en la figura 7 se muestran ocho porciones, la presente invención no se limita a ocho ni a ningún otro múltiplo específico de anchuras del camino de datos. Una vez que las porciones 730A-H del operando amplio se cargan secuencialmente, pueden ser utilizadas como un único operando amplio 735 por el elemento funcional 740, que puede ser cualquier elemento o elementos de la figura 1 conectados al mismo. A continuación, el resultado del operando amplio se proporciona a un registro de resultados 745, que en una realización actualmente preferida tiene la misma anchura que la anchura de la memoria.

[0034] Una vez que el operando amplio se ha cargado satisfactoriamente en el almacenamiento dedicado 714, se puede apreciar un segundo aspecto de la presente invención. La ejecución posterior de esta instrucción o de otras instrucciones similares que especifican la misma dirección de memoria, puede leer el almacenamiento dedicado para obtener el valor del operando bajo condiciones específicas que determinan si el operando de memoria ha sido modificado por instrucciones intermedias. Asumiendo que se satisfacen estas condiciones, el operando de memoria recuperado del almacenamiento dedicado se combina con uno o varios operandos de registro en la unidad funcional, produciendo un resultado. En algunas realizaciones, el tamaño del resultado está limitado al de un registro general, de manera que no se requiere un almacenamiento dedicado similar para el resultado. Sin embargo, en algunas realizaciones diferentes, el resultado puede ser un operando amplio, para mejorar adicionalmente el rendimiento.

[0035] Para permitir que el valor del operando amplio sea direccionado mediante instrucciones posteriores especificando la misma dirección de memoria, se tienen que comprobar y confirmar varias condiciones.

[0036] Estas condiciones incluyen:

Cada instrucción de almacenamiento de memoria comprueba la dirección de memoria frente a las direcciones de memoria registradas para el almacenamiento dedicado. Cualquier coincidencia provoca que el almacenamiento se marque como no valido, dado que una instrucción de almacenamiento de memoria dirigida a alguna dirección de memoria almacenada en el almacenamiento dedicado 714 significa que se han sobrescrito datos.

[0037] Se registra el número de registro utilizado para direccionar el almacenamiento. Si no se ha escrito ninguna instrucción intermedia en el registro, y se utiliza el mismo registro en la instrucción posterior, el almacenamiento es válido (salvo que se haya marcado como no valido mediante la regla #1).

[0038] Si el registro ha sido modificado o se utiliza un número de registro diferente, el valor del registro se lee y se compara con la dirección registrada para el almacenamiento dedicado. Esto utiliza más recursos que #1 debido a la necesidad de recuperar los contenidos del registro, y debido a que la anchura del registro es mayor que la del propio número del registro. Si la dirección coincide, el almacenamiento es válido. El nuevo número de registro se registra para el almacenamiento dedicado.

[0039] Si no se satisfacen las condiciones #2 ó #3, se utilizan los contenidos del registro para dirigirse a la memoria del procesador de propósito general y cargar el almacenamiento dedicado. Si el almacenamiento dedicado está ya totalmente cargado, es necesario descartar (sacrificar) una porción del almacenamiento dedicado para dejar espacio para el nuevo valor. A continuación se realiza la instrucción utilizando el almacenamiento dedicado recién actualizado. La dirección y el número de registro se registra para el almacenamiento dedicado.

[0040] Al comprobar las condiciones anteriores, se elimina la necesidad de guardar y restablecer el almacenamiento dedicado. Además, si se varía el contexto del procesador y el nuevo contexto no utiliza instrucciones amplias que hacen referencia al mismo almacenamiento dedicado, cuando se restablece el contexto original, se permite utilizar los contenidos del almacenamiento dedicado sin refrescar el valor desde la memoria, utilizando la regla de verificación #3. Debido a que los valores en el almacenamiento dedicado se leen desde la memoria y no se modifican directamente realizando operaciones amplias, los valores se pueden desechar en cualquier momento sin guardar los resultados en la memoria general. Esta propiedad simplifica la implementación de la regla #4 anterior.

[0041] Una realización alternativa de la presente invención puede sustituir la anterior regla #1 con la regla siguiente:

1a. Cada instrucción de almacenamiento de memoria comprueba la dirección de memoria frente a las direcciones de memoria registradas para el almacenamiento dedicado. Cualquier coincidencia provoca que se actualice el almacenamiento dedicado, así como la memoria general.

[0042] Mediante la utilización de la anterior regla 1.a, las instrucciones de almacenamiento en memoria pueden modificar el almacenamiento dedicado, actualizando solamente la pieza del almacenamiento dedicado que ha cambiado, dejando intacto el resto. Continuando con la actualización de la memoria general, sigue siendo cierto que los contenidos de la memoria dedicada se pueden desechar en cualquier momento sin guardar los resultados en la memoria general. Esta regla #4 no se complica mediante esta opción. La ventaja de esta realización alternativa es que no es necesario desechar (invalidar) el almacenamiento dedicado mediante operaciones de almacenamiento en memoria.

Estructuras de datos de microcaché amplia

[0043] Haciendo referencia a continuación a la figura 9, se puede apreciar mejor una disposición a modo de ejemplo de las estructuras de datos de la microcaché amplia o almacenamiento dedicado 114. Se puede ver que los contenidos de la microcaché amplia, wmc.c, forman una serie de anchuras del camino de datos 900A-n, aunque en el ejemplo mostrado el número es de ocho. La dirección física, wmc.pa, se muestra como 64 bits en el ejemplo mostrado, si bien la invención no se limita a una anchura específica. El tamaño de los contenidos, wmc.size, se proporciona asimismo en un campo que se muestra como de 10 bits en una realización a modo de ejemplo. Se incluye asimismo en la estructura de datos un indicador de "contenido válido", wmc.cv, de un bit, junto con un campo de dos bits para el último hilo utilizado, o wmc.th. Además, en una realización a modo de ejemplo se proporciona un campo de seis bits para el último registro utilizado, wmc.reg. Adicionalmente, se puede disponer un indicador de un bit para registro e hilo válido, o wmc.rtv.

Control de microcaché amplia – software

[0044] El proceso mediante el que la microcaché se escribe inicialmente con un operando amplio, y a continuación se verifica como válida para las operaciones posteriores, se puede apreciar mejor en la figura 8. El proceso comienza en 800, y avanza a la etapa 805 en la que se realiza una verificación de los contenidos del registro frente al valor almacenado wmc.rc. Si es correcta, se realiza una comprobación en la etapa 810 para verificar el hilo. Si es correcta, el proceso avanza a continuación a la etapa 815 para verificar si el registro y el hilo son válidos. Si la etapa 815 indica correcto, se realiza una comprobación en la etapa 820 para verificar si los contenidos son válidos. Si todas las etapas 805 a 820 indican correcto, la instrucción posterior puede utilizar el operando amplio existente tal como se muestra en la etapa 825, después de lo cual el proceso finaliza. Sin embargo, si alguna de las etapas 805 a 820 indica falso, el proceso se bifurca a la etapa 830, donde se establece el contenido, la dirección física y el tamaño. Dado que las etapas 805 a 820 conducen todas a alguna de las etapas 825 u 830, las etapas 805 a 820 se pueden realizar en cualquier orden o simultáneamente, sin alterar el proceso. El proceso avanza a continuación a la etapa 835, donde se comprueba el tamaño. Esta comprobación asegura básicamente que el tamaño de la unidad de traducción es mayor o igual que el tamaño del operando amplio, de tal modo que una dirección física puede sustituir directamente la utilización de una dirección virtual. La cuestión es que, en algunas realizaciones, los operandos amplios pueden ser mayores que la zona mínima que es capaz de mapear el sistema de memoria virtual. Como resultado, sería posible que un único rango de direcciones virtuales contiguas fuera mapeado a múltiples rangos de direcciones físicas disjuntas, complicando la tarea de comparar direcciones físicas. Determinando el tamaño del operando amplio y comparando este tamaño con el tamaño de la zona de mapeo de direcciones virtuales a la que se hace referencia, si el operando amplio es mayor que la zona de mapeo la instrucción se aborta con una interrupción de excepción. Esto garantiza el funcionamiento seguro del procesador. El software puede volver a mapear la zona utilizando un mapa de mayor tamaño para continuar la ejecución, si se desea. Por lo tanto, si se notifica el tamaño como inaceptable en la etapa 835, se genera una excepción en la etapa 840. Si el tamaño es aceptable, el proceso avanza a la etapa 845, donde se comprueba la dirección física. Si la comprobación indica que es satisfactoria, el proceso avanza a la etapa 850, donde se realiza una comprobación del indicador de contenido válido. Si alguna comprobación en las etapas 845 ú 850 indica falso, el proceso se bifurca y se escribe contenido nuevo en el almacenamiento dedicado 114, ajustándose en consecuencia los campos del mismo. Si la comprobación de la etapa 850 indica verdadero, o si se ha escrito contenido nuevo en la etapa 855, el proceso avanza a la etapa 860, en la que se configuran los campos adecuados para indicar la validez de los datos, después de lo cual se puede llevar a cabo la función solicitada, en la etapa 825. A continuación, el proceso finaliza.

Control de microcaché amplia – hardware

[0045] Haciendo referencia a continuación a las figuras 10 y 11, que muestran juntas el funcionamiento del controlador de la microcaché desde el punto de vista del hardware, se puede comprender mejor el funcionamiento del controlador de la microcaché. En la implementación del hardware, resulta evidente que las condiciones que se indican como etapas secuenciales en las anteriores figuras 8 y 9 se pueden llevar a cabo en paralelo, reduciendo el

retardo para dicha verificación de operando amplio. Además, se puede incluir una copia del hardware indicado para cada microcaché amplia, y de ese modo la totalidad de dichas microcachés, a las que se puede hacer referencia alternativamente mediante una instrucción, se pueden verificar en paralelo. Se considera que no es necesaria ninguna discusión adicional de las figuras 10 y 11 en vista de la discusión extensiva de las anteriores figuras 8 y 9.

[0046] Existen varias alternativas al enfoque anterior para la utilización de los operandos amplios, incluyendo una implementación en la que una única instrucción puede aceptar dos operandos amplios, la división de los operandos en símbolos, multiplicar juntos símbolos correspondientes, y sumar los productos para producir un único valor escalar o un vector de valores divididos de la anchura del archivo de registros, posiblemente después de la extracción de una parte de las sumas. Dicha instrucción puede ser valiosa para la detección de movimiento o la estimación de movimiento en la compresión de video. Una mejora adicional de dicha instrucción puede actualizar incrementalmente el almacenamiento dedicado si la dirección de un operando amplio está dentro del intervalo de operandos amplios especificados anteriormente en el almacenamiento dedicado, mediante cargar solamente la parte que no está ya dentro del intervalo y desplazar según se requiera la parte comprendida en el intervalo. Dicha mejora permite que el funcionamiento se realice sobre una "ventana móvil" de posibles valores. En una instrucción semejante, un operando amplio está alineado y suministra la información de tamaño y forma, mientras que el segundo operando amplio, actualizado incrementalmente, no está alineado.

[0047] La siguiente tabla muestra la notación aritmética y descriptiva utilizada en el pseudocódigo de las figuras a las que se hace referencia a continuación:

x + y	suma de dos complementos, de x e y. El resultado tiene el mismo tamaño que los operandos, y los operandos tienen que tener el mismo tamaño.
x - y	resta de dos complementos, de y respecto de x. El resultado tiene el mismo tamaño que los operandos, y los operandos tienen que tener el mismo tamaño.
x * y	multiplicación de dos complementos, de x e y. El resultado tiene el mismo tamaño que los operandos, y los operandos tienen que tener el mismo tamaño.
x / y	división de dos complementos, de x por y. El resultado tiene el mismo tamaño que los operandos, y los operandos tienen que tener el mismo tamaño.
x & y	operación bit a bit AND, de x e y. El resultado tiene el mismo tamaño que los operandos, y los operandos tienen que tener el mismo tamaño.
x | y	operación bit a bit OR, de x e y. El resultado tiene el mismo tamaño que los operandos, y los operandos tienen que tener el mismo tamaño.
x ^ y	operación bit a bit OR exclusiva, de x e y. El resultado tiene el mismo tamaño que los operandos, y los operandos tienen que tener el mismo tamaño.
~x	inversión bit a bit de x. El resultado tiene mismo tamaño que el operando.
x = y	comparación de la igualdad de dos complementos, entre x e y. El resultado es un solo bit, y los operandos tienen que ser del mismo tamaño.
x ≠ y	comparación de la desigualdad de dos complementos, entre x e y. El resultado es un solo bit, y los operandos tienen que ser del mismo tamaño.
x < y	comparación de menor que, de dos complementos, entre x e y. El resultado es un solo bit, y los operandos tienen que ser del mismo tamaño.
x ≥ y	comparación de mayor o igual que, de dos complementos, entre x e y. El resultado es un solo bit, y los operandos tienen que ser del mismo tamaño.
$\sqrt{x}$	raíz cuadrada con coma flotante, de x.
x || y	concatenación del campo de bits x a la izquierda del campo de bits y.
x^y	dígito binario x repetido, concatenado y veces. el tamaño del resultado es y.
x_y	extracción del bit y (utilizando numeración de bits con comienzo por el extremo bajo) del valor x. El resultado es un solo bit.
$x_{y..z}$	extracción del campo de bits formado por los bits y hasta z del valor x. El tamaño de los resultados es de - z+1; si z > y, el resultado es una cadena vacía.
x?y:z	valor de y, si x es cierto, de lo contrario valor de z. El valor de x es de un solo bit.
x ← y	asignación bit a bit, de x al valor de y.
x.y	campo secundario del campo de bits estructurado x.
Sn	formato de datos binarios de dos complementos, con signo, de n octetos.
Un	formato de datos binarios sin signo, de n octetos.
Fn	formato de datos con coma flotante, de n octetos.

Operaciones amplias

[0048] Los ejemplos particulares de operaciones amplias incluyen Galois de matriz de multiplicación amplia. Si bien la discusión siguiente se centra en tamaños particulares para las instrucciones a modo de ejemplo, se apreciará que la invención no se limita a una anchura particular.

Galois de matriz de multiplicación amplia

[0049] En las figuras 12A a 12D se muestra una realización a modo de ejemplo de la instrucción de Galois de matriz de multiplicación amplia. En una realización a modo de ejemplo, la instrucción de Galois de matriz de multiplicación amplia realiza una multiplicación de matrices de la misma forma que anteriormente, excepto en que las multiplicaciones y sumas se realizan en aritmética de campos de Galois. Se puede especificar un tamaño de 8 bits dentro de la instrucción. Los contenidos de un registro general especifican el polinomio con el que llevar a cabo la operación del resto en campos de Galois. La naturaleza de la multiplicación de matrices es nueva y se describe en detalle a continuación.

[0050] En una realización a modo de ejemplo, estas instrucciones toman una dirección desde un registro general para recuperar de la memoria un operando grande, un segundo y tercer operandos desde los registros generales, realizan un grupo de operaciones en divisiones de bits en los operandos, y concatenan juntos los resultados, colocando el resultado en un registro general. En la figura 12A se muestra una realización a modo de ejemplo del formato 1810 de la instrucción de Galois de matriz de multiplicación amplia.

[0051] En la figura 12B se muestra una realización a modo de ejemplo del esquema 1830 de la instrucción de Galois de matriz de multiplicación amplia. En una realización a modo de ejemplo, los contenidos de un registro re se utilizan como dirección virtual, y se carga desde la memoria un valor de un tamaño específico. Un segundo y un tercer valores son los contenidos de los registros de rd y rb. Los valores se dividen en grupos de operandos del tamaño especificado. Los segundos valores se multiplican como polinomios por el primer valor, produciendo un resultado que se reduce al campo de Galois especificado por el tercer valor, produciendo un grupo de valores de resultado. El grupo de valores de resultado se concatena y se coloca en el registro ra.

[0052] En una realización a modo de ejemplo, la instrucción de octetos de Galois de matriz de multiplicación amplia (W.MUL.MAT.G.8) realiza una multiplicación matricial dividida de hasta 16 384 bits, es decir 128 x 128 bits. La anchura de la matriz se puede limitar a 128, 64, 32 ó 16 bits, pero sin ser menor que el doble del tamaño de grupo de 8 bits, añadiendo la mitad del tamaño deseado en octetos al operando de dirección virtual: 8, 4, 2 ó 1. La matriz se puede limitar verticalmente a 128, 64, 32 ó 16 bits, pero sin ser menor que el doble del tamaño del grupo de 8 bits, añadiendo la mitad del tamaño de operando de memoria deseado en octetos, al operando de dirección virtual.

[0053] En una realización a modo de ejemplo, la dirección virtual tiene que estar alineada a 256 octetos, o bien ser la suma de una dirección alineada y la mitad del tamaño del operando de memoria en octetos y/o la mitad del tamaño del resultado en octetos. Una dirección alineada tiene que ser un múltiplo exacto del tamaño expresado en octetos. Si la dirección no es válida se produce una excepción de "acceso denegado por dirección virtual".

[0054] Tal como se muestra en la figura 12B, una realización a modo de ejemplo de una instrucción de octetos de Galois de matriz de multiplicación amplia (W.MUL.MAT.G.8) multiplica la memoria [m255 m254 ... m1 m0] por el vector [ponmlkjihgfedcba], reduciendo el módulo polinómico [q] del resultado, proporcionando los productos [(pm255 + om247 + ... + bm31 + am15 mod q) (pm254 + om246 + ... + bm30 + am14 mod q) ... (pm248 + om240 + ... + bm16 + am0 mod q)].

[0055] En la figura 12C se muestra una realización a modo de ejemplo del pseudocódigo 1860 de la instrucción de Galois de matriz de multiplicación amplia. En la figura 12D se muestra una realización a modo de ejemplo de las excepciones 1890 de la instrucción de Galois de matriz de multiplicación amplia.

Operandos de memoria de ordenación convencional de octetos con comienzo por el extremo bajo o bien con comienzo por el extremo alto

[0056] Se facilitan operandos de memoria de ordenaciones de octetos convencionales con comienzo por el extremo bajo o bien con comienzo por el extremo alto. Por consiguiente, todas las instrucciones de operandos amplios se especifican en dos formas, una para ordenación de octetos con comienzo por el extremo bajo y una para ordenación de octetos con comienzo por el extremo alto, tal como se especifica mediante una parte de la instrucción. El orden de los octetos especifica al sistema de memoria el orden en el que entregar los octetos dentro de unidades de la anchura del camino de datos (128 bits), así como el orden para colocar múltiples palabras de memoria (128 bits) dentro de un operando amplio mayor.

Extracción de una parte de orden superior de un producto o suma de productos de multiplicador

[0057] La extracción de una parte de orden superior de un producto o suma de productos de multiplicador, es una manera de utilizar eficientemente una matriz grande de multiplicadores. La patente U.S.A. 5 742 840 y la patente U.S.A. 5 953 241 relacionadas describen un sistema y un método para mejorar la utilización de una serie de multiplicadores, añadiendo clases específicas de instrucciones a un procesador de propósito general. Esto aborda el problema de realizar la máxima utilización de una matriz grande de multiplicadores que se utiliza totalmente para aritmética de gran precisión - por ejemplo un multiplicador de 64 x 64 bits se utiliza por completo mediante una multiplicación de 64 bits por 64 bits, pero se utiliza solamente una cuarta parte para una multiplicación de 32 bits por

32 bits) para (en relación con la anchura de datos del multiplicador y los registros) operaciones aritméticas de baja precisión. En particular, se especifican operaciones que realizan multiplicaciones de precisión muy baja que se combinan (suman) juntas de varias maneras. Una de las consideraciones de anulación en la selección del conjunto de operaciones es una limitación del tamaño del operando de resultado. En una realización a modo de ejemplo, por ejemplo, ese tamaño se puede limitar al orden de 128 bits, o de un único registro, aunque no existe una necesidad de limitación específica del tamaño.

[0058] El tamaño de un resultado de multiplicación, un producto, es generalmente la suma de los tamaños de los operandos, de los multiplicandos y del multiplicador. Por consiguiente, las instrucciones de multiplicación especifican operaciones en las que el tamaño del resultado es el doble del tamaño de operandos de entrada de idéntico tamaño. Para nuestro diseño de la técnica anterior, por ejemplo, una instrucción de multiplicación acepta dos fuentes de registro de 64 bits y produce un único resultado de un par de registros de 128 bits, utilizando una matriz entera de 64 x 64 multiplicadores para símbolos de 64 bits, o la mitad de la matriz de multiplicadores para pares de símbolos de 32 bits, o una cuarta parte de la matriz multiplicadora para cuádruplas de símbolos de 16 bits. Para todos estos casos, se debe observar que se combinan dos fuentes de registro de 64 bits, produciendo un resultado de 128 bits.

[0059] En algunas de las operaciones, incluyendo multiplicaciones complejas, convolución, y multiplicación de matrices, se suman juntos productos de multiplicadores de baja precisión. Las sumas incrementan más la precisión requerida. La suma de dos productos requiere un bit adicional de precisión; sumar cuatro productos requiere dos, sumar ocho productos requiere tres, sumar dieciséis productos requiere cuatro. En algunos diseños anteriores, se pierde parte de esta precisión, lo que requiere escalar los operandos multiplicadores para evitar desbordamiento, reduciendo adicionalmente la precisión del resultado.

[0060] La utilización de los pares de registros crea una complejidad no deseable, porque tanto el par de registros como los valores individuales de los registros tienen que ser omitidos en las instrucciones subsiguientes. Como resultado, con las técnicas de la técnica anterior solamente la mitad de los valores del registro de 128 bits de operando fuente se podrían utilizar para producir un resultado de 128 bits de un solo registro.

[0061] Una parte de orden superior del producto o suma de productos de multiplicador es extraída, ajustada por una cantidad de desplazamiento dinámica procedente de un registro general o de un ajuste especificado como parte de la instrucción, y redondeada mediante un valor de control a partir de un registro o parte de instrucción como redondeo al entero/par más cercano, hacia cero, techo o piso. Los desbordamientos se tratan limitando el resultado a los valores máximo y mínimo que se pueden representar con precisión en el resultado de salida.

Extracción controlada por un registro

[0062] Cuando la extracción está controlada por un registro, el tamaño del resultado se puede especificar, lo que permite el redondeo y la limitación a un número menor de bits que pueden caber en el resultado. Esto permite escalar el resultado para su utilización en operaciones posteriores sin preocuparse del desbordamiento o el redondeo, mejorando el rendimiento.

[0063] Cuando la extracción está controlada por un registro, un único valor de registro define el tamaño de los operandos, la cantidad de desplazamiento y el tamaño del resultado, y el control de redondeo. Colocando toda esta información de control en un único registro, se reduce el tamaño de la instrucción respecto del número de bits que dicha instrucción requeriría de otro modo, mejorando el rendimiento e incrementando la flexibilidad del procesador.

[0064] Las instrucciones particulares son extracción convolución de conjunto, extracción multiplicación de conjunto, extracción multiplicación suma de conjunto y extracción suma escalar de conjunto.

Extracción de conjunto in situ

[0065] Las figuras 13A a 13G muestran una instrucción de extracción de conjunto in situ. En una realización a modo de ejemplo, varias de estas instrucciones (extracción convolución de conjunto, extracción multiplicación suma de conjunto) están disponibles habitualmente solamente en formas en las que la extracción se especifica como parte de la instrucción. Una **alternativa** puede incorporar formas de operaciones en las que el tamaño del operando, la cantidad de desplazamiento y el redondeo pueden estar controlados por los contenidos de un registro general (tal como lo están en la instrucción de extracción multiplicación de conjunto). La definición de esa clase de instrucción para extracción convolución de conjunto, y extracción multiplicación suma de conjunto requeriría cuatro registros fuente, lo que incrementa la complejidad al requerir puertos adicionales de lectura de registros generales.

[0066] Estas operaciones toman operandos desde cuatro registros, realizan operaciones sobre divisiones de bits en los operandos, y colocan los resultados concatenados en un cuarto registro. En la figura 13A se muestra una realización a modo de ejemplo del formato y de los códigos de operación 1910 de la instrucción de extracción de conjunto in situ.

[0067] Los esquemas 1930, 1945, 1960 y 1975 de la instrucción de extracción de conjunto in situ se muestran en

las figuras 3C, 13D, 13E y 13F. En una realización a modo de ejemplo, se recuperan los contenidos de los registros rd, rc, rb y ra. La operación especificada se lleva a cabo sobre estos operandos. El resultado se coloca en el registro rd.

[0068] En una realización a modo de ejemplo, para la instrucción E.CON.X, se concatenan los contenidos de los registros rd y rc, como c // d, y se utilizan como un primer valor. Un segundo valor consisten los contenidos del registro rb. Los valores se dividen en grupos de operandos del tamaño especificado y se convolucionan, produciendo un grupo de valores. El grupo de valores es redondeado, limitado y extraído tal como se especifica, produciendo un grupo de resultados de tamaño especificado. El grupo de resultados es concatenado y colocado en el registro rd.

[0069] Para la instrucción E.MUL.ADD.X, los contenidos de los registros rc y rb se dividen en grupos de operandos del tamaño especificado y se multiplican, produciendo un grupo de valores a los que se suman los contenidos divididos extendidos del registro rd. El grupo de valores es redondeado, limitado y extraído tal como se especifica, proporcionando un grupo de resultados de tamaño especificado. El grupo de resultados es concatenado y colocado en el registro rd.

[0070] Tal como se muestra en la figura 13B, los bits 31..0 de los contenidos del registro ra especifican varios parámetros que controlan la manera en que se extraen los datos y, para ciertas operaciones, la manera en que se lleva a cabo la operación. La posición de los campos de control permite que la posición de la fuente se sume a un valor de control fijo para computación dinámica, y permite que los 16 bits inferiores del campo de control se configuren para alguno de los casos de extracción más simples, mediante una única instrucción GCOPYI.128. Los campos de control están dispuestos además de tal modo que es si solamente los 8 bits de orden bajo son distintos de cero, se realiza una extracción de 128 bits con truncado y sin redondeo.

[0071] La siguiente tabla describe el significado de cada etiqueta:

etiqueta	bits	significado
fsize	8	tamaño del campo
dpos	8	posición de destino
x	1	resultado extendido vs. tamaño de grupo
s	1	con signo vs. sin signo
n	1	multiplicación compleja vs. real
m	1	multiplicación de signo mezclado vs. mismo signo
l	1	límite: saturación vs. truncado
rnd	2	redondeo
gssp	9	tamaño del grupo y posición de la fuente

[0072] En una realización a modo de ejemplo, el campo **gssp** de 9 bits codifica tanto el tamaño del grupo, **gsize**, como la posición de la fuente, spos, de acuerdo con la fórmula **gssp** = 512 - 4***gsize + spos**. El tamaño del grupo, **gsize**, es una potencia de dos comprendida en el intervalo de 1..128. La posición de la fuente, **spos**, está comprendida en el intervalo de 0..(2*gsize) - 1.

[0073] En una realización a modo de ejemplo, los valores en los campos **x, s, n, m, l** y **rnd** tienen el siguiente significado:

valores	x	s	n	m	l	rnd
0	grupo	sin signo	real	mismo signo	truncado	F
1	extendido	con signo	complejo	signo mezclado	saturar	Z
2						N
3						C

Extracción multiplicación suma de conjunto

[0074] Tal como se muestra en la figura 13C una instrucción de extracción multiplicación suma de conjunto dobletes (E.MULADDX) multiplica el vector rc [h g f e d c b a] por el vector rb [p o n m l k j i], y suma el vector rd [x w v u t s r q], proporcionando el vector resultado rd [hp+x go+w fn+v em+u dl+t ck+s bj+r ai+q], redondeado y limitado tal como se especifica mediante ra31..0.

[0075] Tal como se muestra en la figura 13D, una instrucción de extracción multiplicación suma de conjunto dobletes complejos (E.MUL.X con n activado) multiplica el vector operando rc [h g f e d c b a] por el vector operando rb [p o n m l k j i], proporcionando el vector resultado rd [gp + ho go - hp en + fm em - fn cl + dk ck - dl aj + bi ai - bj], redondeado y limitado tal como se especifica mediante ra31..0. Se debe observar que esta instrucción prefiere una organización de números complejos en la que la parte real esté situada a la derecha (menor precisión) de la parte

imaginaria.

Extracción convolución de conjunto

[0076] Tal como se muestra en la figura 13E, una instrucción de extracción convolución de conjunto dobletes (ECON.X con n = 0) convoluciona el vector rc // rd [x w v u t s r q p o n m l kj i] con el vector rb [h g f e d c b a], proporcionando el vector de productos rd

[ax + bw + cv + du + et + fs + gr + hq ... as + br + cq + dp + eo + fn + gm + hl ar + bq + cp + do + en + fm + gl + hk aq + bp + co + dn + em + fl + gk + hj], redondeado y limitado tal como se especifica por ra31..0.

[0077] Tal como se muestra en la figura 13F, una instrucción de extracción convolución de conjunto dobletes complejos (ECON.X con n=1) convoluciona el vector rd // rc [x w v u t s r q p o n m l k j i] con el vector rb [h g f e d c b a], proporcionando el vector de productos rd

[0078] [ax + bw + cv + du + et + fs + gr + hq ... as - bt + cq - dr + eo - fp + gm - hn ar + bq + cp + do + en + fm + gl + hk aq - br + co - dp + em - fn + gk + hl], redondeado y limitado tal como se especifica por ra31..0.

[0079] El pseudocódigo 1990 de la instrucción de extracción de conjunto in situ se muestra en la figura 13G. Puede no haber excepciones para la instrucción de extracción de conjunto in situ.

Extracción de conjunto

[0080] En las figuras 14A a 14J se muestra una instrucción de extracción de conjunto. Estas operaciones toman operandos de tres registros, realizan las operaciones sobre divisiones de bits en los operandos, y colocan los resultados concatenados en un cuarto registro. El formato y los códigos de operación 2010 de la instrucción de extracción conjunto se muestran en la figura 14A.

[0081] En las figuras 14C, 14D, 14E, 20F, 20G, 20H y 20I se muestra a un esquema 2020, 2030, 2040, 2050, 2060, 2070 y 2080 de la instrucción de extracción de conjunto in situ. Los contenidos de los registros rd, rc y rb son recuperados. La operación especificada se lleva a cabo sobre estos operandos. El resultado se coloca en el registro ra.

[0082] Tal como se muestra en la figura 14B, los bits 31..0 de los contenidos del registro rb especifican varios parámetros que controlan la manera en que se extraen los datos y, para ciertas operaciones, la manera en que se lleva a cabo la operación. La posición de los campos de control permite que la posición de la fuente se sume a un valor de control fijo para computación dinámica, y permite que los 16 bits inferiores del campo de control se configuren para alguno de los casos de extracción más simples, mediante una única instrucción GCOPYI.128. Los campos de control están dispuestos además de tal modo que es si solamente los 8 bits de orden bajo son distintos de cero, se realiza una extracción de 128 bits con truncado y sin redondeo.

[0083] La siguiente tabla describe el significado de cada etiqueta:

etiqueta	bits	significado
fsize	8	tamaño del campo
dpos	8	posición de destino
x	1	resultado extendido vs. tamaño de grupo
s	1	con signo vs. sin signo
n	1	multiplicación compleja vs. real
m	1	fusionar vs. extraer, o multiplicación de signo mezclado vs. mismo signo
l	1	límite: saturación vs. truncado
rnd	2	redondeo
gssp	9	tamaño del grupo y posición de la fuente

[0084] El campo gssp de 9 bits codifica tanto el tamaño del grupo, gsize, como la posición de la fuente, spos, de acuerdo con la fórmula gssp = 512 4*gsize + spos. El tamaño del grupo, gsize, es una potencia de dos comprendida en el intervalo de 1..128. La posición de la fuente, spos, está comprendida en el intervalo de 0..(2*gsize) - 1.

[0085] Los valores en los campos x, s, n, m, l y rnd tienen el significado siguiente:

valores	x	s	n	m	l	rnd
0	grupo	sin signo	real	extraer/mismo signo	truncado	F
1	extendido	con signo	complejo	fusionar/signo mezclado	saturar	Z
2						N
3						C

[0086] Para la instrucción E.SCAL.ADD.X, los bits 127..64 de los contenidos del registro rb especifican los multiplicadores para los multiplicandos en los registros rd y rc. Específicamente, los bits 64 + 2*gsize-1 .. 64 + gsize consisten en el multiplicador para los contenidos del registro rd, y los bits 64 + gsize-1 .. 64 consisten en el multiplicador para los contenidos del registro rc.

Extracción multiplicación de conjunto

[0087] Tal como se muestra en la figura 14C, una instrucción de extracción multiplicación de conjunto dobletes (E.MULX) multiplica el vector rd [h g f e d c b a] por el vector rc [p o n m l kj i], proporcionando el vector de resultado ra [hp go fn em dl ck bj ai], redondeado y limitado tal como se especifica mediante $rb_{31..0}$.

[0088] Tal como se muestra en la figura 14D, una instrucción de extracción multiplicación de conjunto dobletes complejos (E.MUL.X con n activado) multiplica el vector rd [h g f e d c b a] por el vector rc [p o n m l k j i], proporcionando el vector de resultado ra [gp + ho go - hp en + fm em - fn cl + dk ck - dl aj + bi ai - bj], redondeado y limitado tal como se especifica mediante $rb_{31..0}$. Se debe observar que esta instrucción prefiere una organización de números complejos en la que la parte real esté situada a la derecha (menor precisión) de la parte imaginaria.

Extracción suma escalar de conjunto

[0089] La instrucción de extracción suma escalar de conjunto, combina la información de control de extracción en un registro, con dos valores que se utilizan como multiplicadores escalares para los contenidos de dos multiplicandos vectoriales.

[0090] Esta combinación reduce el número de registros que serían necesarios en otro caso, o el número de bits que de lo contrario requeriría la instrucción, mejorando el rendimiento. Otra ventaja de la presente invención es que la operación combinada se puede llevar a cabo mediante una realización a modo de ejemplo con la suficiente precisión interna en el nodo de sumación como para que no se produzca redondeo o desbordamiento intermedio, mejorando la precisión sobre la operación de la técnica anterior, en la que era necesario llevar a cabo más de una instrucción para este cálculo.

[0091] Tal como se muestra en la figura 14E, una instrucción de extracción de suma escalar de conjunto dobletes (E.SCAL.ADD.X) multiplica el vector rd [h g f e d c b a] por $rb_{95.80}$ [r] y suma el producto al producto del vector rc [p o n m l k j i] por $rb_{79..64}$ [q], proporcionando el resultado [hr + pq gr + oq fr + nq er + mq dr + lq cr + kq br + jq ar + iq], redondeado y limitado tal como se especifica por $rb_{31..0}$.

[0092] Tal como se muestra en la figura 14F, una instrucción de extracción suma escalar de conjunto dobletes complejos (E.SCLADD.X con n activado) multiplica el vector rd [h g fe d c b a] por $rb_{127..96}$ [t s] y suma el producto, al producto del vector rc [p o n m l k j i] por $rb_{95..64}$ [r q], proporcionando el resultado [hs + gt + pq + or gs-ht + oq - pr fs + et + nq + mr es - ft + mq - nr ds + ct + lq + kr cs - dt + kq - lr bs + at + jq + ir as - bt + iq - jr], redondeado y limitado tal como se especifica mediante $rb_{31..0}$.

Extracción de conjunto

[0093] Tal como se muestra en la figura 14G, para la instrucción E.EXTRACT, cuando m=0 y x=0, los parámetros especificados por los contenidos del registro rb se interpretan para seleccionar campos de símbolos de doble tamaño de los contenidos concatenados de los registros rd y rc, extrayendo valores que son concatenados y colocados en el registro ra.

[0094] Tal como se muestra en la figura 14H, para una extracción fusión de conjunto (E.EXTRACT cuando m=1), los parámetros especificados por los contenidos del registro rb se interpretan para fusionar campos a partir de símbolos de los contenidos del registro rd con los contenidos del registro rc. Los resultados son concatenados y colocados en el registro ra. El campo x no tiene ningún efecto cuando m=1.

[0095] Tal como se muestra en la figura 14I, para una extracción expansión de conjunto (E.EXTRACT cuando m=0 y x=1), los parámetros especificados por los contenidos del registro rb se interpretan para extraer campos de símbolos de los contenidos del registro rd. Los resultados son concatenados y colocados en el registro ra. Se debe observar que no se utiliza el valor de rc.

[0096] En la figura 14J se muestra el pseudocódigo 2090 de la instrucción de extracción de conjunto. No existen excepciones para la instrucción de extracción de conjunto.

