(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 988 858**

(51) Int. Cl.:

G06F 9/38 (2008.01)
G06F 15/78 (2006.01)
G06F 9/30 (2008.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **13.02.2019 E 19157043 (1)**
(97) Fecha y número de publicación de la concesión europea: **29.05.2024 EP 3547120**

(54) Título: **Sistemas y métodos para implementar operaciones de teselas en cadena**

(30) Prioridad:

30.03.2018 US 201815942201

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
21.11.2024

(73) Titular/es:

INTEL CORPORATION (100.0%)
2200 Mission College Blvd.
Santa Clara, CA 95054, US

(72) Inventor/es:

HUGHES, CHRISTOPHER J.;
HEINECKE, ALEXANDER F.;
VALENTINE, ROBERT;
TOLL, BRET;
CORBAL, JESUS y
OULD-AHMED-VALL, ELMOUSTAPHA

(74) Agente/Representante:

LEHMANN NOVO, María Isabel

Aviso: En el plazo de nueve meses a contar desde la fecha de publicación en el Boletín Europeo de Patentes, de la mención de concesión de la patente europea, cualquier persona podrá oponerse ante la Oficina Europea de Patentes a la patente concedida. La oposición deberá formularse por escrito y estar motivada; sólo se considerará como formulada una vez que se haya realizado el pago de la tasa de oposición (art. 99.1 del Convenio sobre Concesión de Patentes Europeas).

DESCRIPCIÓN

Sistemas y métodos para implementar operaciones de teselas en cadena

CAMPO DE LA INVENCIÓN

El campo de la invención se refiere en general a la arquitectura de procesadores informáticos y, más específicamente, a sistemas y métodos para implementar operaciones de teselas en cadena.

ANTECEDENTES

Las matrices son cada vez más importantes en muchas tareas informáticas, tales como el aprendizaje automático y otros procesamientos de datos masivos. El documento US 2016/0179551 A1 se refiere a la canalización de instrucciones en desorden y más específicamente con problemas que surgen de instrucciones de diferentes tipos de latencia. Se pueden encadenar al menos dos instrucciones cortas juntas para llegar a una latencia total general que sea igual a la latencia de una instrucción de larga duración. En una cadena, el resultado de la primera instrucción puede reenviarse directamente a la segunda instrucción. El documento US 2009/0144522 A1 se refiere a mejoras en la arquitectura de dispositivos reconfigurables. Se puede establecer un bit válido en cada registro de datos interno cuando se escriben datos en el registro. El bit válido se puede restablecer cuando se leen los datos.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

La presente invención se ilustra a modo de ejemplo y sin limitación en las figuras de los dibujos adjuntos, en los que referencias similares indican elementos similares y en los que:

La **Figura 1A** ilustra una realización de teselas configuradas;

La **Figura 1B** ilustra una realización de teselas configuradas;

La **Figura 2** ilustra varios ejemplos de almacenamiento de matrices;

La **Figura 3** ilustra una realización de un sistema que utiliza un acelerador de operaciones de tesela;

Las **Figuras 4 y 5** muestran diferentes realizaciones de cómo se comparte la memoria usando un acelerador de operaciones de matrices;

La **Figura 6** ilustra una realización de la operación de multiplicación acumulación de matrices usando teselas ("TMMA")

La **Figura 7** ilustra una realización de un subconjunto de la ejecución de una iteración de una instrucción de multiplicación acumulación fusionada en cadena;

La **Figura 8** ilustra una realización de un subconjunto de la ejecución de una iteración de una instrucción de multiplicación acumulación fusionada en cadena;

La **Figura 9** ilustra una realización de un subconjunto de la ejecución de una iteración de una instrucción de multiplicación acumulación fusionada en cadena;

La **Figura 10** ilustra una realización de un subconjunto de la ejecución de una iteración de instrucción de multiplicación acumulación fusionada en cadena;

La **Figura 11** ilustra implementaciones de SIMD con tamaño de potencia de dos en las que los acumuladores usan tamaños de entrada que son mayores que las entradas a los multiplicadores de acuerdo con una realización;

La **Figura 12** ilustra una realización de un sistema que utiliza una circuitería de operaciones de matriz;

La **Figura 13** ilustra una realización de una canalización central de procesador que soporta operaciones de matrices usando teselas;

La **Figura 14** ilustra una realización de una canalización central de procesador que soporta operaciones de matrices usando teselas;

La **Figura 15** ilustra un ejemplo de una matriz expresada en formato de fila principal y formato de columna principal;

La **Figura 16** ilustra un ejemplo de uso de matrices (teselas);

La **Figura 17** ilustra una realización de un método de uso de matrices (teselas);

La **Figura 18** ilustra el soporte para la configuración del uso de teselas de acuerdo con una realización;

La **Figura 19** ilustra una realización de una descripción de las matrices (teselas) que se van a soportar;

Las **Figuras 20(A)-(D)** ilustrar ejemplos de registro o registros;

La **Figura 21A** es un diagrama de flujo de bloques que ilustra una ejecución ilustrativa de una cadena optimizable de instrucciones de teselas;

La **Figura 21B** es un diagrama de flujo de bloques que ilustra una ejecución optimizada de una cadena de instrucciones de tesela, de acuerdo con algunas realizaciones;

La **Figura 22** es un diagrama de flujo de bloques que ilustra un flujo de ejecución de un procesador que responde a instrucciones de teselas en cadena, de acuerdo con una realización;

La **Figura 23** ilustra una descripción más detallada de una ejecución de instrucciones de teselas en cadena, de acuerdo con una realización;

La **Figura 24A** es un pseudocódigo ilustrativito que describe una realización de un procesador que ejecuta instrucciones de teselas en cadena, de acuerdo con una realización;

La **Figura 24B** es un pseudocódigo que ilustra una cadena no optimizada de instrucciones de tesela, de acuerdo con algunas realizaciones;

La **Figura 24C** es un pseudocódigo que ilustra un flujo de ejecución mediante una circuitería de ejecución de procesador para ejecutar una cadena optimizada de instrucciones de tesela, de acuerdo con algunas realizaciones;

Las **Figuras 25A-25D** son diagramas de bloques que ilustran formatos de instrucción de acuerdo con realizaciones de la invención;

La **Figura 25A** es un diagrama de bloques que ilustra un formato de instrucción compatible con cadenas de acuerdo con realizaciones de la invención;

La **Figura 25B** es un diagrama de bloques que ilustra un formato de instrucción de inicio de cadena de acuerdo con realizaciones de la invención;

La **Figura 25C** es un diagrama de bloques que ilustra un formato de instrucción compatible con vectores genérico y plantillas de instrucciones de clase A del mismo de acuerdo con realizaciones de la invención;

La **Figura 25D** es un diagrama de bloques que ilustra el formato de instrucción compatible con vectores genérico y plantillas de instrucciones de clase B del mismo acuerdo con realizaciones de la invención;

La **Figura 26A** es un diagrama de bloques que ilustra un formato de instrucción compatible con vectores específico ilustrativo de acuerdo con realizaciones de la invención;

La **Figura 26B** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico que componen el campo de código de operación completo de acuerdo con una realización de la invención;

La **Figura 26C** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico que componen el campo de índice de registro de acuerdo con una realización de la invención;

La **Figura 26D** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico que componen el campo de operación de aumento de acuerdo con una realización de la invención;

La **Figura 27** es un diagrama de bloques de una arquitectura de registro de acuerdo con una realización de la invención;

La **Figura 28A** es un diagrama de bloques que ilustra tanto una canalización en orden como una canalización de emisión/ejecución en desorden y cambio de nombre de registro ilustrativas de acuerdo con realizaciones de la invención;

La **Figura 28B** es un diagrama de bloques que ilustra tanto una realización ilustrativa de un núcleo de arquitectura en orden como un núcleo de arquitectura de emisión/ejecución en desorden y cambio de nombre de registro ilustrativos que se van a incluir en un procesador de acuerdo con realizaciones de la invención;

Las Figuras 29A-B ilustran un ejemplo más específico de un diagrama de bloques de una arquitectura de núcleo en orden, cuyo núcleo sería uno de varios bloques lógicos (que incluyen otros núcleos del mismo tipo y/o tipos diferentes) en un chip;

La **Figura 29A** es un diagrama de bloques de un único núcleo de procesador, junto con su conexión a la red de interconexión en el encapsulado y con su subconjunto local de la caché de nivel 2 (L2), de acuerdo con realizaciones de la invención;

La **Figura 29B** es una vista ampliada de parte del núcleo del procesador en la **Figura 29A** de acuerdo con realizaciones de la invención;

La **Figura 30** es un diagrama de bloques de un procesador que puede tener más de un núcleo, puede tener un controlador de memoria integrado y puede tener gráficos integrados de acuerdo con realizaciones de la invención;

Las **Figuras 31-34** son diagramas de bloques de arquitecturas informáticas ilustrativas;

La **Figura 31** muestra un diagrama de bloques de un sistema de acuerdo con una realización de la presente invención;

La **Figura 32** es un diagrama de bloques de un primer sistema ilustrativo más específico de acuerdo con una realización de la presente invención;

La **Figura 33** es un diagrama de bloques de un segundo sistema ilustrativo más específico de acuerdo con una realización de la presente invención;

La **Figura 34** es un diagrama de bloques de un sistema en un chip (SoC) de acuerdo con una realización de la presente invención; y

La **Figura 35** es un diagrama de bloques que contrasta el uso de un convertidor de instrucciones de software para convertir instrucciones binarias en un conjunto de instrucciones de origen en instrucciones binarias en un conjunto de instrucciones de destino de acuerdo con realizaciones de la invención.

DESCRIPCIÓN DETALLADA

En la siguiente descripción, se exponen numerosos detalles específicos. Sin embargo, se entiende que las realizaciones de la invención se pueden poner en práctica sin estos detalles específicos. En otros casos, no se han mostrado en detalle circuitos, estructuras y técnicas bien conocidos para no complicar la comprensión de esta descripción.

Las referencias en la memoria descriptiva a "una realización", "una realización ilustrativa", etc., indican que la realización descrita puede incluir un rasgo, estructura o característica particular, pero cada realización puede no necesariamente incluir el rasgo, estructura, o característica particular. Además, tales expresiones no se refieren necesariamente a la misma realización. Además, cuando se describe un rasgo, estructura o característica particular en relación con una realización, se afirma que está dentro del conocimiento de un experto en la materia afectar a tal rasgo, estructura o característica en relación con otras realizaciones ya se haya descrito o no explícitamente.