Reducción de puertos de lectura de registros

[0097] Es posible reducir el número de puntos de lectura de registro necesarios para la implementación de instrucciones en las que el tamaño, el desplazamiento y el redondeo de los operandos están controlados por un

registro. El valor del registro de control de extracción puede ser recuperado utilizando un ciclo adicional de una ejecución inicial, y retenido dentro de la unidad funcional o cerca de la misma para ejecuciones subsiguientes, reduciendo de este modo la cantidad de hardware necesario para la implementación con una pequeña penalización adicional sobre el rendimiento. El valor retenido se marcaría como no válido, provocando una nueva recuperación del registro de control extraído, mediante instrucciones que modifican el registro o, alternativamente, el valor retenido se puede actualizar mediante dicha operación. Una nueva recuperación del registro del control de la extracción sería necesaria asimismo si se especificara un número de registro diferente en una ejecución posterior. Debería resultar evidente que las propiedades de las anteriores dos realizaciones alternativas se pueden combinar.

Aritmética de campos de Galois

[0098] En la aritmética de campos de Galois, las multiplicaciones se llevan a cabo mediante una multiplicación polinómica binaria inicial (multiplicaciones binarias sin signo con acarreos suprimidos), seguida por una operación de modulo polinómico/resto (división binaria sin signo con acarreos suprimidos). La operación de resto es relativamente costosa en superficie y retardo. En la aritmética de campos de Galois, la suma se lleva a cabo mediante la adición binaria con acarreos suprimidos o, de manera equivalente, una operación OR exclusiva bit a bit. Se puede llevar a cabo una multiplicación de matrices utilizando aritmética de campos de Galois, donde las multiplicaciones y las sumas son multiplicaciones y sumas de campos de Galois.

[0099] Utilizando métodos de la técnica anterior, la multiplicación de un vector de 16 octetos por una matriz de 16 x 16 octetos se puede realizar como 256 multiplicaciones de campos de Galois de 8 bits y 16* 15=240 sumas de campos de Galois de 8 bits. En las 256 multiplicaciones de campos de Galois están incluidas 256 multiplicaciones polinómicas y 256 operaciones de resto polinómico.

[0100] El cálculo total se reduce significativamente llevando a cabo 256 multiplicaciones polinómicas, 240 sumas polinómicas 16 bits y 16 operaciones de resto polinómico. Se debe observar que el coste de las sumas polinómicas se ha duplicado en comparación con las sumas de campos de Galois, dado que estas son ahora operaciones de 16 bits en lugar de operaciones de 8 bits, pero el coste de las funciones de resto polinómico se ha reducido en un factor de 16. En conjunto, esto constituye una solución favorable, dado que el coste de la suma es mucho menor que el coste del resto.

Acceso desacoplado de los segmentos de ejecución y multihilos simultáneos

[0101] Tal como se muestra mejor en la figura 4, la presente invención utiliza un acceso desacoplado respecto de las segmentaciones de ejecución y multihilos simultáneos, de una manera única. Las segmentaciones multihilos simultáneas se han utilizado en la técnica anterior para mejorar la utilización de las unidades de camino de datos permitiendo que se emitan instrucciones desde uno de varios hilos de ejecución a cada unidad funcional (por ejemplo Dean M. Tullsen, Susan J. Eggers, y Henry M. Levy, "Simultaneous Multithreading: Maximizing On Chip Parallelism," Proceedings of the 22nd Annual International Symposium on Computer Architecture, Santa Margherita Ligure, Italia, junio de 1995).

[0102] El acceso desacoplado respecto de las segmentaciones de ejecución se ha utilizado en la técnica anterior para mejorar la utilización de las unidades de camino de datos de ejecución, mediante almacenar en memoria tampón resultados procedentes de una unidad de acceso, que calcula direcciones para una unidad de memoria que, a su vez, recupera de la memoria los elementos solicitados, y presentarlos a continuación a una unidad de ejecución (por ejemplo, J. E. Smith, "Decoupled Access/Execute Computer Architectures", Proceedings of the Ninth Annual International Symposium on Computer Architecture, Austin, Tejas (26 a 29 de abril de 1982), páginas 112 a 119).

[0103] En comparación con las segmentaciones convencionales, la técnica anterior de Eggers utilizada en el ciclo de segmentación adicional antes de las instrucciones podría ser emitida para unidades funcionales, el ciclo adicional siendo necesario para determinar a qué hilos se debería permitir emitir instrucciones. Por consiguiente, en relación con las segmentaciones convencionales, el diseño de la técnica anterior tenía un retardo adicional, incluyendo un retardo dependiente de las ramificaciones.

[0104] En el presente documento se describen unidades de camino de datos de acceso, con archivos de registros asociados, para cada hilo de ejecución. Estas unidades de acceso producen direcciones, que se agregan juntas a una unidad de memoria común, que recupera todas las direcciones y coloca los contenidos de la memoria en una o varias memorias tampón. Las instrucciones para las unidades de ejecución, que se comparten en grados variables entre los hilos, son asimismo almacenadas en memoria tampón para su ejecución posterior. Las unidades de ejecución llevan a cabo a continuación operaciones desde todos los hilos activos utilizando unidades de camino de datos funcionales que son compartidas.

[0105] Para las instrucciones llevadas a cabo por las unidades de ejecución, el ciclo adicional requerido por los diseños multihilo simultáneos de la técnica anterior se solapa con el tiempo de acceso a los datos de memoria del acceso desacoplado de la técnica anterior, de los ciclos de ejecución, de tal modo que las unidades funcionales no incurren en ningún retardo adicional para planificar recursos. Para las instrucciones llevadas a cabo por las unidades

de acceso, al utilizar unidades de acceso individuales para cada hilo se elimina asimismo el ciclo adicional para planificar recursos compartidos.

[0106] Ésta constituye una solución favorable debido a que, si bien los hilos no comparten las unidades funcionales de acceso, estas unidades son relativamente pequeñas en comparación con las unidades funcionales de ejecución, que son compartidas por los hilos.

[0107] En relación con la compartición de unidades de ejecución, la presente invención utiliza varias clases diferentes de unidades funcionales para la unidad de ejecución, con coste, utilización y rendimiento variables. En particular, las unidades G, que llevan a cabo operaciones simples de suma y bit a bit, son relativamente costosas (en superficie y potencia) en comparación con las otras unidades, y su utilización es relativamente alta. Por consiguiente, el diseño utiliza cuatro de dichas unidades, donde cada unidad puede estar compartida entre dos hilos. La unidad X, que lleva a cabo una clase extensa de funciones de conmutación de datos, es más costosa y menos utilizada, de manera que se disponen dos unidades que se comparten, cada una, entre dos hilos. La unidad T, que lleva a cabo la instrucción de traducción amplia, es costosa y de utilización reducida, de manera que la única unidad se comparte entre la totalidad de los cuatro hilos. La unidad E, que lleva a cabo la clase de instrucciones de conjunto, es muy costosa en superficie y potencia en comparación con las otras unidades funcionales, pero su utilización es relativamente alta, de manera que se disponen dos de dichas unidades, estando cada unidad compartida por dos hilos.

[0108] En la figura 4, se muestran cuatro copias de una unidad de acceso, cada una con una cola de recuperación de instrucciones de acceso cola-A 401 a 404, acoplada a un archivo de registros de acceso AR 405 a 408, cada uno de los cuales está acoplado, a su vez, a dos unidades funcionales de acceso A 409 a 416. Las unidades de acceso funcionan independientemente para cuatro hilos simultáneos de ejecución. Estas ocho unidades funcionales de acceso A 409 a 416 producen resultados para archivos de registros de acceso AR 405 a 408, y direcciones para un sistema de memoria compartido 417. Los contenidos de memoria recuperados del sistema de memoria 417 se combinan con instrucciones de ejecución no realizadas por la unidad de acceso, y se introducen en las cuatro colas de instrucción de ejecución E-colas 421 a 424. Los datos de memoria e instrucciones de la cola-E 421 a 424 se presentan a archivos de registros de ejecución 425 a 428, que recuperan operandos fuente de archivo de registros de ejecución. Las instrucciones se acoplan a la unidad de arbitraje de las unidades de ejecución Arbitraje 431, que selecciona qué instrucciones de los cuatro hilos tienen que ser encaminadas a las unidades de ejecución disponibles E 441 y 449, X 442 y 448, G 443 a 444 y 446 a 447 y T 445. Los operandos fuente del archivo de registros de ejecución ER 425 a 428 se acoplan las unidades de ejecución 441 a 445 utilizando buses de operandos fuente 451 a 454, y a las unidades de ejecución 445 a 449 utilizando buses de operandos fuente 455 a 458. Los operandos de resultado de unidades de función, procedentes de las unidades de ejecución 441 a 445 se acoplan al archivo de registros de ejecución utilizando el bus de resultados 461, y los operandos de resultado de unidades de función, procedentes de las unidades de ejecución 445 a 449 se acoplan al archivo de registros de ejecución utilizando el bus de resultados 462.

Suma de grupos

[0109] La figura 16A presenta varios ejemplos de instrucciones de suma de grupos que alojan los diferentes tamaños de operandos, tales como un octeto (8 bits), un doblete (16 bits), una cuádrupla (32 bits), una 8-tupla (64 bits) y una 16-tupla (128 bits). Las figuras 16B y 16C muestran una realización a modo de ejemplo de un formato y códigos de operación que pueden ser utilizados para llevar a cabo las diversas instrucciones de suma de grupos mostradas en la figura 16A. Tal como se muestra en las figuras 16B y 16C en esta realización a modo de ejemplo, los contenidos de los registros rc y rb se dividen en grupos de operando del tamaño especificado y se suman y, si se especifica, se verifica su desbordamiento o se limitan, proporcionando un grupo de resultados, cada uno de los cuales tiene el tamaño especificado. El grupo de resultados es concatenado y colocado en el registro rd. Si bien en este caso y en otros de la presente descripción se describe la utilización de dos registros de operandos y un registro de resultado diferente, pueden ser implementadas asimismo otras disposiciones, tal como la utilización de valores inmediatos.

[0110] Por ejemplo, si el tamaño de operando especificado es un octeto (8 bits), y cada registro tiene 128 bits de amplitud, entonces el contenido de cada registro se puede dividir en 16 operandos individuales, y pueden tener lugar 16 operaciones de suma individuales diferentes, como resultado de una única instrucción de suma de grupo. Otras instrucciones que involucran grupos de operandos pueden llevar a cabo operaciones de grupo de manera similar.

Establecimiento de grupo y resta de grupo

[0111] Análogamente, la figura 17A presenta varios ejemplos de instrucciones de establecimiento de grupo e instrucciones de resta de grupo que alojan diferentes tamaños de operando. Las figuras 17B y 17C muestran una realización a modo de ejemplo de un formato y códigos de operación que pueden ser utilizados para llevar a cabo diversas instrucciones de establecimiento de grupo e instrucciones de resta del grupo. Tal como se muestra en las figuras 17B y 17C, los contenidos de los registros rc y rb están divididos en grupos de operandos del tamaño especificado, y para las instrucciones de establecimiento de grupo se comparan para una condición aritmética

especificada, o para instrucciones de resta de grupo se restan y, si se especifica, se verifica su desbordamiento o se limitan, proporcionando un grupo de resultados, cada uno de los cuales tiene el tamaño especificado. El grupo de resultados es concatenado y colocado en el registro rd.

Convolución, división, multiplicación, suma de multiplicaciones de conjunto

[0112] Están disponibles asimismo otras operaciones de grupo de punto fijo. La figura 18A representa varios ejemplos de instrucciones de convolución de conjunto, división de conjunto, multiplicación de conjunto y suma de multiplicaciones de conjunto, que alojan diferentes tamaños de operandos. Las figuras 18B y 18C muestran una realización a modo de ejemplo de un formato y códigos de operación que pueden ser utilizados para llevar a cabo las diversas instrucciones de convolución de conjunto, división de conjunto, multiplicación de conjunto y suma de multiplicaciones de conjunto. Tal como se muestra en la figura 18B y 18C, en esta realización a modo de ejemplo, los contenidos de los registros rc y rb son divididos en grupos de operandos del tamaño especificado y convolucionados o divididos o multiplicados, proporcionando un grupo de resultados, o multiplicados y sumados en un único resultado. El grupo de resultados es concatenado y colocado, o el resultado único es colocado, en el registro rd.

Suma, división, multiplicación y resta de conjunto con coma flotante

[0113] De acuerdo con una realización de la invención, el procesador maneja asimismo diversas operaciones de grupo con coma flotante que alojan diferentes tamaños de operandos. En este caso, los diferentes tamaños de operandos pueden representar operandos con coma flotante de precisiones diferentes, tal como precisión media (16 bits), precisión simple (32 bits), precisión doble (64 bits) y precisión cuádruple (128 bits). La figura 19 muestra funciones a modo de ejemplo están definidas para su utilización dentro de las definiciones de instrucción detalladas en otras secciones y figuras. En las funciones presentadas en la figura 19, un formato interno representa valores con coma flotante con precisión infinita como una estructura de cuatro elementos que consiste en (1) s (bit de signo): 0 para positivo, 1 para negativo, (2) t (tipo): NORM, ZERO, SNAN, QNAN, INFINITY, (3) e (exponente) y (4) f: (fracción). La interpretación matemática de un valor normal coloca el punto binario en las unidades de la fracción, ajustado por el exponente: $(-1)^{\wedge s}*(2^{\wedge e})*f$. La función F transforma un valor con coma flotante IEEE empaquetado al formato interno. La función PackF transforma un formato interno de nuevo al formato de coma flotante IEEE, con control de redondeo y excepciones.

[0114] Las figuras 20A y 21A presentan varios ejemplos de instrucciones de suma, división, multiplicación y resta con coma flotante de conjunto. Las figuras 20B a C y 21B a C muestran una realización a modo de ejemplo de formatos y códigos de operación que pueden ser utilizados para llevar a cabo las diversas instrucciones de suma, división, multiplicación y resta con coma flotante de conjunto. En estos ejemplos, las instrucciones de conjunto de suma, división y multiplicación con coma flotante se han identificado como "EnsembleFloatingPoint". Asimismo, las instrucciones de conjunto de resta con coma flotante se han identificado como "EnsembleReversedFloatingPoint". Tal como se muestra en las figuras 20B a C y 21B a C, en esta realización a modo de ejemplo, los contenidos de los registros rc y rb se dividen en grupos de operandos del tamaño especificado, y se lleva a cabo la operación de grupo especificado, proporcionando un grupo de resultados. El grupo de resultados se concatena y se coloca en el registro rd.

[0115] En la presente realización, la operación se redondea utilizando la opción especificada de redondear, o utilizando redondear al más próximo, si no se especifica. Si se especifica una opción de redondeo, la operación presenta una excepción de coma flotante si se produce una operación no válida de coma flotante, división por cero, desbordamiento o subdesbordamiento o, cuando se especifica, si el resultado es inexacto. Si no se especifica una opción de redondeo, no se presentan excepciones de coma flotante, y se manejan de acuerdo con las normas por defecto de IEEE 754.

Realización de una operación booleana bit a bit de tres entradas en una única instrucción (booleana de grupo)

[0116] Se da a conocer un sistema y un método para la realización de una operación Booleana bit a bit de tres entradas en una única instrucción. Se utiliza un método nuevo para codificar los ocho posibles estados de salida de dicha operación en solamente siete bits, y se descodifican estos siete bits de nuevo en los ocho estados.

[0117] En las figuras 15A a 15C se muestra una instrucción Booleana de grupo. En una realización a modo de ejemplo, estas operaciones toman operandos de tres registros, realizan operaciones booleanas sobre bits correspondientes en los operandos, y colocan los resultados concatenados en el tercer registro. En la figura 15A se muestra un formato 2310 de la instrucción booleana de grupo.

[0118] En la figura 15B se muestra un procedimiento 2320 de la instrucción booleana de grupo. Se toman tres valores de los contenidos de los registros rd, rc y rb. Los campos ih e il especifican una función de tres bits, que produce un resultado de un solo bit. La función especificada se evalúa para cada posición de bit, y los resultados se concatenan y se colocan en el registro rd. El registro rd es tanto una fuente como un destino de esta instrucción.

[0119] La función se especifica mediante ocho bits, que proporcionan el resultado para cada posible valor de los tres bits fuente en cada posición de bit:

d	1	1	1	1	0	0	0	0
c	1	1	0	0	1	1	0	0
b	1	0	1	0	1	0	1	0
$f_{(d,c,b)}$	f_7	f_6	f_5	f_4	f_3	f_2	f_1	f_0

[0120] Una función se puede modificar reordenando los bits del valor inmediato. La tabla siguiente muestra cómo el reordenamiento del valor inmediato $f_{7.0}$ puede reordenar los operandos d, c, b para la misma función.

operación	inmediato
f(d,c,b)	f_7 f_6 f_5 f_4 f_3 f_2 f_1 f_0
f(c,d,b)	f_7 f_6 f_3 f_2 f_5 f_4 f_1 f_0
f(d,b,c)	f_7 f_5 f_6 f_4 f_3 f_1 f_2 f_0
f(b,c,d)	f_7 f_3 f_5 f_1 f_6 f_2 f_4 f_0
f(c,b,d)	f_7 f_5 f_3 f_1 f_6 f_4 f_2 f_0
f(b,d,c)	f_7 f_3 f_6 f_2 f_5 f_1 f_4 f_0

[0121] Utilizando este reordenamiento, una operación de la forma: b = f(d, c, b) puede ser decodificada en la forma lícita: b = f(b, d, c). Por ejemplo, la función de: b=f(d, c, b)=d?c:b no se puede codificar, pero la función equivalente: d=c?b:d se puede determinar reordenando el código para d=f(d, c, b)=d?c: b, que es 11001010, de acuerdo con la norma para f(d, c, b) => f(c, b, d), al código 11011000.

Codificación

[0122] Algunas características especiales de este reordenamiento constituyen la base del modo en que ocho bits de especificación de la función se comprimen en siete bits inmediatos, en esta instrucción. Tal como se ve en la tabla anterior, en el caso general, un reordenamiento de operandos desde f(d,c,b) hasta f(d,b,c) (intercambiando rc y rb) requiere intercambiar los valores de f_6 y f_5 y los valores de f_2 y f_1.

[0123] Entre las 256 posibles funciones que puede llevar a cabo esta instrucción, una cuarta parte de las mismas (64 funciones) permanecen inalteradas mediante este reordenamiento. Estas funciones tienen la propiedad de que $f_6 = f_5$ y $f_2 = f_1$. Los valores de rc y rb (se debe observar que rc y rb son especificadores de registros, no los contenidos de los registros) se pueden intercambiar libremente, y por lo tanto están ordenados en orden ascendente o descendente para indicar el valor de f_2. (Se produce un caso especial cuando rc = rb, de manera que el ordenamiento de rc y rb no puede contener información. Sin embargo, dado que en este caso solamente pueden resultar los valores f_7, f_4, f_3 y f_0, no es necesario codificar f_6, f_5, f_2 y f_1 para este caso, de manera que no se requiere ningún tratamiento especial.) Estas funciones se codifican por los valores de f_7, f_6, f_4, f_3 y f_0 en el campo intermedio y f_2 en función de si rc > rb, utilizando por lo tanto 32 valores intermedios para 64 funciones.

[0124] Otra cuarta parte de las funciones tienen $f_6 = 1$ y $f_5 = 0$. Estas funciones se registran intercambiando rc y rb, f_6 y f_5, f_2 y f_1. Éstas comparten la misma codificación que la cuarta parte de las funciones en las que $f_6 = 0$ y $f_5 = 1$, y se codifican mediante los valores f_7, f_4, f_3, f_2, f_1 y f_0 en el campo intermedio inmediato, utilizando por lo tanto 64 valores intermedios para 128 funciones.

[0125] La cuarta parte restante de las funciones tienen $f_6 = f_5$ y $f_2 \neq f_1$. La mitad de éstas, en las que $f_2 = 1$ y $f_1 = 0$, se registran intercambiando rc y rb, f_6 y f_5, f_2 y f_1. Éstas comparten entonces la misma codificación de las ocho de las funciones en las que $f_2 = 0$ y $f_1 = 1$, y se codifican mediante los valores f_7, f_6, f_4, f_3 y f_0 en el campo inmediato, utilizando por lo tanto 32 valores inmediatos para 64 funciones.

[0126] La codificación de funciones se resume mediante la tabla siguiente:

f_7	f_6	f_5	f_4	f_3	f_2	f_1	f_0	trc>trb	ih	il_5	il_4	il_3	il_2	il_1	il_0	rc	rb
		f_6				f_2		f_2	0	0	f_6	f_7	f_4	f_3	f_0	trc	trb
		f_6				f_2		~f_2	0	0	f_6	f_7	f_4	f_3	f_0	trb	trc
		f_6			0	1			0	1	f_6	f_7	f_4	f_3	f_0	trc	trb
		f_6			1	0			0	1	f_6	f_7	f_4	f_3	f_0	trb	trc
	0	1							1	f_2	f_1	f_7	f_4	f_3	f_0	trc	trb
	1	0							1	f_1	f_2	f_7	f_4	f_3	f_0	trb	trc

[0127] La descodificación de funciones se resume en mediante la tabla siguiente:

ih	il_5	il_4	il_3	il_2	il_1	il_0	rc> rb	f_7	f_6	f_5	f_4	f_3	f_2	f_1	f_0
0	0						0	il_3	il_4	il_4	il_2	il_1	0	0	il_0
0	0						1	il_3	il_4	il_4	il_2	il_1	1	1	il_0
0	1							il_3	il_4	il_4	il_2	il_1	0	1	il_0
1								il_3	0	1	il_2	il_1	il_5	il_4	il_0

[0128] Por la discusión anterior, se puede apreciar que un compilador o ensamblador que produce la instrucción codificada lleva a cabo las etapas anteriores para codificar la instrucción, comparando los valores f6 y f5 y los valores f2 y fl del campo inmediato para determinar cuál de los diversos medios de codificación del campo inmediato debe ser utilizado, y que la colocación de los especificadores del registro trb y trc en la instrucción codificada depende de los valores de f2 (o fl) y f6 (o f5).

[0129] El pseudocódigo 2330 de la instrucción booleana de grupo se muestra en la figura 15C. A partir del código se puede apreciar que una realización a modo de ejemplo de un circuito que descodifica esta instrucción produce los valores f2 y f1, cuando los bits inmediatos ih e il5 son cero, mediante la comparación aritmética de los especificadores de registro rc y rb, produciendo un valor de uno (1) para f2 y f1 cuando rc > rb. No existen excepciones para la instrucción booleana de grupo.

Compresión, expansión, rotación y desplazamiento de barras cruzadas

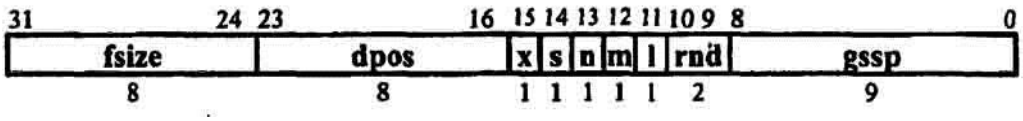
[0130] Las unidades de conmutación de barras cruzadas, tales como las unidades 142 y 148 llevan a cabo operaciones de manipulación de datos, tal como se ha descrito anteriormente. Tal como se muestra en la figura 22A, dichas operaciones de manipulación de datos pueden incluir varios ejemplos de operaciones de compresión de barras cruzadas, expansión de barras cruzadas, rotación de barras cruzadas y desplazamiento de barras cruzadas. Las figuras 22B y 22C muestran una realización a modo de ejemplo de un formato y códigos de operación que pueden ser utilizados para llevar a cabo las diversas instrucciones de compresión de barras cruzadas, rotación de barras cruzadas, expansión de barras cruzadas y desplazamiento de barras cruzadas. Tal como se muestra en las figuras 22B y 22C. Los contenidos del registro rc son divididos en grupos de operandos del tamaño especificado, y comprimidos, expandidos, rotados o desplazados en la cantidad especificada mediante una parte de los contenidos del registro rb, proporcionando un grupo de resultados. El grupo de resultados es concatenado y colocado en el registro rd.

[0131] Varias operaciones de compresión de grupo puedan convertir grupos de operandos, desde datos de mayor precisión a datos de menor precisión. Se puede seleccionar un subcampo de cada campo de bits, de la mitad de tamaño, arbitrario, para que se muestre en el resultado. Por ejemplo, la figura 22D muestra una operación X.COMPRESS rd = rc, 16, 4, que lleva a cabo una selección de bits 19..4 de cada cuádrupla en una 16-tupla. Varias operaciones de desplazamiento del grupo pueden permitir el desplazamiento de grupos de operandos en un número especificado de bits, en un sentido especificado, tal como desplazamiento a la derecha o desplazamiento a la izquierda. Tal como se puede ver en la figura 22C, cierto grupo de instrucciones de desplazamiento a la izquierda de grupo pueden involucrar asimismo borrar (poner a cero) los bits vacíos de orden bajo asociados con el desplazamiento, para cada operando. Cierto grupo de instrucciones de desplazamiento a la derecha de grupo puede involucrar borrar (poner a cero) los bits vacíos de orden alto asociados con el desplazamiento, para cada operando. Además, ciertas instrucciones de desplazamiento a la derecha de grupo pueden involucrar rellenar bits vacíos de orden alto asociados con el desplazamiento, con copias del bit de signo, para cada operando.

Extracción

[0132] Las operaciones de manipulación de datos pueden incluir asimismo una instrucción de extracción de barras cruzadas. Las figuras 23A y 23B muestran una realización a modo de ejemplo de un formato y códigos de operación que pueden ser utilizados para llevar a cabo la instrucción de extracción de barras cruzadas. Tal como se muestra en las figuras 23A y 23B, se recuperan los contenidos de los registros rd, rc y rb. La operación especificada se lleva a cabo sobre estos operandos. El resultado se coloca en el registro ra.

[0133] La instrucción de extracción de barras cruzadas permite que se extraigan bits de operandos diferentes, de varias maneras. Específicamente, los bits 31..0 de los contenidos del registro rb especifican varios parámetros que controlan la manera mediante la que se extraen datos, y para ciertas operaciones, la manera mediante la que se lleva a cabo la operación. La posición de los campos de control permite que la posición de la fuente se sume a un valor de control fijo para computación dinámica, y permite que los 16 bits inferiores del campo de control se configuren para alguno de los casos de extracción más simples, mediante una única instrucción GCOPYI.128 (ver el apéndice). Los campos de control están dispuestos además de tal modo que es si solamente los 8 bits de orden bajo son distintos de cero, se realiza una extracción de 128 bits con truncado y sin redondeo.



[0134] La siguiente tabla describe el significado de cada etiqueta:

etiqueta	bits	significado
fsize	8	tamaño del campo
dpos	8	posición de destino
x	1	reservado
s	1	con signo vs. sin signo
n	1	reservado
m	1	fusión vs. extracción
l	1	reservado
rnd	2	reservado
gssp	9	tamaño del grupo y posición de la fuente

[0135] El campo gssp de 9 bits codifica tanto el tamaño del grupo, **gsize**, como la posición de la fuente, **spos**, de acuerdo con la fórmula **gssp = 512 -4*gsize + spos**. El tamaño del grupo, **gsize**, es una potencia de dos comprendida en el intervalo de 1..128. La posición de la fuente, **spos**, está comprendida en el intervalo de **0..(2*gsize) - 1**.

[0136] Los valores en los campos **s, n, m, l** y **rnd** tienen el siguiente significado:

valores	s	n	m	l	rnd
0	sin signo		extracción		
1	con signo		fusión		
2					
3					

[0137] Tal como se muestra en la figura 23C para la instrucción X.EXTRACT, cuando, m = 0, se interpreta que los parámetros seleccionan campos de los contenidos concatenados de los registros rd y rc, extrayendo valores que están concatenados y colocados en el registro ra. Tal como se muestra en la figura 23D, para una fusión extracción de barras cruzadas (X.EXTRACT cuando m=1), se interpreta que los parámetros fusionan campos de los contenidos del registro rd con los contenidos del registro rc. Los resultados se concatenan y se colocan en el registro ra.

Galois de resolución amplia

[0138] En las figuras 24A a 24B se muestra una instrucción de Galois de resolución amplia. La figura 24A muestra un método y un aparato para resolver ecuaciones iterativamente. La operación particular descrita es un resolutor amplio para la clase de ecuaciones de congruencia polinómica de Galois L*S = W (mod z**2T), donde S, L y W son polinomios en un campo de Galois, tal como GF(256) de grado 2T, T + 1 y T respectivamente. La solución de este problema es una etapa computacional central de ciertos códigos de corrección de errores, tales como códigos de Reed-Solomon, que corrigen óptimamente hasta T errores que en un bloque de símbolos a efectos debe proporcionar un medio de almacenamiento o de comunicación digital más fiable. Se pueden obtener detalles adicionales de las matemáticas que sustentan esta implementación en el documento de Sarwate, Dilip V. y Shanbhag, Naresh R., "High-Speed Architectures for Reed-Solomon Decoders", revisado el 7 de junio de 2000, presentado a IEEE Trans VLSI Systems, accesible en http://icims.csl.uiuc.edu/~shanbhag/vips/publications/bma.pdf.

[0139] El aparato de la figura 24A contiene placas de memoria, multiplicadores de Galois, sumadores de Galois, multiplexores y circuitos de control que están ya contenidos en las realizaciones a modo de ejemplo a las que se hace referencia en la presente invención. Tal como se puede apreciar en la descripción de la instrucción de Galois de multiplicación de matriz ampliada, la etapa del resto polinómico asociada habitualmente con la multiplicación de Galois se puede desplazar a continuación de la suma de Galois sustituyendo las etapas de resto y suma con una etapa de suma polinómica y resto.

[0140] Este aparato lee y escribe, en las placas de memoria incorporadas, múltiples etapas de iteraciones sucesivas, tal como se especifica mediante el bloqueo de control de iteraciones dispuesto a la izquierda. Cada placa de memoria se carga inicialmente con el polinomio S, y cuando se han completado 2T iteraciones (en el ejemplo mostrado, T=4), la placa de memoria superior contiene los polinomios de solución deseados L y W. El bloque de código de la figura 24B describe detalles de la operación del aparato de la figura 24A, utilizando notación de lenguaje C.

[0141] Se pueden desarrollar códigos y aparatos similares para resolutores de ecuaciones iterativas de

multiplicación escalar-suma en otros dominios matemáticos, tales como enteros y números con coma flotante de varios tamaños, y para operandos matriciales de propiedades particulares, tales como matrices definidas positivas, o matrices simétricas, o matrices triangulares superiores o inferiores.

Rebanada de transformada amplia

[0142] En las figuras 25A a 25B se muestra una realización a modo de ejemplo de la instrucción de rebanada de transformada amplia. La figura 25A muestra un método y un aparato para un cálculo extremadamente rápido de transformadas, tales como la transformada de Fourier, que se requiere para comunicaciones en el dominio de frecuencias, análisis de imágenes, etc. En este aparato, se muestra una matriz 4 x 4 de 16 multiplicadores complejos, cada uno adyacente a una primera caché de operandos amplios. Una segunda caché de operando amplio o matriz de memorias de coeficientes incorporada suministra operandos que se multiplican por los multiplicadores con el acceso de datos de la caché incorporada amplia. Los productos resultantes se suministran a placas de transformadas atómicas - en esta realización preferida, unidades de mariposa de base-4 o base-2. Estas unidades reciben los productos desde una fila o columna de multiplicadores, y depositan resultados con un paso especificado e inversión digital, de nuevo en la primera caché de operandos amplios.

[0143] Un registro general ra contiene tanto la dirección del primer operando amplio como especificadores de tamaño y forma, y un segundo registro general rb contiene tanto la dirección del segundo operando amplio como especificadores de tamaño y forma.

[0144] Un registro general adicional rc especifica parámetros adicionales, tales como parámetros de precisión, de extracción de resultados (tal como en las diversas instrucciones de extracción descritas en el presente documento).

[0145] En una realización alternativa, el segundo operando de memoria puede estar situado junto al primer operando de memoria en una memoria ampliada, utilizando un direccionamiento de memoria distintivo para obtener cualquiera del primero o el segundo operandos de memoria.

[0146] En una realización alternativa, los resultados se depositan en una tercera memoria caché de operandos amplios. Este tercer operando de memoria se puede combinar con el primer operando de memoria, de nuevo utilizando direccionamiento de memoria distintivo. Renombrando las etiquetas de caché de operandos amplios, la tercera memoria puede alternar posiciones de almacenamiento con la primera memoria. De este modo, tras la finalización de la instrucción de rebanada de transformada amplia, se renombran las etiquetas de caché de operandos amplios de manera que el resultado aparece en una posición especificada por el primer operando de memoria. Esta alternancia permite la especificación de etapas de transformada no in situ, y permite que la operación se aborte y reinicie a continuación, si es necesario como resultado de la interrupción del flujo de ejecución.

[0147] El bloque de código de la figura 25B describe los detalles de la operación del aparato de la figura 25A, utilizando notación de lenguaje C. Se pueden desarrollar códigos y aparatos similares para otras transformadas y otros dominios matemáticos, tales como polinomios, campos de Galois, y números de varios tamaños enteros, y reales y complejos con coma flotante,

[0148] La instrucción de rebanada de transformada amplia calcula asimismo la posición del bit más significativo de todos los elementos resultantes, devolviendo dicho valor como un resultado escalar de la instrucción a colocar en el registro general rc. Éste es el mismo operando en el que se coloca la información de control de extracción y otra. Notablemente, está localización del bit más significativo se calcula mediante una serie de operaciones booleanas sobre subconjuntos paralelos de los elementos resultantes, que produce resultados booleanos vectoriales y, al término de la operación, mediante una reducción del vector de resultados booleanos a un valor booleano escalar, seguida por la determinación del bit más significativo del valor booleano escalar.

[0149] Añadiendo los valores que representan el control de extracción y otra información a esta posición del bit más significativo, se obtiene un parámetro de escala adecuado, para su utilización en la fase posterior de la instrucción de rebanada de transformada amplia. Acumulando la información del bit más significativo, se puede obtener un valor de escala global para toda la transformada, y los resultados de la transformada se mantienen con precisión adicional sobre el de los esquemas de escala fija de la técnica anterior.

Extracción convolución amplia

[0150] Las figuras 26A a 26K muestran una instrucción de extracción convolución amplia. Un método y un aparato similares se pueden aplicar al filtrado digital mediante métodos de convolución de 1-D o 2-D, o a la estimación del movimiento mediante el método de correlación 1-D o 2-D. La misma operación puede ser utilizada para correlación, dado que la correlación se puede calcular invirtiendo el orden del patrón 1-D o 2-D y realizando una convolución. Por lo tanto, la instrucción de convolución descrita en el presente documento puede ser utilizada para correlación, o se puede construir una instrucción de extracción correlación amplia que es similar a la instrucción de convolución descrita en el presente documento, excepto en que el orden del bloque de operandos de coeficientes se invierte en 1-D o 2-D.

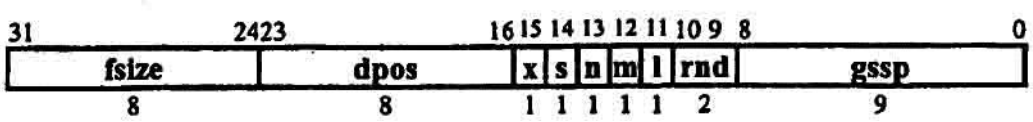
[0151] Los coeficientes del filtro digital o un bloque de plantilla de estimación se almacenan en una memoria de operandos amplios, y los datos de imagen se almacenan en una segunda memoria de operandos amplios. Una sola fila o columna de datos de imagen puede ser desplazada en la matriz de la imagen, con un desplazamiento correspondiente, de la relación de las posiciones de los datos de la imagen con respecto al bloque de plantilla y los multiplicadores. Mediante este método de actualizar parcialmente y desplazar los datos a la segunda memoria incorporada, la correlación de la plantilla frente a la imagen se puede calcular con un ancho de banda eficaz muy mejorado para la matriz multiplicadora. Se debe observar que en la presente realización, en lugar de desplazar la matriz, se utiliza direccionamiento circular, y se especifica una cantidad o posición inicial del desplazamiento, como parámetro de la instrucción.

[0152] Las figuras 26A y 26B muestran un formato y códigos de operación que pueden ser utilizados para realizar la instrucción de extracción convolución amplia. Tal como se muestra en las figuras 26A y 26B, en esta realización a modo de ejemplo, los contenidos de los registros generales rc y rd se utilizan como especificadores de operandos amplios. Estos especificadores determinan las direcciones virtuales, el tamaño de los operandos amplios y la forma para los operandos amplios. Utilizando las direcciones virtuales y los tamaños de operandos, un primer y un segundo valores del tamaño especificado se cargan desde la memoria. El tamaño del grupo y otros parámetros se especifican a partir de los contenidos del registro general rb. Los valores se dividen en grupos de operandos del tamaño y de la forma especificados y se convolucionan, produciendo un grupo de valores. El grupo de valores se redondea, y se limita según se especifica, produciendo un grupo de resultados que tiene el tamaño especificado. El grupo de resultados se concatena y se coloca en el registro general ra.

[0153] Las instrucciones de extracción convolución amplia (W.CONVOLVE.X.B, W.CONVOLVE.X.L) realizan una multiplicación matricial dividida de un tamaño máximo limitado solamente por la extensión de los operandos de memoria, no por el tamaño del camino de datos. La extensión, el tamaño y la forma de los parámetros de los operandos de memoria se especifican siempre como potencias de dos; parámetros adicionales pueden limitar la extensión de los operandos válidos dentro de la zona de una potencia de dos.

[0154] Tal como se muestra en la figura 26C, cada uno de los especificadores de operandos amplios especifica una extensión de un operando de memoria sumando la mitad de la extensión deseada del operando de memoria, en octetos, a los especificadores. Cada uno de los especificadores de operandos amplios especifica una forma de operando de memoria sumando un cuarto de la anchura deseada, en octetos, a los especificadores. Las alturas de cada uno de los operandos de memoria se pueden deducir dividiendo la extensión del operando por la anchura del operando. Los vectores unidimensionales se representan como matrices con una altura de uno y con anchura igual a la extensión. En una realización alternativa, algunas de las especificaciones del presente documento pueden estar incluidas como parte de la instrucción.

[0155] La instrucción de extracción convolución amplia permite calcular de varias maneras los bits a extraer del grupo de valores. Por ejemplo, los bits 31..0 de los contenidos del registro general rb especifican varios parámetros que controlan la manera en que son extraídos los datos. La posición y los valores por defecto de los campos de control permiten que la posición de la fuente se sume a un valor de control fijo para cálculo dinámico, y permiten que los 16 bits inferiores del campo de control se configuren para alguno de los casos más simples mediante una única instrucción GCOPYI, pudiendo estar incluidas algunas de las especificaciones del presente documento como parte de dicha instrucción.



[0156] La siguiente tabla describe el significado de cada etiqueta:

etiqueta	bits	significado
fsize	8	tamaño del campo
dpos	8	posición de destino
x	1	resultado extendido vs. tamaño de grupo
s	1	con signo vs. sin signo
n	1	multiplicación compleja vs. real
m	1	multiplicación con signo mezclado vs. mismo signo
l	1	saturación vs. truncado
rnd	2	redondeo
gssp	9	tamaño del grupo y posición de la fuente

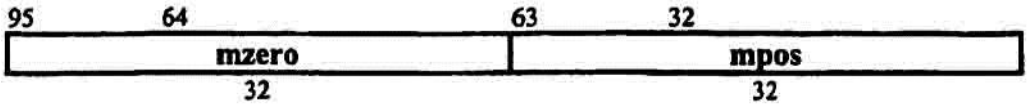
[0157] El campo gssp de 9 bits codifica tanto el tamaño del grupo, gsize, como la posición de la fuente, spos, de acuerdo con la fórmula gssp = 512 -4*gsize + spos. El tamaño del grupo, gsize, es una potencia de dos comprendida

en el intervalo de 1..128. La posición de la fuente, spos, está comprendida en el intervalo de 0..(2*gsize) - 1.

[0158] Los valores en los campos x, s, n, m, l y rnd tienen el significado siguiente:

valores	x	s	n	m	l	rnd
0	grupo	sin signo	real	mismo signo	truncado	F
1	extendido	con signo	complejo	signo mezclado	saturar	Z
2						N
3						C

[0159] Los bits 95..32 de los contenidos del registro general rb especifican varios parámetros que controlan la selección de divisiones de los operandos de memoria. La posición y los valores por defecto de los campos de control permiten que el campo de longitud de ceros del multiplicador valga cero por defecto, y que el cálculo del campo de posición del origen del multiplicando fluctúe sin desbordar ningún otro campo utilizando aritmética de 32 bits.



[0160] La siguiente tabla describe el significado de cada etiqueta:

etiqueta	bits	significado
mpos	32	posición de origen del multiplicando
mzero	32	longitud de ceros del multiplicador

[0161] El campo mpos de 32 bits codifica la posición tanto horizontal como vertical del origen del multiplicando, que es la posición del elemento multiplicando en que el elemento cero-ésimo del multiplicador se combina para producir el elemento 0-ésimo del resultado. Valores variables en este campo permiten calcular diversos resultados sin cambios en los dos operandos amplios. El campo mpos está a un desplazamiento de un octeto desde el comienzo del operando del multiplicando.

[0162] El campo mzero de 32 bits codifica una parte del operando multiplicador que tiene valor cero, y que se puede omitir del cálculo de multiplicación y de suma. Las implementaciones pueden utilizar un valor distinto de cero en este campo, para reducir el tiempo y/o la potencia para la realización de la instrucción, o pueden ignorar el contenido de este campo. La implementación puede presumir un valor cero para el operando de multiplicador en los bits dmsize-1..dmsize-(mzero*8), y saltarse la multiplicación de cualquier multiplicador obtenido en este intervalo de bits. El campo mzero está a un desplazamiento de un octeto desde el final del operando multiplicador.

[0163] Las direcciones virtuales de los operandos amplios tienen que estar alineadas, es decir, las direcciones de octeto tienen que ser un múltiplo exacto de la extensión del operando expresada en octetos. Si las direcciones no están alineadas, las direcciones virtuales no se pueden codificar en un especificador válido. Algunos especificadores no válidos provocan una extensión de "límite de operando".

[0164] El redondeo Z (cero) no se define para operaciones de extracción sin signo, de manera que se sustituye por el redondeo F (piso), que redondeará adecuadamente hacia abajo los resultados sin signo.

[0165] Una implementación puede limitar la extensión de los operandos debido a límites en la memoria de operandos o en la caché, o debido al número de valores que se pueden sumar con precisión, y provocar de ese modo una excepción ReservedInstruction.

[0166] Tal como se muestra en las figuras 26D y 26E, como un ejemplo con valores de registro específicos, una instrucción de extracción convolución amplia dobletes (W.CONVOLVE.X.B o W.CONVOLVE.X.L), con inicio en rb = 24, convoluciona el vector de memoria rc [c31 c30.. c1 c0] con el vector de memoria rd [d15 d14 ... d1 d0], proporcionando los productos [c16d15 + c17d14 + ... + c30d1 + c31d0 cl5d15 + c16dl4 + ... + c29dl + c30d0 ... c10d15 + c11d14 + ... + c24d1 + c25d0 c9d15 + c10d14 + ... + c23d1 + c24d0], redondeados y limitados tal como se especifica mediante los contenidos del registro general rb. Los valores c8...c0 no se utilizan en el cálculo y pueden ser cualquier valor.

[0167] Tal como se muestra en las figuras 26F y 26G, como un ejemplo con valores de registro específicos, una instrucción de extracción convolución amplia dobletes (W.CONVOLVE.X.L), con mpos en rb = 8 y **mzero** en rb = 48 (de manera que longitud = (512-mzero)*dmsize/512 = 13), convoluciona el vector de memoria rc [c31 c30.. c1 c0] con el vector de memoria rd [d15 d14 ... d1 d0], produciendo los productos [c3d12 + c4d11 + ... + c14d1 + c15d0 c2d12 + c3d11 + ... + c13d1 + c14d0 ... c29d12 + c30d11 + ... + c8d1 + c9d0 c28d12 + c29d11 + ... + c7d1 + c8d0], redondeados y limitados tal como se especifica. En este caso, la posición de inicio está situada de manera que el

rango útil de valores fluctúa por debajo de c0, hasta c31...28. Los valores c27...c16 no se utilizan en el cálculo y pueden ser cualquier valor. El parámetro de longitud está fijado en 13, de manera que los valores d15...d13 tienen que ser cero.

[0168] Tal como se muestra en las figuras 26H y 26I, como un ejemplo con valores de registro específicos, una instrucción de extracción convolución amplia dobletes bidimensional (W.CONVOLVE.X.B ó W.CONVOLVE.X.L), con mpos en rb = 24 y vsize en rc y rd = 4, convoluciona el vector de memoria rc [c127 c126 ... c31 c30 ... c1 c0] con el vector de memoria rd [d63 d62 ... d15 d14 ... d1 d0], produciendo los productos [c113d63 + c112d62 + ... + c16d15 + c17d14 + ... + c30d1 + c31d0 c112d63 + c111d62 + ... + c15d15 + c16d14 + ... + c29d1 + c30d0 ... c107d63 + c106d62 + ... + c10d15 + c11d14 + ... + c24d1 + c25d0 c106d63 + c105d62 + ... + c9d15 + c10d14 + ... + c23d1 + c24d0], redondeados y limitados tal como se especifica por los contenidos del registro general rb.

[0169] Tal como se muestra en las figuras 26J y 26K, con un ejemplo con valores de registro específicos, una instrucción de extracción convolución amplia dobletes complejos (W.CONVOLVE.X.B o W.CONVOLVE.X.L con n configurado en rb), con **mpos** en rb = 12, convoluciona el vector de memoria rc [c15 c14.. c1 c0] con el vector de memoria rd [d7 d6 ... d1 d0], produciendo los productos [c8d7 + c9d6 + ... + c16d1 + c15d0 c7d7 + c8d6 + ... + c13d1 + c14d0 c6d7 + c7d6 + ... + c12d1 + c13d0 c5d7 + c6d6 + ... + c11d1 + c12d0], redondeados y limitados tal como se especifica mediante los contenidos del registro general rb.

Coma flotante de convolución amplia

[0170] Una inserción de coma flotante de convolución amplia funciona de manera similar a la instrucción de extracción convolución amplia descrita anteriormente, excepto en que las multiplicaciones y sumas de los operandos se llevan a cabo utilizando aritmética de coma flotante. La representación de los productos de multiplicación y de las sumas intermedias en una realización a modo de ejemplo se lleva a cabo sin redondeo con precisión esencialmente no limitada, con los resultados finales sujetos a un redondeo simple a la precisión del operando de resultado. En una realización alternativa, los productos y las sumas se calculan con precisión extendida, pero limitada. En otra realización alternativa, los productos y las sumas se calculan con precisión limitada al tamaño de los operandos.

[0171] La instrucción de coma flotante de convolución amplia en una realización a modo de ejemplo puede utilizar el mismo formato para los campos de registro general rb en la instrucción de extracción convolución amplia, excepto para campos que no son aplicables a la aritmética de coma flotante. Por ejemplo, los campos **fsize, dpos, s, m** y **l**, y el parámetro **spos** del campo **gssp** se pueden ignorar para esta instrucción. En una realización alternativa, parte de la información restante se puede especificar dentro de la instrucción, tal como el parámetro gsize o el parámetro n, o se pueden fijar los valores especificados, de manera que el parámetro de redondeo se puede fijar a redondeo al más próximo. En una realización alternativa, los campos restantes se pueden reordenar, por ejemplo, si todos menos el campo mpos están contenidos dentro de la instrucción o se ignoran, el campo mpos solamente por estar contenido en la parte menos significativa de los contenidos del registro general rb.

Descodificación amplia

[0172] Otra categoría de operaciones amplias mejoradas es útil para corrección de errores por medio de descodificación de Viterbi o turbo. En este caso, las placas de memoria incorporada se utilizan para contener métricas de estado y dígitos de decisión pre-rastreo. Una matriz de unidades de suma-comparación-intercambio o log-MAP reciben un pequeño número de métricas de ramificación, tal como 128 bits procedentes desde un registro externo en la realización preferida por los inventores. A continuación esta matriz lee, recalcula y actualiza las entradas de memoria de métricas de estado que, para muchos códigos prácticos, son mucho mayores. Una serie de dígitos de decisión, habitualmente de 4 bits cada uno con un método de pre-rastreo en base 16, se acumulan en una segunda memoria de rastreo. Los cálculos de la matriz y las actualizaciones de la métrica de estado se llevan a cabo de manera interactiva durante una serie especificada de ciclos. A continuación, una segunda operación iterativa recorre la memoria de rastreo para resolver el camino más probable a través de la malla de estados.

Booleana amplia

[0173] Otra categoría de operaciones amplias mejoradas son las operaciones booleanas amplias, que involucran un conjunto de tablas de consulta pequeñas (LUTs, small look up tables), habitualmente con 8 ó 16 entradas, cada una especificada mediante 3 ó 4 bits de dirección de entrada, interconectadas con multiplexores y cerrojos próximos. El control de las entradas LUT, selecciones de multiplexores y activaciones de relojes de cerrojo está especificado mediante una memoria caché amplia incorporada. Esta estructura proporciona un medio para disponer una placa de matriz de puertas programable in situ, que realiza operaciones iterativas sobre operandos proporcionados por los registros de un microprocesador de propósito general. Estas operaciones pueden iterar sobre múltiples ciclos, llevando a cabo operaciones lógicas especificables aleatoriamente que actualizan tanto los cerrojos internos como la propia placa de memoria.

Transferencias entre memorias de operandos amplios

[0174] El método y el aparato descritos en el presente documento son ampliamente aplicables al problema de aumentar el ancho de banda eficaz de unidades funcionales de microprocesador hasta aproximadamente lo que se alcanza en los circuitos integrados de aplicación específica (ASICs, application-specific integrated circuits). Cuando están presentes al mismo tiempo dos o más unidades funcionales que pueden manejar operandos amplios, se produce el problema de transferir datos desde una unidad funcional que los está produciendo en memoria incorporada, y a través del sistema de memoria o en torno al mismo, hasta una segunda unidad funcional que puede asimismo manejar operandos amplios, que tiene que consumir dichos datos después de cargarlos en su memoria de operandos amplios. Copiar explícitamente los datos desde una localización de memoria a la otra conseguiría dicha transferencia, pero la sobrecarga involucrada reduciría la eficacia del procesador en general.

[0175] La figura 27 describe un método y un aparato para resolver este problema de transferencia entre dos o más unidades, con sobrecarga reducida. Las matrices de memoria incorporada funcionan como cachés que retienen copias locales de datos que están presentes conceptualmente en un único espacio global de memoria. Un controlador de coherencia de cachés monitoriza los flujos de direcciones de las actividades de las cachés, y utiliza uno de los protocolos de coherencia, tal como MOESI o MESI, para mantener la consistencia con un estándar especificado. Mediante la iniciación adecuada del controlador de coherencia de cachés, un software que se ejecuta en el microprocesador de propósito general puede permitir que se produzca en segundo plano una transferencia de datos entre unidades amplias, solapada con el cálculo en las unidades amplias, reduciendo la sobrecarga de cargas explícitas y almacenamientos.

Conclusión

[0176] Los expertos en la materia reconocerán, dadas las explicaciones del presente documento, que existen numerosas alternativas que no se salen de la invención. Por lo tanto, se prevé que la invención no está limitada por la descripción anterior, sino solamente por las reivindicaciones adjuntas.

REIVINDICACIONES

1. Un procesador, que comprende:

un primer camino de datos (137, 714 y 715) que tiene una primera anchura de bits;

un segundo camino de datos (entre 136 y 149; 714 y 720n) que tiene una segunda anchura de bits mayor que la primera anchura de bits;

una serie de terceros caminos de datos (155 a 158; 720A y 740) que tienen una anchura de bits combinada menor que la segunda anchura de bits;

un primer almacenamiento de operando amplio (136, 714) acoplado al primer camino de datos y al segundo camino de datos para almacenar un primer operando amplio recibido sobre el primer camino de datos, teniendo el primer operando amplio un tamaño con un número de bits mayor que la primera anchura de bits;

un segundo almacenamiento de operando amplio (134, 714) acoplado al primer camino de datos y al segundo camino de datos para almacenar un segundo operando amplio recibido sobre el primer camino de datos, teniendo el segundo operando amplio un tamaño con un número de bits mayor que la primera anchura de bits;

un archivo de registros (720A a 720n) que incluye registros que tienen la primera anchura de bits, estando conectado el archivo de registros al primer camino de datos y a los terceros caminos de datos, e incluyendo almacenamiento (710) para un primer especificador de operando amplio que especifica una dirección de primer operando amplio y un segundo especificador de operando amplio que especifica una dirección del segundo operando amplio;

una unidad funcional (149, 740) que puede iniciar instrucciones, la unidad funcional estando acoplada mediante el segundo camino de datos al primer almacenamiento de operando amplio y estando acoplada mediante los terceros caminos de datos al archivo de registros; y

en el que la unidad funcional está dispuesta para ejecutar una instrucción de rebanada de transformada de Fourier amplia que contiene campos de instrucción que especifican (i) un primer registro de operando amplio (710) para provocar la recuperación del primer operando amplio para su almacenamiento en el primer almacenamiento de operando amplio, (ii) un segundo registro de operando amplio (710) para provocar la recuperación del segundo operando amplio para su almacenamiento en el segundo almacenamiento de operando amplio, (iii) por lo menos un registro de operando de control en forma de un registro general en el archivo de registros que almacena un operando de control y especifica parámetros de control de precisión y de extracción, y (iv) un registro de resultados (745) en el archivo de registros, y

en el que la instrucción de rebanada de transformada de Fourier amplia está configurada para provocar que la unidad funcional (i) tome el primer operando amplio del primer almacenamiento de operando amplio, (ii) tome el segundo operando amplio del segundo almacenamiento de operando amplio, (iii) lleve a cabo multiplicaciones complejas paralelas de subconjuntos de elementos de datos del primer operando amplio, por subconjuntos de elementos de datos del segundo operando amplio para producir primeros resultados, (iv) lleve a cabo operaciones de mariposa paralelas y operaciones de extracción paralelas sobre subconjuntos de los primeros resultados para proporcionar de ese modo segundos resultados;

en el que el resultado escalar de la instrucción de rebanada de transformada de Fourier amplia contiene información a partir de la cual se puede determinar la posición del bit más significativo de los segundos resultados;

en el que la posición del bit más significativo del segundo resultado es:

calculada mediante una serie de operaciones booleanas sobre subconjuntos paralelos de los segundos resultados, produciendo resultados booleanos vectoriales, y reduciendo además los resultados booleanos vectoriales a un valor booleano escalar, seguido por una determinación de la posición del bit más significativo del valor booleano escalar; y

colocado en el registro general para proporcionar un parámetro de escala para su utilización en la fase subsiguiente de la instrucción de rebanada de transformada de Fourier amplia.

2. Un procesador acorde con la reivindicación 1, en el que cuando se lleva a cabo una operación posterior de especificación de un operando amplio, el procesador determina si el operando amplio está ya almacenado en el almacenamiento de operando amplio y, en caso afirmativo, el procesador reutiliza el operando amplio de almacenamiento de operando amplio en la operación posterior.

3. Un procesador acorde con cualquiera de las reivindicaciones anteriores, en el que cuando se ejecuta una única instrucción que contiene campos de instrucción que especifican un registro de operando amplio, el

procesador hace referencia a un único registro que especifica tanto la dirección como el tamaño del operando amplio.

4. Un procesador acorde con cualquiera de las reivindicaciones anteriores, en el que la unidad funcional es operativa asimismo para ejecutar una única instrucción booleana amplia que contiene campos de instrucción que especifican (i) un registro de operando amplio (710) para provocar la recuperación del operando amplio para su almacenamiento en el almacenamiento de operando amplio, (ii) por lo menos un registro de operando fuente en el archivo de registros que almacena un operando fuente, y (iii) un registro de resultados (745) en el archivo de registros, provocando la única instrucción que la unidad funcional lleve a cabo operaciones que involucran un conjunto de tablas de consulta interconectadas con multiplexores y cerrojos, en el que los contenidos de las tablas de consulta y el control de los multiplexores y de los cerrojos se especifican mediante información en el almacenamiento de operando amplio, provocando de ese modo que una placa de una matriz de puertas programables in situ realice operaciones sobre dicho por lo menos un operando fuente, produciendo resultados a colocar en el registro de resultados.

5. Un procesador acorde con la reivindicación 4, en el que la unidad funcional lleva a cabo operaciones iterativas sobre múltiples ciclos que actualizan cerrojos internos representados por información en el almacenamiento de operando amplio.

6. Un procesador acorde con cualquiera de las reivindicaciones anteriores, en el que la unidad funcional es operativa asimismo para ejecutar una única instrucción de Galois de resolución amplia que especifica un registro de operando amplio para provocar la recuperación del operando amplio para su almacenamiento en el almacenamiento de operando amplio, realizando la unidad funcional operaciones iterativas de multiplicación-suma sobre elementos concatenados del operando amplio contenido en el almacenamiento de operando amplio a efectos de resolver un sistema de ecuaciones, produciendo un resultado que tiene una anchura de bits mayor que la primera anchura de bits, para su almacenamiento en el almacenamiento de operando amplio.

7. Un procesador acorde con la reivindicación 6, en el que los elementos concatenados comprenden valores de campos de Galois y las operaciones de multiplicación-suma son operaciones de multiplicación-suma de campos de Galois.

8. Un procesador acorde con la reivindicación 6 ó 7, en el que los elementos concatenados comprenden operandos enteros y las operaciones de multiplicación-suma son operaciones de multiplicación-suma de enteros.

9. Un procesador acorde con la reivindicación 6, 7 ó 8, en el que los elementos concatenados comprenden valores con coma flotante y las operaciones de multiplicación-suma son operaciones de multiplicación-suma con coma flotante.

10. Un procesador acorde con cualquiera de las reivindicaciones 6 a 9, en el que los elementos concatenados comprenden una matriz definida positiva.

11. Un operador acorde con cualquiera de las reivindicaciones 6 a 10, en el que los elementos concatenados comprenden una matriz simétrica.

12. Un procesador acorde con cualquiera de las reivindicaciones 6 a 11, en el que los elementos concatenados comprenden una matriz triangular superior o una matriz triangular inferior.

13. Un procesador acorde con cualquiera de las reivindicaciones 1 a 5, en el que la unidad funcional puede asimismo iniciar una instrucción de Galois de resolución amplia y en el que el operando amplio contiene métricas de estado de corrección de errores que se actualizan iterativamente y se recorren a continuación para resolver un camino más probable, y el camino más probable es un resultado de la instrucción.

14. Un procesador acorde con la reivindicación 13, en el que el camino más probable es un resultado devuelto a un registro en el archivo de registros.

15. Un procesador acorde con la reivindicación 13 ó 14, en el que la instrucción produce métricas de estado actualizadas.

16. Un procesador acorde con cualquiera de las reivindicaciones anteriores, en el que la unidad funcional puede asimismo ejecutar una instrucción de descodificación amplia para llevar a cabo corrección de errores por medio de descodificación Viterbi o turbo especificando (i) un registro del archivo de registros que proporciona una serie de métricas de ramificación de corrección de errores; (ii) un registro que contiene un especificador de operando amplio que especifica un operando amplio que contiene métricas de estado de corrección de errores, en el que las métricas de estado se actualizan iterativamente utilizando la serie de métricas de ramificación, y las métricas de estado se recorren a continuación para resolver un camino más probable, como resultado de la instrucción.

17. Un procesador acorde con la reivindicación 16, en el que el camino más probable es un resultado devuelto a un registro en el archivo de registros.

18. Un procesador acorde con la reivindicación 16 ó 17, en el que la instrucción produce métricas de estado actualizadas del operando amplio para su almacenamiento en el almacenamiento de operando amplio.

19. Un procesador acorde con la reivindicación 1, en el que la parte extraída de los segundos resultados se coloca en el registro de resultados.

20. Un procesador acorde con la reivindicación 1, en el que la parte extraída de los segundos resultados se coloca en un registro de operando amplio.

21. Un procesador acorde con la reivindicación 1, en el que los segundos resultados se almacenan en el primer almacenamiento de operando amplio o en un tercer almacenamiento de operando amplio.

22. Un procesador acorde con la reivindicación 1, en el que el resultado escalar se coloca en un registro o en uno de dicho por lo menos un registro de operando de control en el archivo de registros.

23. Un sistema de procesamiento de datos que comprende:

un procesador acorde con cualquiera de las reivindicaciones anteriores, siendo el procesador un solo circuito integrado;

la memoria principal (120) externa a dicho único circuito integrado; y

un bus acoplado a la memoria principal, incluyendo el procesador una interfaz de bus que acopla el procesador al bus.

24. Un método de funcionamiento de un aparato que incluye un primer camino de datos que tiene una primera anchura de bits, un segundo camino de datos que tiene una segunda anchura de bits mayor que la primera anchura de bits, una serie de terceros caminos de datos que tienen una anchura de bits combinada menor que la segunda anchura de bits, un primer almacenamiento de operando amplio acoplado al primer camino de datos y al segundo camino de datos para almacenar un primer operando amplio recibido sobre el primer camino de datos, teniendo el primer operando amplio un tamaño con un número de bits mayor que la primera anchura de bits, un segundo almacenamiento de operando amplio acoplado al primer camino de datos y al segundo camino de datos para almacenar un segundo operando amplio recibido sobre el primer camino de datos, teniendo el segundo operando amplio un tamaño con un número de bits mayor que la primera anchura de bits, un archivo de registros incluyendo registros que tienen la primera anchura de bits, estando conectado el archivo de registros al primer camino de datos y a los terceros caminos de datos, e incluyendo un primer almacenamiento para un primer especificador de operando amplio que especifica la dirección del primer operando amplio, e incluyendo un segundo almacenamiento para un segundo especificador de operando amplio que especifica una dirección del segundo operando amplio, y una unidad funcional que puede iniciar instrucciones, estando la unidad funcional acoplada mediante el segundo camino de datos al primer almacenamiento de operando amplio y acoplada mediante los terceros caminos de datos al archivo de registros, comprendiendo el método:

ejecutar una sola instrucción de rebanada de transformada de Fourier amplia que contiene campos de instrucción que especifican (i) un primer registro de operando amplio (710) para provocar la recuperación del primer operando amplio para su almacenamiento en el primer almacenamiento de operando amplio, (ii) un segundo registro de operando amplio (710) para provocar la recuperación del segundo operando amplio para su almacenamiento en el segundo almacenamiento de operando amplio, (iii) por lo menos un registro de operando de control en la forma de un registro general en el archivo de registros que almacena un operando de control y especifica parámetros de control de precisión y de extracción, y (iv) un registro de resultados (745) en el archivo de registros, y

en el que la instrucción de rebanada de transformada de Fourier amplia provoca que la unidad funcional (i) tome el primer operando amplio del primer almacenamiento de operando amplio, (ii) tome el segundo operando amplio del segundo almacenamiento de operando amplio, (iii) lleve a cabo multiplicaciones complejas paralelas de subconjuntos de elementos de datos del primer operando amplio por subconjuntos de elementos de datos del segundo operando amplio para producir primeros resultados, (iv) lleve a cabo operaciones de mariposa paralelas y operaciones de extracción paralelas sobre subconjuntos de los primeros resultados para proporcionar de ese modo segundos resultados;

en el que el resultado escalar de la instrucción de rebanada de transformada de Fourier amplia contiene información a partir de la cual se puede determinar la posición del bit más significativo de los segundos resultados;

en el que la posición del bit más significativo del segundo resultado es:

calculada mediante una serie de operaciones booleanas sobre subconjuntos paralelos de los segundos resultados, produciendo resultados booleanos vectoriales, y reduciendo además los resultados booleanos vectoriales a un valor booleano escalar, seguido por una determinación de la posición del bit más significativo del valor booleano escalar; y

colocado en el registro general para proporcionar un parámetro de escala para su utilización en la subsiguiente instrucción de rebanada de transformada de Fourier amplia.

25. Un método acorde con la reivindicación 24, en el que cuando se lleva a cabo una operación posterior de especificación de un operando amplio, el método comprende además:

determinar si el operando amplio está ya almacenado en el almacenamiento de operando amplio; y

si el operando amplio está ya almacenado dentro del almacenamiento de operando amplio, reutilizar entonces el operando amplio del almacenamiento de operando amplio en la operación posterior.

26. Un método acorde con la reivindicación 24 ó 25, en el que la etapa de ejecutar una única instrucción que contiene campos de instrucción que especifican un registro de operando amplio comprende además hacer referencia a un único registro que especifica tanto la dirección como el tamaño del operando amplio.

27. Un método acorde con la reivindicación 24, que comprende además almacenar los segundos resultados en el primer almacenamiento de operando amplio o en un tercer almacenamiento de operando amplio.

28. Un método acorde con la reivindicación 24, en el que se puede determinar el resultado escalar para la determinación del bit más significativo del mayor de los terceros resultados.

29. Un procesador acorde con la reivindicación 24, en el que el resultado escalar se coloca en un registro o en uno de dicho por lo menos un registro de operando de control en el archivo de registros.

30. Un medio legible por ordenador que tiene código legible por ordenador en el mismo, para provocar que un procesador lleve a cabo el método acorde con cualquiera de las reivindicaciones 1 a 23.

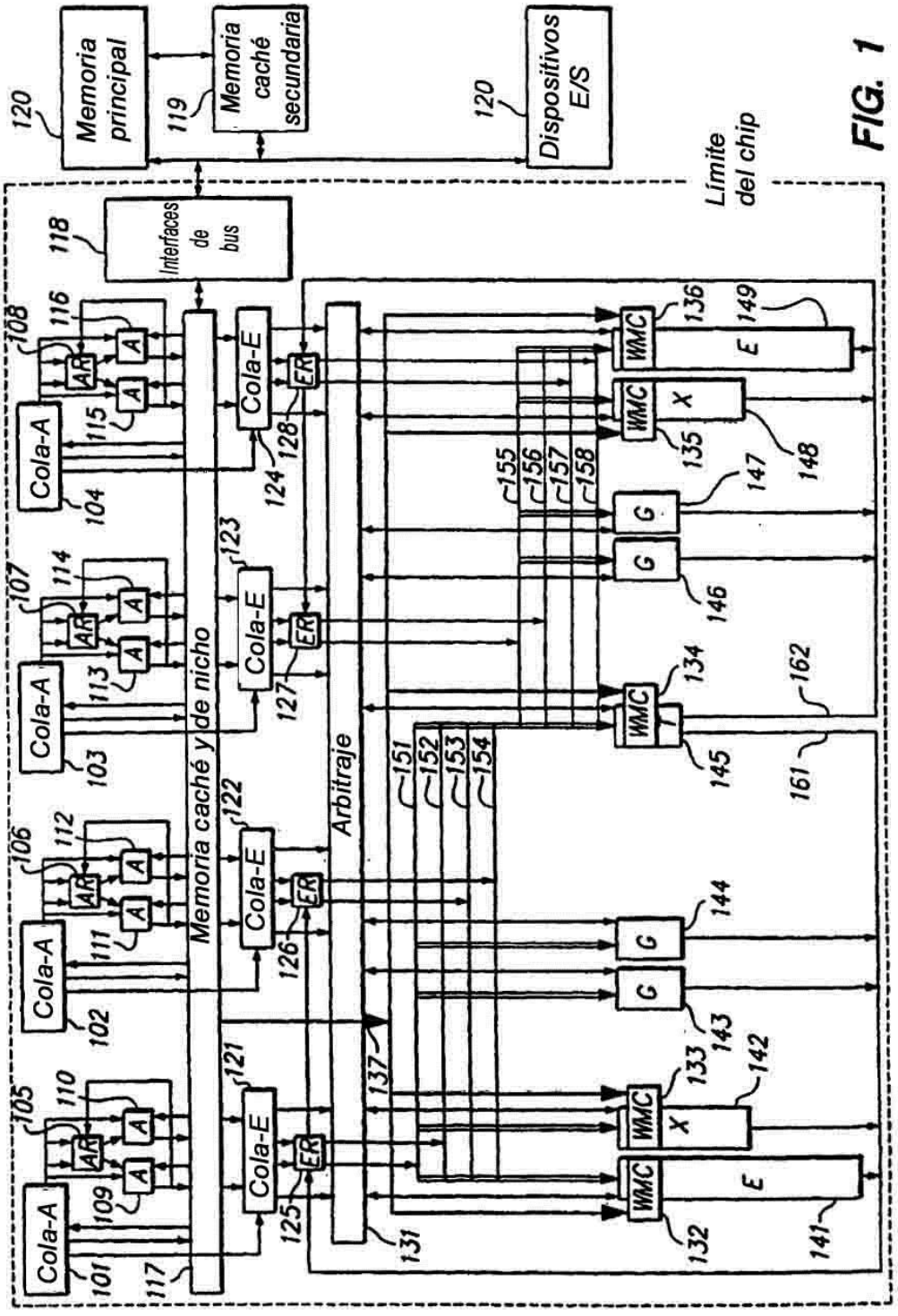

Memoria principal
Memoria caché secundaria
Dispositivos E/S
Límite del chip
Interfaces de bus
Memoria caché y de nicho
Arbitraje
Cola-A
Cola-E
AR
A
ER
G
WMC
X
E

FIG. 1

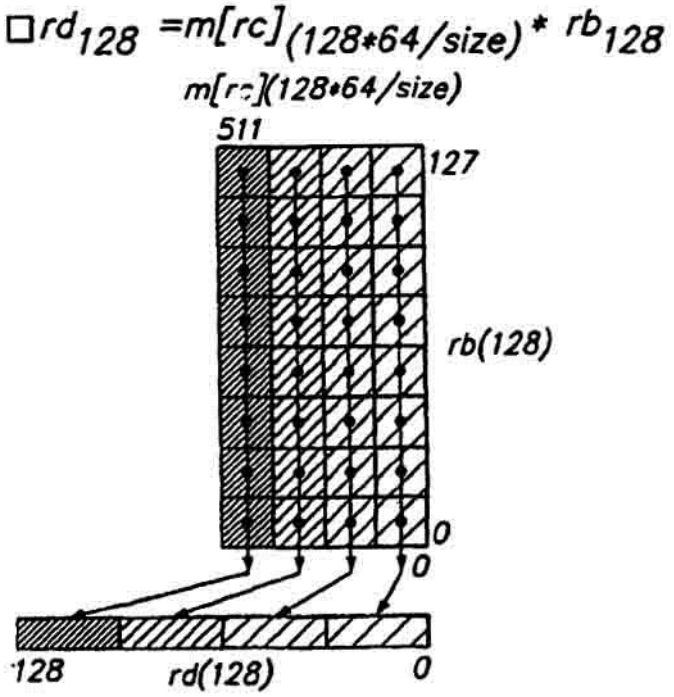

□rd128 =m[rc](128*64/size) * rb128
m[rc](128*64/size)
511
127
rb(128)
0
0
128
rd(128)
0

FIG. 2

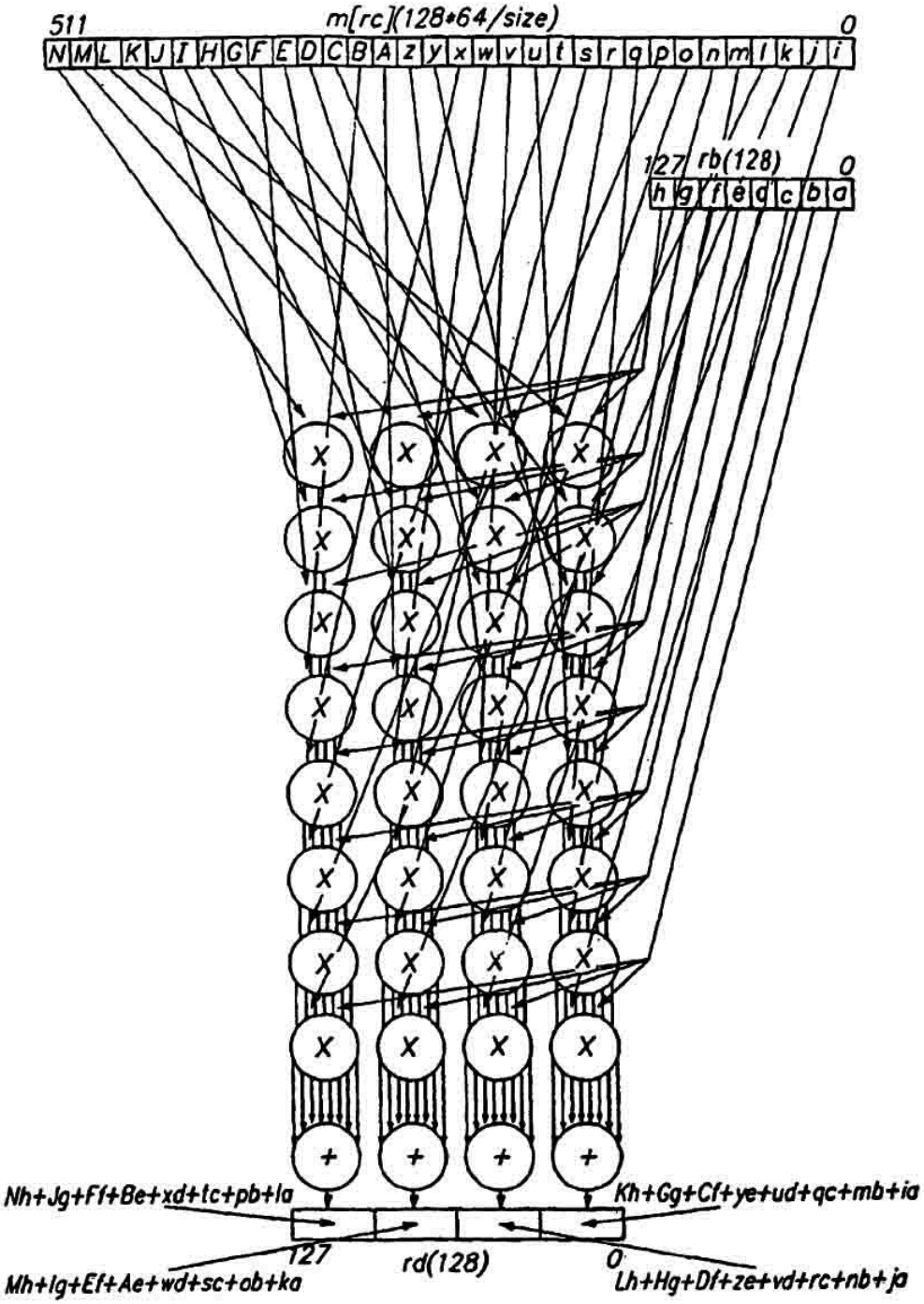

511
m[rc](128+64/size)
0
N M L K J I H G F E D C B A z y x w v u t s r q p o n m l k j i
127
rb(128)
0
h g f e d c b a
Nh+Jg+Ff+Be+xd+tc+pb+la
Kh+Gg+Cf+ye+ud+qc+mb+ia
127
rd(128)
0
Mh+lg+Ef+Ae+wd+sc+ob+ka
Lh+Hg+Df+ze+vd+rc+nb+ja