En muchos procesadores convencionales, el manejo de matrices es una tarea difícil y/o intensiva en cuanto a instrucciones. Por ejemplo, las filas de una matriz podrían colocarse en una pluralidad de registros de datos empaquetados (por ejemplo, SIMD o vectoriales) y, a continuación, operarse individualmente. Por ejemplo, sumar dos matrices de 8x2 puede requerir una carga o reunirlas en cuatro registros de datos empaquetados dependiendo de los tamaños de los datos. A continuación, se realiza una primera suma de registros de datos empaquetados correspondientes a una primera fila de cada matriz y se realiza una segunda suma de registros de datos empaquetados correspondientes a una segunda fila de cada matriz. A continuación, los registros de datos empaquetados resultantes se devuelven a la memoria. Si bien para matrices pequeñas esta situación puede ser aceptable, a menudo no lo es con matrices más grandes.

I. ANÁLISIS DE ALTO NIVEL

En el presente documento se describen mecanismos para soportar operaciones de matrices en hardware informático tales como unidades centrales de procesamiento (CPU), unidades de procesamiento gráfico (GPU) y aceleradores. Las operaciones de matrices utilizan estructuras de datos bidimensionales (2-D) que representan una o más regiones empaquetadas de memoria tales como registros. A lo largo de esta descripción, estas estructuras de datos 2-D se denominan teselas. Obsérvese que, una matriz puede ser más pequeña que una tesela (usar menos que toda una tesela) o utilizar una pluralidad de teselas (la matriz es más grande que el tamaño de cualquier tesela). A lo largo de toda la descripción, el lenguaje de teselas se usa para indicar operaciones realizadas usando teselas que impactan una matriz; si esa matriz es o no más grande que cualquier tesela no es típicamente relevante.

Cada tesela puede verse afectada por diferentes operaciones, tales como las que se detallan en el presente documento e incluyen, pero sin limitación: multiplicación de teselas, suma de teselas, resta de teselas, diagonal de teselas, cero de teselas, transposición de teselas, producto escalar de teselas, difusión de teselas, difusión de filas de teselas, difusión de columnas de teselas, multiplicación de teselas, multiplicación y acumulación de teselas, movimiento de teselas, etc. Además, se puede usar soporte para operadores tales como el uso de una escala y/o desvío con estas operaciones o para soporte de aplicaciones no numéricas en el futuro, por ejemplo, "memoria local" de OpenCL, compresión/descompresión de datos, etc.

Las porciones de almacenamiento (tales como la memoria (no volátil y volátil), registros, caché, etc.) están dispuestas en teselas de diferentes dimensiones horizontales y verticales. Por ejemplo, una tesela puede tener una dimensión horizontal de 4 (por ejemplo, cuatro filas de una matriz) y una dimensión vertical de 8 (por ejemplo, 8 columnas de la matriz). Típicamente, la dimensión horizontal está relacionada con los tamaños de los elementos (por ejemplo, 2, 4, 8, 16, 32, 64, 128 bits, etc.). Se pueden soportar múltiples tipos de datos (coma flotante de precisión sencilla, coma flotante de precisión doble, entero, etc.).

A. USO ILUSTRATIVO DE TESELAS CONFIGURADAS

En algunas realizaciones, se pueden configurar los parámetros de teselas. Por ejemplo, una tesela dada se puede configurar para proporcionar opciones de tesela. Las opciones de teselas ilustrativas incluyen, pero sin limitación: un número de filas de la tesela, un número de columnas de la tesela, si la tesela es VÁLIDA y si la tesela consiste en un PAR de teselas con el mismo tamaño.

La **Figura 1A** ilustra una realización de teselas configuradas. Como se muestra, 4 kB de memoria de aplicación 102 tienen almacenados en los mismos 4 teselas de 1 kB, la tesela 0 104, la tesela 1106, la tesela 2 108 y la tesela 3 110. En este ejemplo, las 4 teselas no consisten en pares y cada una tiene elementos dispuestos en filas y columnas. Las teselas t0 104 y t1 106 tienen K filas y N columnas de elementos de 4 bytes (por ejemplo, datos de precisión sencilla), donde K es igual a 8 y N=32. Las teselas t2 108 y t3 110 tienen K filas y N/2 columnas de elementos de 8 bytes (por ejemplo, datos de precisión doble). Como los operandos de doble precisión tienen el doble de anchura que los de precisión sencilla, esta configuración es consistente con una paleta, usada para proporcionar opciones de teselas, que proporciona al menos 4 nombres con un almacenamiento total de al menos 4 kB. En operación, las teselas se pueden cargar desde y almacenar en la memoria usando operaciones de carga y almacenamiento. Dependiendo del esquema de codificación de instrucciones usado, varía la cantidad de memoria de aplicación disponible, así como el tamaño, número y configuración de las teselas disponibles.

La **Figura 1B** ilustra una realización de teselas configuradas. Como se muestra, 4 kB de memoria de aplicación 122 tienen almacenados en los mismos 2 pares de teselas de 1 kB, siendo el primer par la tesela t4L 124 y la tesela t4R 126, y siendo el segundo par la tesela t5L 128 y la tesela t5R 130. Como se muestra, los pares de teselas se dividen en una tesela izquierda y una tesela derecha. En otras realizaciones, el par de teselas se divide en una tesela par y una tesela impar. En este ejemplo, cada una de las 4 teselas tiene elementos dispuestos en filas y columnas. Las teselas t4L 124 y t4R 126 tienen K filas y N columnas de elementos de 4 bytes (por ejemplo, datos de precisión sencilla), donde K es igual a 8 y N equivale a 32. Las teselas t5L 128 y t5R 130 tienen K filas y N/2 columnas de elementos de 8 bytes (por ejemplo, datos de precisión doble). Como los operandos de doble precisión tienen el doble de anchura que los de precisión sencilla, esta configuración es consistente con una paleta, usada para proporcionar opciones de teselas, que proporciona al menos 2 nombres con un almacenamiento total de al menos 4 kB. Las cuatro teselas de la **Figura 1A** usan 4 nombres, nombrando cada uno una tesela de 1 kB, mientras que los 2 pares de teselas en la **Figura 1B** puede usar 2 nombres para especificar las teselas emparejadas. En algunas realizaciones, las instrucciones de tesela aceptan el nombre de una tesela emparejada como operando. En operación, las teselas se pueden cargar desde y almacenar en la memoria usando operaciones de carga y almacenamiento. Dependiendo del esquema de codificación de instrucciones usado, varía la cantidad de memoria de aplicación disponible, así como el tamaño, número y configuración de las teselas disponibles.

En algunas realizaciones, los parámetros de las teselas son definibles. Por ejemplo, se usa una "paleta" para proporcionar opciones de teselas. Las opciones ilustrativas incluyen, pero sin limitación: el número de nombres de teselas, el número de bytes en una fila de almacenamiento, el número de filas y columnas en una tesela, etc. Por ejemplo, una "altura" máxima (número de filas) de una tesela se puede definir como:

Como tal, una aplicación se puede escribir de manera que un uso fijo de nombres pueda aprovechar diferentes tamaños de almacenamiento en todas las implementaciones.

La configuración de teselas se realiza mediante una instrucción de configuración de teselas ("TILECONFIG"), donde se define un uso de tesela particular en una paleta seleccionada. Esta declaración incluye el número de nombres de teselas que se van a usar, el número solicitado de filas y columnas por nombre (tesela) y, en algunas realizaciones, el tipo de datos solicitado de cada tesela. En algunas realizaciones, se realizan comprobaciones de consistencia durante la ejecución de una instrucción TILECONFIG para determinar que coincide con las restricciones de la entrada de la paleta.

B. TIPOS DE ALMACENAMIENTO DE TESELAS ILUSTRATIVOS

La **Figura 2** ilustra varios ejemplos de almacenamiento de matrices. En (A), una tesela se almacena en la memoria. Como se muestra, cada "fila" consiste en cuatro elementos de datos empaquetados. Para pasar a la siguiente "fila", se usa un valor de paso. Obsérvese que las filas pueden almacenarse consecutivamente en la memoria. Los accesos a la memoria con paso permiten el acceso de una fila a la siguiente cuando el almacenamiento de teselas no mapea la anchura de fila de la matriz de memoria subyacente.

Las cargas de teselas desde la memoria y los almacenes en la memoria típicamente son accesos con pasos desde la memoria de la aplicación a filas empaquetadas de datos. Las instrucciones TILELOAD y TILESTORE ilustrativas, u otras referencias de instrucciones a la memoria de la aplicación como operando TILE en instrucciones de operación de carga, son, en algunas realizaciones, reiniciables para manejar (hasta) 2*filas de errores de página, excepciones de coma flotante no enmascaradas y/o interrupciones por instrucción.

En (B), una matriz se almacena en una tesela compuesta por una pluralidad de registros, tales como registros de datos empaquetados (datos múltiples de instrucción única (SIMD) o registros vectoriales). En este ejemplo, la tesela se superpone en tres registros físicos. Típicamente, se usan registros consecutivos, aunque no tiene por qué ser así.

En (C), una matriz se almacena en una tesela en un almacenamiento sin registro accesible a un circuito de multiplicación acumulación fusionado (FMA) usado en las instrucciones de tesela. Este almacenamiento puede estar dentro de un FMA o adyacente a él. Además, en algunas realizaciones, que se analizan a continuación, el almacenamiento puede ser para un elemento de datos y no para una fila o tesela completa.

Los parámetros admitidos para la arquitectura TMMA se informan a través de CPUID. En algunas realizaciones, la lista de información incluye una altura máxima y una dimensión de SIMD máxima. Configurar la arquitectura de TMMA requiere especificar las dimensiones de cada tesela, el tamaño de elemento para cada tesela y el identificador de paleta. Esta configuración se realiza ejecutando la instrucción TILECONFIG.

La ejecución con éxito de una instrucción TILECONFIG habilita a los operadores TILE posteriores. Una instrucción TILERELEASEALL borra la configuración de la tesela y deshabilita las operaciones TILE (hasta que se ejecute la siguiente instrucción TILECONFIG). En algunas realizaciones, XSAVE, XSTORE, etc. se usan en la conmutación de contexto usando teselas. En algunas realizaciones, se usan 2 bits XCR0 en XSAVE, uno para los metadatos TILECONFIF y un bit correspondiente a los datos de carga útil de tesela reales.

TILECONFIG no únicamente configura el uso de teselas, sino que también establece una variable de estado que indica que el programa se encuentra en una región de código con teselas configuradas. Una implementación puede enumerar restricciones sobre otras instrucciones que se pueden usar con una región de tesela, tal como no usar un conjunto de registros existente, etc.