FIG. 3

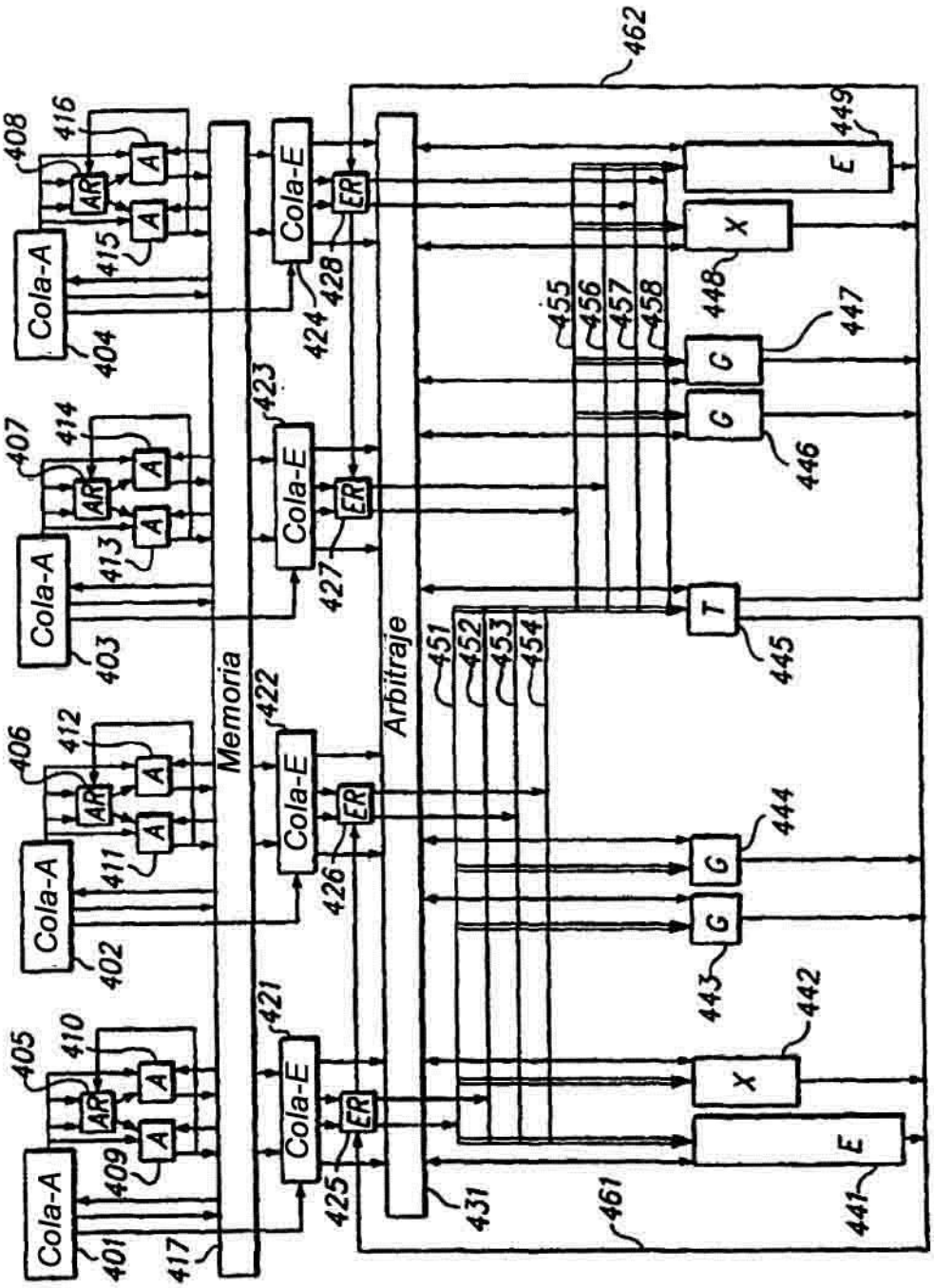

Cola-A
Cola-E
Memoria
Arbitraje
AR
A
ER
E
X
G
T
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
421
422
423
424
425
426
427
428
431
441
442
443
444
445
446
447
448
449
451
452
453
454
455
456
457
458
461
462

FIG. 4

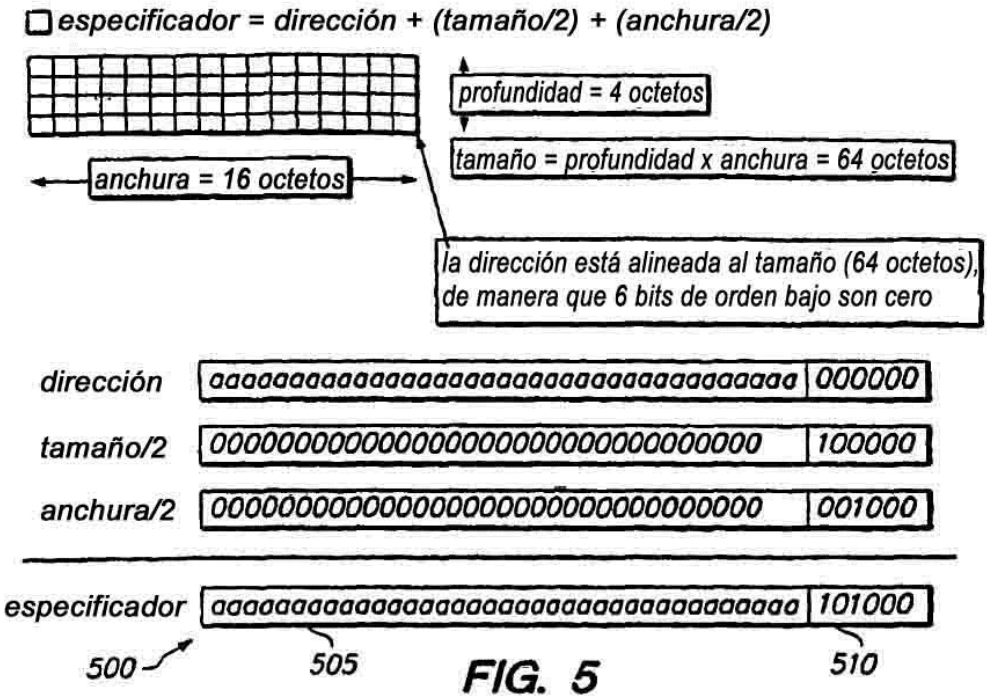

especificador = dirección + (tamaño/2) + (anchura/2)
profundidad = 4 octetos
tamaño = profundidad x anchura = 64 octetos
anchura = 16 octetos
la dirección está alineada al tamaño (64 octetos), de manera que 6 bits de orden bajo son cero
dirección
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
000000
tamaño/2
00000000000000000000000000000000
100000
anchura/2
00000000000000000000000000000000
001000
especificador
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
101000
500
505
510

FIG. 5

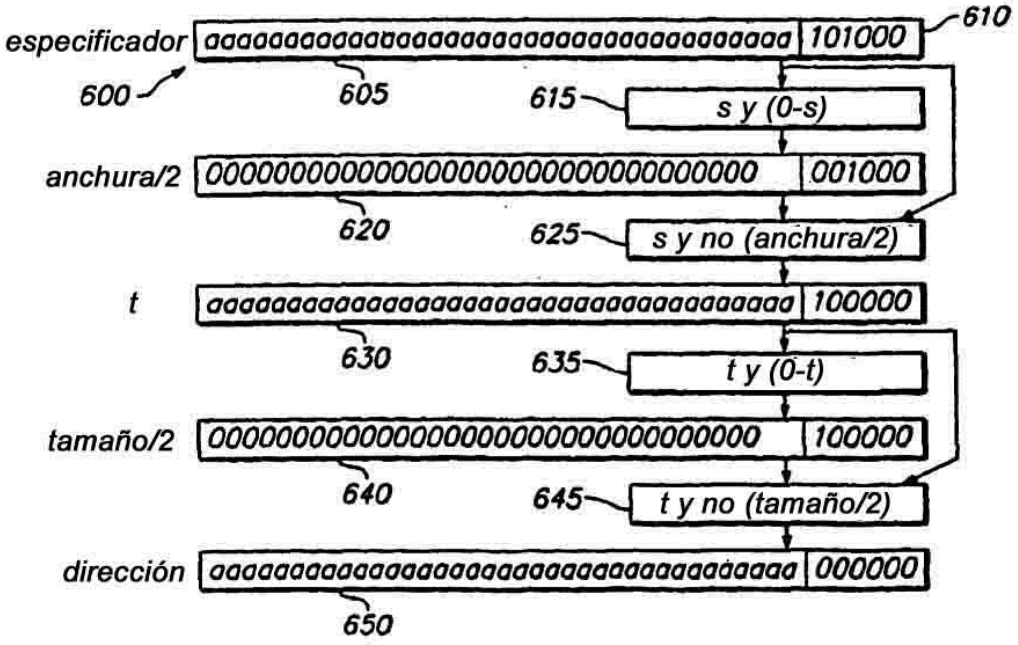

especificador
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
101000
610
600
605
615
s y (0-s)
anchura/2
00000000000000000000000000000000
001000
620
625
s y no (anchura/2)
t
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
100000
630
635
t y (0-t)
tamaño/2
00000000000000000000000000000000
100000
640
645
t y no (tamaño/2)
dirección
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
000000
650

FIG. 6

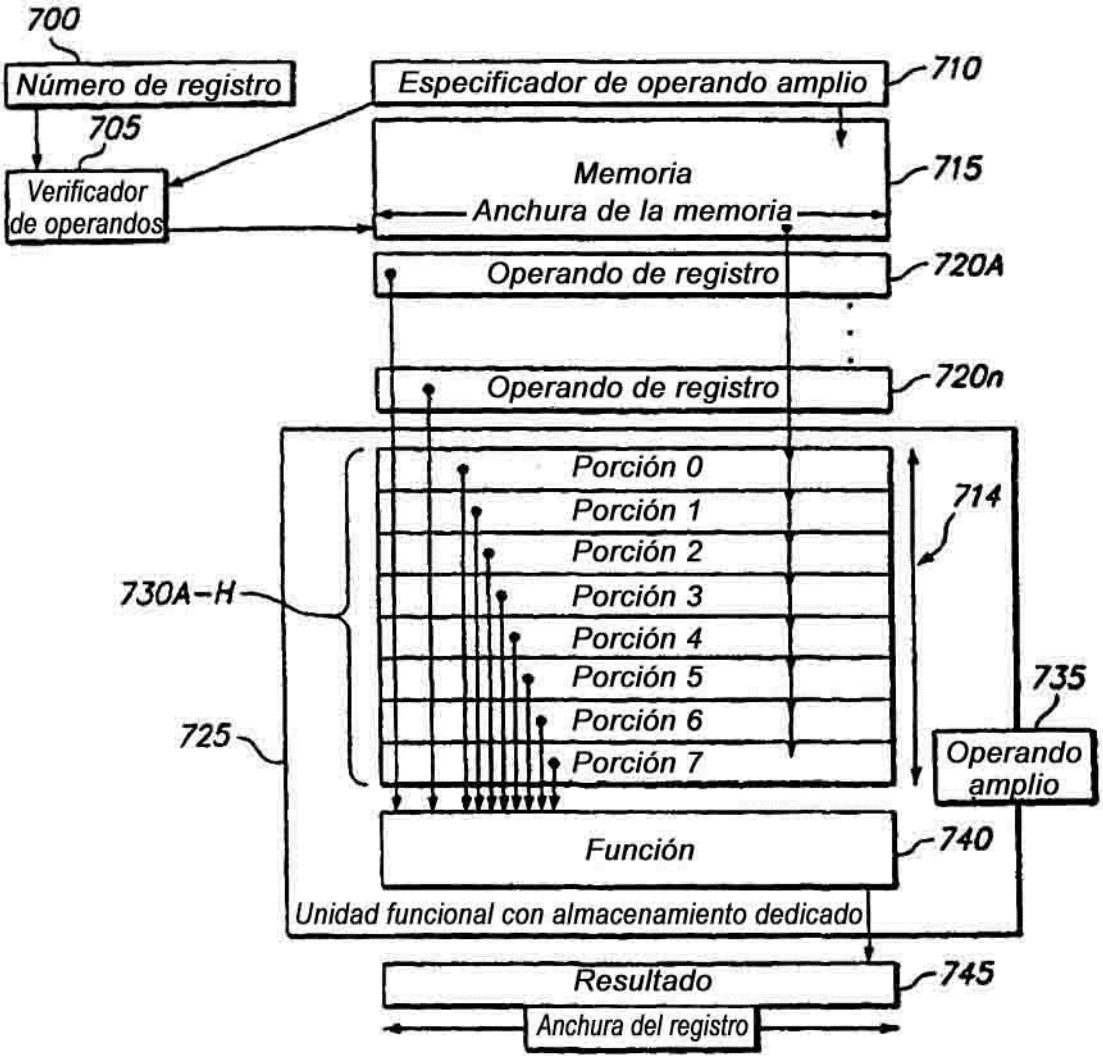

700
Número de registro
705
Verificador de operandos
Especificador de operando amplio
710
Memoria
Anchura de la memoria
715
Operando de registro
720A
Operando de registro
720n
Porción 0
Porción 1
Porción 2
Porción 3
Porción 4
Porción 5
Porción 6
Porción 7
714
730A-H
725
735
Operando amplio
Función
740
Unidad funcional con almacenamiento dedicado
Resultado
745
Anchura del registro

FIG. 7

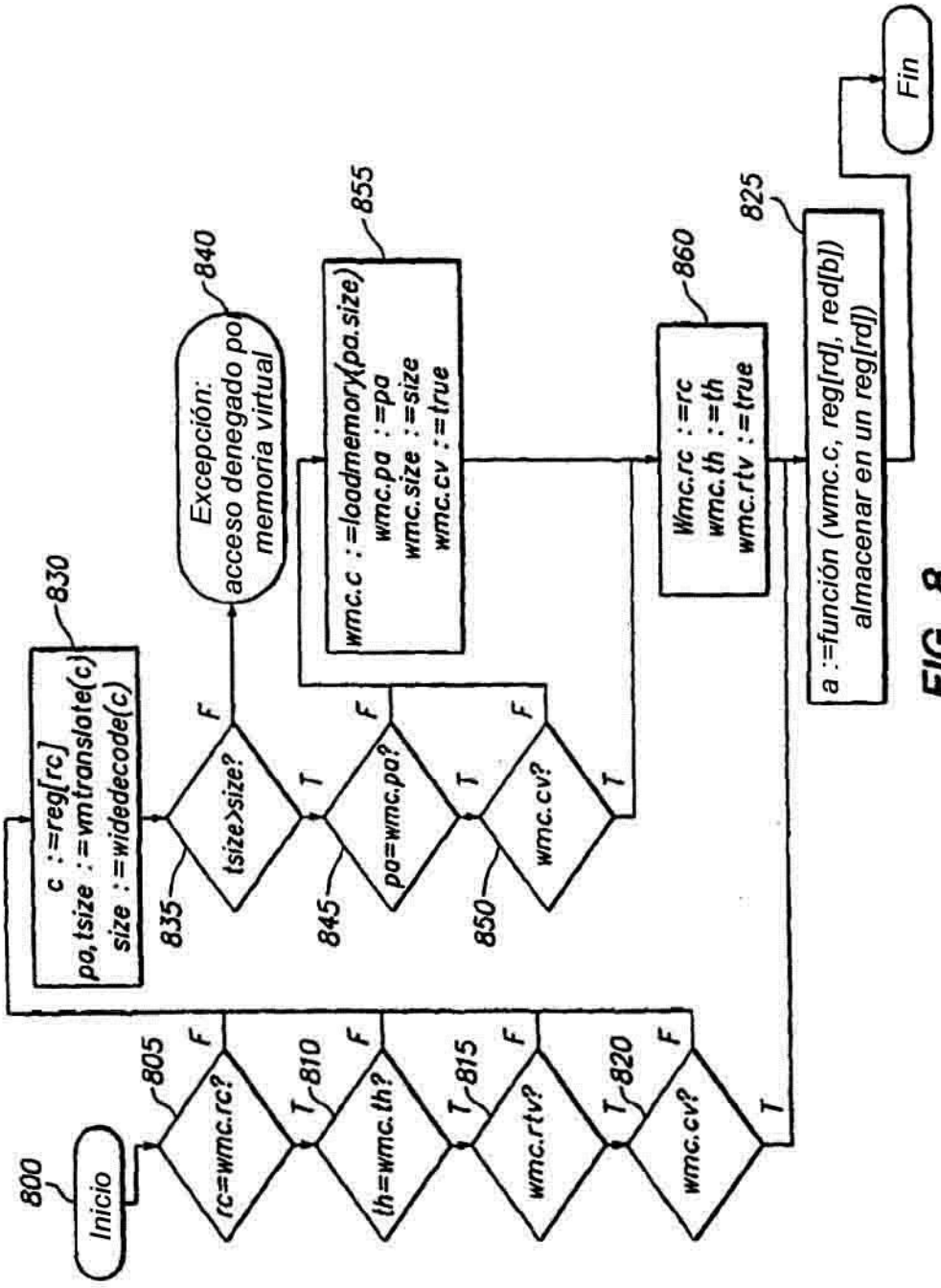

800
Inicio
805
rc=wmc.rc?
810
th=wmc.th?
815
wmc.rtv?
820
wmc.cv?
825
a :=función (wmc.c, reg[rd], red[b])
almacenar en un reg[rd]
830
c :=reg[rc]
pa, tsize :=vmtranslate(c)
size :=widedecode(c)
835
tsize>size?
840
Excepción:
acceso denegado po
memoria virtual
845
pa=wmc.pa?
850
wmc.cv?
855
wmc.c :=loadmemory(pa.size)
wmc.pa :=pa
wmc.size :=size
wmc.cv :=true
860
Wmc.rc :=rc
wmc.th :=th
wmc.rtv :=true
Fin
F
T

FIG. 8

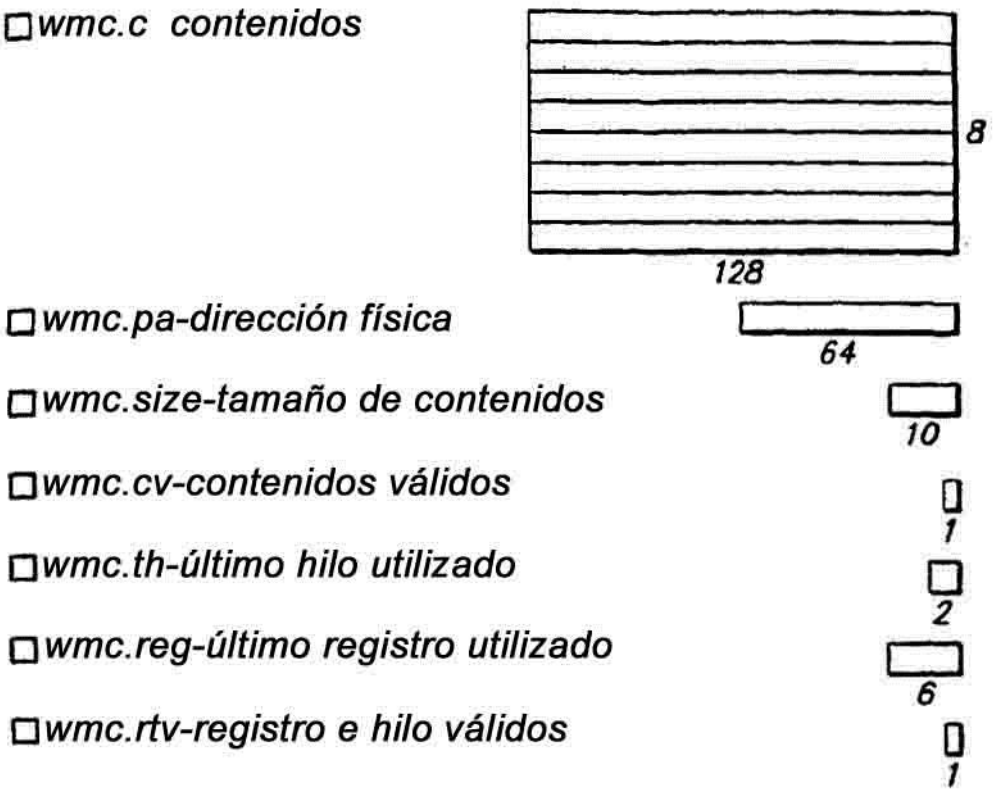

wmc.c contenidos
8
128
wmc.pa-dirección física
64
wmc.size-tamaño de contenidos
10
wmc.cv-contenidos válidos
1
wmc.th-último hilo utilizado
2
wmc.reg-último registro utilizado
6
wmc.rtv-registro e hilo válidos
1

FIG. 9

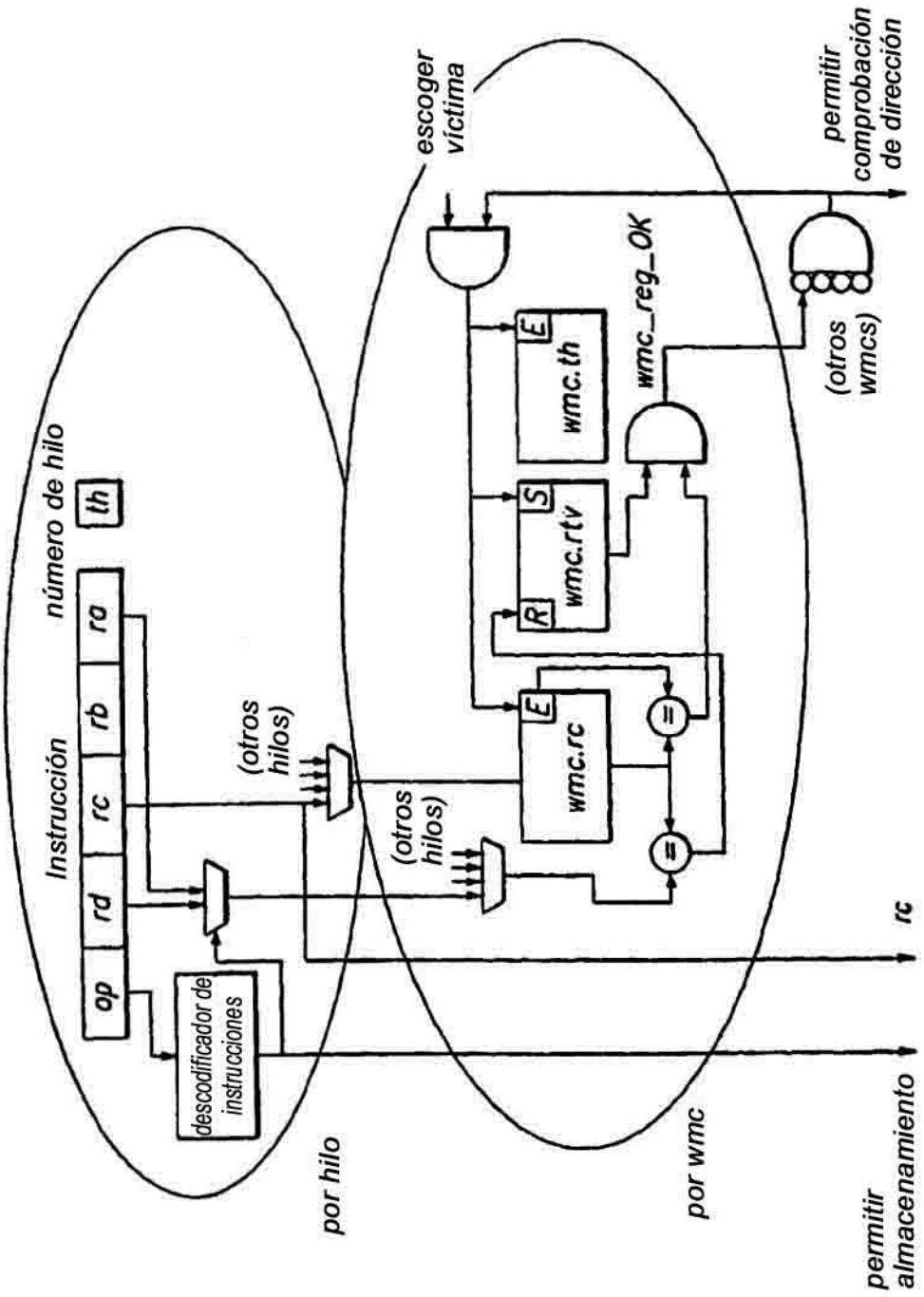

Instrucción
número de hilo
th
op
rd
rc
rb
ra
descodificador de instrucciones
(otros hilos)
(otros hilos)
por hilo
escoger víctima
E
wmc.th
S
R
wmc.rtv
E
wmc.rc
wmc_reg_OK
(otros wmcs)
permitir comprobación de dirección
rc
por wmc
permitir almacenamiento

FIG. 10

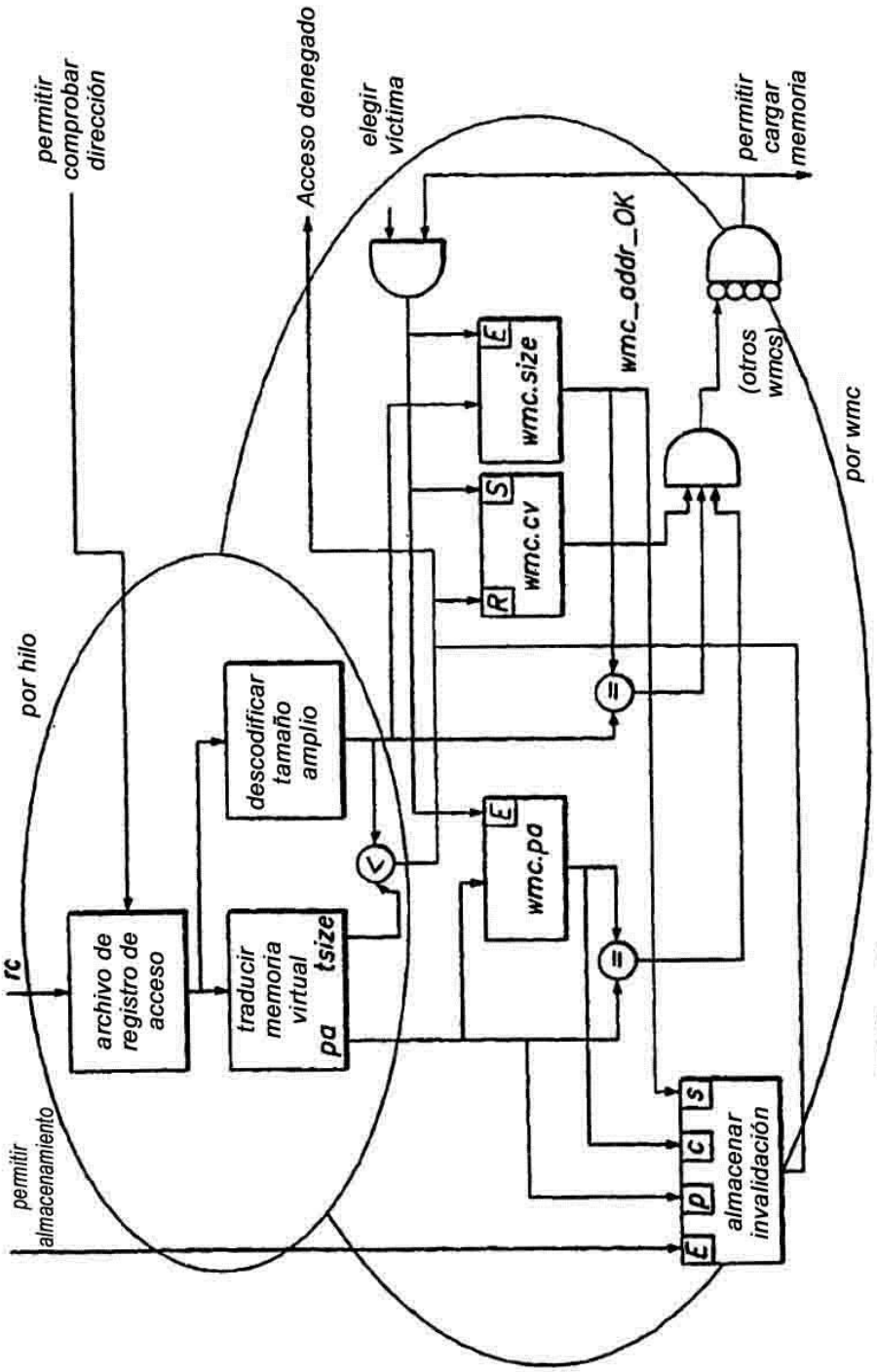

permitir comprobar dirección
Acceso denegado
elegir víctima
permitir cargar memoria
por hilo
por wmc
rc
archivo de registro de acceso
traducir memoria virtual
pa
tsize
descodificar tamaño amplio
wmc.size
wmc.cv
wmc.pa
wmc_addr_OK
(otros wmcs)
permitir almacenamiento
almacenar invalidación

FIG. 11

1810

Códigos de operación

W.MUL.MAT.G.8.B	Galois de matriz de multiplicación amplia octetos comenzando por el extremo alto
W.MUL.MAT.G.8.L	Galois de matriz de multiplicación amplia octetos comenzando por el extremo bajo

Selección

clase	op	tamaño	orden
Galois de matriz de multiplicación	W.MUL.MAT.G	8	B L

Formato

W.op.order ra=rc,rd,rb

ra=woporder(rc,rd,rb)

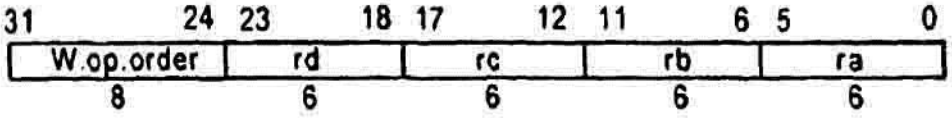


Fig 12A

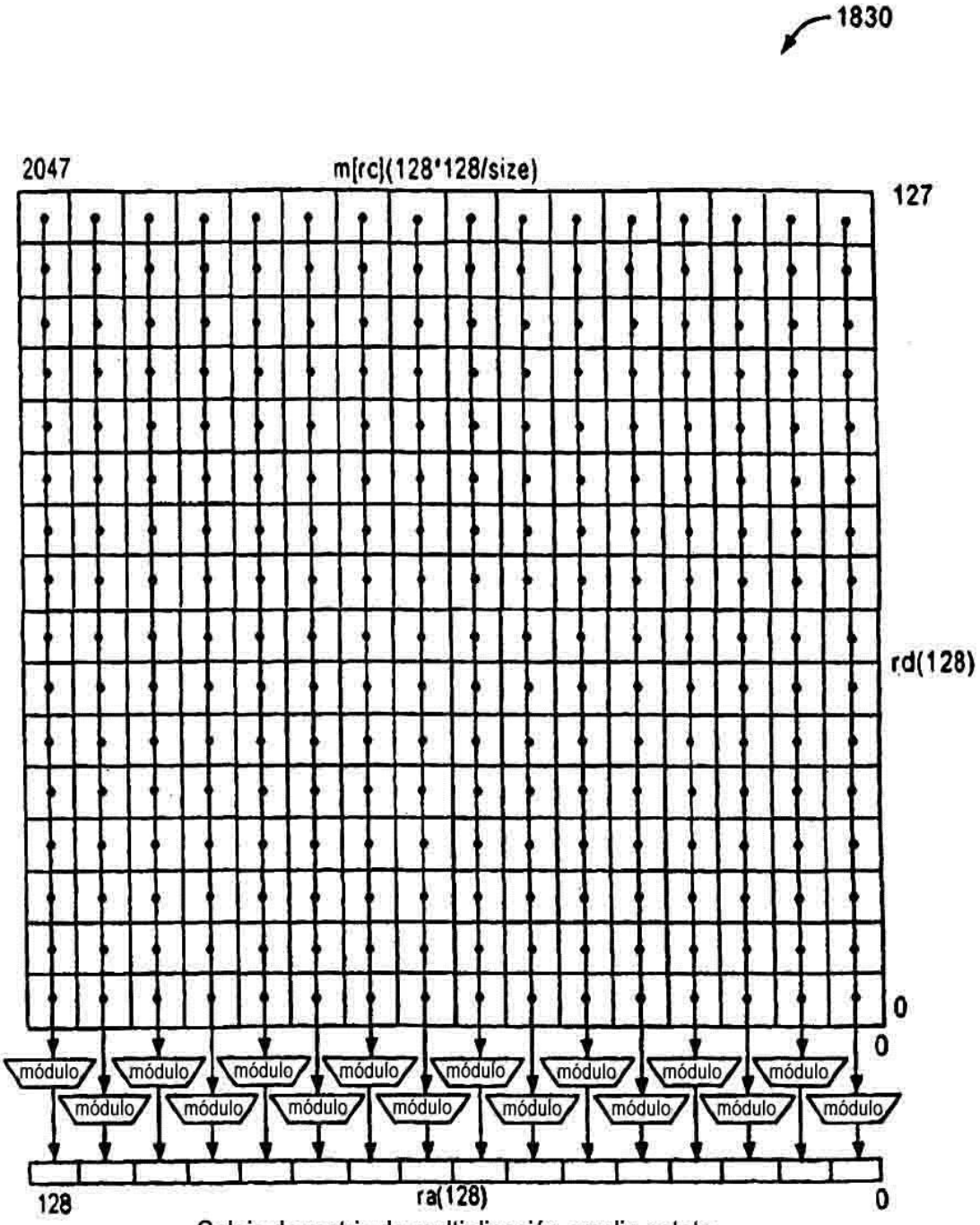

1830
2047
m[rc](128*128/size)
127
rd(128)
0
0
módulo
128
ra(128)
0
Galois de matriz de multiplicación amplia octeto

Fig 12B

Definición

1860

```
def c ← PolyMultiply(size,a,b) as
    p[0] ← 0^(2*size)
    for k ← 0 to size-1
        p[k+1] ← p[k] ^ a_k ? (0^(size-k) || b || 0^k) : 0^(2*size)
    endfor
    c ← p[size]
enddef

def c ← PolyResidue(size,a,b) as
    p[0] ← a
    for k ← size-1 to 0 by-1
        p[k-1] ← p[k] ^ p[0]_(size+k) ?(0^(size-k) || 1^1 || b || 0^k) : 0^(2*size)
    endfor
    c ← p[size]_(size-1..0)
enddef
def WideMultiplyMatrixGalois(op,gsize,rd,rc,rb,ra)
    d ← RegRead(rd, 128)
    c ← RegRead(rc, 64)
    b ← RegRead(rb,128)
    lgsize ← log(gsize)
    if c_(lgsize-4..0) ≠ 0 then
        raise AccessDisallowedByVirtualAddress
    endif
    if c_(3..lgsize-3) ≠ 0 then
        wsize ← (c and (0-c)) || 0^4
        t ← c and (c-1)
    else
        wsize ← 128
        t ← c
    endif
    lwsize ← log(wsize)
    if t_(lwsize+6-lgsize..lwsize-3) ≠ 0 then
        msize ← (t and (0-t)) || 0^4
        VirtAddr ← t and (t-1)
    else
        msize ← 128*wsize/gsize
        VirtAddr ← t
    endif
    case op of
        W.MUL.MAT.G.8.B:
            order ← B
        W.MUL.MAT.G.8.L:
            order ← L
    endcase
```

Fig 12C-1

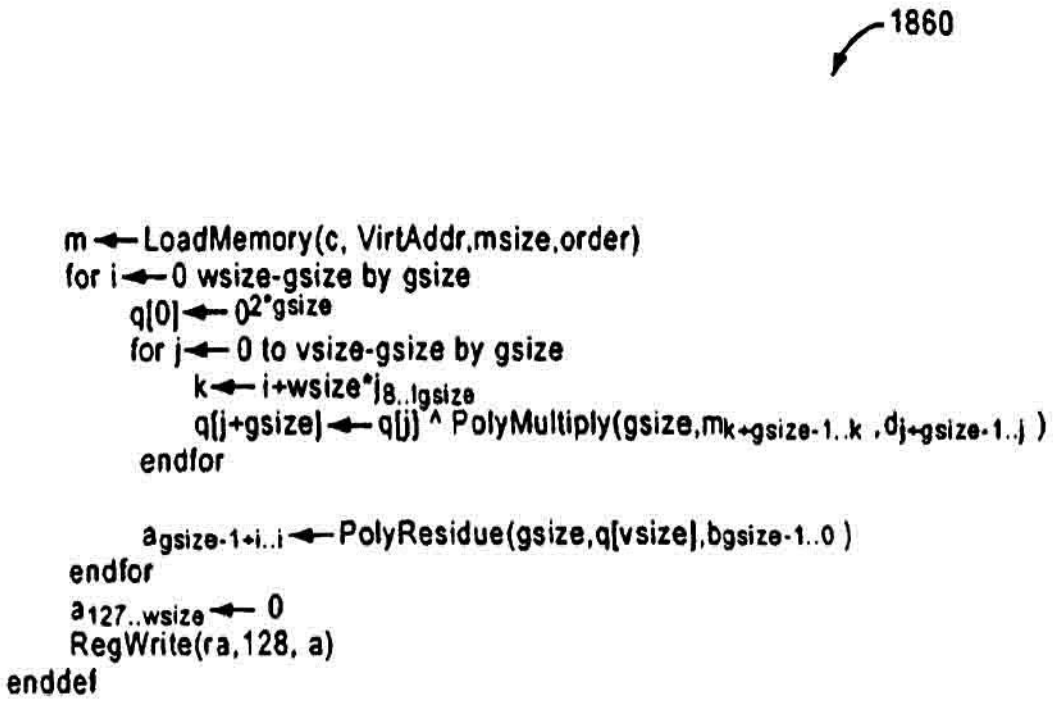
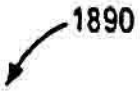

1860
m ← LoadMemory(c, VirtAddr,msize,order)
for i ← 0 wsize-gsize by gsize
q[0] ← 0^2*gsize
for j ← 0 to vsize-gsize by gsize
k ← i+wsize*j8..lgsize
q[j+gsize] ← q[j] ^ PolyMultiply(gsize,mk+gsize-1..k ,dj+gsize-1..j)
endfor
agsize-1+i..i ← PolyResidue(gsize,q[vsize],bgsize-1..0)
endfor
a127..wsize ← 0
RegWrite(ra,128, a)
enddef

Fig 12C-2



Excepciones

Acceso denegado por dirección virtual
Acceso denegado por etiqueta
Acceso denegado por TB global
Acceso denegado por TB local
Detalle de acceso requerido por etiqueta
Detalle de acceso requerido por TB local
Detalle de acceso requerido por TB global
Pérdida de TB local
Pérdida de TB global

Fig 12D

1910

Códigos de operación

E.MUL.ADD.X	Extracción suma de multiplicaciones de conjunto
E.CON.X	Extracción convolución de conjunto

Formato

E.op rd@rc,rb,ra

rd=gop(rd,rc,rb,ra)

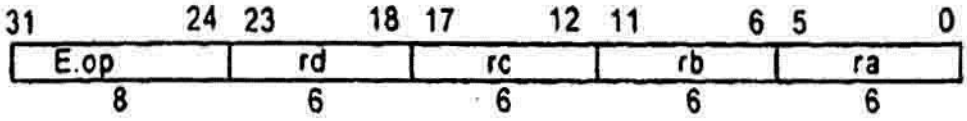
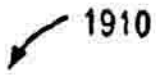


Fig 13A



Figuras 19B y 20B tienen campos en blanco: debería ser.

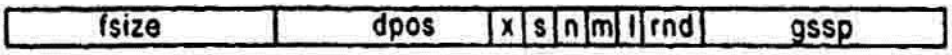


Fig 13B

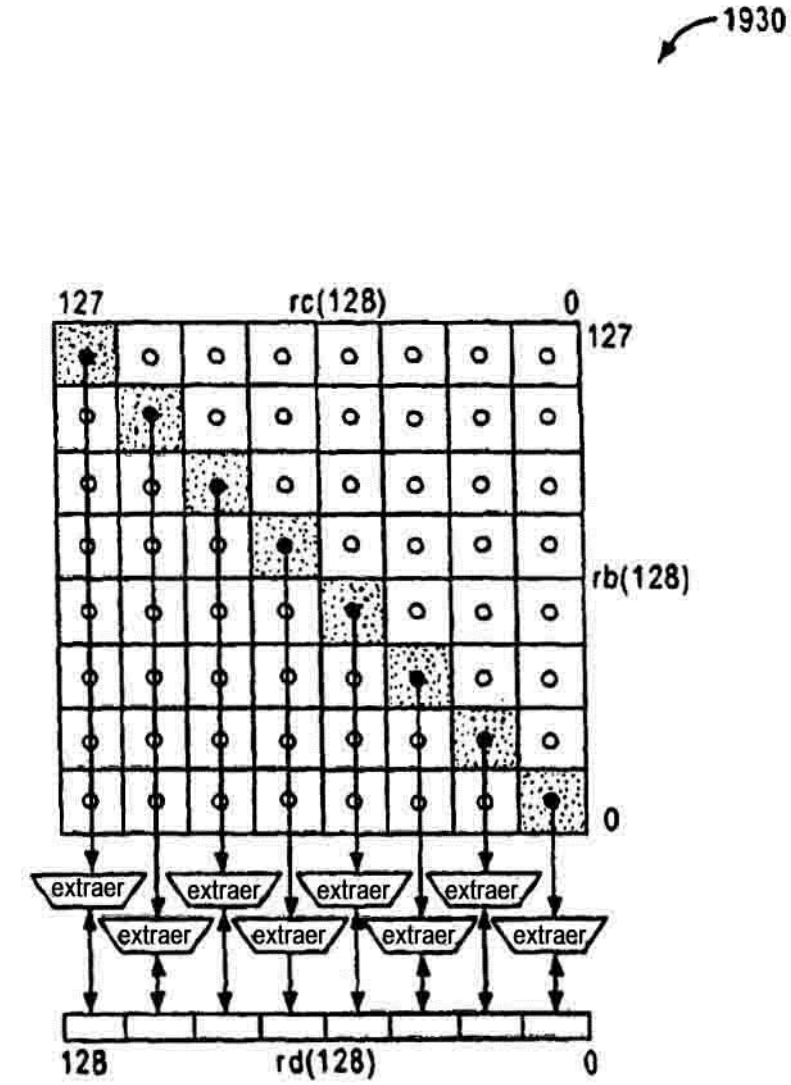
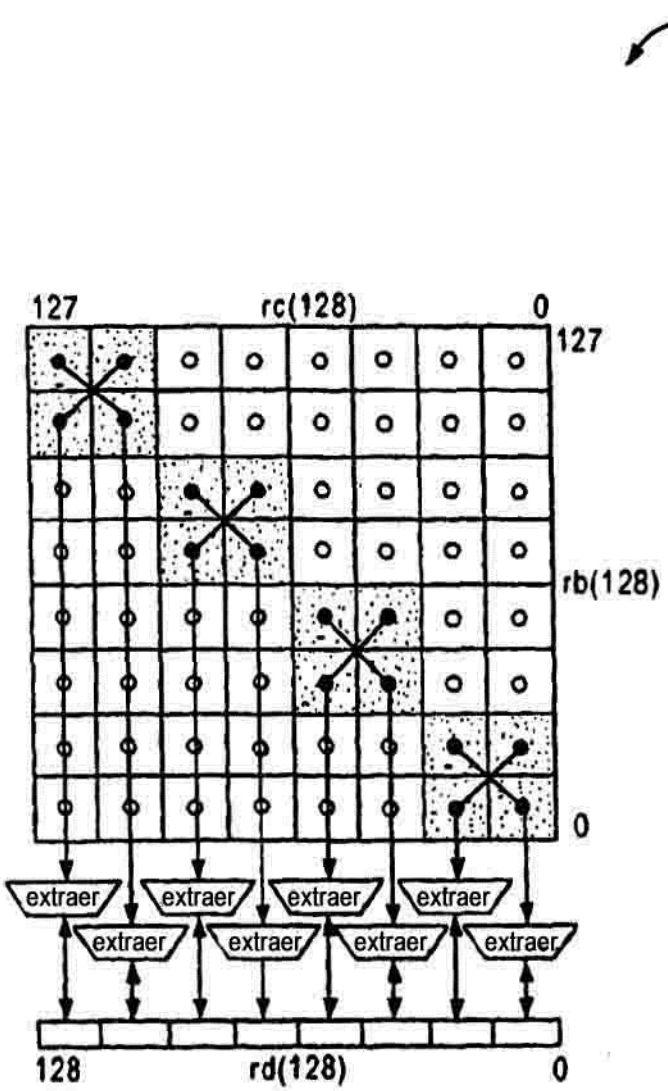

1930
127
rc(128)
0
127
rb(128)
0
extraer
extraer
extraer
extraer
extraer
extraer
extraer
extraer
128
rd(128)
0
Extracción multiplicación suma de conjunto dobletes

Fig 13C



Extracción multiplicación suma compleja del conjunto dobletes

Estas instrucciones de extracción multiplicación suma de conjunto (E.MUL.ADD.X), cuando el bit x está activado, multiplican los 64 bits de orden inferior de cada uno de los registros rc y rb, y producen resultados extendidos (doble tamaño).

Fig 13D

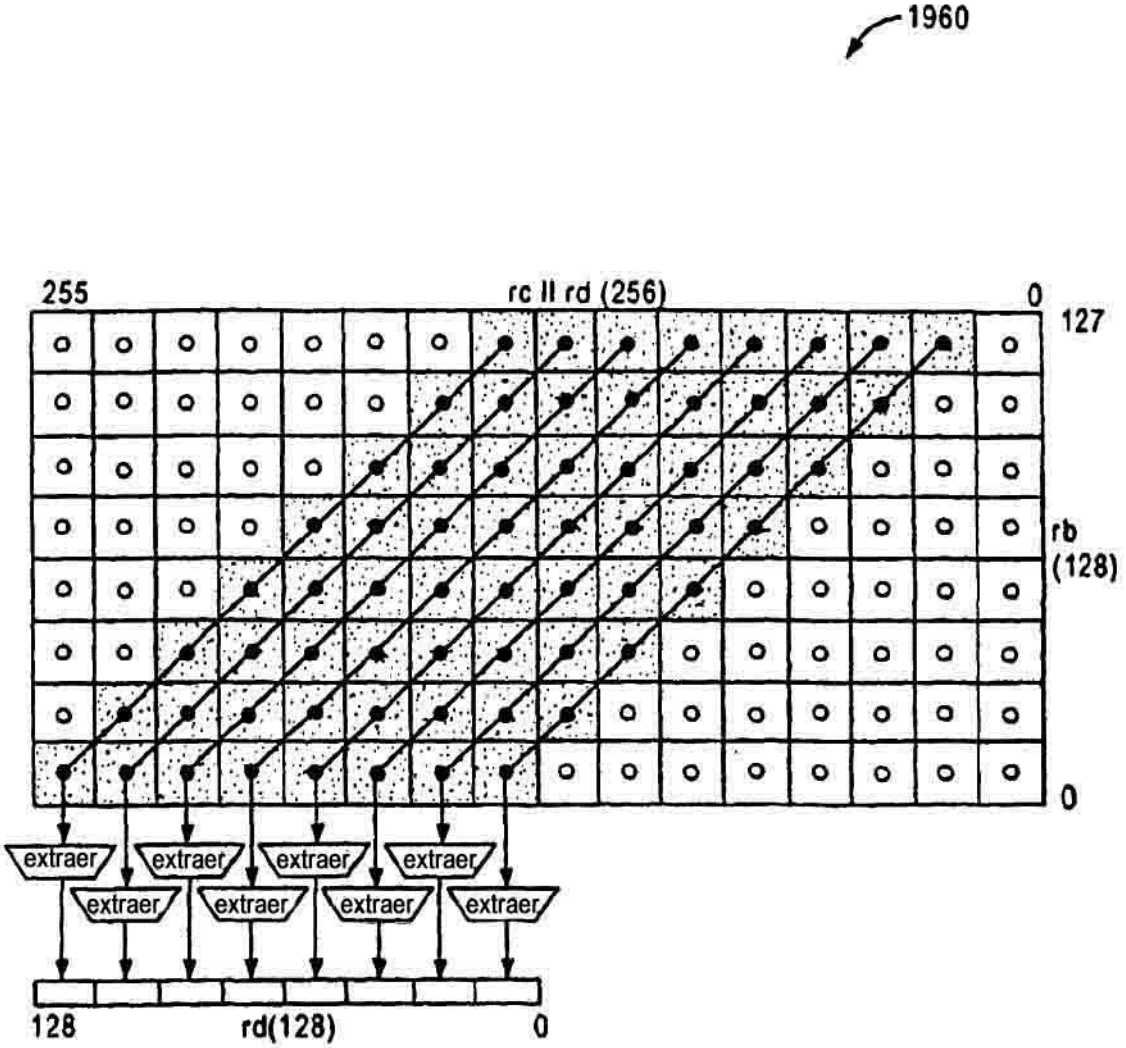
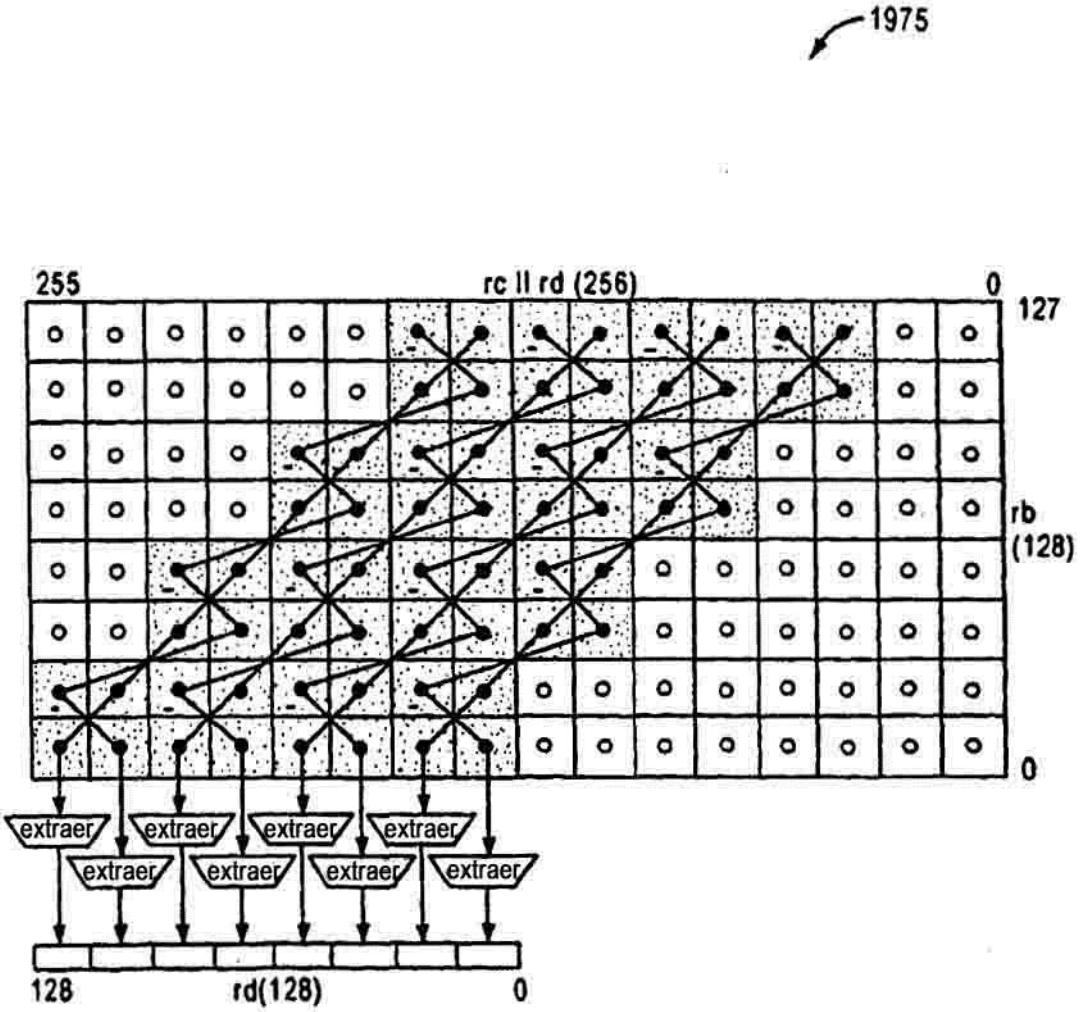

1960
255
rc II rd (256)
0
127
rb
(128)
0
extraer
extraer
extraer
extraer
extraer
extraer
extraer
extraer
extraer
128
rd(128)
0
Extracción convolución de conjunto dobletes

Fig 13E



Extracción convolución de conjunto dobletes complejos

Fig 13F

Definición

1990

```
def mul(size,h,vs,v,i,ws,w,j) as
    mul← ((vs&v size-1+i)h-size||v size-1+i..i) * ((ws&w size-1+j)h-size||w size-1+j..j)
enddef

def EnsembleExtractInplace(op,ra,rb,rc,rd) as
    d←RegRead(rd, 128)
    c←RegRead(rc, 128)
    b←RegRead(rb, 128)
    case b8..0 of
        0..255:
            sgsize←128
        256..383:
            sgsize←64
        384..447:
            sgsize←32
        448..479:
            sgsize←16
        480..495:
            sgsize←8
        496..503:
            sgsize←4
        504..507:
            sgsize←2
        508..511:
            sgsize←1
    endcase
    l←a11
    m←a12
    n←a13
    signed←a14
    x←a15
    case op of
        E.CON.X:
            if (sgsize < 8) then
                gsize← 8
            elseif (sgsize*(n-1)*(x+1) > 128 then
                gsize← 128/(n-1)/(x+1)
            else
                gsize← sgsize
            endif
            lgsize← log(gsize)
            wsize ← 128/(x+1)
```

Fig 13G-1

1990

```
        vsize ← 128
        ds ← cs ← signed
        bs ← signed ^ m
        zs ← signed or m or n
        zsize ← gsize*(x+1)
        h ← (2*gsize) + log(vsize) - lgsize
        spos ← (a8..0) and (2*gsize-1)

E.MUL.ADD.X:
    if(sgsize < 9) then
        gsize ← 8
    elseif (sgsize*(n+1)*(x+1) > 128) then
        gsize ← 128/(n+1)/(x+1)
    else
        gsize ← sgsize
    endif
    ds ← signed
    cs ← signed ^ m
    zs ← signed or m or n
    zsize ← gsize*(x+1)
    h ← (2*gsize) + n
    spos ← (a8..0) and (2*gssize-1)
endcase
dpos ← (0|| a23..16) and (zsize-1)
r ← spos
sfsize ← (0|| a31..24) and (zsize-1)
tfsize ← (sfsize = 0) or ((sfsize+dpos) > zsize) ? zsize-dpos : sfsize
fsize ← (tfsize + spos > h) ? h - spos : tfsize
if (b10..9 = Z) and not as then
    rnd ← F
else
    rnd ← b10..9
endif
```

Fig 13G-2

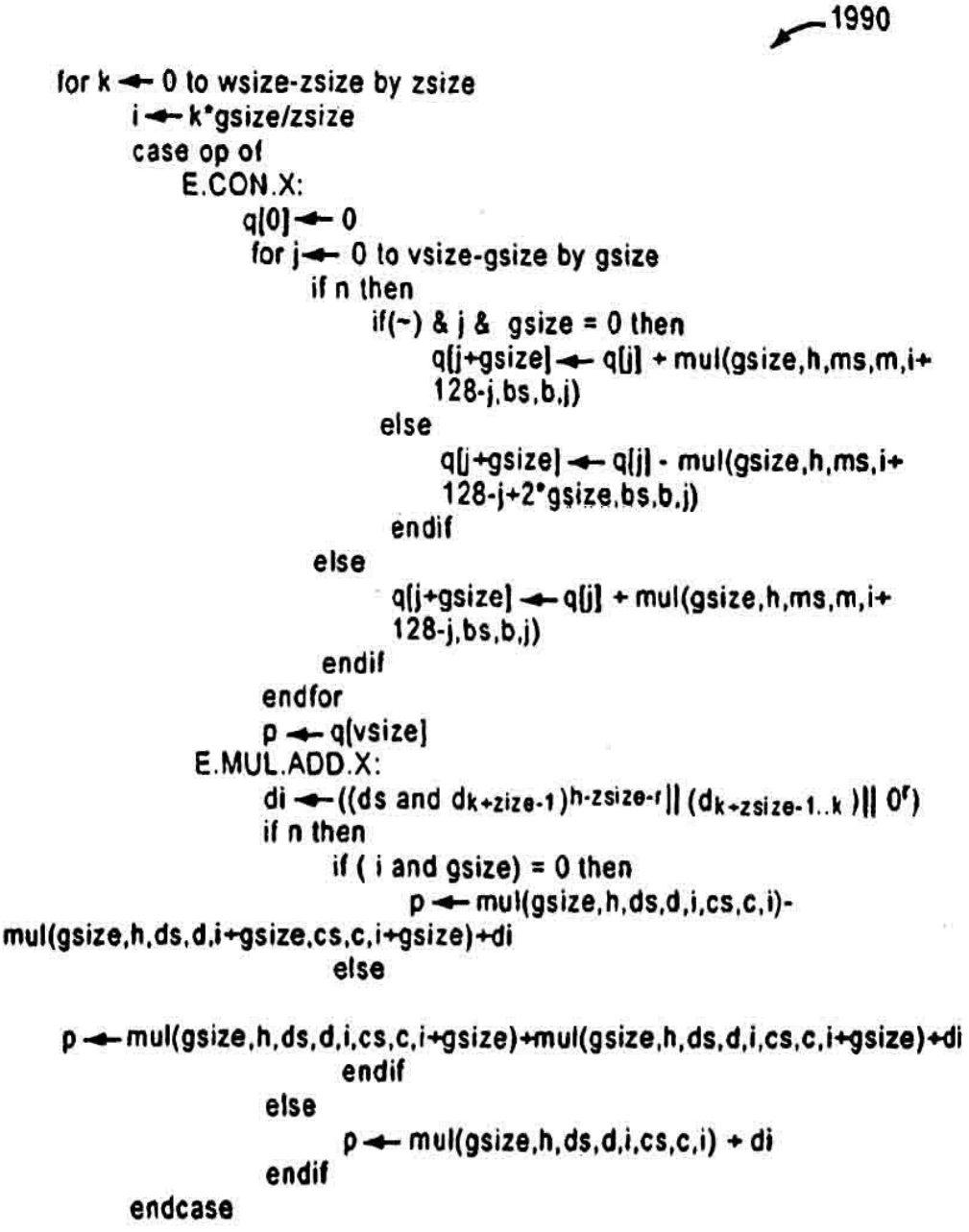

1990
for k ← 0 to wsize-zsize by zsize
i ← k*gsize/zsize
case op of
E.CON.X:
q[0] ← 0
for j ← 0 to vsize-gsize by gsize
if n then
if(~) & j & gsize = 0 then
q[j+gsize] ← q[j] + mul(gsize,h,ms,m,i+
128-j,bs,b,j)
else
q[j+gsize] ← q[j] - mul(gsize,h,ms,i+
128-j+2*gsize,bs,b,j)
endif
else
q[j+gsize] ← q[j] + mul(gsize,h,ms,m,i+
128-j,bs,b,j)
endif
endfor
p ← q[vsize]
E.MUL.ADD.X:
di ← ((ds and dk+zize-1)h-zsize-r || (dk+zsize-1..k)|| 0r)
if n then
if (i and gsize) = 0 then
p ← mul(gsize,h,ds,d,i,cs,c,i)-
mul(gsize,h,ds,d,i+gsize,cs,c,i+gsize)+di
else
p ← mul(gsize,h,ds,d,i,cs,c,i+gsize)+mul(gsize,h,ds,d,i,cs,c,i+gsize)+di
endif
else
p ← mul(gsize,h,ds,d,i,cs,c,i) + di
endif
endcase

Fig 13G-3ans

1990

```
        case rnd of
            N:
                s ← 0^(h-r) || ~p_r || p_r^(r-1)
            Z:
                s ← 0^(h-r) || p_(h-1)^r
            F:
                s ← 0^h
            C:
                s ← 0^(h-r) || 1^r
        endcase
        v ← ((zs & p_(h-1)) || p) + (0 || s)
        if (v_(h..r+fsize) = (zs & v_(r+fsize-1))^(h+1-r-fsize)) or not (l and (op =
        EXTRACT)) then
            w ← (zs & v_(r+fsize-1))^(zsize-fsize-dpos) || v_(fsize-1+r..r) || 0^dpos
        else
            w ← (zs ? (v_h || ~v_h^(zsize-dpos-1)) : 1^(zsize-dpos)) || 0^dpos
        endif
        z_(zsize-1_k..k) ← w
    endfor
    RegWrite(rd, 128, z)
enddef
```

Fig 13G-4

2010

Códigos de operación

E.MUL.X	Extracción multiplicación de conjunto
E.EXTRACT	Extracción de conjunto
E.SCAL.ADD.X	Escalar y extracción de conjunto

Formato

E.op ra=rd,rc,rb

ra=eop(rd,rc,rb)

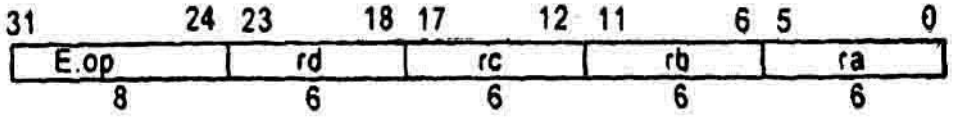
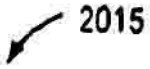
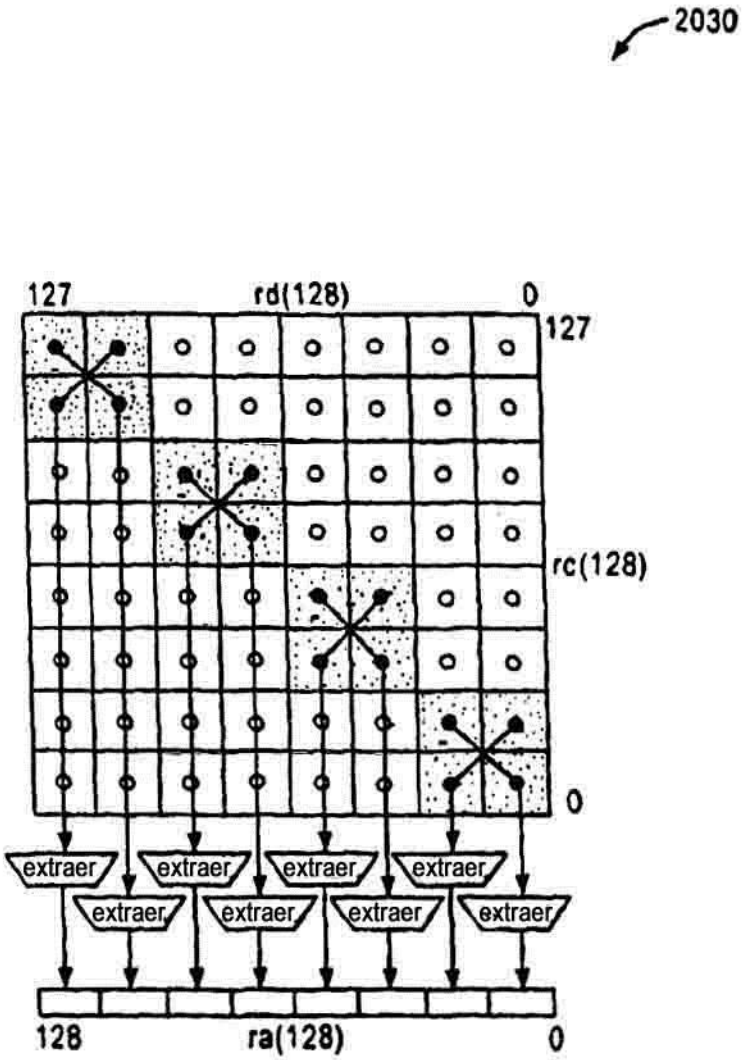


Fig 14A



Las figuras 19B y 20B tienen campos en blanco: debería ser.

Fig 14B



Extracción dobletes de multiplicación compleja de conjunto

Estas instrucciones de extracción multiplicación de conjunto (E.MUL.X), cuando el bit x está activado, multiplican los 64 bits de orden inferior de cada uno de los registros rc y rb, y producen resultados extendidos (doble tamaño).

Fig 14D

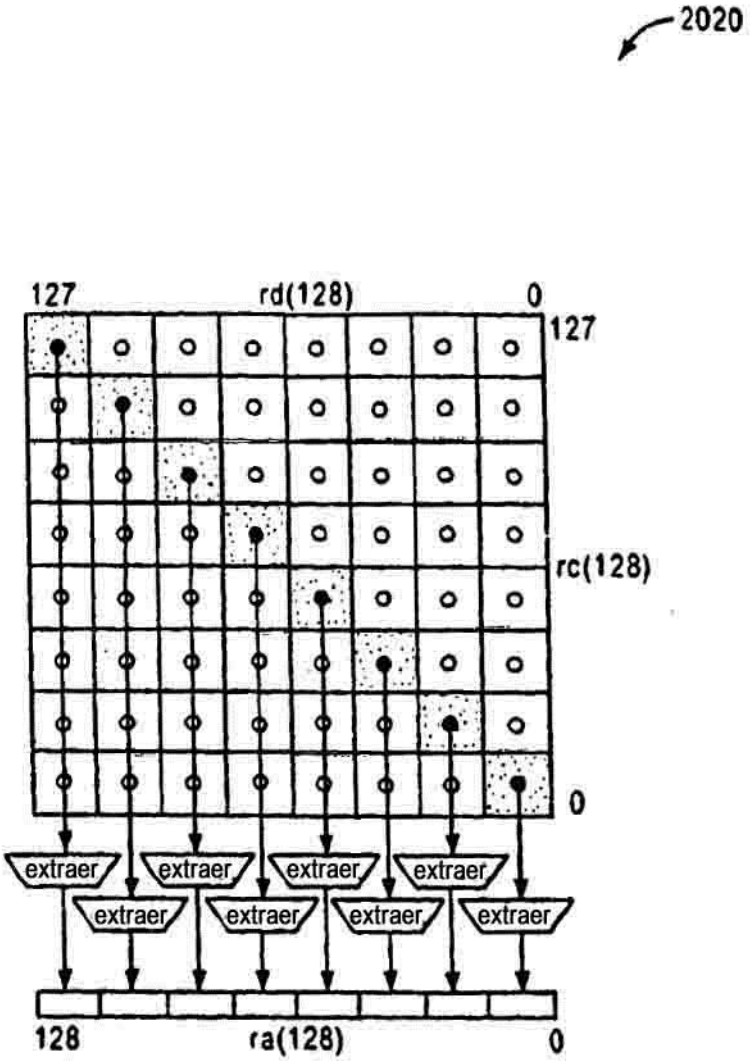
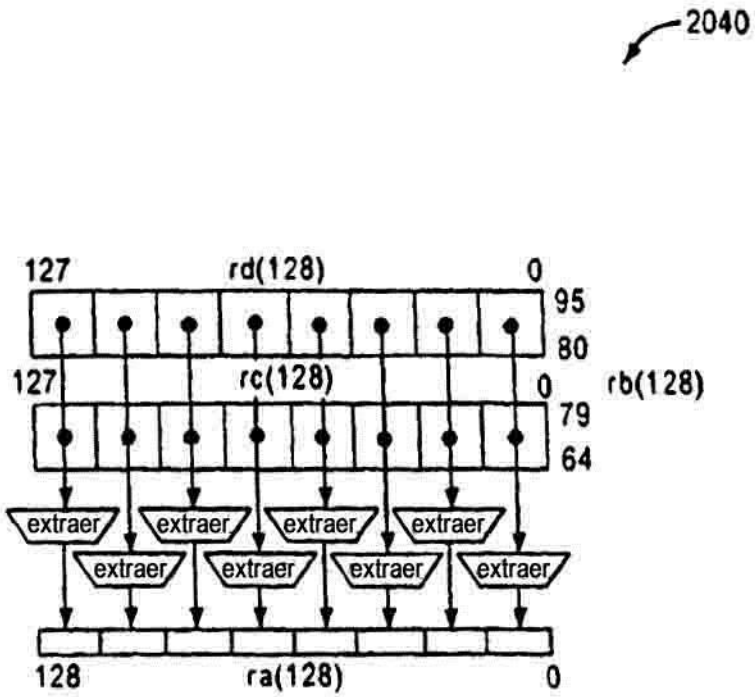
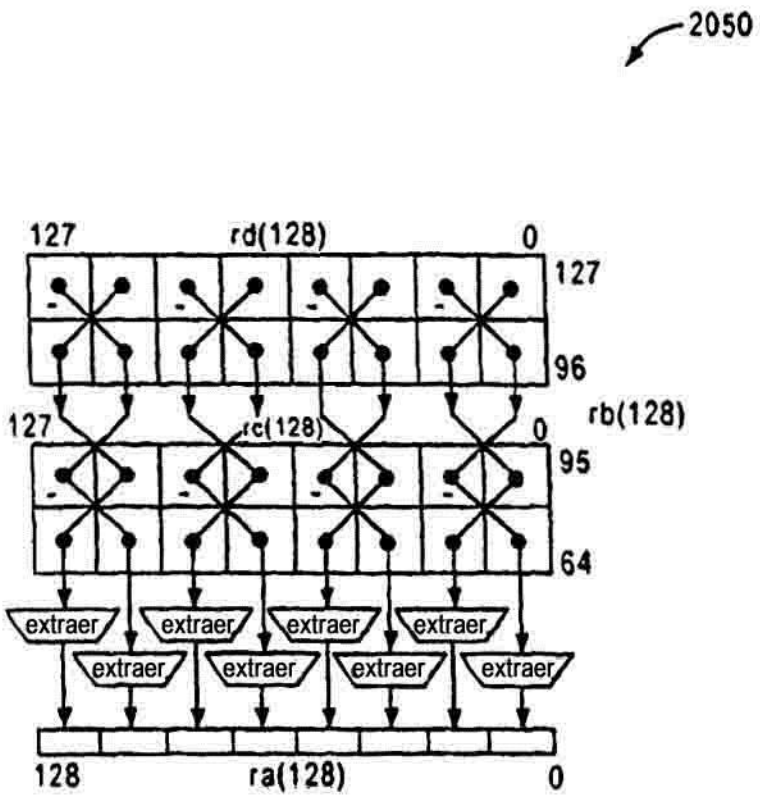

2020
127
rd(128)
0
127
rc(128)
0
extraer
extraer
extraer
extraer
extraer
extraer
extraer
extraer
extraer
128
ra(128)
0

Fig 14C



Extracción dobletes suma escalar de conjunto

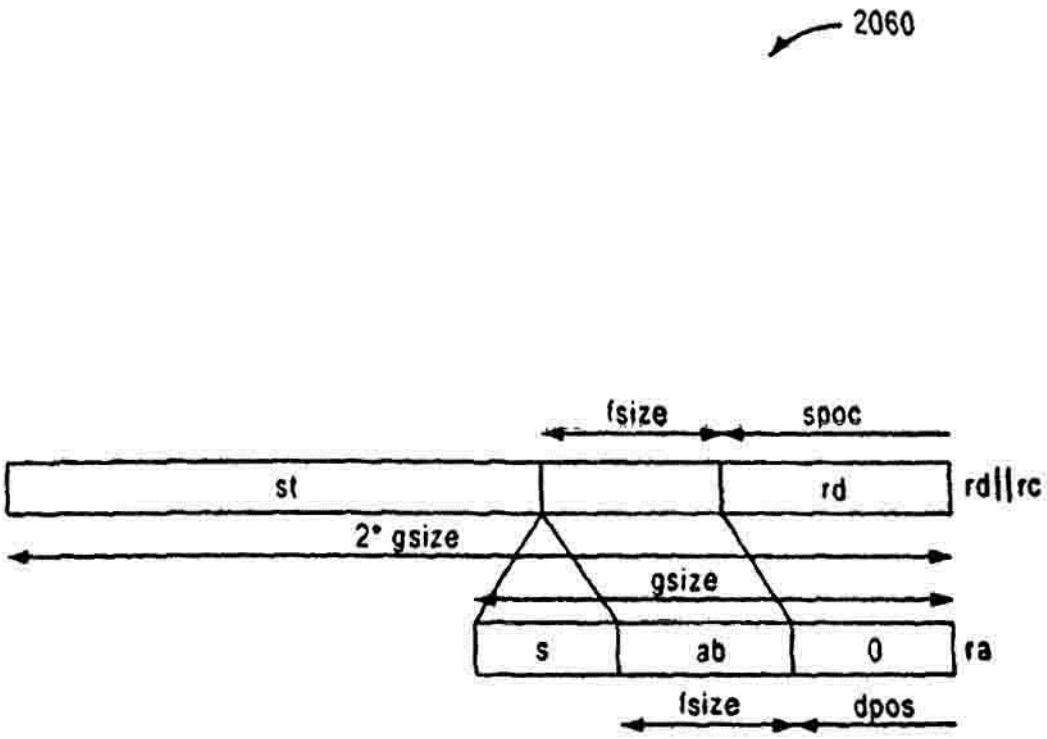
Fig 14E



Extracción dobletes suma escalar compleja de conjunto

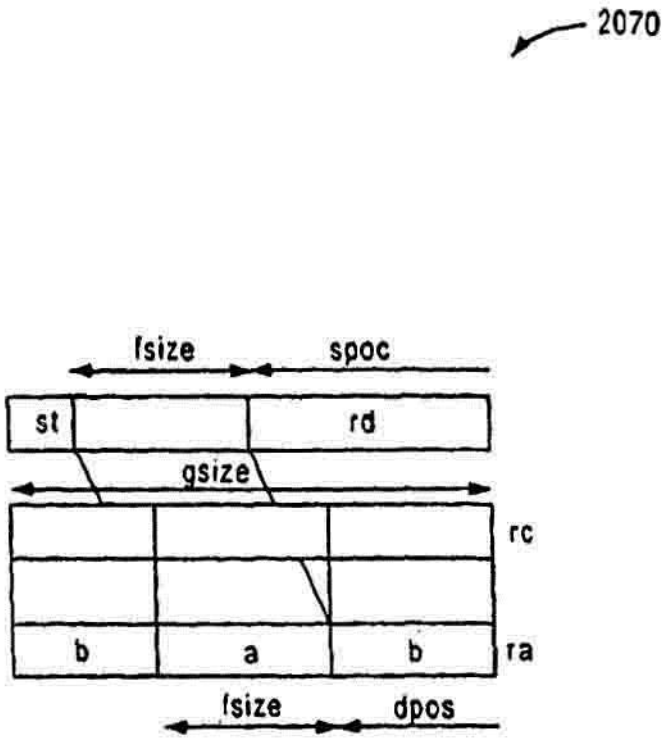
Las instrucciones de extracción suma escalar de conjunto (E.SCLADD.X), cuando el bit x está activado, multiplican los 64 bits de orden inferior de cada uno de los registros rd y re, por los campos del registro rd y producen resultados extendidos (doble tamaño).