Típicamente, salir de una región de tesela se realiza con la instrucción TILERELEASEALL. No requiere parámetros e invalida rápidamente todas las teselas (lo que indica que los datos ya no necesitan guardarse ni restaurarse) y borra el estado interno correspondiente a estar en una región de tesela.

En algunas realizaciones, las operaciones de tesela pondrán a cero cualquier fila y columna más allá de las dimensiones especificadas por la configuración de tesela. Por ejemplo, las operaciones de tesela pondrán a cero los datos más allá del número configurado de columnas (teniendo en cuenta el tamaño de los elementos) a medida que se escribe cada fila. Por ejemplo, con filas de 64 bytes y una tesela configurada con 10 filas y 12 columnas, una operación que escribe elementos FP32 escribiría cada una de las primeras 10 filas con 12*4 bytes con datos de salida/resultado y pondría a cero los 4*4 bytes restantes en cada fila. Las operaciones en tesela también ponen a cero por completo cualquier fila después de las primeras 10 filas configuradas. Cuando se usa tesela de 1K con filas de 64 bytes, habrá 16 filas, por lo que, en este ejemplo, las últimas 6 filas también se pondrán a cero.

En algunas realizaciones, una restauración de contexto (por ejemplo, XRSTOR), cuando se cargan datos, exige que los datos más allá de las filas configuradas para una tesela se mantengan en cero. Si no hay una configuración válida, todas las filas se ponen a cero. XRSTOR de datos de teselas puede cargar basura en las columnas más allá de las

configuradas. No debería ser posible que XRSTOR borre más allá del número de columnas configuradas porque no hay una anchura de elemento asociada con la configuración de la tesela.

El guardado de contexto (por ejemplo, XSAVE) expone toda el área de almacenamiento TILE al escribirla en la memoria. Si XRSTOR cargó datos basura en la parte más a la derecha de una tesela, XSAVE guardará esos datos. XSAVE pondrá ceros en las filas más allá del número especificado para cada tesela.

En algunas realizaciones, las instrucciones de las teselas se pueden reiniciar. Las operaciones que acceden a la memoria permiten reiniciar después de fallos de página. Las instrucciones computacionales que tratan con operaciones de coma flotante también permiten excepciones de coma flotante no enmascaradas, con el enmascaramiento de las excepciones controlado por un registro de control y/o estado.

Para soportar instrucciones de reinicio después de estos eventos, las instrucciones almacenan información en los registros de inicio que se detallan a continuación.

II. SISTEMAS DE OPERACIÓN DE MATRICES (TESELAS)

A. SOPORTE DE HARDWARE ILUSTRATIVO

La **Figura** 3 ilustra una realización de un sistema que utiliza un acelerador de operaciones de matriz (tesela). En esta ilustración, un procesador/sistema de procesamiento de anfitrión 301 comunica comandos 311 (por ejemplo, operaciones de manipulación de matrices tales como operaciones aritméticas o de manipulación de matrices, u operaciones de carga y almacenamiento) a un acelerador de operaciones de matrices 307. Sin embargo, esto se muestra de esta manera solo por motivos de análisis. Como se detalla más adelante, este acelerador de operaciones de matrices 307 puede ser parte de un núcleo de procesamiento. Típicamente, los comandos 311 que son instrucciones del operador de manipulación de teselas se referirán a las teselas como formato registro-registro ("reg-reg") o registro-memoria ("reg-mem"). Otros comandos tales como TILESTORE, TILELOAD, TILECONFIG, etc., no realizan operaciones de datos en una tesela. Los comandos pueden ser instrucciones decodificadas (por ejemplo, microoperaciones) o macroinstrucciones para que las maneje el acelerador de operaciones de matrices 307.

En este ejemplo, una interfaz de memoria coherente 303 está acoplada al procesador/sistema de procesamiento anfitrión 301 y al acelerador de operaciones de matrices 307 de manera que puedan compartir memoria. Las **Figuras 4 y 5** muestran diferentes realizaciones de cómo se comparte la memoria usando un acelerador de operaciones de matrices. Como se muestra en la **Figura 4,** el procesador de anfitrión 401 y la circuitería de acelerador de operaciones de matrices 405 comparten la misma memoria 403. **Figura 5** ilustra una realización en la que el procesador principal 501 y el acelerador de operaciones de matrices 505 no comparten memoria, pero pueden acceder a la memoria de cada uno. Por ejemplo, el procesador 501 puede acceder a la memoria de tesela 507 y utilizar su memoria de anfitrión 503 de forma normal. De manera similar, el acelerador de operaciones de matrices 505 puede acceder a la memoria de anfitrión 503, pero más típicamente usa la memoria de tesela 507. Obsérvese que estas memorias pueden ser de diferentes tipos.

En algunas realizaciones, el acelerador de operaciones de matrices 307 incluye una pluralidad de FMA 309 acoplados a memorias intermedias de datos 305 (en algunas implementaciones, una o más de estas memorias intermedias de datos 305 se almacenan en los FMA de la cuadrícula como se muestra). Las memorias intermedias de datos 305 que almacenan de forma intermedia teselas cargadas desde la memoria y/o teselas que se van a almacenar en la memoria (por ejemplo, usando una instrucción de tileload o de tilestore). Las memorias intermedias de datos pueden ser, por ejemplo, una pluralidad de registros. Típicamente, estas FMA están dispuestas como una cuadrícula de FMA 309 en cadena que pueden leer y escribir teselas. En este ejemplo, el acelerador de operaciones de matrices 307 ha de realizar una operación de multiplicación matricial usando las teselas T0, T1 y T2. Al menos una de las teselas está alojada en la cuadrícula de FMA 309. En algunas realizaciones, todas las teselas en una operación se almacenan en la cuadrícula de FMA 309. En otras realizaciones, únicamente un subconjunto se almacena en la cuadrícula de FMA 309. Como se muestra, T1 está alojada y T0 y T2 no lo están. Obsérvese que A, B y C se refieren a las matrices de estas teselas que pueden o no ocupar todo el espacio de la tesela.

La **Figura 6** ilustra una realización de la operación de multiplicación acumulación de matrices usando teselas ("TMMA").

El número de filas en la matriz (TILE A 601) coincide con el número de FMA en serie (en cadena) que comprenden la latencia del cálculo. Una implementación es libre de recircular sobre una cuadrícula de menor altura, pero el cálculo sigue siendo el mismo.

El vector de origen/destino proviene de una tesela de N filas (TESELA C 605) y la cuadrícula de FMA 611 realiza N operaciones vector-matriz que dan como resultado una instrucción completa que realiza una multiplicación matricial de teselas. La tesela B 603 es el otro origen de vector y proporciona términos de "difusión" a las FMA en cada etapa.

En operación, en algunas realizaciones, los elementos de la matriz B (almacenados en una tesela B 603) se distribuyen a través de la cuadrícula rectangular de los FMA. La matriz B (almacenada en la tesela A 601) tiene sus elementos de una fila transpuestos para que coincidan con la dimensión de la columna de la cuadrícula rectangular de los FMA. En cada FMA de la cuadrícula, un elemento de A y B se multiplica y se suma al sumando entrante (desde arriba en la Figura) y la suma saliente se pasa a la siguiente fila de FMA (o la salida final).

La latencia de una única etapa es proporcional a K (altura de fila de la matriz B) y las TMMA típicamente tienen suficientes filas de origen-destino (ya sea en una única tesela o a través de teselas) para ocultar esa latencia. Una implementación también puede dividir la dimensión de SIMD (elemento de datos empaquetados) M (altura de fila de la matriz A) a través de etapas de tiempo, pero esto simplemente cambia la constante por la que se multiplica K. Cuando un programa especifica un K menor que el máximo enumerado por TMACC, una implementación es libre de implementarlo con "enmascaramiento" o "salidas anticipadas".

La latencia de una TMMA completa es proporcional a N*K. La tasa de repetición es proporcional a N. El número de MAC por instrucción TMMA es N*K*M.

La **Figura 7** ilustra una realización de un subconjunto de la ejecución de una iteración de una instrucción de multiplicación acumulación fusionada en cadena. En particular, esto ilustra la circuitería de ejecución de una iteración de la posición de un elemento de datos empaquetado del destino. En esta realización, la multiplicación acumulación fusionada en cadena opera en orígenes con signo en los que el acumulador tiene el doble del tamaño de los datos de entrada.

Un primer origen con signo (origen 1 701) y un segundo origen con signo (origen 2 703) tiene cada uno cuatro elementos de datos empaquetados. Cada uno de estos elementos de datos empaquetados almacena datos con signo, tales como datos de coma flotante. Un tercer origen con signo (origen 3 709) tiene dos elementos de datos empaquetados, cada uno de los cuales almacena datos con signo. Los tamaños del primer y segundo orígenes con signo 701 y 703 son la mitad que los del tercer origen con signo (valor inicial o resultado anterior) 709. Por ejemplo, el primer y el segundo orígenes con signo 701 y 703 podrían tener elementos de datos empaquetados de 32 bits (por ejemplo, coma flotante de precisión sencilla) mientras que el tercer origen con signo 709 podría tener elementos de datos empaquetados de 64 bits (por ejemplo, coma flotante de precisión doble).

En esta ilustración, únicamente se muestran las dos posiciones de elementos de datos empaquetados más significativas del primer y segundo orígenes con signo 701 y 703 y la posición de elementos de datos empaquetados más significativa del tercer origen con signo 709. Por supuesto, también se procesarían las demás posiciones de los elementos de datos empaquetados.

Como se ilustra, los elementos de datos empaquetados se procesan en pares. Por ejemplo, los datos de las posiciones de elementos de datos empaquetados más significativas del primer y segundo orígenes con signo 701 y 703 se multiplican usando un circuito multiplicador 705, y los datos de las posiciones de los segundos elementos de datos empaquetados más significativos del primer y segundo orígenes con signo 701 y 703 se multiplican usando un circuito multiplicador 707. En algunas realizaciones, estos circuitos multiplicadores 705 y 707 se reutilizan para otras posiciones de elementos de datos empaquetados. En otras realizaciones, se usan circuitos multiplicadores adicionales para que los elementos de datos empaquetados se procesen en paralelo. En algunos contextos, la ejecución paralela se realiza usando carriles que son del tamaño del tercer origen con signo 709. Los resultados de cada una de las multiplicaciones se suman usando la circuitería de sumador 711.

El resultado de la suma de los resultados de las multiplicaciones se suma a los datos de la posición de elemento de datos empaquetados más significativo del origen con signo 3 709 (usando un sumador diferente 713 o el mismo sumador 711).