Fig 14F



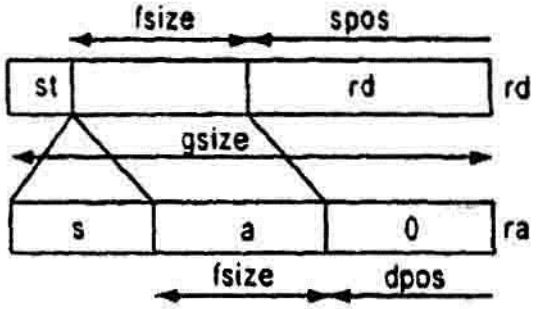
Extracción de conjunto

Fig 14G



Extracción fusionar conjunto

Fig 14H



Extracción expansión conjunto

Fig 14I

Definición

2090

```
def mul(size,h,vs,v,i,ws,w,j) as
    mul ← ((vs&v_{size-1+i})^{h-size} || v_{size-1+i..i}) * ((ws&w_{size-1+j})^{h-size} || w_{size-1+j..j})
enddef

def EnsembleExtract(op,ra,rb,rc,rd) as
    d ← RegRead(rd, 128)
    c ← RegRead(rc, 128)
    b ← RegRead(rb, 128)
    case b_{8..0} of
        0..255:
            sgsize ← 128
        256..383:
            sgsize ← 64
        384..447:
            sgsize ← 32
        448..479:
            sgsize ← 16
        480..495:
            sgsize ← 8
        496..503:
            sgsize ← 4
        504..507:
            sgsize ← 2
        508..511:
            sgsize ← 1
    endcase
    l ← b_{11}
    m ← b_{12}
    n ← b_{13}
    signed ← b_{14}
    x ← b_{15}
    case op of
        E.EXTRACT:
            gsize ← sgsize*2(2-(m or x))
            zsize ← sgsize
            h ← gsize
            as ← signed
            spos ← (b_{8..0}) and (gsize-1)
```

Fig 14J-1

2090

```
E.SCAL.ADD.X:
        if (sgsize < 8) then
              gsize ← 8
        elseif (sgsize*(n+1) > 32) then
              gsize ← 32/(n+1)
        else
              gsize ← sgsize
        endif
        ds ← cs ← signed
        bs ← signed ^ m
        as ← signed or m or n
        zsize ← gsize*(x+1)
        h ← (2*gsize) + 1 + n
        spos ← (b8..0) and (2*gsize-1)
    E.MUL.X:
        if (sgsize < 8) then
              gsize ← 8
        elseif (sgsize*(n+1)*(x+1) > 128) then
              gsize ← 128/(n+1)/(x+1)
        else
              gsize ← sgsize
        endif
        ds ← signed
        cs ← signed ^ m
        as ← signed or m or n
        zsize ← gsize*(x+1)
        h ← (2*gsize) + n
        spos ← (b8..0) and (2*gsize-1)
endcase
dpos ← (0|| b23..16) and (zsize-1)
r ← spos
sfsize ← (0|| b31..24) and (zsize-1)
lfsize ← (sfsize =0) or ((sfsize+dpos) > zsize) ? zsize-dpos : sfsize
fsize ← (lfsize + spos > h) ? h - spos : lfsize
if (b10..9=Z) and not as then
     rnd ← F
else
     rnd ← b
endif
```

Fig 14J-2

```
for j ← 0 to 128-zsize by zsize                                    ← 2090
    i ← j*gsize/zsize
    case op of
        E.EXTRACT:
            if m or x then
                p ← dgsize←i-1..i
            else
                p ← (d|| c)gsize←i-1..i
            endif
        E.MUL.X:
            if n then
                if (i and gsize) = 0 then
                    p ← mul(gsize,h,ds,d,i,cs,c,i)-
mul(gsize,h,ds,d,i+gsize,cs,c,i+gsize)
                else
                    p ←
mul(gsize,h,ds,d,i,cs,c,i+gsize)+mul(gsize,h,ds,d,i,cs,c,i+gsize)
                endif
            else
                p ← mul(gsize,h,ds,d,i,cs,c,i)
            endif
        E.SCAL.ADD.X:
            if n then
                if (i and gsize) = 0 then
                    p ← mul(gsize,h,ds,d,i,bs,b,64+2*gsize)
                        + mul(gsize,h,cs,c,i,bs,b,64)
                        - mul(gsize,h,ds,d,i+gsize,bs,b,64+3*gsize)
                        - mul(gsize,h,cs,c,i+gsize,bs,b,64+gsize)
                else
                    p ← mul(gsize,h,ds,d,i,bs,b,64+3*gsize)
                        + mul(gsize,h,cs,c,i,bs,b,64+gsize)
                        + mul(gsize,h,ds,d,i+gsize,bs,b,64+2*gsize)
                        + mul(gsize,h,cs,c,i+gsize,bs,b,64)
                endif
            else
                p ← mul(gsize,h,ds,d,i,bs,b,64+gsize) + mul(gsize
                    ,h,cs,c,i,bs,b,64)
            endif
    endcase
```

Fig 14J-3

```
        case rnd of                                                    2090
            N:
                s ← 0^(h-r) || ~p_r || p_r^(r-1)
            Z:
                s ← 0^(h-r) || p_(h-1)^r
            F:
                s ← 0^h
            C:
                s ← 0^(h-r) || 1^r
        endcase
        v ← ((as & p_(h-1)) || p) + (0 || s)
        if (v_(h..r+fsize) = (as & v_(r+fsize-1))^(h+1-r-fsize)) or not (l and (op =
                        E.EXTRACT)) then
            w ← (as & v_(r+fsize=1))^(zsize-fsize-dpos) || v_(fsize-1+r..r) || 0^dpos
        else
            w ← (s ? (v_h || ~v_h^(zsize-dpos-1)) : 1^(zsize-dpos)) || 0^dpos
        endif
        if m and (op = E.EXTRACT) then
            z_(zsize-1+j..j) ← c_(asize-1+j..dpos+fsize+j) || w_(dpos+fsize-1..dpos) ||
                               c_(dpos-1+j..j)
        else
            z_(zsize-1+j..j) ← w
        endif
    endfor
    RegWrite(ra, 128, z)
enddef
```

Fig 14J-4

2310

Códigos de operación

G.BOOLEAN	Booleana de grupo

Selección

operación	función (binaria)	función (binaria) decimal
d	11110000	240
c	11001100	204
b	10101010	176
d&c&b	10000000	128
(d&c)|b	11101010	234
d|c|b	11111110	254
d?c:b	11001010	202
d^c^b	10010110	150
~d^c^b	01101001	105
0	00000000	0

Formato

G.BOOLEAN rd@trc,trb,f

rd=gbooleani(rd,rc,rb,f)

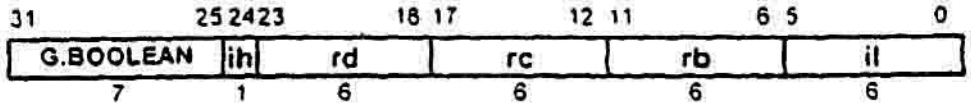
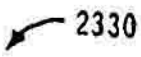


Fig 15A

2320

```
if f6=f5 then
        if f2=f1 then
                if f2 then
                        rc ← max(trc,trb)
                        rb ← min(trc,trb)
                else
                        rc ← min(trc,trb)
                        rb ← max(trc,trb)
                endif
                ih ← 0
                il ← 0 || f6 || f7 || f4 || f3 || f0
        else
                if f2 then
                        rc ← trb
                        rb ← trc
                else
                        rc ← trc
                        rb ← trb
                endif
                ih ← 0
                il ← 1 || f6 || f7 || f4 || f3 || f0
        endif
else
        ih ← 1
        if f6 then
                rc ← trb
                rb ← trc
                il ← f1 || f2 || f7 || f4 || f3 || f0
        else
                rc ← trc
                rb ← trb
                il ← f2 || f1 || f7 || f4 || f3 || f0
        endif
endif
```

Fig 15B



Definición

```
def GroupBoolean (ih,rd,rc,rb,il)
      d ← RegRead(rd, 128)
      c ← RegRead(rc, 128)
      b ← RegRead(rb, 128)
      if ih=0 then
            if il5=0 then
                  f ← il3 || il4 || il4 || il2 || il1 || (rc>rb)2 || il0
            else
                  f ← il3 || il4 || il4 || il2 || il1 || 0 || 1 || il0
            endif
      else
            f ← il3 || 0 || 1 || il2 || il1 || il5 || il4 || il0
      endif
      for i ← 0 to 127 by size
            ai ← f(ai||ci||bi)
      endfor
      RegWrite(rd, 128, a)
enddef
```

Fig 15C

Códigos de operación

G.ADD.8	Suma de grupo octetos
G.ADD.16	Suma de grupo dobletes
G.ADD.32	Suma de grupo cuádruplas
G.ADD.64	Suma de grupo 8-tuplas
G.ADD.128	Suma de grupo 16-tuplas
G.ADD.L.8	Suma de grupo límite octetos con signo
G.ADD.L.16	Suma de grupo límite dobletes con signo
G.ADD.L.32	Suma de grupo límite cuádruplas con signo
G.ADD.L.64	Suma de grupo límite 8- tuplas con signo
G.ADD.L.128	Suma de grupo límite 16-tuplas con signo
G.ADD.L.U.8	Suma de grupo límite octetos sin signo
G.ADD.L.U.16	Suma de grupo límite dobletes sin signo
G.ADD.L.U.32	Suma de grupo límite cuádruplas sin signo
G.ADD.L.U.64	Suma de grupo límite 8- tuplas sin signo
G.ADD.L.U.128	Suma de grupo límite 16-tuplas sin signo
G.ADD.8.O	Suma de grupo octetos con signo comprobar desbordamiento
G.ADD.16.O	Suma de grupo dobletes con signo comprobar desbordamiento
G.ADD.32.O	Suma de grupo cuádruplas con signo comprobar desbordamiento
G.ADD.64.O	Suma de grupo 8- tuplas con signo comprobar desbordamiento
G.ADD.128.O	Suma de grupo 16-tuplas con signo comprobar desbordamiento
G.ADD.U.8.O	Suma de grupo octetos sin signo comprobar desbordamiento
G.ADD.U.16.O	Suma de grupo dobletes sin signo comprobar desbordamiento
G.ADD.U.32.O	Suma de grupo cuádruplas sin signo comprobar desbordamiento
G.ADD.U.64.O	Suma de grupo 8- tuplas sin signo comprobar desbordamiento
G.ADD.U.128.O	Suma de grupo 16-tuplas sin signo comprobar desbordamiento

Fig 16A

Formato

G.op.size rd=rc,rb

rd=gopsize(rc,rb)

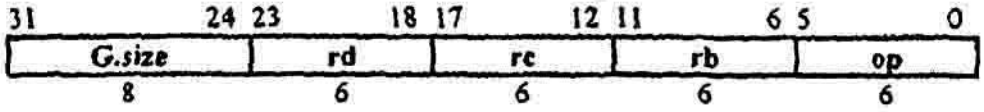


Fig 16B

Definición

```
def Group(op,size,rd,rc,rb)
    c ← RegRead(rc, 128)
    b ← RegRead(rb, 128)
    case op of
        G.ADD:
            for i ← 0 to 128-size by size
                a_{i+size-1..i} ← c_{i+size-1..i} + b_{i+size-1..i}
            endfor
        G.ADD.L:
            for i ← 0 to 128-size by size
                t ← (c_{i+size-1} || c_{i+size-1..i}) + (b_{i+size-1} || b_{i+size-1..i})
                a_{i+size-1..i} ← (t_size ≠ t_{size-1}) ? (t_size || t_{size-1}^{size-1}) : t_{size-1..0}
            endfor
        G.ADD.L.U:
            for i ← 0 to 128-size by size
                t ← (0^1 || c_{i+size-1..i}) + (0^1 || b_{i+size-1..i})
                a_{i+size-1..i} ← (t_size ≠ 0) ? (1^size) : t_{size-1..0}
            endfor
        G.ADD.O:
            for i ← 0 to 128-size by size
                t ← (c_{i+size-1} || c_{i+size-1..i}) + (b_{i+size-1} || b_{i+size-1..i})
                if t_size ≠ t_{size-1} then
                    raise FixedPointArithmetic
                endif
                a_{i+size-1..i} ← t_{size-1..0}
            endfor
        G.ADD.U.O:
            for i ← 0 to 128-size by size
                t ← (0^1 || c_{i+size-1..i}) + (0^1 || b_{i+size-1..i})
                if t_size ≠ 0 then
                    raise FixedPointArithmetic
                endif
                a_{i+size-1..i} ← t_{size-1..0}
            endfor
    endcase
    RegWrite(rd, 128, a)
enddef
```

Fig 16C

Códigos de operación

G.SET.AND.E.8	Establecimiento de grupo e igual a cero octetos
G.SET.AND.E.16	Establecimiento de grupo e igual a cero dobletes
G.SET.AND.E.32	Establecimiento de grupo e igual a cero cuádruplas
G.SET.AND.E.64	Establecimiento de grupo e igual a cero 8-tuplas
G.SET.AND.E.128	Establecimiento de grupo e igual a cero 16-tuplas
G.SET.AND.NE.8	Establecimiento de grupo y distinto de cero octetos
G.SET.AND.NE.16	Establecimiento de grupo y distinto de cero dobletes
G.SET.AND.NE.32	Establecimiento de grupo y distinto de cero cuádruplas
G.SET.AND.NE.64	Establecimiento de grupo y distinto de cero 8-tuplas
G.SET.AND.NE.128	Establecimiento de grupo y distinto de cero 16-tuplas
G.SET.E.8	Establecimiento de grupo igual octetos
G.SET.E.16	Establecimiento de grupo igual dobletes
G.SET.E.32	Establecimiento de grupo igual cuádruplas
G.SET.E.64	Establecimiento de grupo igual 8-tuplas
G.SET.E.128	Establecimiento de grupo igual 16-tuplas
G.SET.GE.8	Establecimiento de grupo mayor o igual con signo octetos
G.SET.GE.16	Establecimiento de grupo mayor o igual con signo dobletes
G.SET.GE.32	Establecimiento de grupo mayor o igual con signo cuádruplas
G.SET.GE.64	Establecimiento de grupo mayor o igual con signo 8-tuplas
G.SET.GE.128	Establecimiento de grupo mayor o igual con signo 16-tuplas
G.SET.GE.U.8	Establecimiento de grupo mayor o igual sin signo octetos
G.SET.GE.U.16	Establecimiento de grupo mayor o igual sin signo dobletes
G.SET.GE.U.32	Establecimiento de grupo mayor o igual sin signo cuádruplas
G.SET.GE.U.64	Establecimiento de grupo mayor o igual sin signo 8-tuplas
G.SET.GE.U.128	Establecimiento de grupo mayor o igual sin signo 16-tuplas
G.SET.L.8	Establecimiento de grupo con signo menos octetos
G.SET.L.16	Establecimiento de grupo con signo menos dobletes
G.SET.L.32	Establecimiento de grupo con signo menos cuádruplas
G.SET.L.64	Establecimiento de grupo con signo menos 8-tuplas
G.SET.L.128	Establecimiento de grupo con signo menos 16-tuplas
G.SET.L.U.8	Establecimiento de grupo menos sin signo octetos
G.SET.L.U.16	Establecimiento de grupo menos sin signo dobletes
G.SET.L.U.32	Establecimiento de grupo menos sin signo cuádruplas
G.SET.L.U.64	Establecimiento de grupo menos sin signo 8-tuplas
G.SET.L.U.128	Establecimiento de grupo menos sin signo 16-tuplas
G.SET.NE.8	Establecimiento de grupo no igual octetos
G.SET.NE.16	Establecimiento de grupo no igual dobletes
G.SET.NE.32	Establecimiento de grupo no igual cuádruplas
G.SET.NE.64	Establecimiento de grupo no igual 8-tuplas
G.SET.NE.128	Establecimiento de grupo no igual 16-tuplas
G.SUB.8	Resta de grupo octetos
G.SUB.8.O	Resta de grupo con signo octetos comprobar desbordamiento

Fig 17A-1

G.SUB.16	Resta de grupo dobletes
G.SUB.16.O	Resta de grupo con signo dobletes comprobar desbordamiento
G.SUB.32	Resta de grupo cuádruplas
G.SUB.32.O	Resta de grupo con signo cuádruplas comprobar desbordamiento
G.SUB.64	Resta de grupo 8-tuplas
G.SUB.64.O	Resta de grupo con signo 8-tuplas comprobar desbordamiento
G.SUB.128	Resta de grupo 16-tuplas
G.SUB.128.O	Resta de grupo con signo 16-tuplas comprobar desbordamiento
G.SUB.L.8	Resta de grupo límite con signo octetos
G.SUB.L.16	Resta de grupo límite con signo dobletes
G.SUB.L.32	Resta de grupo límite con signo cuádruplas
G.SUB.L.64	Resta de grupo límite con signo 8-tuplas
G.SUB.L.128	Resta de grupo límite con signo 16-tuplas
G.SUB.L.U.8	Resta de grupo límite sin signo octetos
G.SUB.L.U.16	Resta de grupo límite sin signo dobletes
G.SUB.L.U.32	Resta de grupo límite sin signo cuádruplas
G.SUB.L.U.64	Resta de grupo límite sin signo 8-tuplas
G.SUB.L.U.128	Resta de grupo límite sin signo 16-tuplas
G.SUB.U.8.O	Resta de grupo sin signo octetos comprobar desbordamiento
G.SUB.U.16.O	Resta de grupo sin signo dobletes comprobar desbordamiento
G.SUB.U.32.O	Resta de grupo sin signo cuádruplas comprobar desbordamiento
G.SUB.U.64.O	Resta de grupo sin signo 8-tuplas comprobar desbordamiento
G.SUB.U.128.O	Resta de grupo sin signo 16-tuplas comprobar desbordamiento

Fig 17A-2

Formato

G.op.size rd=rb,rc

rd=gopsize(rb,rc)

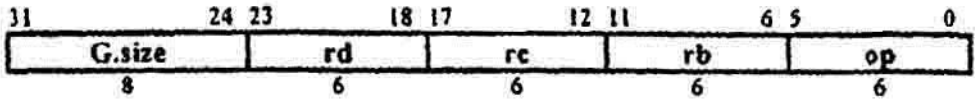


Fig 17B

Definición

```
def GroupReversed(op,size,rd,rc,rb)
    c ← RegRead(rc, 128)
    b ← RegRead(rb, 128)
    case op of
        G.SUB:
            for i ← 0 to 128-size by size
                a_{i+size-1..i} ← b_{i+size-1..i} - c_{i+size-1..i}
            endfor
        G.SUB.L:
            for i ← 0 to 128-size by size
                t ← (b_{i+size-1} || b_{i+size-1..i}) - (c_{i+size-1} || c_{i+size-1..i})
                a_{i+size-1..i} ← (t_size ≠ t_{size-1}) ? (t_size || t_{size-1}^{size-1}) : t_{size-1..0}
            endfor
        G.SUB.LU:
            for i ← 0 to 128-size by size
                t ← (0^1 || b_{i+size-1..i}) - (0^1 || c_{i+size-1..i})
                a_{i+size-1..i} ← (t_size ≠ 0) ? 0^size : t_{size-1..0}
            endfor
        G.SUB.O:
            for i ← 0 to 128-size by size
                t ← (b_{i+size-1} || b_{i+size-1..i}) - (c_{i+size-1} || c_{i+size-1..i})
                if (t_size ≠ t_{size-1}) then
                    raise FixedPointArithmetic
                endif
                a_{i+size-1..i} ← t_{size-1..0}
            endfor
        G.SUB.U.O:
            for i ← 0 to 128-size by size
                t ← (0^1 || b_{i+size-1..i}) - (0^1 || c_{i+size-1..i})
                if (t_size ≠ 0) then
                    raise FixedPointArithmetic
                endif
                a_{i+size-1..i} ← t_{size-1..0}
            endfor
        G.SET.E:
            for i ← 0 to 128-size by size
                a_{i+size-1..i} ← (b_{i+size-1..i} = c_{i+size-1..i})^size
            endfor
        G.SET.NE:
            for i ← 0 to 128-size by size
                a_{i+size-1..i} ← (b_{i+size-1..i} ≠ c_{i+size-1..i})^size
            endfor
        G.SET.AND.E:
            for i ← 0 to 128-size by size
                a_{i+size-1..i} ← ((b_{i+size-1..i} and c_{i+size-1..i}) = 0)^size
            endfor
```

Fig 17C-1

```
        G.SET.AND.NE:
            for i ← 0 to 128-size by size
                a_{i+size-1..i} ← ((b_{i+size-1..i} and c_{i+size-1..i}) ≠ 0)^size
            endfor
        G.SET.L:
            for i ← 0 to 128-size by size
                a_{i+size-1..i} ← ((rc = rb) ? (b_{i+size-1..i} < 0) : (b_{i+size-1..i} < c_{i+size-1..i}))^size
            endfor
        G.SET.GE:
            for i ← 0 to 128-size by size
                a_{i+size-1..i} ← ((rc = rb) ? (b_{i+size-1..i} ≥ 0) : (b_{i+size-1..i} ≥ c_{i+size-1..i}))^size
            endfor
        G.SET.L.U:
            for i ← 0 to 128-size by size
                a_{i+size-1..i} ← ((rc = rb) ? (b_{i+size-1..i} > 0) :
                    ((0 || b_{i+size-1..i}) < (0 || c_{i+size-1..i})))^size
            endfor
        G.SET.GE.U:
            for i ← 0 to 128-size by size
                a_{i+size-1..i} ← ((rc = rb) ? (b_{i+size-1..i} ≤ 0) :
                    ((0 || b_{i+size-1..i}) ≥ (0 || c_{i+size-1..i})))^size
            endfor
    endcase
    RegWrite(rd, 128, a)
enddef
```

Fig 17C-2

Códigos de operación

E.CON.8	Convolución de conjunto con signo octetos
E.CON.16	Convolución de conjunto con signo dobletes
E.CON.32	Convolución de conjunto con signo cuádruplas
E.CON.64	Convolución de conjunto con signo 8-tuplas
E.CON.C.8	Convolución de conjunto compleja octetos
E.CON.C.16	Convolución de conjunto compleja dobletes
E.CON.C.32	Convolución de conjunto compleja cuádruplas
E.CON.M.8	Convolución de conjunto signo mezclado octetos
E.CON.M.16	Convolución de conjunto signo mezclado dobletes
E.CON.M.32	Convolución de conjunto signo mezclado cuádruplas
E.CON.M.64	Convolución de conjunto signo mezclado 8-tuplas
E.CON.U.8	Convolución de conjunto sin signo octetos
E.CON.U.16	Convolución de conjunto sin signo dobletes
E.CON.U.32	Convolución de conjunto sin signo cuádruplas
E.CON.U.64	Convolución de conjunto sin signo 8-tuplas
E.DIV.64	División de conjunto con signo 8-tuplas
E.DIV.U.64	División de conjunto sin signo 8-tuplas
E.MUL.8	Multiplicación de conjunto con signo octetos
E.MUL.16	Multiplicación de conjunto con signo dobletes
E.MUL.32	Multiplicación de conjunto con signo cuádruplas
E.MUL.64	Multiplicación de conjunto con signo 8-tuplas
E.MUL.SUM.8	Suma de multiplicaciones de conjunto con signo octetos
E.MUL.SUM.16	Suma de multiplicaciones de conjunto con signo dobletes
E.MUL.SUM.32	Suma de multiplicaciones de conjunto con signo cuádruplas
E.MUL.SUM.64	Suma de multiplicaciones de conjunto con signo 8-tuplas
E.MUL.C.8	Multiplicación compleja de conjunto octetos
E.MUL.C.16	Multiplicación compleja de conjunto dobletes
E.MUL.C.32	Multiplicación compleja de conjunto cuádruplas
E.MUL.M.8	Multiplicación de conjunto signo mezclado octetos
E.MUL.M.16	Multiplicación de conjunto signo mezclado dobletes
E.MUL.M.32	Multiplicación de conjunto signo mezclado cuádruplas
E.MUL.M.64	Multiplicación de conjunto signo mezclado 8-tuplas
E.MUL.P.8	Multiplicación polinomial de conjunto octetos
E.MUL.P.16	Multiplicación polinomial de conjunto dobletes
E.MUL.P.32	Multiplicación polinomial de conjunto cuádruplas
E.MUL.P.64	Multiplicación polinomial de conjunto 8-tuplas
E.MUL.SUM.C.8	Suma de multiplicaciones de conjunto compleja octetos
E.MUL.SUM.C.16	Suma de multiplicaciones de conjunto compleja dobletes
E.MUL.SUM.C.32	Suma de multiplicaciones de conjunto compleja cuádruplas
E.MUL.SUM.M.8	Suma de multiplicaciones de conjunto signo mezclado octetos
E.MUL.SUM.M.16	Suma de multiplicaciones de conjunto signo mezclado dobletes
E.MUL.SUM.M.32	Suma de multiplicaciones de conjunto signo mezclado cuádruplas
E.MUL.SUM.M.64	Suma de multiplicaciones de conjunto signo mezclado 8-tuplas

Fig 18A-1

E.MUL.SUM.U.8	Suma de multiplicaciones de conjunto sin signo octetos
E.MUL.SUM.U.16	Suma de multiplicaciones de conjunto sin signo dobletes
E.MUL.SUM.U.32	Suma de multiplicaciones de conjunto sin signo cuádruplas
E.MUL.SUM.U.64	Suma de multiplicaciones de conjunto sin signo 8-tuplas
E.MUL.U.8	Multiplicación de conjunto sin signo octetos
E.MUL.U.16	Multiplicación de conjunto sin signo dobletes
E.MUL.U.32	Multiplicación de conjunto sin signo cuádruplas
E.MUL.U.64	Multiplicación de conjunto sin signo 8-tuplas

Fig 18A-2

Formato

E.op.size rd=rc,rb

rd=eopsize(rc,rb)

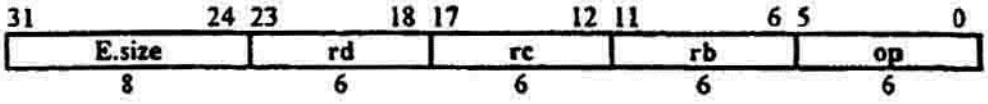


Fig 18B

Definición

```
def mul(size,h,vs,v,i,ws,w,j) as
    mul ← ((vs&v_{size-1+i})^{h-size} || v_{size-1+i..i}) * ((ws&w_{size-1+j})^{h-size} || w_{size-1+j..j})
enddef

def c ← PolyMultiply(size,a,b) as
    p[0] ← 0^{2*size}
    for k ← 0 to size-1
        p[k+1] ← p[k] ^ a_k ? (0^{size-k} || b || 0^k) : 0^{2*size}
    endfor
    c ← p[size]
enddef

def Ensemble(op,size,rd,rc,rb)
    c ← RegRead(rc, 128)
    b ← RegRead(rb, 128)
    case op of
        E.MUL:, E.MUL.C:, EMUL.SUM, E.MUL.SUM.C, E.CON, E.CON.C, E.DIV:
            cs ← bs ← 1
        E.MUL.M:, EMUL.SUM.M, E.CON.M:
            cs ← 0
            bs ← 1
        E.MUL.U:, EMUL.SUM.U, E.CON.U, E.DIV.U, E.MUL.P:
            cs ← bs ← 0
    endcase
    case op of
        E.MUL, E.MUL.U, E.MUL.M:
            for i ← 0 to 64-size by size
                d_{2*(i+size)-1..2*i} ← mul(size,2*size,cs,c,i,bs,b,i)
            endfor
        E.MUL.P:
            for i ← 0 to 64-size by size
                d_{2*(i+size)-1..2*i} ← PolyMultiply(size,c_{size-1+i..i},b_{size-1+i..i})
            endfor
        E.MUL.C:
            for i ← 0 to 64-size by size
                if (i and size) = 0 then
                    p ← mul(size,2*size,1,c,i,1,b,i) - mul(size,2*size,1,c,i+size,1,b,i+size)
                else
                    p ← mul(size,2*size,1,c,i,1,b,i+size) + mul(size,2*size,1,c,i,1,b,i+size)
                endif
                d_{2*(i+size)-1..2*i} ← p
            endfor
        E.MUL.SUM, E.MUL.SUM.U, E.MUL.SUM.M:
            p[0] ← 0^{128}
            for i ← 0 to 128-size by size
                p[i+size] ← p[i] + mul(size,128,cs,c,i,bs,b,i)
            endfor
```

Fig 18C-1

```
a ← p[128]
E.MUL.SUM.C:
    p[0] ← 0^64
    p[size] ← 0^64
    for i ← 0 to 128-size by size
        if (i and size) = 0 then
            p[i+2*size] ← p[i] + mul(size,64,1,c,i,1,b,i)
                               - mul(size,64,1,c,i+size,1,b,i+size)
        else
            p[i+2*size] ← p[i] + mul(size,64,1,c,i,1,b,i+size)
                               + mul(size,64,1,c,i+size,1,b,i)
        endif
    endfor
    a ← p[128+size] || p[128]
E.CON, E.CON.U, E.CON.M:
    p[0] ← 0^128
    for j ← 0 to 64-size by size
        for i ← 0 to 64-size by size
            p[j+size]2*(i+size)-1..2*i ← p[j]2*(i+size)-1..2*i +
                mul(size,2*size,cs,c,i+64-j,bs,b,j)
        endfor
    endfor
    a ← p[64]
E.CON.C:
    p[0] ← 0^128
    for j ← 0 to 64-size by size
        for i ← 0 to 64-size by size
            if ((~i) and j and size) = 0 then
                p[j+size]2*(i+size)-1..2*i ← p[j]2*(i+size)-1..2*i +
                    mul(size,2*size,1,c,i+64-j,1,b,j)
            else
                p[j+size]2*(i+size)-1..2*i ← p[j]2*(i+size)-1..2*i -
                    mul(size,2*size,1,c,i+64-j+2*size,1,b,j)
            endif
        endfor
    endfor
    a ← p[64]
E.DIV:
    if (b = 0) or ( (c = (1||0^63)) and (b = 1^64) ) then
        a ← undefined
```

Fig 18C-2

```
    else
                q ← c / b
                r ← c - q*b
                a ← r63..0 || q63..0
            endif
        E.DIV.U:
            if b = 0 then
                a ← undefined
            else
                q ← (0 || c) / (0 || b)
                r ← c - (0 || q)*(0 || b)
                a ← r63..0 || q63..0
            endif
    endcase
    RegWrite(rd, 128, a)
enddef
```

Fig 18C-3

Definiciones de función con coma flotante

```
def eb ← ebits(prec) as
    case pref of
        16:
            eb ← 5
        32:
            eb ← 8
        64:
            eb ← 11
        128:
            eb ← 15
    endcase
enddef

def eb ← ebias(prec) as
    eb ← 0 || 1^(ebits(prec)-1)
enddef

def fb ← fbits(prec) as
    fb ← prec - 1 - eb
enddef

def a ← F(prec, ai) as
    a.s ← ai_(prec-1)
    ae ← ai_(prec-2..fbits(prec))
    af ← ai_(fbits(prec)-1..0)
    if ae = 1^(ebits(prec)) then
        if af = 0 then
            a.t ← INFINITY
        elseif af_(fbits(prec)-1) then
            a.t ← SNaN
            a.e ← -fbits(prec)
            a.f ← 1 || af_(fbits(prec)-2..0)
        else
            a.t ← QNaN
            a.e ← -fbits(prec)
            a.f ← af
        endif
```

Fig 19-1

```
    elseif ae = 0 then
        if af = 0 then
            a.t ← ZERO
        else
            a.t ← NORM
            a.e ← 1-ebias(prec)-fbits(prec)
            a.f ← 0 || af
        endif
    else
        a.t ← NORM
        a.e ← ae-ebias(prec)-fbits(prec)
        a.f ← 1 || af
    endif
enddef

def a ← DEFAULTQNAN as
    a.s ← 0
    a.t ← QNAN
    a.e ← -1
    a.f ← 1
enddef

def a ← DEFAULTSNAN as
    a.s ← 0
    a.t ← SNAN
    a.e ← -1
    a.f ← 1
enddef

def fadd(a,b) as faddr(a,b,N) enddef

def c ← faddr(a,b,round) as
    if a.t=NORM and b.t=NORM then
        // d,e are a,b with exponent aligned and fraction adjusted
        if a.e > b.e then
            d ← a
            e.t ← b.t
            e.s ← b.s
            e.e ← a.e
            e.f ← b.f || 0^(a.e-b.e)
        else if a.e < b.e then
            d.t ← a.t
            d.s ← a.s
            d.e ← b.e
            d.f ← a.f || 0^(b.e-a.e)
            e ← b
```

Fig 19-2

```
            endif
            c.t ← d.t
            c.e ← d.e
            if d.s = e.s then
                  c.s ← d.s
                  c.f ← d.f + e.f
            elseif d.f > e.f then
                  c.s ← d.s
                  c.f ← d.f - e.f
            elseif d.f < e.f then
                  c.s ← e.s
                  c.f ← e.f - d.f
            else
                  c.s ← r=F
                  c.t ← ZERO
            endif
      // priority is given to b operand for NaN propagation
      elseif (b.t=SNAN) or (b.t=QNAN) then
            c ← b
      elseif (a.t=SNAN) or (a.t=QNAN) then
            c ← a
      elseif a.t=ZERO and b.t=ZERO then
            c.t ← ZERO
            c.s ← (a.s and b.s) or (round=F and (a.s or b.s))
      // NULL values are like zero, but do not combine with ZERO to alter sign
      elseif a.t=ZERO or a.t=NULL then
            c ← b
      elseif b.t=ZERO or b.t=NULL then
            c ← a
      elseif a.t=INFINITY and b.t=INFINITY then
            if a.s ≠ b.s then
                  c ← DEFAULTSNAN // Invalid
            else
                  c ← a
            endif
      elseif a.t=INFINITY then
            c ← a
      elseif b.t=INFINITY then
            c ← b
      else
            assert FALSE // should have covered al the cases above
      endif
enddef

def b ← fneg(a) as
      b.s ← ~a.s
      b.t ← a.t
      b.e ← a.e
      b.f ← a.f
enddef
```

Fig 19-3

```
def fsubr(a,b,round) as faddr(a,fneg(b),round) enddef

def frsub(a,b) as frsubr(a,b,N) enddef

def frsubr(a,b,round) as faddr(fneg(a),b,round) enddef

def c ← fcom(a,b) as
    if (a.t=SNAN) or (a.t=QNAN) or (b.t=SNAN) or (b.t=QNAN) then
        c ← U
    elseif a.t=INFINITY and b.t=INFINITY then
        if a.s ≠ b.s then
            c ← (a.s=0) ? G: L

        else
            c ← E
        endif
    elseif a.t=INFINITY then
        c ← (a.s=0) ? G: L
    elseif b.t=INFINITY then
        c ← (b.s=0) ? G: L
    elseif a.t=NORM and b.t=NORM then
        if a.s ≠ b.s then
            c ← (a.s=0) ? G: L
        else
            if a.e > b.e then
                af ← a.f
                bf ← b.f || 0^(a.e-b.e)
            else
                af ← a.f || 0^(b.e-a.e)
                bf ← b.f
            endif
            if af = bf then
                c ← E
            else
                c ← ((a.s=0) ^ (af > bf)) ? G : L
            endif
        endif
    elseif a.t=NORM then
        c ← (a.s=0) ? G: L
    elseif b.t=NORM then
        c ← (b.s=0) ? G: L
    elseif a.t=ZERO and b.t=ZERO then
        c ← E
    else
        assert FALSE // should have covered al the cases above
    endif
enddef
```

Fig 19-4

```
def c ← fmul(a,b) as
      if a.t=NORM and b.t=NORM then
            c.s ← a.s ^ b.s
            c.t ← NORM
            c.e ← a.e + b.e
            c.f ← a.f * b.f
      // priority is given to b operand for NaN propagation
      elseif (b.t=SNAN) or (b.t=QNAN) then
            c.s ← a.s ^ b.s
            c.t ← b.t
            c.e ← b.e
            c.f ← b.f
    elseif (a.t=SNAN) or (a.t=QNAN) then
            c.s ← a.s ^ b.s
            c.t ← a.t
            c.e ← a.e
            c.f ← a.f
      elseif a.t=ZERO and b.t=INFINITY then
            c ← DEFAULTSNAN // Invalid
      elseif a.t=INFINITY and b.t=ZERO then
            c ← DEFAULTSNAN // Invalid
      elseif a.t=ZERO or b.t=ZERO then
            c.s ← a.s ^ b.s
            c.t ← ZERO
      else
            assert FALSE // should have covered al the cases above
      endif
enddef

def c ← fdivr(a,b) as
      if a.t=NORM and b.t=NORM then
            c.s ← a.s ^ b.s
            c.t ← NORM
            c.e ← a.e - b.e + 256
            c.f ← (a.f || 0^256) / b.f
      // priority is given to b operand for NaN propagation
      elseif (b.t=SNAN) or (b.t=QNAN) then
            c.s ← a.s ^ b.s
            c.t ← b.t
            c.e ← b.e
            c.f ← b.f
      elseif (a.t=SNAN) or (a.t=QNAN) then
            c.s ← a.s ^ b.s
            c.t ← a.t
            c.e ← a.e
            c.f ← a.f
```

Fig 19-5

```
    elseif a.t=ZERO and b.t=ZERO then
        c ← DEFAULTSNAN // Invalid
    elseif a.t=INFINITY and b.t=INFINITY then
        c ← DEFAULTSNAN // Invalid
    elseif a.t=ZERO then
        c.s ← a.s ^ b.s
        c.t ← ZERO
    elseif a.t=INFINITY then
        c.s ← a.s ^ b.s
        c.t ← INFINITY
    else
        assert FALSE // should have covered al the cases above
    endif
enddef

def msb ← findmsb(a) as
    MAXF ← 2^18 // Largest possible f value after matrix multiply
    for j ← 0 to MAXF
        if a[MAXF-1..j] = (0^(MAXF-1-j) || 1) then
            msb ← j
        endif
    endfor
enddef

def ai ← PackF(prec,a,round) as
    case a.t of
        NORM:
            msb ← findmsb(a.f)
            rn ← msb-1-fbits(prec) // lsb for normal
            rdn ← -ebias(prec)-a.e-1-fbits(prec) // lsb if a denormal
            rb ← (rn > rdn) ? rn : rdn
```

Fig 19-6

```
if rb ≤ 0 then
      aifr ← a.f_{msb-1..0} || 0^{-rb}
      eadj ← 0
else
      case round of
            C:
                  s ← 0^{msb-rb} || (~a.s)^{rb}
            F:
                  s ← 0^{msb-rb} || (a.s)^{rb}
            N, NONE:
                  s ← 0^{msb-rb} || ~a.f_{rb} || a.f_{rb-1}^{rb-1}
            X:
                  if a.f_{rb-1..0} ≠ 0 then
                        raise FloatingPointArithmetic // Inexact
                  endif
                  s ← 0
            Z:
                  s ← 0
      endcase
      v ← (0||a.f_{msb..0}) + (0||s)
      if v_{msb} = 1 then
            aifr ← v_{msb-1..rb}
            eadj ← 0
      else
            aifr ← 0^{fbits(prec)}
            eadj ← 1
      endif
endif
aien ← a.e + msb - 1 + eadj + ebias(prec)
if aien ≤ 0 then
      if round = NONE then
            ai ← a.s || 0^{ebits(prec)} || aifr
      else
            raise FloatingPointArithmetic //Underflow
      endif
elseif aien ≥ 1^{ebits(prec)} then
      if round = NONE then
            //default: round-to-nearest overflow handling
            ai ← a.s || 1^{ebits(prec)} || 0^{fbits(prec)}
      else
            raise FloatingPointArithmetic //Underflow
      endif
else
      ai ← a.s || aien_{ebits(prec)-1..0} || aifr
endif
```

Fig 19-7

```
        SNAN:
            if round ≠ NONE then
                raise FloatingPointArithmetic //Invalid
            endif
            if -a.e < fbits(prec) then
                ai ← a.s || 1^ebits(prec) || a.f_{-a.e-1..0} || 0^(fbits(prec)+a.e)
            else
                lsb ← a.f_{-a.e-1-fbits(prec)+1..0} ≠ 0
                ai ← a.s || 1^ebits(prec) || a.f_{-a.e-1..-a.e-1-fbits(prec)+2} || lsb
            endif
        QNAN:
            if -a.e < fbits(prec) then
                ai ← a.s || 1^ebits(prec) || a.f_{-a.e-1..0} || 0^(fbits(prec)+a.e)
            else
                lsb ← a.f_{-a.e-1-fbits(prec)+1..0} ≠ 0
                ai ← a.s || 1^ebits(prec) || a.f_{-a.e-1..-a.e-1-fbits(prec)+2} || lsb
            endif
        ZERO:
            ai ← a.s || 0^ebits(prec) || 0^fbits(prec)
        INFINITY:
            ai ← a.s || 1^ebits(prec) || 0^fbits(prec)
    endcase
defdef

def ai ← fsinkr(prec, a, round) as
    case a.t of
        NORM:
            msb ← findmsb(a.f)
            rb ← -a.e
            if rb ≤ 0 then
                aifr ← a.f_{msb..0} || 0^-rb
                aims ← msb - rb
            else
                case round of
                    C, C.D:
                        s ← 0^(msb-rb) || (~ai.s)^rb
                    F, F.D:
                        s ← 0^(msb-rb) || (ai.s)^rb
                    N, NONE:
                        s ← 0^(msb-rb) || ~ai.f_rb || ai.f_{rb-1}
                    X:
                        if ai.f_{rb-1..0} ≠ 0 then
                            raise FloatingPointArithmetic // Inexact
                        endif
                        s ← 0
                    Z, Z.D:
                        s ← 0
```

Fig 19-8

```
                endcase
                v ← (0||a.f_msb..0) + (0||s)
                if v_msb = 1 then
                    aims ← msb + 1 - rb
                else
                    aims ← msb - rb
                endif
                aifr ← v_aims..rb
            endif
            if aims > prec then
                case round of
                    C.D, F.D, NONE, Z.D:
                        ai ← a.s || (~as)^prec-1

                    C, F, N, X, Z:
                        raise FloatingPointArithmetic // Overflow
                endcase
            elseif a.s = 0 then
                ai ← aifr
            else
                ai ← -aifr
            endif
        ZERO:
            ai ← 0^prec
        SNAN, QNAN:
            case round of
                C.D, F.D, NONE, Z.D:
                    ai ← 0^prec
                C, F, N, X, Z:
                    raise FloatingPointArithmetic // Invalid
            endcase
        INFINITY:
            case round of
                C.D, F.D, NONE, Z.D:
                    ai ← a.s || (~as)^prec-1
                C, F, N, X, Z:
                    raise FloatingPointArithmetic // Invalid
            endcase
    endcase
enddef

def c ← frecrest(a) as
    b.s ← 0
    b.t ← NORM
    b.e ← 0
    b.f ← 1
    c ← fest(fdiv(b,a))
enddef
```

Fig 19-9

```
def c ← frsqrest(a) as
      b.s ← 0
      b.t ← NORM
      b.e ← 0
      b.f ← 1
      c ← fest(fsqr(fdiv(b,a)))
enddef

def c ← fest(a) as
      if (a.t=NORM) then
            msb ← findmsb(a.f)
            a.e ← a.e + msb - 13
            a.f ← a.fmsb..msb-12 || 1
      else
            c ← a
      endif
enddef

def c ← fsqr(a) as
      if (a.t=NORM) and (a.s=0) then
            c.s ← 0
            c.t ← NORM
            if (a.e0 = 1) then
                  c.e ← (a.e-127) / 2
                  c.f ← sqr(a.f || 0^127)
            else
                  c.e ← (a.e-128) / 2
                  c.f ← sqr(a.f || 0^128)
            endif
      elseif (a.t=SNAN) or (a.t=QNAN) or a.t=ZERO or ((a.t=INFINITY) and (a.s=0)) then
            c ← a
      elseif ((a.t=NORM) or (a.t=INFINITY)) and (a.s=1) then
            c ← DEFAULTSNAN // Invalid
      else
            assert FALSE // should have covered al the cases above
      endif
enddef
```

Fig 19-10

Códigos de operación

E.ADD.F.16	Suma de conjunto coma flotante medio
E.ADD.F.16.C	Suma de conjunto coma flotante medio techo
E.ADD.F.16.F	Suma de conjunto coma flotante medio piso
E.ADD.F.16.N	Suma de conjunto coma flotante medio más próximo
E.ADD.F.16.X	Suma de conjunto coma flotante medio exacto
E.ADD.F.16.Z	Suma de conjunto coma flotante medio cero
E.ADD.F.32	Suma de conjunto coma flotante simple
E.ADD.F.32.C	Suma de conjunto coma flotante simple techo
E.ADD.F.32.F	Suma de conjunto coma flotante simple piso
E.ADD.F.32.N	Suma de conjunto coma flotante simple más próximo
E.ADD.F.32.X	Suma de conjunto coma flotante simple exacto
E.ADD.F.32.Z	Suma de conjunto coma flotante simple cero
E.ADD.F.64	Suma de conjunto coma flotante doble
E.ADD.F.64.C	Suma de conjunto coma flotante doble techo
E.ADD.F.64.F	Suma de conjunto coma flotante doble piso
E.ADD.F.64.N	Suma de conjunto coma flotante doble más próximo
E.ADD.F.64.X	Suma de conjunto coma flotante doble exacto
E.ADD.F.64.Z	Suma de conjunto coma flotante doble cero
E.ADD.F.128	Suma de conjunto coma flotante cuádruple
E.ADD.F.128.C	Suma de conjunto coma flotante cuádruple techo
E.ADD.F.128.F	Suma de conjunto coma flotante cuádruple piso
E.ADD.F.128.N	Suma de conjunto coma flotante cuádruple más próximo
E.ADD.F.128.X	Suma de conjunto coma flotante cuádruple exacto
E.ADD.F.128.Z	Suma de conjunto coma flotante cuádruple cero
E.DIV.F.16	División de conjunto coma flotante medio
E.DIV.F.16.C	División de conjunto coma flotante medio techo
E.DIV.F.16.F	División de conjunto coma flotante medio piso
E.DIV.F.16.N	División de conjunto coma flotante medio más próximo
E.DIV.F.16.X	División de conjunto coma flotante medio exacto
E.DIV.F.16.Z	División de conjunto coma flotante medio cero
E.DIV.F.32	División de conjunto coma flotante simple
E.DIV.F.32.C	División de conjunto coma flotante simple techo
E.DIV.F.32.F	División de conjunto coma flotante simple piso
E.DIV.F.32.N	División de conjunto coma flotante simple más próximo
E.DIV.F.32.X	División de conjunto coma flotante simple exacto
E.DIV.F.32.Z	División de conjunto coma flotante simple cero
E.DIV.F.64	División de conjunto coma flotante doble

Fig 20A-1

E.DIV.F.64.C	División de conjunto coma flotante doble techo
E.DIV.F.64.F	División de conjunto coma flotante doble piso
E.DIV.F.64.N	División de conjunto coma flotante doble más próximo
E.DIV.F.64.X	División de conjunto coma flotante doble exacto
E.DIV.F.64.Z	División de conjunto coma flotante doble cero
E.DIV.F.128	División de conjunto coma flotante cuádruple
E.DIV.F.128.C	División de conjunto coma flotante cuádruple techo
E.DIV.F.128.F	División de conjunto coma flotante cuádruple piso
E.DIV.F.128.N	División de conjunto coma flotante cuádruple más próximo
E.DIV.F.128.X	División de conjunto coma flotante cuádruple exacto
E.DIV.F.128.Z	División de conjunto coma flotante cuádruple cero
E.MUL.C.F.16	Multiplicación de conjunto coma flotante medio
E.MUL.C.F.32	Multiplicación de conjunto coma flotante simple
E.MUL.C.F.64	Multiplicación de conjunto coma flotante doble
E.MUL.F.16	Multiplicación de conjunto coma flotante medio
E.MUL.F.16.C	Multiplicación de conjunto coma flotante medio techo
E.MUL.F.16.F	Multiplicación de conjunto coma flotante medio piso
E.MUL.F.16.N	Multiplicación de conjunto coma flotante medio más próximo
E.MUL.F.16.X	Multiplicación de conjunto coma flotante medio exacto
E.MUL.F.16.Z	Multiplicación de conjunto coma flotante medio cero
E.MUL.F.32	Multiplicación de conjunto coma flotante simple
E.MUL.F.32.C	Multiplicación de conjunto coma flotante simple techo
E.MUL.F.32.F	Multiplicación de conjunto coma flotante simple piso
E.MUL.F.32.N	Multiplicación de conjunto coma flotante simple más próximo
E.MUL.F.32.X	Multiplicación de conjunto coma flotante simple exacto
E.MUL.F.32.Z	Multiplicación de conjunto coma flotante simple cero
E.MUL.F.64	Multiplicación de conjunto coma flotante doble
E.MUL.F.64.C	Multiplicación de conjunto coma flotante doble techo
E.MUL.F.64.F	Multiplicación de conjunto coma flotante doble piso
E.MUL.F.64.N	Multiplicación de conjunto coma flotante doble más próximo
E.MUL.F.64.X	Multiplicación de conjunto coma flotante doble exacto
E.MUL.F.64.Z	Multiplicación de conjunto coma flotante doble cero
E.MUL.F.128	Multiplicación de conjunto coma flotante cuádruple
E.MUL.F.128.C	Multiplicación de conjunto coma flotante cuádruple techo
E.MUL.F.128.F	Multiplicación de conjunto coma flotante cuádruple piso
E.MUL.F.128.N	Multiplicación de conjunto coma flotante cuádruple más próximo
E.MUL.F.128.X	Multiplicación de conjunto coma flotante cuádruple exacto
E.MUL.F.128.Z	Multiplicación de conjunto coma flotante cuádruple cero

Fig 20A-2

Selección

class	op	prec				round/trap
add	EADDF	16	32	64	128	NONE C F N X Z
divide	EDIVF	16	32	64	128	NONE C F N X Z
multiply	EMULF	16	32	64	128	NONE C F N X Z
complex multiply	EMUL.CF	16	32	64		NONE

Formato

E.op.prec.round rd=rc,rb

rd=eopprecround(rc,rb)

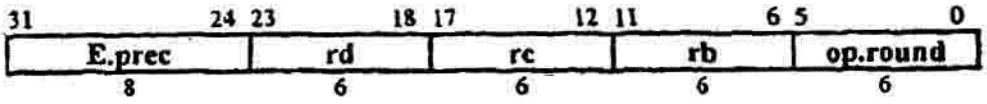


Fig 20B

Definición

```
def mul(size,v,i,w,j) as
    mul ← fmul(F(size,v_size-1+i..i),F(size,w_size-1+j..j))
enddef

def EnsembleFloatingPoint(op,prec,round,ra,rb,rc) as
    c ← RegRead(rc, 128)
    b ← RegRead(rb, 128)
    for i ← 0 to 128-prec by prec
        ci ← F(prec,c_i+prec-1..i)
        bi ← F(prec,b_i+prec-1..i)
        case op of
            E.ADD.F:
                ai ← faddr(ci,bi,round)
            E.MUL.F:
                ai ← fmul(ci,bi)
            E.MUL.C.F:
                if (i and prec) then
                    ai ← fadd(mul(prec,c,i,b,i-prec), mul(prec,c,i-prec,b,i))
                else
                    ai ← fsub(mul(prec,c,l,b,l), mul(prec,c,i+prec,b,i+prec))
                endif
            E.DIV.F.:
                ai ← fdiv(ci,bi)
        endcase
        a_i+prec-1..i ← PackF(prec, ai, round)
    endfor
    RegWrite(rd, 128, a)
enddef
```

Fig 20C

Códigos de operación

E.SUB.F.16	Resta de conjunto coma flotante medio
E.SUB.F.16.C	Resta de conjunto coma flotante medio techo
E.SUB.F.16.F	Resta de conjunto coma flotante medio piso
E.SUB.F.16.N	Resta de conjunto coma flotante medio más próximo
E.SUB.F.16.Z	Resta de conjunto coma flotante medio exacto
E.SUB.F.16.X	Resta de conjunto coma flotante medio cero
E.SUB.F.32	Resta de conjunto coma flotante simple
E.SUB.F.32.C	Resta de conjunto coma flotante simple techo
E.SUB.F.32.F	Resta de conjunto coma flotante simple piso
E.SUB.F.32.N	Resta de conjunto coma flotante simple más próximo
E.SUB.F.32.Z	Resta de conjunto coma flotante simple exacto
E.SUB.F.32.X	Resta de conjunto coma flotante simple cero
E.SUB.F.64	Resta de conjunto coma flotante doble
E.SUB.F.64.C	Resta de conjunto coma flotante doble techo
E.SUB.F.64.F	Resta de conjunto coma flotante doble piso
E.SUB.F.64.N	Resta de conjunto coma flotante doble más próximo
E.SUB.F.64.Z	Resta de conjunto coma flotante doble exacto
E.SUB.F.64.X	Resta de conjunto coma flotante doble cero
E.SUB.F.128	Resta de conjunto coma flotante cuádruple
E.SUB.F.128.C	Resta de conjunto coma flotante cuádruple techo
E.SUB.F.128.F	Resta de conjunto coma flotante cuádruple piso
E.SUB.F.128.N	Resta de conjunto coma flotante cuádruple más próximo
E.SUB.F.128.Z	Resta de conjunto coma flotante cuádruple exacto
E.SUB.F.128.X	Resta de conjunto coma flotante cuádruple cero

Fig 21A

Selección

class	op	prec	round/trap
set	SET. E LG L GE	16 32 64 128	NONE X
subtract	SUB	16 32 64 128	NONE C F N X Z

Formato

E.op.prec.round rd=rb,rc

rd=eopprecround(rb,rc)

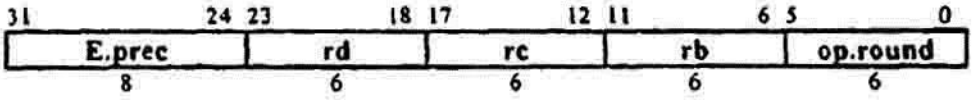


Fig21B

Definición

```
def EnsembleReversedFloatingPoint(op,prec,round,rd,rc,rb) as
      c ← RegRead(rc, 128)
      b ← RegRead(rb, 128)
      for i ← 0 to 128-prec by prec
            ci ← F(prec,ci+prec-1..i)
            bi ← F(prec,bi+prec-1..i)
            ai ← frsubr(ci,-bi, round)
            ai+prec-1..i ← PackF(prec, ai, round)
      endfor
      RegWrite(rd, 128, a)
enddef
```

Fig 21C

Códigos de operación

X.COMPRESS.2	Compresión de barras cruzadas con signo pecks
X.COMPRESS.4	Compresión de barras cruzadas con signo nibbles
X.COMPRESS.8	Compresión de barras cruzadas con signo octetos
X.COMPRESS.16	Compresión de barras cruzadas con signo dobletes
X.COMPRESS.32	Compresión de barras cruzadas con signo cuádruplas
X.COMPRESS.64	Compresión de barras cruzadas con signo 8-tuplas
X.COMPRESS.128	Compresión de barras cruzadas con signo 16-tuplas
X.COMPRESS.U.2	Compresión de barras cruzadas sin signo pecks
X.COMPRESS.U.4	Compresión de barras cruzadas sin signo nibbles
X.COMPRESS.U.8	Compresión de barras cruzadas sin signo octetos
X.COMPRESS.U.16	Compresión de barras cruzadas sin signo dobletes
X.COMPRESS.U.32	Compresión de barras cruzadas sin signo cuádruplas
X.COMPRESS.U.64	Compresión de barras cruzadas sin signo 8-tuplas
X.COMPRESS.U.128	Compresión de barras cruzadas sin signo 16-tuplas
X.EXPAND.2	Expansión de barras cruzadas con signo pecks
X.EXPAND.4	Expansión de barras cruzadas con signo nibbles
X.EXPAND.8	Expansión de barras cruzadas con signo octetos
X.EXPAND.16	Expansión de barras cruzadas con signo dobletes
X.EXPAND.32	Expansión de barras cruzadas con signo cuádruplas
X.EXPAND.64	Expansión de barras cruzadas con signo 8-tuplas
X.EXPAND.128	Expansión de barras cruzadas con signo 16-tuplas
X.EXPAND.U.2	Expansión de barras cruzadas sin signo pecks
X.EXPAND.U.4	Expansión de barras cruzadas sin signo nibbles
X.EXPAND.U.8	Expansión de barras cruzadas sin signo octetos
X.EXPAND.U.16	Expansión de barras cruzadas sin signo dobletes
X.EXPAND.U.32	Expansión de barras cruzadas sin signo cuádruplas
X.EXPAND.U.64	Expansión de barras cruzadas sin signo 8-tuplas
X.EXPAND.U.128	Expansión de barras cruzadas sin signo 16-tuplas
X.ROTL.2	Rotación izquierda de barras cruzadas pecks
X.ROTL.4	Rotación izquierda de barras cruzadas nibbles
X.ROTL.8	Rotación izquierda de barras cruzadas octetos
X.ROTL.16	Rotación izquierda de barras cruzadas dobletes
X.ROTL.32	Rotación izquierda de barras cruzadas cuádruplas
X.ROTL.64	Rotación izquierda de barras cruzadas 8-tuplas
X.ROTL.128	Rotación izquierda de barras cruzadas 16-tuplas
X.ROTR.2	Rotación derecha de barras cruzadas pecks
X.ROTR.4	Rotación derecha de barras cruzadas nibbles
X.ROTR.8	Rotación derecha de barras cruzadas octetos
X.ROTR.16	Rotación derecha de barras cruzadas dobletes

Fig 22A-1

X.ROTR.32	Rotación derecha de barras cruzadas cuádruplas
X.ROTR.64	Rotación derecha de barras cruzadas 8-tuplas
X.ROTR.128	Rotación derecha de barras cruzadas 16-tuplas
X.SHL.2	Desplazamiento izquierda de barras cruzadas pecks
X.SHL.2.O	Desplazamiento izquierda de barras cruzadas pecks con signo verificación desbordamiento
X.SHL.4	Desplazamiento izquierda de barras cruzadas nibbles
X.SHL.4.O	Desplazamiento izquierda de barras cruzadas nibbles con signo verificación desbordamiento
X.SHL.8	Desplazamiento izquierda de barras cruzadas octetos
X.SHL.8.O	Desplazamiento izquierda de barras cruzadas octetos con signo verificación desbordamiento
X.SHL.16	Desplazamiento izquierda de barras cruzadas dobletes
X.SHL.16.O	Desplazamiento izquierda de barras cruzadas dobletes con signo verificación desbordamiento
X.SHL.32	Desplazamiento izquierda de barras cruzadas cuádruplas
X.SHL.32.O	Desplazamiento izquierda de barras cruzadas cuádruplas con signo verificación desbordamiento
X.SHL.64	Desplazamiento izquierda de barras cruzadas 8-tuplas
X.SHL.64.O	Desplazamiento izquierda de barras cruzadas 8-tuplas con signo verificación desbordamiento
X.SHL.128	Desplazamiento izquierda de barras cruzadas 16-tuplas
X.SHL.128.O	Desplazamiento izquierda de barras cruzadas 16-tuplas con signo verificación desbordamiento
X.SHL.U.2.O	Desplazamiento izquierda de barras cruzadas pecks sin signo verificación desbordamiento
X.SHL.U.4.O	Desplazamiento izquierda de barras cruzadas nibbles sin signo verificación desbordamiento
X.SHL.U.8.O	Desplazamiento izquierda de barras cruzadas octetos sin signo verificación desbordamiento
X.SHL.U.16.O	Desplazamiento izquierda de barras cruzadas dobletes sin signo verificación desbordamiento
X.SHL.U.32.O	Desplazamiento izquierda de barras cruzadas cuádruplas sin signo verificación desbordamiento
X.SHL.U.64.O	Desplazamiento izquierda de barras cruzadas 8-tuplas sin signo verificación desbordamiento
X.SHL.U.128.O	Desplazamiento izquierda de barras cruzadas 16-tuplas sin signo verificación desbordamiento
X.SHR.2	Desplazamiento derecha con signo barras cruzadas pecks
X.SHR.4	Desplazamiento derecha con signo barras cruzadas nibbles
X.SHR.8	Desplazamiento derecha con signo barras cruzadas octetos
X.SHR.16	Desplazamiento derecha con signo barras cruzadas dobletes
X.SHR.32	Desplazamiento derecha con signo barras cruzadas cuádruplas
X.SHR.64	Desplazamiento derecha con signo barras cruzadas 8-tuplas
X.SHR.128	Desplazamiento derecha con signo barras cruzadas 16-tuplas
X.SHR.U.2	Desplazamiento derecha sin signo barras cruzadas pecks
X.SHR.U.4	Desplazamiento derecha sin signo barras cruzadas nibbles
X.SHR.U.8	Desplazamiento derecha sin signo barras cruzadas octetos
X.SHR.U.16	Desplazamiento derecha sin signo barras cruzadas dobletes
X.SHR.U.32	Desplazamiento derecha sin signo barras cruzadas cuádruplas
X.SHR.U.64	Desplazamiento derecha sin signo barras cruzadas 8-tuplas
X.SHR.U.128	Desplazamiento derecha sin signo barras cruzadas 16-tuplas

Fig 22A-2

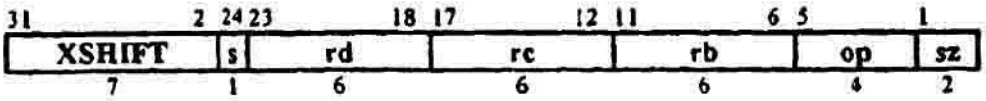
Selección

class	op	size
precision	EXPAND EXPAND.U COMPRESS COMPRESS.U	2 4 8 16 32 64 128
shift	ROTR ROTL SHR SHL SHL.O SHL.U.O SHR.U	2 4 8 16 32 64 128

Formato

X.op.size rd=rc,rb

rd=xopsize(rc,rb)



lsize ← log(size)
s ← lsize$_2$
sz ← lsize$_{1..0}$

Fig 22B

Definición

```
def Crossbar(op,size,rd,rc,rb)
        c ← RegRead(rc, 128)
        b ← RegRead(rb, 128)
        shift ← b and (size-1)
        case op5..2 || 0^2 of
                X.COMPRESS:
                        hsize ← size/2
                        for i ← 0 to 64-hsize by hsize
                                if shift ≤ hsize then
                                        a_{i+hsize-1..i} ← c_{i+i+shift+hsize-1..i+i+shift}
                                else
                                        a_{i+hsize-1..i} ← c_{i+i+size-1}^{shift-hsize} || c_{i+i+size-1..i+i+shift}
                                endif
                        endfor
                        a_{127..64} ← 0
                X.COMPRESS.U:
                        hsize ← size/2
                        for i ← 0 to 64-hsize by hsize
                                if shift ≤ hsize then
                                        a_{i+hsize-1..i} ← c_{i+i+shift+hsize-1..i+i+shift}
                                else
                                        a_{i+hsize-1..i} ← 0^{shift-hsize} || c_{i+i+size-1..i+i+shift}
                                endif
                        endfor
                        a_{127..64} ← 0
                X.EXPAND:
                        hsize ← size/2
                        for i ← 0 to 64-hsize by hsize
                                if shift ≤ hsize then
                                        a_{i+i+size-1..i+i} ← c_{i+hsize-1}^{hsize-shift} || c_{i+hsize-1..i} || 0^{shift}
                                else
                                        a_{i+i+size-1..i+i} ← c_{i+size-shift-1..i} || 0^{shift}
                                endif
                        endfor
```

Fig 22C

```
X.EXPAND.U:
        hsize ← size/2
        for i ← 0 to 64-hsize by hsize
            if shift ≤ hsize then
                a_{i+i+size-1..i+i} ← 0^{hsize-shift} || c_{i+hsize-1..i} || 0^{shift}
            else
                a_{i+i+size-1..i+i} ← c_{i+size-shift-1..i} || 0^{shift}
            endif
        endfor
    X.ROTL:
        for i ← 0 to 128-size by size
            a_{i+size-1..i} ← c_{i+size-1-shift..i} || c_{i+size-1..i+size-1-shift}
        endfor

X.ROTR:
    for i ← 0 to 128-size by size
        a_{i+size-1..i} ← c_{i+shift-1..i} || c_{i+size-1..i+shift}
    endfor
X.SHL:
    for i ← 0 to 128-size by size
        a_{i+size-1..i} ← c_{i+size-1-shift..i} || 0^{shift}
    endfor
X.SHL.O:
    for i ← 0 to 128-size by size
        if c_{i+size-1..i+size-1-shift} ≠ c^{shift+1}_{i+size-1-shift} then
            raise FixedPointArithmetic
        endif
        a_{i+size-1..i} ← c_{i+size-1-shift..i} || 0^{shift}
    endfor
```

Fig 22C-2

```
            X.SHL.U.O:
                for i ← 0 to 128-size by size
                    if c<sub>i+size-1..i+size-shift</sub> ≠ 0^shift then
                        raise FixedPointArithmetic
                    endif
                    a<sub>i+size-1..i</sub> ← c<sub>i+size-1-shift..i</sub> || 0^shift
                endfor
            X.SHR:
                for i ← 0 to 128-size by size
                    a<sub>i+size-1..i</sub> ← c^shift<sub>i+size-1</sub> || c<sub>i+size-1..i+shift</sub>
                endfor
            X.SHR.U:
                for i ← 0 to 128-size by size
                    a<sub>i+size-1..i</sub> ← 0^shift || c<sub>i+size-1..i+shift</sub>
                endfor
        endcase
        RegWrite(rd, 128, a)
enddef
```

Fig 22C-3

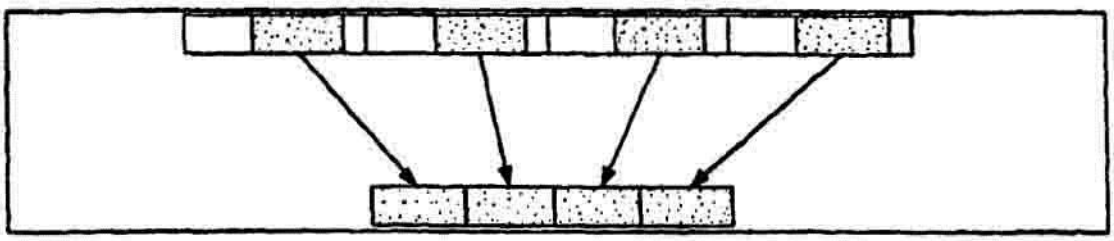

Comprimir 32 bits a 16, con desplazamiento a derecha de 4 bits

Fig 22D

Formato

X.EXTRACT ra=rd,rc,rb

ra=xextract(rd,rc,rb)

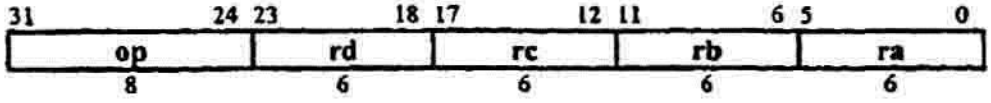
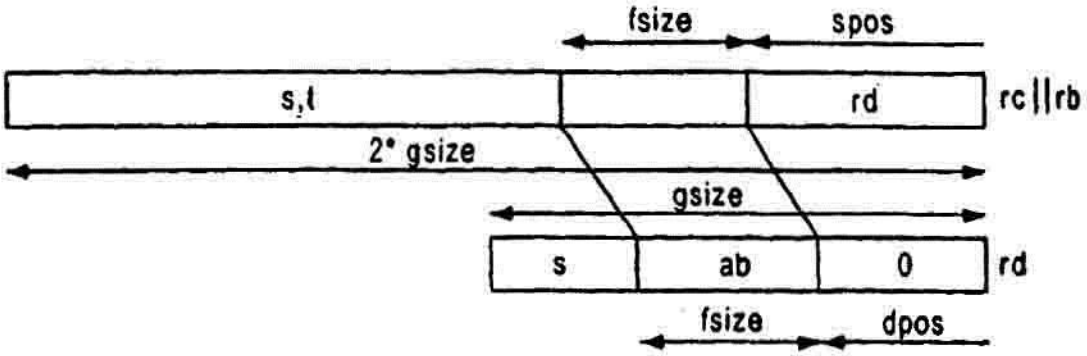


Fig 23A

Definición

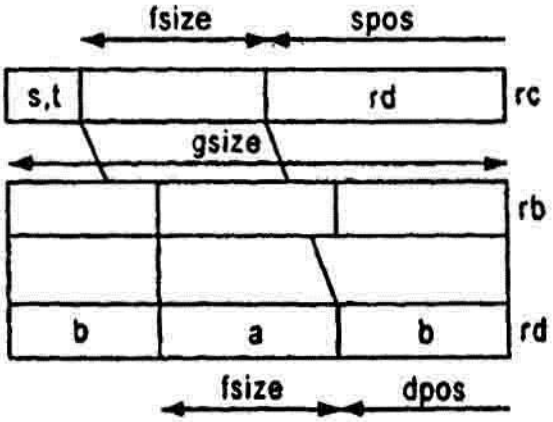
```
def CrossbarExtract(op,ra,rb,rc,rd) as
    d ← RegRead(rd, 128)
    c ← RegRead(rc, 128)
    b ← RegRead(rb, 128)
    case b8..0 of
        0..255:
            gsize ← 128
        256..383:
            gsize ← 64
        384..447:
            gsize ← 32
        448..479:
            gsize ← 16
        480..495:
            gsize ← 8
        496..503:
            gsize ← 4
        504..507:
            gsize ← 2
        508..511:
            gsize ← 1
    endcase
    m ← b12
    as ← signed ← b14
    h ← (2-m)*gsize
    spos ← (b8..0) and ((2-m)*gsize-1)
    dpos ← (0 || b23..16) and (gsize-1)
    sfsize ← (0 || b31..24) and (gsize-1)
    tfsize ← (sfsize = 0) or ((sfsize+dpos) > gsize) ? gsize-dpos : sfsize
    fsize ← (tfsize + spos > h) ? h - spos : tfsize
    for i ← 0 to 128-gsize by gsize
        case op of
            X.EXTRACT:
                if m then
                    p ← dgsize+i-1..i
                else
                    p ← (d || c)2*(gsize+i)-1..2*i
                endif
        endcase
        v ← (as & ph-1)||p
        w ← (as & vspos+fsize-1)gsize-fsize-dpos || vfsize-1+spos..spos || 0dpos
        if m then
            asize-1+i..i ← cgsize-1+i..dpos+fsize+i || wdpos+fsize-1..dpos || cdpos-1+1..i
        else
            asize-1+i..i ← w
        endif
    endfor
    RegWrite(ra, 128, a)
enddef
```