Finalmente, el resultado de la segunda adición se almacena en el destino con signo 715 en una posición de elemento de datos empaquetados que corresponde a la posición de elemento de datos empaquetados usada desde el tercer origen con signo 709 o se pasa a la siguiente iteración, si la hay. En algunas realizaciones, se aplica una máscara de escritura a este almacenamiento de modo que si se establece una máscara de escritura (bit) correspondiente, se realiza el almacenamiento y, si no se establece, no se realiza el almacenamiento.

La **Figura 8** ilustra una realización de un subconjunto de la ejecución de una iteración de una instrucción de multiplicación acumulación fusionada en cadena. En particular, esto ilustra la circuitería de ejecución de una iteración de la posición de un elemento de datos empaquetado del destino. En esta realización, la multiplicación acumulación fusionada en cadena opera en orígenes con signo en los que el acumulador tiene el doble del tamaño de los datos de entrada.

Un primer origen con signo (origen 1801) y un segundo origen con signo (origen 2803) tiene cada uno cuatro elementos de datos empaquetados. Cada uno de estos elementos de datos empaquetados almacena datos con signo, tales como datos de números enteros. Un tercer origen con signo (origen 3 809) tiene dos elementos de datos empaquetados, cada uno de los cuales almacena datos con signo. Los tamaños del primer y segundo orígenes con signo 801 y 803

son la mitad que los del tercer origen con signo 809. Por ejemplo, el primer y el segundo orígenes con signo 801 y 803 podrían tener elementos de datos empaquetados de 32 bits (por ejemplo, coma flotante de precisión sencilla), el tercer origen con signo 809 podría tener elementos de datos empaquetados de 64 bits (por ejemplo, coma flotante de precisión doble).

En esta ilustración, únicamente se muestran las dos posiciones de elementos de datos empaquetados más significativas del primer y segundo orígenes con signo 801 y 803 y la posición de elementos de datos empaquetados más significativa del tercer origen con signo 809. Por supuesto, también se procesarían las demás posiciones de los elementos de datos empaquetados.

Como se ilustra, los elementos de datos empaquetados se procesan en pares. Por ejemplo, los datos de las posiciones de elementos de datos empaquetados más significativas del primer y segundo orígenes con signo 801 y 803 se multiplican usando un circuito multiplicador 805, y los datos de las posiciones de los segundos elementos de datos empaquetados más significativos del primer y segundo orígenes con signo 801 y 803 se multiplican usando un circuito multiplicador 807. En algunas realizaciones, estos circuitos multiplicadores 805 y 807 se reutilizan para otras posiciones de elementos de datos empaquetados. En otras realizaciones, se usan circuitos multiplicadores adicionales para que los elementos de datos empaquetados se procesen en paralelo. En algunos contextos, la ejecución paralela se realiza usando carriles que son del tamaño del tercer origen con signo (valor inicial o resultado de la iteración anterior) 809. Los resultados de cada una de las multiplicaciones se suman al tercer origen con signo 809 usando circuitería de suma/saturación 813.

La circuitería de suma/saturación 813 (acumulador) conserva un signo de un operando cuando la suma da como resultado un valor que es demasiado grande. En particular, la evaluación de saturación ocurre en el resultado de precisión infinita entre la adición multidireccional y la escritura en el destino o la siguiente iteración. Cuando el acumulador 813 es de coma flotante y los términos de entrada son números enteros, la suma de productos y el valor de entrada del acumulador de coma flotante se convierten en valores de precisión infinita (números de coma fija de cientos de bits), se realiza la suma de los resultados de la multiplicación y la tercera entrada y se realiza un redondeo único al tipo de acumulador real.

La saturación sin signo significa que los valores de salida están limitados a un número máximo sin signo para esa anchura de elemento (todos 1). La saturación con signo significa que un valor está limitado a estar en el rango entre un número negativo mínimo y un número positivo máximo para esa anchura de elemento (para bytes, por ejemplo, el rango es de -128 (= - 2^7) a 127(= 2^7-1)).

Se almacena el resultado de la adición y la comprobación de saturación en el destino con signo 815 en una posición de elemento de datos empaquetados que corresponde a la posición de elemento de datos empaquetados usada desde el tercer origen con signo 809 o se pasa a la siguiente iteración, si la hay. En algunas realizaciones, se aplica una máscara de escritura a este almacenamiento de modo que si se establece una máscara de escritura (bit) correspondiente, se realiza el almacenamiento y, si no se establece, no se realiza el almacenamiento.

La **Figura 9** ilustra una realización de un subconjunto de la ejecución de una iteración de una instrucción de multiplicación acumulación fusionada en cadena. En particular, esto ilustra la circuitería de ejecución de una iteración de la posición de un elemento de datos empaquetado del destino. En esta realización, la multiplicación acumulación fusionada en cadena opera en un origen con signo y en un origen sin signo en donde el acumulador tiene 4 veces el tamaño de los datos de entrada.

Un primer origen con signo (origen 1901) y un segundo origen sin signo (origen 2 903) tiene cada uno cuatro elementos de datos empaquetados. Cada uno de estos elementos de datos empaquetados tiene datos como datos de coma flotante o de números enteros. Un tercer origen con signo (valor inicial o resultado 915) tiene un elemento de datos empaquetados del cual almacena datos con signo. Los tamaños del primer y segundo orígenes 901 y 903 son una cuarta parte de los del tercer origen con signo 915. Por ejemplo, el primer y el segundo orígenes 901 y 903 podrían tener elementos de datos empaquetados de 16 bits (por ejemplo, palabra) y el tercer origen con signo 915 podría tener elementos de datos empaquetados de 64 bits (por ejemplo, coma flotante de precisión doble o número entero de 64 bits).

En esta ilustración, se muestran las cuatro posiciones de elementos de datos empaquetados más significativas del primer y segundo orígenes 901 y 903 y la posición de elementos de datos empaquetados más significativa del tercer origen con signo 915. Por supuesto, también se procesarían otras posiciones de elementos de datos empaquetados, si las hubiera.

Como se ilustra, los elementos de datos empaquetados se procesan en cuartetos. Por ejemplo, los datos de las posiciones de elementos de datos empaquetados más significativas del primer y segundo orígenes 901 y 903 se multiplican usando un circuito multiplicador 905, los datos de las segundas posiciones de elementos de datos empaquetados más significativas del primer y segundo orígenes 901 y 903 se multiplican usando un circuito multiplicador 907, los datos de las terceras posiciones de elementos de datos empaquetados más significativas del primer y segundo orígenes 901 y 903 se multiplican usando un circuito multiplicador 909, y los datos de las posiciones

de elementos de datos empaquetados menos significativas del primer y segundo orígenes 901 y 903 se multiplican usando un circuito multiplicador 911. En algunas realizaciones, los elementos de datos empaquetados con signo del primer origen 901 tienen signo extendido y los elementos de datos empaquetados sin signo del segundo origen 903 tienen extensión cero antes de las multiplicaciones.

En algunas realizaciones, estos circuitos multiplicadores 905-911 se reutilizan para otras posiciones de elementos de datos empaquetados. En otras realizaciones, se usan circuitos multiplicadores adicionales para que los elementos de datos empaquetados se procesen en paralelo. En algunos contextos, la ejecución paralela se realiza usando carriles que son del tamaño del tercer origen con signo 915. Los resultados de cada una de las multiplicaciones se suman usando circuitería de suma 911.

El resultado de la suma de los resultados de las multiplicaciones se suma a los datos de la posición de elemento de datos empaquetados más significativo del origen con signo 3 915 (usando un sumador diferente 913 o el mismo sumador 911).

Finalmente, el resultado 919 de la segunda adición se almacena en el destino con signo en una posición de elemento de datos empaquetados que corresponde a la posición de elemento de datos empaquetados usada desde el tercer origen con signo 915 o se pasa a la siguiente iteración. En algunas realizaciones, se aplica una máscara de escritura a este almacenamiento de modo que si se establece una máscara de escritura (bit) correspondiente, se realiza el almacenamiento y, si no se establece, no se realiza el almacenamiento.

La **Figura 10** ilustra una realización de un subconjunto de la ejecución de una iteración de instrucción de multiplicación acumulación fusionada en cadena. En particular, esto ilustra la circuitería de ejecución de una iteración de la posición de un elemento de datos empaquetado del destino. En esta realización, la multiplicación acumulación fusionada en cadena opera en un origen con signo y en un origen sin signo en donde el acumulador tiene 4 veces el tamaño de los datos de entrada.

Un primer origen con signo (origen con signo 11001) y un segundo origen sin signo (origen 2 sin signo 1003) tiene cada uno cuatro elementos de datos empaquetados. Cada uno de estos elementos de datos empaquetados almacena datos como datos de coma flotante o de números enteros. Un tercer origen con signo (resultado inicial o anterior 1015) tiene un elemento de datos empaquetados del cual almacena datos con signo. Los tamaños del origen con signo 11001 y del origen sin signo 2 1003 son una cuarta parte del tercer origen con signo conectado al resultado inicial o anterior 1015. Por ejemplo, el origen con signo 11001 y el origen sin signo 2 1003 podrían tener elementos de datos empaquetados de 16 bits (por ejemplo, palabra) y el resultado inicial o anterior 1015 podría tener elementos de datos empaquetados de 64 bits (por ejemplo, coma flotante de precisión doble o número entero de 64 bits).

En esta ilustración, se muestran las cuatro posiciones de elementos de datos empaquetados más significativas del origen con signo 11001 y del origen 2 sin signo 1003 y la posición de elementos de datos empaquetados más significativa del resultado inicial o anterior 1015. Por supuesto, también se procesarían otras posiciones de elementos de datos empaquetados, si las hubiera.

Como se ilustra, los elementos de datos empaquetados se procesan en cuartetos. Por ejemplo, los datos de las posiciones de elementos de datos empaquetados más significativas del origen con signo 1 1001 y del origen sin signo 2 1003 se multiplican usando un circuito multiplicador 1005, los datos de las segundas posiciones más significativas de los elementos de datos empaquetados del origen con signo 11001 y del origen sin signo 2 1003 se multiplican usando un circuito multiplicador 1007, los datos de las posiciones de los terceros elementos de datos empaquetados más significativos del origen con signo 11001 y del origen sin signo 2 1003 se multiplican usando un circuito multiplicador 1009, y los datos de las posiciones de los elementos de datos empaquetados menos significativos del origen con signo 11001 y del origen sin signo 2 1003 se multiplican usando un circuito multiplicador 1011. En algunas realizaciones, los elementos de datos empaquetados con signo del origen con signo 11001 tienen signo extendido y los elementos de datos empaquetados sin signo del origen sin signo 2 1003 tienen extensión cero antes de las multiplicaciones.

En algunas realizaciones, estos circuitos multiplicadores 1005-1011 se reutilizan para otras posiciones de elementos de datos empaquetados. En otras realizaciones, se usan circuitos multiplicadores adicionales para que los elementos de datos empaquetados se procesen en paralelo. En algunos contextos, la ejecución paralela se realiza usando carriles que son del tamaño del resultado inicial o anterior 1015. El resultado de la suma de los resultados de las multiplicaciones se suma a los datos de la posición de elemento de datos empaquetados más significativo del origen con signo 3, conectado al resultado inicial o anterior 1015 usando circuitería de suma/saturación 1013.

La circuitería de suma/saturación 1013 (acumulador) conserva un signo de un operando cuando la suma da como resultado un valor que es demasiado grande o demasiado pequeño para saturación con signo. Como se usa en el presente documento, la circuitería de adición/saturación 1013 en ocasiones se denomina acumulador. En particular, la evaluación de saturación ocurre en el resultado de precisión infinita entre la adición multidireccional y la escritura en el destino. Cuando la circuitería de suma/saturación 1013 usa coma flotante y los términos de entrada son números enteros, la suma de productos y el valor de entrada del acumulador de coma flotante se convierten en valores de

precisión infinita (números de coma fija de cientos de bits), se realiza la suma de los resultados de la multiplicación y la tercera entrada y se realiza un redondeo único al tipo de acumulador real.

El resultado 1019 de la comprobación de adición y saturación se almacena en el destino con signo en una posición de elemento de datos empaquetados que corresponde a la posición de elemento de datos empaquetados usada desde el tercer origen con signo, conectado al resultado inicial o anterior 1015, o se pasa a la siguiente iteración. En algunas realizaciones, se aplica una máscara de escritura a este almacenamiento de modo que si se establece una máscara de escritura (bit) correspondiente, se realiza el almacenamiento y, si no se establece, no se realiza el almacenamiento.

La **Figura 11** ilustra implementaciones de SIMD con tamaño de potencia de dos en las que los acumuladores usan tamaños de entrada que son mayores que las entradas a los multiplicadores de acuerdo con una realización. Obsérvese que los valores origen (a los multiplicadores) y acumulador pueden ser valores con signo o sin signo. Para un acumulador que tiene tamaños de entrada 2X (en otras palabras, el valor de entrada del acumulador es el doble del tamaño de los tamaños de elementos de datos empaquetados de los orígenes), la tabla 1101 ilustra diferentes configuraciones. Para orígenes con tamaño de bytes, el acumulador usa valores de palabra o de coma flotante de precisión media (HPFP) que tienen un tamaño de 16 bits. Para orígenes con tamaño de palabra, el acumulador usa valores de números enteros de 32 bits o valores de coma flotante de precisión sencilla (SPFP) que tienen un tamaño de 32 bits. Para orígenes de SPFP o con tamaño de números entero de 32 bits, el acumulador usa valores de coma flotante de números enteros de 64 o de precisión doble (DPFP) que tienen un tamaño de 64 bits.

Para un acumulador que tiene tamaños de entrada 4X (en otras palabras, el valor de entrada del acumulador es cuatro veces el tamaño de los tamaños de elementos de datos empaquetados de los orígenes), la tabla 1103 ilustra diferentes configuraciones. Para orígenes con tamaño de bytes, el acumulador usa valores de números enteros de 32 bits o valores de coma flotante de precisión sencilla (SPFP) que tienen un tamaño de 32 bits. Para orígenes con tamaño de palabra, el acumulador usa valores de números enteros de 64 bits o de coma flotante de precisión doble (DPFP) que tienen un tamaño de 64 bits en algunas realizaciones.

Para un acumulador que tiene tamaños de entrada 8X (en otras palabras, el valor de entrada del acumulador es ocho veces el tamaño de los tamaños de elementos de datos empaquetados de los orígenes), la tabla 1105 ilustra una configuración. Para orígenes con tamaño de bytes, el acumulador usa un número entero de 64 bits.

Como se indicó anteriormente, la circuitería de operaciones de matrices puede incluirse en un núcleo o como un acelerador externo. La **Figura 12** ilustra una realización de un sistema que utiliza una circuitería de operaciones de matriz. En esta ilustración, una pluralidad de entidades están acopladas con una interconexión en anillo 1245.

Una pluralidad de núcleos 1201, 1203, 1205 y 1207 proporcionan soporte de instrucción no basado en teselas. En algunas realizaciones, la circuitería de operaciones de matrices 1251 se proporciona en un núcleo 1203, y en otras realizaciones, la circuitería de operaciones de matrices 1211 y 1213 es accesible en la interconexión en anillo 1245.

Adicionalmente, se proporciona uno o más controladores de memoria 1223-1225 para comunicarse con la memoria 1233 y 1231 en nombre de los núcleos y/o la circuitería de operaciones de matrices.

La **Figura 13** ilustra una realización de una canalización central de procesador que soporta operaciones de matrices usando teselas. La circuitería de predicción de bifurcación y decodificación 1303 realiza predicción de bifurcación de instrucciones, decodificación de instrucciones y/o ambas a partir de instrucciones almacenadas en el almacenamiento de instrucciones 1301. Por ejemplo, las instrucciones detalladas en el presente documento pueden almacenarse en un almacenamiento de instrucciones. En algunas implementaciones, se usa circuitería separada para la predicción de ramificaciones y, en algunas realizaciones, al menos algunas instrucciones se decodifican en una o más microoperaciones, puntos de entrada de microcódigo, microinstrucciones, otras instrucciones u otras señales de control usando el microcódigo 1305. La circuitería de predicción de bifurcación y decodificación 1303 puede implementarse usando diversos mecanismos diferentes. Ejemplos de mecanismos adecuados incluyen, pero sin limitación, tablas de consulta, implementaciones de hardware, matrices lógicas programables (PLA), memorias de sólo lectura de microcódigo (ROM), etc.

La circuitería de predicción de bifurcación y decodificación 1303 está acoplada a una circuitería de cambio de nombre/asignación 1307 que está acoplada, en algunas realizaciones, a la circuitería de planificador 1309. En algunas realizaciones, estos circuitos proporcionan funcionalidad de cambio de nombre de registro, asignación de registro y/o planificación realizando uno o más de: 1) cambiar el nombre de los valores de operandos lógicos a valores de operandos físicos (por ejemplo, una tabla de alias de registros en algunas realizaciones), 2) asignar bits de estado y banderas a la instrucción decodificada, y 3) planificar la instrucción decodificada para su ejecución en la circuitería de ejecución fuera de una agrupación de instrucciones (por ejemplo, usando una estación de reserva en algunas realizaciones).

La circuitería de planificador 1309 representa cualquier número de planificadores diferentes, incluyendo estaciones de reserva, ventana de instrucciones central, etc. La circuitería de planificador de la unidad o unidades de planificador 1309 está acoplada a, o incluye, el archivo o archivos de registro físico 1315. Cada uno del archivo o archivos de

registro físico 1315 representa uno o más archivos de registro físico, diferentes de los cuales almacenan uno o más tipos de datos diferentes, como número entero escalar, coma flotante escalar, número entero empaquetado, coma flotante empaquetado, número entero vectorial, coma flotante vectorial, estado (por ejemplo, un puntero de instrucción que es la dirección de la siguiente instrucción que se va a ejecutar), teselas, etc. En una realización, el archivo o archivos de registro físico 1315 comprenden una circuitería de registros vectoriales, circuitería de registros de máscara de escritura y circuitería de registros escalares. Estos circuitos de registro pueden proporcionar registros vectoriales arquitectónicos, registros de máscara vectorial y registros de propósito general. El archivo o archivos de registro físico 1315 se superponen con un circuito de retiro 1317 para ilustrar diversas formas en las que se puede implementar el cambio de nombre de registro y la ejecución en desorden (por ejemplo, usando una memoria o memorias intermedias de reordenación y un archivo o archivos de registro de retiro; usando un archivo o archivos futuros, una memoria o memorias intermedias de historial y un archivo o archivos de registro de retiro; usando mapas de registros y una agrupación de registros; etc.). El circuito de retiro 1317 y el archivo o archivos de registro físico 1315 están acoplados al circuito o circuitos de ejecución 1311.

Si bien el cambio de nombre de registros se describe en el contexto de la ejecución en desorden, se debe entender que el cambio de nombre de registros se puede usar en una arquitectura en orden. Si bien la realización ilustrada del procesador también puede incluir unidades de caché de datos e instrucciones separadas y una unidad de caché L2 compartida, unas realizaciones alternativas pueden tener una única caché interna tanto para instrucciones como para datos, tal como, por ejemplo, una caché interna de nivel 1 (L1), o múltiples niveles de caché interna. En algunas realizaciones, el sistema puede incluir una combinación de una memoria caché interna y una memoria caché externa que es externa al núcleo y/o al procesador. Como alternativa, toda la caché puede ser externa al núcleo y/o al procesador.

La circuitería de ejecución 1311 es un conjunto de uno o más circuitos de ejecución 1321, 1323 y 1327 y un conjunto de uno o más circuitos de acceso a memoria 1325. Los circuitos de ejecución 1321, 1323 y 1327 realizan diversas operaciones (por ejemplo, desplazamientos, suma, resta, multiplicación) y en diversos tipos de datos (por ejemplo, coma flotante escalar, número entero empaquetado, coma flotante empaquetado, número entero vectorial, vector coma flotante). Si bien algunas realizaciones pueden incluir un número de unidades de ejecución especializadas a funciones específicas o conjuntos de funciones, otras realizaciones pueden incluir únicamente una unidad de ejecución o múltiples unidades de ejecución que realizan todas las funciones. La circuitería escalar 1321 realiza operaciones escalares, la circuitería vectorial/SIMD 1323 realiza operaciones vectoriales/SIMD y la circuitería de operaciones de matrices 1327 realiza operaciones de matrices (teselas) detalladas en el presente documento.