Fig 23B



Extracción barras cruzadas

Fig 23C



Fusión extracción barras cruzadas

Fig 23D

Fig 24A **Galois de resolución amplia**

wminor	*galpoly	*galpoly	solvepar	wsolveg
8	6	6	6	6

Resuelve L*2 = W mod z8 en 8 iteraciones**

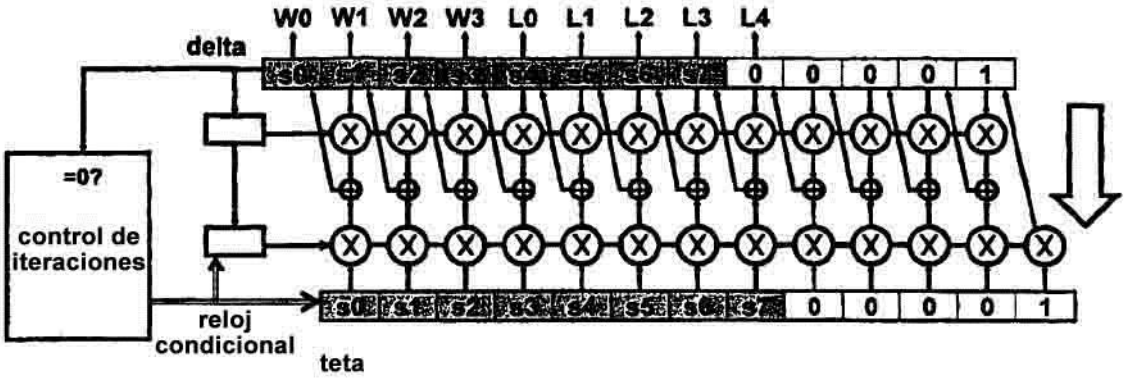
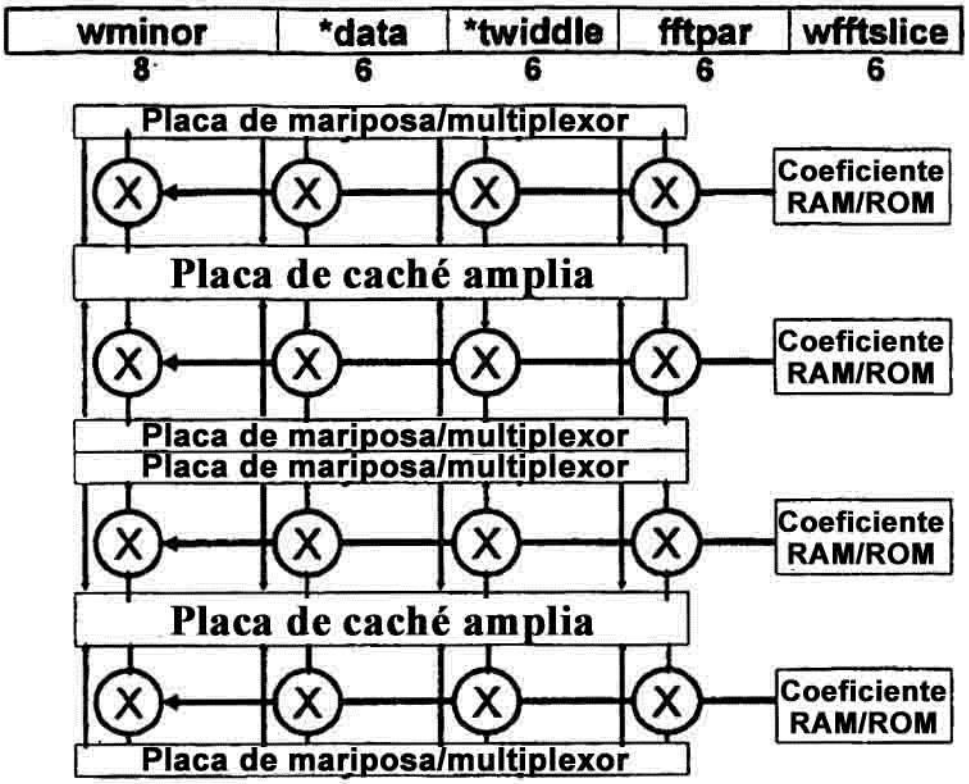


Fig 24B **Galois de resolución amplia**

```
static     v8_t      wsolveg(v8_t hh, v8_t syndrome, v8_t *omega)

for ( r=0; r < N_PARITY; r++)                                   /*: A + 16*(B+A):*/
  {
   delta = _xcopyl8(delta0,0);                                  /*: 16*X :*/
   delta0s = _castv8(_xshrm128(_castv128(delta0),_castv128(delta1),8));  /*: 16*X :*/
   delta1s = _reindex8(delta1, -1);                             /*: 16*X :*/
   delta0 = _gxor8(_emulg8(gamma, delta0s, hh),_emulg8(delta,theta0, hh));  /*: 16*(2*E+G) :*/
   delta1 = _gxor8(_emulg8(gamma, delta1s, hh),_emulg8(delta,theta1, hh));  /*: 16*(2*E+G) :*/
   s = _gsetandne8(delta, _gsetge8(k,_gzero8));                 /*: 16*2*G :*/
   theta0 = _gmux8(s,delta0s,theta0);                           /*: 16*G :*/
   theta1 = _gmux8(s,delta1s,theta1);                           /*: 16*G :*/
   gamma = _gmux8(s,delta,gamma);                               /*: 16*G :*/
   k = _gmux8(s,_gnot8(k),_gadd8(k,_gone8));                    /*: 16*3*G :*/
  }
  lambda = _xselect8(delta1,delta0,USE_VCONST(lambdai));        /*: X :*/
  *omega = _castv8(_xwithdrawu128(_castv128(delta0),64,0));     /*: X :*/
```

Fig 25A **Rebanada de FFT amplia**



```
/*****************************************************************************/
/* DSP library module:  Inverse FFT, selectable length,                      */
/*                                   16-bit complex integers,                */
/*                                   split-radix algorithm                   */
/*                                                                           */
/*****************************************************************************/

/* include files */
#include <stdio.h>
#include "broadmx.h"
#include "affirm.h"
#include "dspFFTud.h"
#include <math.h>

#define  SHOW            0


/* typed version of _gboolean: should be part of gops */
static INLINE v16_t      _gboolean16(v16_t src1, v16_t src2, v16_t src3, int imm)
{
 return _gboolean(src1.rr, src2.rr, src3.rr, imm).v16;
}

/*----------------------------------------
 * I * (a - b) / 2
 */
static   inline   vc16_t  _sub_mul_by_i_c16(vc16_t aa, vc16_t bb)
{
   v16_t muxmask          = _castv16(_gcopyi32(0xFFFF));
   v16_t xx;

   /* xx = _gsubh16n(_gmux16(muxmask,aa,bb),_gmux16(muxmask,bb,aa)); */
   xx = _gsubh16n(_gxor16(muxmask,bb),_gxor16(muxmask,aa));
   xx = _xswizzle16(xx, 7, 1);
   return xx;
}
```

Fig 25B

```
/*-----------------------------------------
 * Perform 4 independent 4-point fft's
 *
 * x0..x3 holds the input to the transform, 4 sets of 4 complex numbers.
 * Each set is inverse-fourier transformed independently of the others.
 * The results appear in x0..x3. The original values of y0..y3 are corrupted.
 */
#define  QUAD_IFFT_4PT_c16(_y0,_y1,_y2,_y3, _x0,_x1,_x2,_x3)  {\
   _y0 = _gaddh16n(_x0,_x2);                \
   _y1 = _gaddh16n(_x1,_x3);                \
   _y2 = _gsubh16n(_x0,_x2);                \
   _y3 = _sub_mul_by_i_c16(_x1,_x3);        \
   _x0 = _gaddh16n(_y0,_y1);                \
   _x2 = _gsubh16n(_y0,_y1);                \
   _x1 = _gaddh16n(_y2,_y3);                \
   _x3 = _gsubh16n(_y2,_y3);                \
}

/*-----------------------------------------
 * Perform 4 independent 2-point fft's
 *
 * x0..x1 holds the input to the transform, 4 sets of 2 complex numbers.
 * Each set is inverse-fourier transformed independently of the others.
 * The results appear in y0..y1.
 */
#define  QUAD_IFFT_2PT_c16(_y0,_y1, _x0,_x1) {\
   _y0 = _gaddh16n(_x0,_x1);                \
   _y1 = _gsubh16n(_x0,_x1);                \
}
```

Fig 25B (cont.)

```
static int _wfftslicect6(vc16_t *dp, vc16_t *tp, int dn, int ds, int tn, int radix, int reorder, int extract)
{
 int i,j,ii, logmost;
 vc16_t *dwp, *twp;
 vc16_t t0,t1,t2,t3, d0,d1,d2,d3, p0,p1,p2,p3, z0,z1,z2,z3, m, n;
                                . .
 if(SHOW) printf("extract = %d\n",extract&0xf);
 n = m = _gcopyi16(0);
 if (radix==4) {
  if (ds==1) {
   for (twp=tp,i=0; i<tn; dp++,twp++,i+=NELEMC16) {
        t0 = twp[0];
        d0 = dp[0];
        p0 = _emulx16(t0,d0,extract);
        z0 = _xshri16(p0,1);
        n = _gboolean16(n,p0,z0,0xf6);
        d0 = _vput16(d0,0,(_vget16(p0,0)+_vget16(p0,2)+_vget16(p0,4)+_vget16(p0,6)+2)>>2);
        d0 = _vput16(d0,1,(_vget16(p0,1)+_vget16(p0,3)+_vget16(p0,5)+_vget16(p0,7)+2)>>2);
        d0 = _vput16(d0,4,(_vget16(p0,0)-_vget16(p0,2)+_vget16(p0,4)-_vget16(p0,6)+2)>>2);
        d0 = _vput16(d0,5,(_vget16(p0,1)-_vget16(p0,3)+_vget16(p0,5)-_vget16(p0,7)+2)>>2);
        d0 = _vput16(d0,2,(_vget16(p0,0)-_vget16(p0,3)-_vget16(p0,4)+_vget16(p0,7)+2)>>2);
        d0 = _vput16(d0,3,(_vget16(p0,1)+_vget16(p0,2)-_vget16(p0,5)-_vget16(p0,6)+2)>>2);
        d0 = _vput16(d0,6,(_vget16(p0,0)+_vget16(p0,3)-_vget16(p0,4)-_vget16(p0,7)+2)>>2);
        d0 = _vput16(d0,7,(_vget16(p0,1)-_vget16(p0,2)-_vget16(p0,5)+_vget16(p0,6)+2)>>2);
        z0 = _xshri16(d0,1);
        m = _gboolean16(m,d0,z0,0xf6);
        dp[0] = d0;
   }
  } else {
   ii = ds / NELEMC16;
   for (twp=tp,i=0; i<tn; dp++,twp++,i+=4*NELEMC16) {
        t0 = twp[0*ii];
        t1 = twp[1*ii];
        t2 = twp[2*ii];
        t3 = twp[3*ii];
        for (dwp=dp,j=0; j<dn; dwp+=4*ii,j+=4*ds) {
         d0 = dwp[0*ii];
         d1 = dwp[1*ii];
         d2 = dwp[2*ii];
         d3 = dwp[3*ii];
         d0 = _emulx16(t0,d0,extract); // can be eextract
         d1 = _emulx16(t1,d1,extract);
         d2 = _emulx16(t2,d2,extract);
         d3 = _emulx16(t3,d3,extract);
         z0 = _xshri16(d0,1);
         z1 = _xshri16(d1,1);
         z2 = _xshri16(d2,1);
         z3 = _xshri16(d3,1);
         n = _gboolean16(n,d0,z0,0xf6);
         n = _gboolean16(n,d1,z1,0xf6);
         n = _gboolean16(n,d2,z2,0xf6);
         n = _gboolean16(n,d3,z3,0xf6);
```

Fig 25B (cont.)

```
            QUAD_IFFT_4PT_c16(p0,p1,p2,p3, d0,d1,d2,d3);
            z0 = _xshri16(d0,1);
            z1 = _xshri16(d1,1);
            z2 = _xshri16(d2,1);
            z3 = _xshri16(d3,1);
            m = _gboolean16(m,d0,z0,0xf6);
            m = _gboolean16(m,d1,z1,0xf6);
            m = _gboolean16(m,d2,z2,0xf6);
            m = _gboolean16(m,d3,z3,0xf6);
            dwp[0*ii] = d0;
            dwp[1*ii] = d1;
            dwp[2*ii] = d2;
            dwp[3*ii] = d3;
          }
    }
  }
} else if (radix==2) {
  ii = ds / NELEMC16;
  for (twp=tp,i=0; i<tn; dp++,twp++,i+=2*NELEMC16) {
    t0 = twp[0*ii];
    t1 = twp[1*ii];
    for (dwp=dp,j=0; j<dn; dwp+=2*ii,j+=2*ds) {
          d0 = dwp[0*ii];
          d1 = dwp[1*ii];
          p0 = _emulx16(t0,d0,extract); // can be eextract
          p1 = _emulx16(t1,d1,extract);
          z0 = _xshri16(p0,1);
          z1 = _xshri16(p1,1);
          n = _gboolean16(n,p0,z0,0xf6);
          n = _gboolean16(n,p1,z1,0xf6);
          QUAD_IFFT_2PT_c16(d0,d1, p0,p1);
          z0 = _xshri16(d0,1);
          z1 = _xshri16(d1,1);
          m = _gboolean16(m,d0,z0,0xf6);
          m = _gboolean16(m,d1,z1,0xf6);
          dwp[0*ii] = d0;
          dwp[1*ii] = d1;
    }
  }
} else {
    for (j=0; j<dn; dp++,tp++,j+=NELEMC16) {
          *dp = d0 = *tp;
          z0 = _xshri16(d0,1);
          m = _gboolean16(m,d0,z0,0xf6);
    }
    n = m;
}
```

Fig 25B (cont.)

```
 n = _gor16(n,_castv16(_xshriu128(_castv128(n),64)));
 n = _gor16(n,_castv16(_xshriu128(_castv128(n),32)));
 n = _gor16(n,_castv16(_xshriu128(_castv128(n),16)));
 logmost = _vget16(_elogmost16(n),0);
 if(SHOW) printf("logmost = %d (after mulx)\n",logmost);
 m = _gor16(m,_castv16(_xshriu128(_castv128(m),64)));
 m = _gor16(m,_castv16(_xshriu128(_castv128(m),32)));
 m = _gor16(m,_castv16(_xshriu128(_castv128(m),16)));
 logmost = _vget16(_elogmost16(m),0);
 if(SHOW) printf("logmost = %d (after addh)\n",logmost);
 return logmost;
}

static    cplxi16  const    exptab[][4] =
#define IFFT_COEFS_16
#include "dspIFFT-coefs.h"
#undef IFFT_COEFS_16
;

static    void      make_twiddle(cplxi16 *tw, int ni, int nj, int len, int show)
{
  int               ii, jj;

  for(ii = 0;  ii < ni;  ++ii) {
        for(jj = 0;  jj < nj;  ++jj) {
          tw->re = rint(-32768*cos(2*M_PI/len*ii*jj));
          tw->im = rint(-32768*sin(2*M_PI/len*ii*jj));
          if(show) printf("twiddle[%d][%d] = (%7d,%7d)\n", ii, jj,  tw->re, tw->im);
          ++tw;
        }
  }
}

int dspInverseFourier_slice_c16(cplxi16 *out, cplxi16 const *in, int len)
{
 int logmost, extract, scale;
 static cplxi16 twidtab[12][1024];
 int i, j, k, l;
 int ds, tn;

 for(i = 0; i < len;  ++i) {
        twidtab[0][i].re = -32768;
        twidtab[0][i].im = 0;
 }
 make_twiddle(&twidtab[1][0], 4, 4, 16, 0);
 make_twiddle(&twidtab[2][0], 4, 16, 64, 0);
 make_twiddle(&twidtab[3][0], 4, 64, 256, 0);
 make_twiddle(&twidtab[4][0], 2, 256, 512, 0);
```

Fig 25B (cont.)

```
scale = 0;
logmost = 0;
 if(len == 4) {
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)in, len, 0, 0, 1, 0, 0);
        scale = 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[0], len, 1, len, 4, 0, extract);
 } else if(len == 16) {
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)in, len, 0, 0, 1, 0, 0);
        scale = 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[0], len, 1, len, 4, 0, extract);
        scale += 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[1], len, 4, 16, 4, 0, extract);
 } else if(len == 64) {
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)in, len, 0, 0, 1, 0, 0);
        scale = 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[0], len, 1, len, 4, 0, extract);
        scale += 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[1], len, 4, 16, 4, 0, extract);
        scale += 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[2], len, 16, 64, 4, 0, extract);
        scale -= 2;
 } else if(len == 256) {
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)in, len, 0, 0, 1, 0, 0);
        scale = 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[0], len, 1, len, 4, 0, extract);
        scale += 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[1], len, 4, 16, 4, 0, extract);
        scale += 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[2], len, 16, 64, 4, 0, extract);
        scale += 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[3], len, 64, 256, 4, 0, extract);
        scale -= 4;
```

Fig 25B (cont.)

```
} else if(len == 512) {
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)in, len, 0, 0, 1, 0, 0);
        scale = 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[0], len, 1, len, 4, 0, extract);
        scale += 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[1], len, 4, 16, 4, 0, extract);
        scale += 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[2], len, 16, 64, 4, 0, extract);
        scale += 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[3], len, 64, 256, 4, 0, extract);
        scale += 16 - logmost;
        extract = (1<<14) + (1<<13) + (2<<9) + (512-4*16+logmost+1);
        logmost = _wfftslicec16((vc16_t *)out, (vc16_t *)twidtab[4], len, 256, 512, 2, 0, extract);
        scale -= 7;
  }
  if(SHOW) printf("scale = %d\n",scale);
  return scale;
```

Fig 25B (cont.)

Formato

W.CONVOLVE.X.order ra=rc,rd,rb

ra=wop(rc,rd,rb)

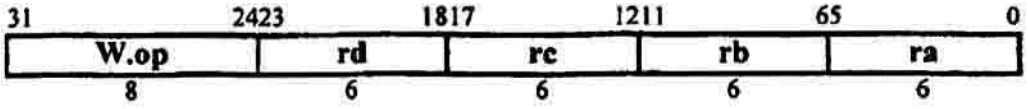


Fig 26A

Definición

```
def mul(size,h,vs,v,i,ws,w,j) as
    mul ← ((vs&v_{size-1+i})^{h-size} || v_{size-1+i..i}) * ((ws&w_{size-1+j})^{h-size} || w_{size-1+j..j})
enddef

def WideConvolveExtract(op,ra,rb,rc,rd)
    d ← RegRead(rd, 64)
    c ← RegRead(rc, 64)
    b ← RegRead(rb, 128)
    case b_{8..0} of
        0..255:
            sgsize ← 128
        256..383:
            sgsize ← 64
        384..447:
            sgsize ← 32
        448..479:
            sgsize ← 16
        480..495:
            sgsize ← 8
        496..503:
            sgsize ← 4
        504..507:
            sgsize ← 2
        508..511:
            sgsize ← 1
    endcase
    l ← b_{11}
    m ← b_{12}
    n ← b_{13}
    signed ← b_{14}
    x ← b_{15}
    if (c_{2..0} ≠ 0) or (d_{2..0} ≠ 0) then
        raise ReservedInstruction
    endif
    cwsize ← (c and (0-c)) || 0^5
    ct ← c and (c-1)
    cmsize ← (ct and (0-ct)) || 0^4
    ca ← ct and (ct-1)
    lcmsize ← log(cmsize)
    lcwsize ← log(cwsize)
    cm ← LoadMemory(c,ca,cmsize,order)
    dwsize ← (d and (0-d)) || 0^5
    dt ← d and (d-1)
    dmsize ← (dt and (0-dt)) || 0^4
    da ← dt and (dt-1)
    ldmsize ← log(dmsize)
    ldwsize ← log(dwsize)
    dm ← LoadMemory(d,da,dmsize,order)
    if (sgsize < 8) or (sgsize > wsize/2) then
        raise ReservedInstruction
```

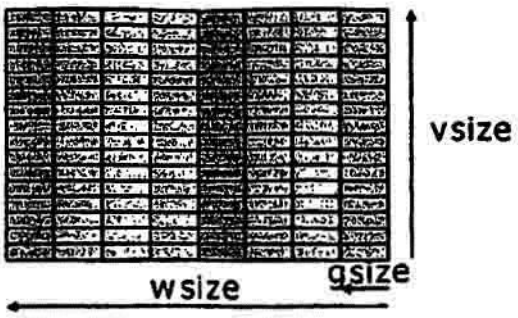
Fig 26B

```
endif
gsize ← sgsize
lgsize ← log(gsize)
case op of
        W.CONVOLVE.X.B:
                order ← B
        W.CONVOLVE.X.L:
                order ← L
endcase
cs ← signed
ds ← signed ^ m
zs ← signed or m or n
zsize ← gsize*(x+1)
h ← (2*gsize) + ldmsize - lgsize
spos ← (b8..0) and (2*gsize-1)
dpos ← (0 || b23..16) and (zsize-1)
r ← spos
sfsize ← (0 || b31..24) and (zsize-1)
tfsize ← (sfsize = 0) or ((sfsize+dpos) > zsize) ? zsize-dpos : sfsize
fsize ← (tfsize + spos > h+1) ? h+1 - spos : tfsize
if (b10..9 = Z) and not zs then
        md ← F
else
        md ← b10..9
endif
mzero ← b95..64
mpos ← b63..32
oo ← mpos || 0^3
ox ← oo[lcwsize-1..lgsize]
oy ← oo[lcmsize-1..lcwsize]
zz ← (~mzero) || 1^3
zx ← zz[ldwsize-1..lgsize]
zy ← zz[ldmsize-1..ldwsize]
```

Fig 26B (Cont.)

```
        for k ← 0 to 128-zsize by zsize
            i ← k*gsize/zsize
            ix ← i[lcwsize-1..lgsize]
            iy ← i[lcmsize-1..lcwsize]
            q[0] ← 0^h
            for j ← 0 to dmsize-gsize by gsize
                jj ← n and j[lgsize] and not i[lgsize]
                jx ← j[ldwsize-1..lgsize]
                jy ← j[ldmsize-1..ldwsize]
                u ← (oy+iy-jy)[lcmsize-lcwsize-1..0] || (ox+ix-jx-2*jj)[lcmsize-lcwsize-1..0] || 0^lgsize
                if (jx>zx) or (jy>zy) and (dm[lgsize-1+j..j]0 ) and undefined then
                    q[j+gsize] ← q[j]
                else
                    if jj then
                        q[j+gsize] ← q[j] - mul(gsize,h,cs,cm,u,ds,dm,j)
                    else
                        q[j+gsize] ← q[j] + mul(gsize,h,cs,cm,u,ds,dm,j)
                    endif
                endif
            endfor
            p ← q[dmsize]
            case rnd of
                none, N:
                    s ← 0^(h-r) || ~p_r || ~p_r^(r-1)
                Z:
                    s ← 0^(h-r) || p[h-1]
                F:
                    s ← 0^h
                C:
                    s ← 0^(h-r) || 1^r
            endcase
            v ← ((zs & p_(h-1))||p) + (0||s)
            if (v[h..r+fsize] = (zs & v_(r+fsize-1))^(h+1-r-fsize)) or not I then
                w ← (zs & v_(r+fsize-1))^(zsize-fsize-dpos) || v[fsize-1+r..r] || 0^dpos
            else
                w ← (zs ? (v_h^(zsize-fsize-dpos+1) || ~v_h^(fsize-1)) : 0^(zsize-fsize-dpos) || 1^fsize) || 0^dpos
            endif
            z[zsize-1+k..k] ← w
        endfor
        RegWrite(ra, 128, z)
    enddef
```

Fig 26B (Cont.)

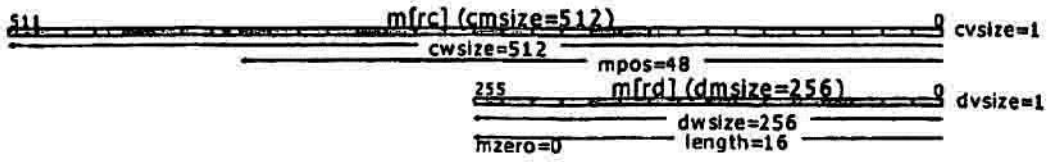


msize = wsize * vsize

spec = base + msize/16 + wsize/32

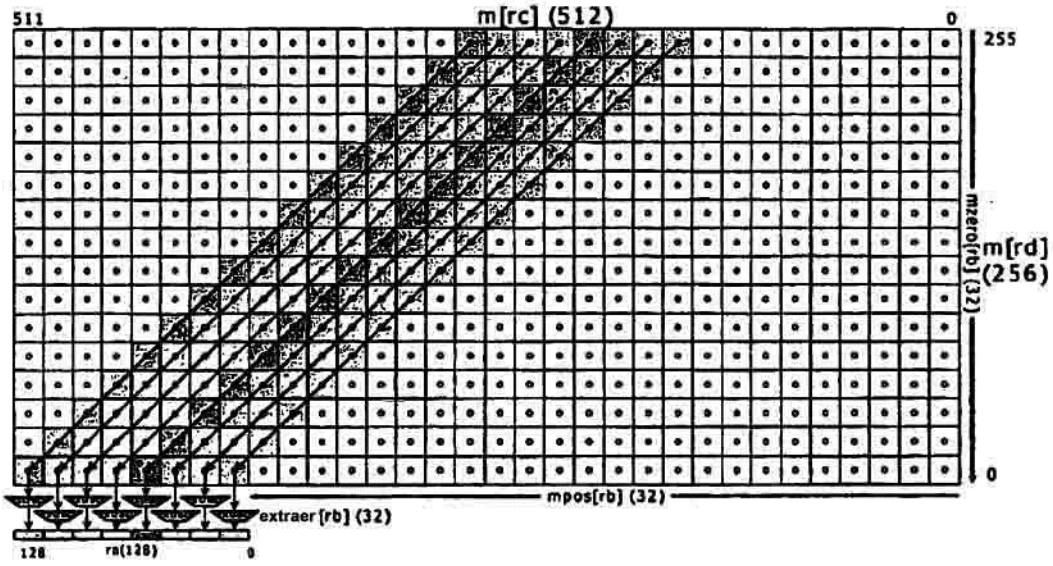
Especificador de operando amplio para extracción convolución amplia

Fig 26C



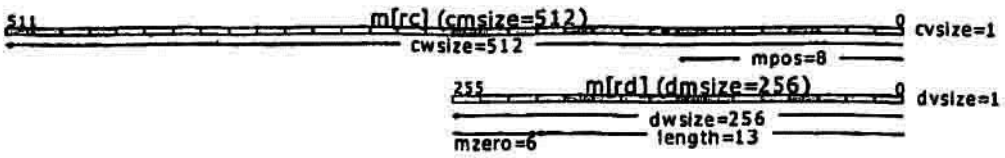
Dobletes de extracción convolución amplia

Fig 26D



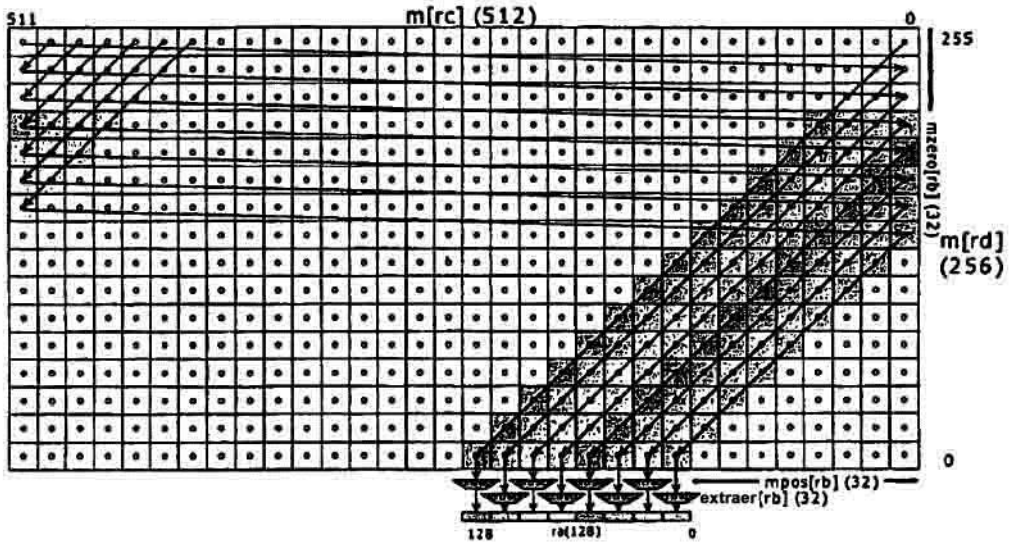
Dobletes de extracción convolución amplia

Fig 26E



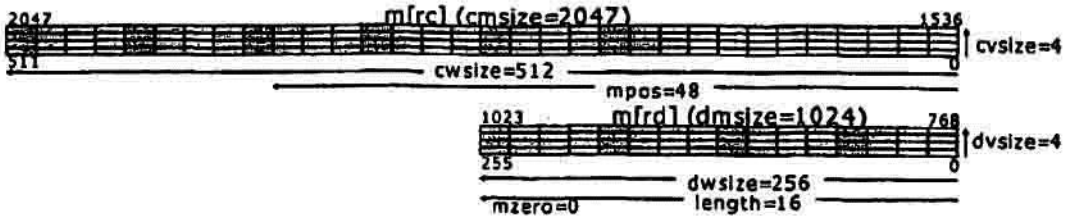
Dobletes de extracción convolución amplia

Fig 26F



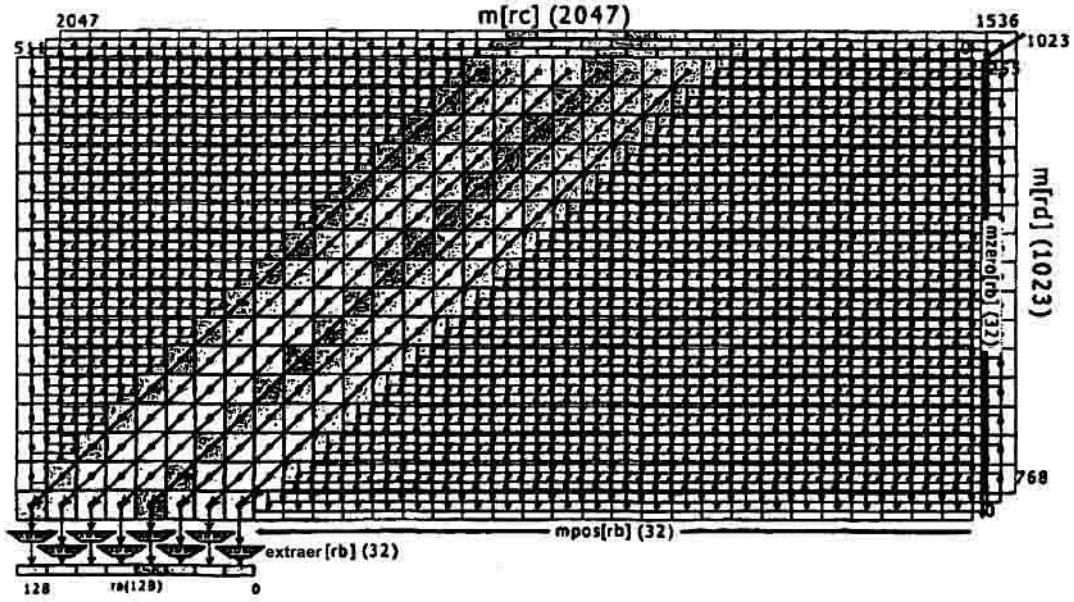
Dobletes de extracción convolución amplia

Fig 26G



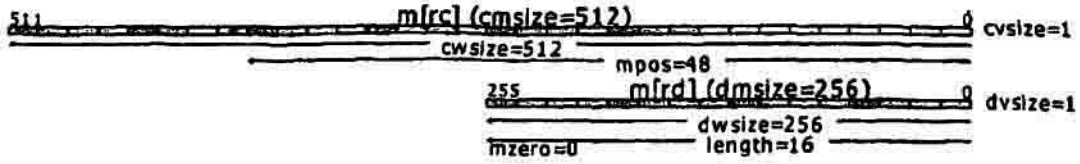
Dobletes de extracción convolución amplia bidimensional

Fig 26H



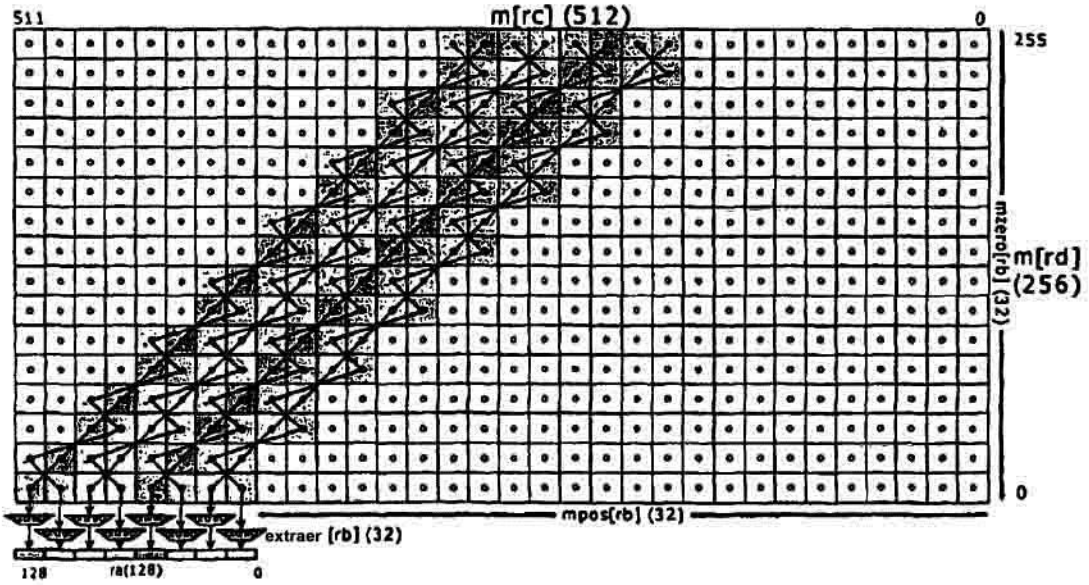
Dobletes de extracción convolución amplia bidimensional

Fig 26I



Dobletes complejos de estación de convolución amplia

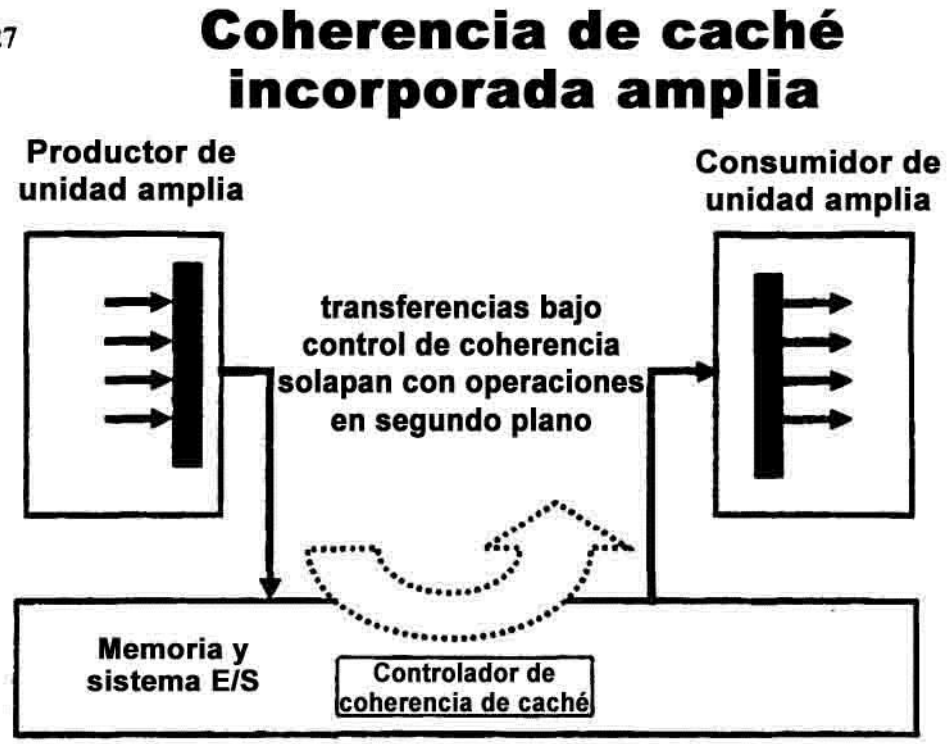
Fig 26J



Dobletes complejos de extracción convolución amplia

Fig 26K

Fig 27


Coherencia de caché incorporada amplia
Productor de unidad amplia
Consumidor de unidad amplia
transferencias bajo control de coherencia solapan con operaciones en segundo plano
Memoria y sistema E/S
Controlador de coherencia de caché