A modo de ejemplo, la arquitectura central ilustrativa de cambio de nombre de registro, emisión/ejecución desordenada puede implementar una canalización de la siguiente manera: 1) un circuito de extracción de instrucciones realiza etapas de extracción y decodificación de longitud; 2) la circuitería de bifurcación y decodificación 1303 realiza una etapa de decodificación; 3) la circuitería de cambio de nombre/asignación 1307 realiza una etapa de asignación y una etapa de cambio de nombre; 4) la circuitería de planificador 1309 realiza una etapa de planificador; 5) archivo o archivos de registro físico (acoplados a o incluidos en la circuitería de planificador 1309 y la circuitería de cambio de nombre/asignación 1307 y una unidad de memoria realizan una etapa de lectura de registro/lectura de memoria; la circuitería de ejecución 1311 realiza una etapa de ejecución; 6) una unidad de memoria y la unidad o unidades de archivo o archivos de registro físico realizan una etapa de rescritura/escritura en memoria; 7) diversas unidades pueden estar involucradas en la etapa de manejo de excepciones; y 8) una unidad de retiro y la unidad o unidades de archivo o archivos de registro físico realizan una etapa de confirmación.

El núcleo puede soportar uno o más conjuntos de instrucciones (por ejemplo, el conjunto de instrucciones x86 (con algunas extensiones que se han añadido con versiones más recientes); el conjunto de instrucciones MIPS de MIPS Technologies de Sunnyvale, CA; el conjunto de instrucciones ARM (con extensiones opcionales adicionales tal como NEON) de ARM Holdings de Sunnyvale, CA), incluyendo la instrucción o instrucciones descritas en el presente documento. En una realización, el núcleo 1390 incluye una lógica para soportar una extensión del conjunto de instrucciones de datos empaquetados (por ejemplo, AVX1, AVX2), permitiendo de este modo que las operaciones usadas por muchas aplicaciones multimedia se realicen usando datos empaquetados.

Debe entenderse que el núcleo puede soportar hilos múltiples (ejecutar dos o más conjuntos paralelos de operaciones o hilos), y puede hacerlo de diversas maneras, incluyendo hilos múltiples en intervalos de tiempo, hilos múltiples simultáneos (donde un único núcleo físico proporciona un núcleo lógico para cada uno de los hilos que el núcleo físico está subprocesando de forma múltiple simultáneamente), o una combinación de los mismos (por ejemplo, recuperación y decodificación en cortes de tiempo e hilos múltiples simultáneos a partir de entonces, como en la tecnología Intel® Hyperthreading).

La **Figura 14** ilustra una realización de una canalización central de procesador que soporta operaciones de matrices usando teselas. La circuitería de predicción de bifurcación y decodificación 1403 realiza predicción de bifurcación de instrucciones, decodificación de instrucciones y/o ambas a partir de instrucciones almacenadas en el almacenamiento de instrucciones 1401. Por ejemplo, las instrucciones detalladas en el presente documento pueden almacenarse en un almacenamiento de instrucciones. En algunas implementaciones, se usa circuitería separada para la predicción de ramificaciones y, en algunas realizaciones, al menos algunas instrucciones se decodifican en una o más

microoperaciones, puntos de entrada de microcódigo, microinstrucciones, otras instrucciones u otras señales de control usando el microcódigo 1405. La circuitería de predicción de bifurcación y decodificación 1403 puede implementarse usando diversos mecanismos diferentes. Ejemplos de mecanismos adecuados incluyen, pero sin limitación, tablas de consulta, implementaciones de hardware, matrices lógicas programables (PLA), memorias de sólo lectura de microcódigo (ROM), etc.

La circuitería de predicción de bifurcación y decodificación 1403 está acoplada a una circuitería de cambio de nombre/asignación 1407 que está acoplada, en algunas realizaciones, a la circuitería de planificador 1409. En algunas realizaciones, estos circuitos proporcionan funcionalidad de cambio de nombre de registro, asignación de registro y/o planificación realizando uno o más de: 1) cambiar el nombre de los valores de operandos lógicos a valores de operandos físicos (por ejemplo, una tabla de alias de registros en algunas realizaciones), 2) asignar bits de estado y banderas a la instrucción decodificada, y 3) planificar la instrucción decodificada para su ejecución en la circuitería de ejecución fuera de una agrupación de instrucciones (por ejemplo, usando una estación de reserva en algunas realizaciones).

La circuitería de planificador 1409 representa cualquier número de planificadores diferentes, incluyendo estaciones de reserva, ventana de instrucciones central, etc. La circuitería de planificador de la unidad o unidades de planificador 1409 está acoplada a, o incluye, el archivo o archivos de registro físico 1415. Cada uno del archivo o archivos de registro físico 1415 representa uno o más archivos de registro físico, diferentes de los cuales almacenan uno o más tipos de datos diferentes, como número entero escalar, coma flotante escalar, número entero empaquetado, coma flotante empaquetado, número entero vectorial, coma flotante vectorial, estado (por ejemplo, un puntero de instrucción que es la dirección de la siguiente instrucción que se va a ejecutar), teselas, etc. En una realización, el archivo o archivos de registro físico 1415 comprenden una circuitería de registros vectoriales, circuitería de registros de máscara de escritura y circuitería de registros escalares. Estos circuitos de registro pueden proporcionar registros vectoriales arquitectónicos, registros de máscara vectorial y registros de propósito general. El archivo o archivos de registro físico 1415 se superponen con un circuito de retiro 1417 para ilustrar diversas formas en las que se puede implementar el cambio de nombre de registro y la ejecución en desorden (por ejemplo, usando una memoria o memorias intermedias de reordenación y un archivo o archivos de registro de retiro; usando un archivo o archivos futuros, una memoria o memorias intermedias de historial y un archivo o archivos de registro de retiro; usando mapas de registros y una agrupación de registros; etc.). El circuito de retiro 1417 y el archivo o archivos de registro físico 1415 están acoplados a la circuitería de ejecución 1411.

Si bien el cambio de nombre de registros se describe en el contexto de la ejecución en desorden, se debe entender que el cambio de nombre de registros se puede usar en una arquitectura en orden. Si bien la realización ilustrada del procesador también puede incluir unidades de caché de datos e instrucciones separadas y una unidad de caché L2 compartida, unas realizaciones alternativas pueden tener una única caché interna tanto para instrucciones como para datos, tal como, por ejemplo, una caché interna de nivel 1 (L1), o múltiples niveles de caché interna. En algunas realizaciones, el sistema puede incluir una combinación de una memoria caché interna y una memoria caché externa que es externa al núcleo y/o al procesador. Como alternativa, toda la caché puede ser externa al núcleo y/o al procesador.

La circuitería de ejecución 1411 es un conjunto de uno o más circuitos de ejecución 1427 y un conjunto de uno o más circuitos de acceso a memoria 1425. Los circuitos de ejecución 1427 realizan operaciones de matrices (teselas) detalladas en el presente documento.

A modo de ejemplo, la arquitectura central ilustrativa de cambio de nombre de registro, emisión/ejecución desordenada puede implementar una canalización de la siguiente manera: 1) un circuito de extracción de instrucciones realiza etapas de extracción y decodificación de longitud; 2) la circuitería de bifurcación y decodificación 1403 realiza una etapa de decodificación; 3) la circuitería de cambio de nombre/asignación 1407 realiza una etapa de asignación y una etapa de cambio de nombre; 4) la circuitería de planificador 1409 realiza una etapa de planificador; 5) archivo o archivos de registro físico (acoplados a o incluidos en la circuitería de planificador 1407 y la circuitería de cambio de nombre/asignación 1407 y una unidad de memoria realizan una etapa de lectura de registro/lectura de memoria; la circuitería de ejecución 1411 realiza una etapa de ejecución; 6) una unidad de memoria y la unidad o unidades de archivo o archivos de registro físico realizan una etapa de rescritura/escritura en memoria; 7) diversas unidades pueden estar involucradas en la etapa de manejo de excepciones; y 8) una unidad de retiro y la unidad o unidades de archivo o archivos de registro físico realizan una etapa de confirmación.

El núcleo puede soportar uno o más conjuntos de instrucciones (por ejemplo, el conjunto de instrucciones x86 (con algunas extensiones que se han añadido con versiones más recientes); el conjunto de instrucciones MIPS de MIPS Technologies de Sunnyvale, CA; el conjunto de instrucciones ARM (con extensiones opcionales adicionales tal como NEON) de ARM Holdings de Sunnyvale, CA), incluyendo la instrucción o instrucciones descritas en el presente documento. En una realización, el núcleo 1490 incluye una lógica para soportar una extensión del conjunto de instrucciones de datos empaquetados (por ejemplo, AVX1, AVX2), permitiendo de este modo que las operaciones usadas por muchas aplicaciones multimedia se realicen usando datos empaquetados.

Debe entenderse que el núcleo puede soportar hilos múltiples (ejecutar dos o más conjuntos paralelos de operaciones o hilos), y puede hacerlo de diversas maneras, incluyendo hilos múltiples en intervalos de tiempo, hilos múltiples simultáneos (donde un único núcleo físico proporciona un núcleo lógico para cada uno de los hilos que el núcleo físico está subprocesando de forma múltiple simultáneamente), o una combinación de los mismos (por ejemplo, recuperación y decodificación en cortes de tiempo e hilos múltiples simultáneos a partir de entonces, como en la tecnología Intel® Hyperthreading).

B. DISTRIBUCIÓN

A través de toda esta descripción, los datos se expresan usando la distribución de datos de filas principales. Los usuarios de columna principal deben trasladar los términos de acuerdo con su orientación. La **Figura 15** ilustra un ejemplo de una matriz expresada en formato de fila principal y formato de columna principal. Como se muestra, la matriz A es una matriz de 2x3. Cuando esta matriz se almacena en formato de fila principal, los elementos de datos de una fila son consecutivos. Cuando esta matriz se almacena en formato de columna principal, los elementos de datos de una columna son consecutivos. Es una propiedad bien conocida de las matrices que $A^T * B^T = (BA)^T$, donde el superíndice T significa transponer. La lectura de los datos de columna principal como datos de fila principal da como resultado que la matriz se parezca a la matriz transpuesta.

En algunas realizaciones, la semántica de fila principal se utiliza en hardware, y los datos de columna principal deben intercambiar el orden de los operandos con el resultado de transposiciones de matriz, pero para las posteriores lecturas de columna mayor de la memoria es la matriz correcta, no transpuesta.

Por ejemplo, si hay dos matrices de columna principal para multiplicar:

a b	g i k	ag+bh ai+bj ak+bl
c d *	h j l =	cg+dh ci+dj ck+dl
e f		eg+fh ei+fj ek+fl
(3x2)	(2x3)	(3x3)

Las matrices de entrada se almacenarían en la memoria lineal (columna principal) como:

a c e b d f

y

g h i j k l.

Cuando se leen estas matrices como fila principal con dimensiones 2x3 y 3x2, aparecerían como:

a c e	y	g h
b d f		i j
k l		

Intercambiando el orden y multiplicando la matriz:

g h		a c e	ag+bh cg+dh eg+fh
i j	*	b d f =	ai+bj ci+dj ei+fj
k l			ak+bl ck+dl ek+fl

la matriz transpuesta está disponible y, a continuación, se puede almacenar en el orden de fila principal:

ag+bh	cg+dh	eg+fh	ai+bj	ci+dj	ei+fj	ak+bl	ck+dl	ek+fl

y usada en cálculos de columnas principal posteriores, es la matriz correcta no transpuesta:

ag+bh	ai+bj	ak+bl

cg+dh	ci+dj	ck+dl
eg+fh	ei+fj	ek+fl

III. USO ILUSTRATIVO

La **Figura 16** ilustra un ejemplo de uso de matrices (teselas). En este ejemplo, la matriz C 1601 incluye dos teselas, la tesela A 1603 incluye una tesela y la tesela B 1605 incluye dos teselas. Esta figura muestra un ejemplo del bucle interno de un algoritmo para calcular una multiplicación de matrices. En este ejemplo, se usan dos teselas de resultados, tmm0 y tmm1, de la matriz C 1601 para acumular los resultados intermedios. Una tesela de la matriz A 1603 (tmm2) se reutiliza dos veces al multiplicarse por dos teselas de la matriz B 1605. Los punteros para cargar una nueva tesela A y dos nuevas teselas B desde las direcciones indicadas por las flechas. Un bucle exterior, no mostrado, ajusta los punteros de las teselas C.

El código ilustrativo como se muestra incluye el uso de una instrucción de configuración de teselas y se ejecuta para configurar el uso de teselas, cargar teselas, un bucle para procesar las teselas, almacenar teselas en la memoria y liberar el uso de teselas.

La **Figura 17** ilustra una realización de uso de matrices (teselas). En 1701, se configura el uso de teselas. Por ejemplo, se ejecuta una instrucción TILECONFIG para configurar el uso de teselas, que incluye la configuración de un número de filas y columnas por tesela. Típicamente, al menos una matriz (tesela) se carga desde la memoria en 1703. Se realiza al menos una operación de matriz (tesela) en 1705 usando las matrices (teselas). En 1707, se almacena al menos una matriz (tesela) en la memoria y puede ocurrir un cambio de contexto en 1709.

IV. CONFIGURACIÓN ILUSTRATIVA

A. Soporte de hardware de configuración de teselas

Como se mencionó anteriormente, el uso de teselas típicamente necesita configurarse antes de su uso. Por ejemplo, es posible que no sea necesario el uso completo de todas las filas y columnas. La configuración de estas filas y columnas no sólo ahorra energía en algunas realizaciones, sino que la configuración puede usarse para determinar si una operación generará un error. Por ejemplo, una multiplicación de matrices de la forma (NxM)*(L*N) típicamente no funcionará si M y L no son iguales.

Antes de usar matrices que usan teselas, en algunas realizaciones, ha de configurarse el soporte de teselas. Por ejemplo, se configura cuántas filas y columnas por tesela, teselas que se van a usar, etc. Una instrucción TILECONFIG es una mejora para un ordenador en sí, ya que proporciona soporte para configurar el ordenador para usar un acelerador matricial (ya sea como parte de un núcleo de procesador o como un dispositivo externo). En particular, una ejecución de la instrucción TILECONFIG hace que se recupere una configuración de la memoria y se aplique a los ajustes de matriz (tesela) dentro de un acelerador de matriz.

I. CONFIGURACIÓN DE USO DE TESELAS

La **Figura 18** ilustra el soporte para la configuración del uso de teselas de acuerdo con una realización. Una memoria 1801 contiene la descripción de tesela 1803 de las matrices (teselas) que se van a soportar.

La circuitería de ejecución 1811 de un procesador/núcleo 1805 almacena aspectos de la descripción de tesela 1803 en configuraciones de tesela 1817. Las configuraciones de teselas 1817 detallan qué teselas para una paleta están configurados (el número de filas y columnas en cada tesela) y una marca de que el soporte de matriz está en uso. La circuitería de ejecución 1811 está configurada para usar teselas como se especifica por las configuraciones de teselas 1817. La circuitería de ejecución 1811 también puede incluir un registro de estado de la máquina, un registro específico de modelo (MSR) o un registro de configuración para indicar el uso de la tesela. También se establecen valores adicionales tales como los valores en uso y de inicio. Las configuraciones de teselas 1817 utilizan uno o más registros 1819 para almacenar información de configuración y uso de teselas.

La **Figura 19** ilustra una realización de una descripción de las matrices (teselas) que se van a soportar. Esta es la descripción que se va a almacenar tras la ejecución de una instrucción STTILECFG. En este ejemplo, cada campo es un byte. En el byte[0] se almacena el ID de paleta 1901. El ID de paleta se usa para indexar una tabla de paleta 1813 que almacena, por ID de paleta, un número de bytes en una tesela y bytes por fila de las teselas que están asociados con este ID según se define por la configuración.

El byte 1 almacena un valor que se va a almacenar en un registro "startRow" 1903 y el byte 2 almacena un valor que se va a almacenar en un registro "startP" 1905. Para soportar el reinicio de instrucciones después de estos eventos, las instrucciones almacenan información de estos registros. Para soportar instrucciones de reinicio después de

eventos de interrupción como los detallados anteriormente, las instrucciones almacenan información en estos registros. El valor de startRow indica la fila que se debe usar para reiniciar. El valor de startP indica la posición dentro de la fila para operaciones de almacenamiento cuando se usan pares y, en algunas realizaciones, indica la mitad inferior de la fila (en la tesela inferior de un par) o la mitad superior de la fila (en la tesela superior de un par). Generalmente, la posición en la fila (la columna) no es necesaria.

Con la excepción de TILECONFIG y STTILECFG, la ejecución con éxito de instrucciones de matriz (tesela) establecerá startRow y startP a cero.

Cada vez que no se reinicia una instrucción de matriz (tesela) interrumpida, es responsabilidad del software poner a cero los valores startRow y startP. Por ejemplo, los manejadores de excepciones de coma flotante no enmascarados podrían decidir finalizar la operación en el software y cambiar el valor de contador de programa a otra instrucción, normalmente la siguiente instrucción. En este caso, el manejador de excepciones de software debe poner a cero los valores de startRow y startP en la excepción que le presenta el sistema operativo antes de reanudar el programa. Posteriormente, el sistema operativo recargará esos valores mediante una instrucción de restauración.

El byte 3 almacena una indicación de pares (1b por tesela) de las teselas 1907.

Los bytes 16-17 almacenan el número de filas 1913 y columnas 1915 para la tesela 0, los bytes 18-19 almacenan el número de filas y columnas para la tesela 1, etc. En otras palabras, cada grupo de 2 bytes especifica un número de filas y columnas para una tesela. Si no se usa un grupo de 2 bytes para especificar los parámetros de la tesela, estos deben tener el valor cero. La especificación de parámetros de tesela para más teselas que el límite de implementación o el límite de paleta da como resultado un fallo. Las teselas no configuradas se configuran en un estado inicial con 0 filas y 0 columnas.

Finalmente, la configuración en la memoria típicamente termina con una delimitación final, tal como todos ceros durante varios bytes consecutivos.

II. TESELAS Y ALMACENAMIENTO DE CONFIGURACIÓN TESELAS ILUSTRATIVOS

Las **Figuras 20(A)-(D)** ilustran ejemplos de registro o registros 1819. La **Figura 20(A)** ilustra una pluralidad de registros 1819. Como se muestra en cada tesela (TMM0 2001... TMMN 2003) tiene un registro separado y cada registro almacena un tamaño de fila y columna para esa tesela particular. StartP 2011 y StartRow 2013 se almacenan en registros separados. Se configura uno o más registros de estado 2015 (por ejemplo, TILES_CONFIGURED = 1) para indicar que las teselas están configuradas para su uso.

La **Figura 20(B)** ilustra una pluralidad de registros 1819. Como se muestra, cada tesela tiene registros separados para sus filas y columnas. Por ejemplo, la configuración de filas TMM0 2021, la configuración de columnas TMM0 2023, StartP 2011 y StartRow 2013 se almacenan en registros separados. Se configura uno o más registros de estado 2015 (por ejemplo, TILES_CONFIGURED = 1) para indicar que las teselas están configuradas para su uso.

La **Figura 20(C)** Ilustra un registro único 1819. Como se muestra, este registro almacena las configuraciones de teselas (filas y columnas por tesela) 2031, StartP 2011 y StartRow 2013 se almacenan en un registro único como registros de datos empaquetados. Se configura uno o más registros de estado 2015 (por ejemplo, TILES_CONFIGURED = 1) para indicar que las teselas están configuradas para su uso.

La **Figura 20(D)** ilustra una pluralidad de registros 1819. Como se muestra, un registro único almacena configuraciones de teselas (filas y columnas por tesela) 2031. StartP 2011 y StartRow 2013 se almacenan en registros separados. Se configura uno o más registros de estado 2015 (por ejemplo, TILES_CONFIGURED = 1) para indicar que las teselas están configuradas para su uso.

Se contemplan otras combinaciones tales como combinar los registros de inicio en un registro único donde se muestran por separado, etc.

OPERACIONES DE TESELAS EN CADENA

Los núcleos de procesador de instrucción única y datos múltiples (SIMD) implementados De acuerdo con las realizaciones descritas capturan un paralelismo de datos significativo a través de la ejecución de SIMD. Sin embargo, en ocasiones existe una limitación en el rendimiento de cálculo de SIMD, ya que el rendimiento está limitado por el número de unidades funcionales de SIMD o la anchura de las unidades funcionales de SIMD. Las unidades funcionales de SIMD en ocasiones se denominan en el presente documento motores de procesamiento (PE). Las realizaciones divulgadas aprovechan la arquitectura de TESELAS (TILES) descrita en el presente documento para su uso en la ejecución de SIMD, lo que permite un mayor número de unidades funcionales de SIMD y una mayor anchura de cada unidad funcional.

Las realizaciones divulgadas proporcionan un aumento en la eficiencia energética y la velocidad de ejecución para muchas instrucciones vectorizables que usan la arquitectura de TESELAS para permitir la reutilización de datos entre operaciones de teselas en cadena que acceden a teselas dentro de la matriz. Como se usa en el presente documento, una cadena es una secuencia de dos o más instrucciones (macrooperaciones y/o microoperaciones) que tienen especificadores de teselas de origen y destino en común. Como se usa en el presente documento, se incluye un especificador de tesela de origen en una instrucción para especificar una ubicación desde donde recuperar una tesela de origen para operar sobre ella. Se incluye un especificador de tesela de destino en una instrucción para especificar una tesela de destino en la que escribir el resultado de la instrucción. Las teselas de origen y destino especificadas pueden almacenarse en uno cualquiera o más de una memoria, una jerarquía de caché, una memoria de bloc de notas y un archivo de registro.

Para describir un ejemplo no limitativo de una cadena de operaciones de tesela, una primera instrucción puede producir un resultado que se va a escribir en una tesela de destino especificada, pero el resultado también se puede usar como una tesela de origen especificada de una segunda instrucción. De acuerdo con algunas realizaciones, el especificador de tesela de destino de la primera instrucción ha de apartarse y, en su lugar, el resultado de la primera instrucción ha de enrutarse a un motor de procesamiento que ejecuta la segunda instrucción. En algunas otras realizaciones, el especificador de tesela de destino de la primera instrucción ha de ignorarse y, en su lugar, el resultado de la primera instrucción ha de enrutarse a un motor de procesamiento que ejecuta una instrucción posterior en la cadena. En algunas otras realizaciones, el especificador de tesela de destino de la primera instrucción ha de incumplirse y, en su lugar, el resultado de la primera instrucción ha de enrutarse a un motor de procesamiento que ejecuta una instrucción posterior en la cadena. En algunas otras realizaciones, el especificador de tesela de destino de la primera instrucción ha de pasarse por alto y, en su lugar, el resultado de la primera instrucción ha de enrutarse a un motor de procesamiento que ejecuta una instrucción posterior en la cadena. En algunas otras realizaciones, el especificador de tesela de destino de la primera instrucción ha de desobedecerse y, en su lugar, el resultado de la primera instrucción ha de enrutarse a un motor de procesamiento que ejecuta una instrucción posterior en la cadena. En algunas otras realizaciones, el especificador de tesela de destino de la primera instrucción ha de rechazarse y, en su lugar, el resultado de la primera instrucción ha de enrutarse a un motor de procesamiento que ejecuta una instrucción posterior en la cadena.

En algunas realizaciones, el resultado de la primera instrucción se enruta a múltiples instrucciones posteriores. En algunas realizaciones, la segunda instrucción es para recibir resultados de múltiples instrucciones anteriores en la cadena.

Las cadenas de instrucciones no están limitadas en su longitud; y muchas instrucciones pueden estar en cadena. Se describe e ilustra además una cadena de instrucciones al menos con respecto a las **Figuras 21-23** (que muestran diagramas de flujo de procesos que ilustran la ejecución de una cadena optimizada de instrucciones), la **Figura 24A** (que muestra pseudocódigo optimizado para ejecutar operaciones eficientes de teselas en cadena), y la **Figura 25B** (que muestra un formato de instrucción para una instrucción de inicio de cadena que especifica los vínculos entre las instrucciones de la cadena).

Para mejorar la eficiencia de procesador permitiendo la reutilización de datos entre múltiples operaciones cuando se ejecuta una cadena de instrucciones, las realizaciones divulgadas mejoran la eficiencia al optar por apartar el especificador de tesela de destino de una primera instrucción y, en su lugar, enrutar un resultado de la primera instrucción directamente a una segunda instrucción, sin antes reescribir el resultado en la memoria. Tal apartamiento se puede implementar para múltiples instrucciones en la cadena, sin limitación.

Ignorar el especificador de tesela de destino de la primera instrucción mejora el rendimiento, al menos porque la segunda instrucción no necesita esperar a que los resultados de la primera instrucción se rescriban en el almacenamiento. En algunas realizaciones, las instrucciones dentro de una cadena de instrucciones incluyen un campo de control de cadena para proporcionar una sugerencia a la circuitería de ejecución en cuanto a si realizar el apartamiento y cómo hacerlo. En algunas realizaciones, la circuitería de ejecución incluye un traductor binario que monitoriza las dependencias entre múltiples instrucciones y determina si y cuándo apartar uno o más especificadores de tesela de destino y, en su lugar, enrutar los resultados de una o las múltiples instrucciones a uno o más PE que ejecutan las siguientes instrucciones en la cadena de instrucciones.

En algunas realizaciones, la circuitería de ejecución realiza comprobaciones de consistencia regulares para determinar cualquier problema provocado por el apartamiento. En algunas realizaciones, la circuitería de ejecución mantiene un registro de las instrucciones cuyos especificadores de tesela de destino se apartaron, de modo que el procesador de SIMD pueda retroceder y volver a ejecutar esas instrucciones grabadas, esta vez sin apartar.

Las realizaciones divulgadas no intentan aumentar la eficiencia de SIMD limitando la elección de núcleos de ejecución a un tipo. Más bien, algunas realizaciones utilizan una mezcla de motores de procesamiento (PE) asimétricos, algunos de los cuales tienen una funcionalidad más limitada y utilizan menos energía que otros. En algunas realizaciones, una ruta de ejecución SIMD junto con la selección de motores de procesamiento entre motores de procesamiento asimétricos se realiza dinámicamente en tiempo de ejecución.

Algunas realizaciones divulgadas aprovechan un paradigma de una matriz de motores de procesamiento (PE) asimétricos. En algunas realizaciones, cada uno de los PE en la matriz asimétrica es una unidad aritmético-lógica (ALU) que puede realizar únicamente un número limitado de operaciones. Por ejemplo, el conjunto de PE asimétricos puede incluir algunos PE especializados al desplazamiento, algunos PE especializados a la suma y resta, algunos PE especializados a la conversión de formato, algunos PE especializados a la división, algunos PE especializados a la multiplicación, algunos PE especializados a operaciones unarias, algunos PE especializados a operaciones lógicas, y así sucesivamente, sin limitación. En algunas realizaciones, al menos algunos de los PE pueden ejecutar más de un tipo de operación. En operación, la circuitería de ejecución es para seleccionar y configurar dinámicamente un conjunto de PE para realizar un conjunto de instrucciones en una cadena de instrucciones de SIMD. Se pueden seleccionar diferentes PE para realizar diferentes operaciones, haciendo coincidir el rendimiento requerido con el rendimiento proporcionado, produciendo de esta manera una implementación eficiente en cuanto a energía. Una ventaja de esta matriz asimétrica es que permite una estrecha coincidencia entre la potencia de procesamiento requerida y la potencia gastada: únicamente se usa la potencia de procesamiento necesaria. En algunas realizaciones, la energía se conserva ejecutando una configuración única de un conjunto de PE para realizar diferentes partes de una cadena de operaciones de tesela y permitiendo que esa configuración se reutilice para los elementos de matriz restantes. Como se describe en el presente documento, se puede hacer referencia a una "tesela" y es un caso especial de una matriz.

En algunas realizaciones, la matriz asimétrica de PE incluye una interconexión dinámicamente configurable para enrutar datos entre múltiples PE en una ruta de procesamiento de SIMD. En operación, la circuitería de ejecución puede seleccionar un conjunto de PE para ejecutar una cadena de instrucciones de SIMD y configurar dinámicamente una red para transferir datos entre ellos. Una ventaja de esta interconexión dinámicamente configurable es permitir conexiones entre un amplio conjunto de PE sin incurrir en el coste y el área de conexión necesaria de cada PE con todos los demás PE.

Una ventaja de esta matriz asimétrica de PE y de interconexión dinámicamente configurable es que no es necesario reescribir los resultados intermedios en un archivo de registro ni colocarlos en una red de desvío; en su lugar, los resultados de un PE pueden enrutarse al siguiente PE en la cadena de procesamiento de SIMD. Por lo tanto, este hardware puede producir un procesador de SIMD con mejor rendimiento y mayor eficiencia en cuanto a la energía.

EJECUCIÓN ILUSTRATIVA

La **Figura 21A** es un diagrama de flujo de bloques que ilustra una ejecución ilustrativa de una cadena optimizable de instrucciones de teselas. Como se muestra, la cadena de instrucciones no optimizada 2100 especifica una secuencia aritmética de operaciones para establecer una tesela de resultados como una función de tres teselas de origen, en concreto, resultado = ((A + B) * 3) -C). Una cadena de instrucciones de este tipo podría ejecutarse N veces, generando cada vez un resultado que se va a almacenar en cada elemento de una tesela que tiene M filas y K columnas de elementos. Sin embargo, por simplicidad y para ilustrar la optimización, únicamente se ilustra una instancia de la instrucción, es decir, únicamente una iteración del bucle para generar una tesela.

Para optimizar la cadena de instrucciones de acuerdo con las realizaciones divulgadas en el presente documento, han de apartarse uno o más de los especificadores de tesela de destino de las instrucciones y, en su lugar, los resultados de esa instrucción han de enrutarse a una instrucción posterior en la cadena. En algunas realizaciones, la circuitería de ejecución mantiene un registro de las instrucciones que han apartado sus especificadores de tesela de destino y retrocede para recuperarse volviendo a ejecutar una o más de las instrucciones registradas si se detecta un problema resultante del apartamiento.

En algunas realizaciones, la circuitería de ejecución es para mantener un registro de las instrucciones que han apartado sus especificadores de tesela de destino, de modo que la circuitería de ejecución pueda retroceder y volver a ejecutar la instrucción de nuevo sin tener que apartarlas. En algunas realizaciones, la circuitería de ejecución es, además, para guardar un estado de máquina antes del apartamiento y, posteriormente, ejecutar una comprobación de consistencia de estado de máquina para detectar cualquier inconsistencia provocada por el apartamiento. En algunas realizaciones, la circuitería de ejecución, además de dejar de lado un especificador de tesela de destino de una instrucción, es para establecer un bit sucio en la configuración de esa tesela y detectar una inconsistencia si una instrucción posterior lee una tesela sucia. La circuitería de ejecución en algunas realizaciones es para generar un fallo si se detectan tales inconsistencias, y, en algunas realizaciones, la circuitería de ejecución es para retroceder y volver a ejecutar una o más instrucciones.

En operación, como se muestra, la cadena de instrucciones 2100 ha de extraerse del almacenamiento como una secuencia de instrucciones (por ejemplo, microoperaciones) o macroinstrucciones, teniendo cada una un especificador de tesela de destino y un especificador de tesela de origen. Típicamente, las instrucciones en la cadena de instrucciones especifican, como orígenes y destinos, teselas intermedias que se almacenan en la memoria de bloc de notas 2102. En algunas realizaciones, las instrucciones en la cadena de instrucciones especifican teselas de origen y/o teselas de destino en un archivo de registro, en una caché o en alguna otra estructura de memoria. En algunas realizaciones, una memoria de bloc de notas es una memoria, que incluye uno o más registros o interruptores o incluso memoria caché y otra memoria estática, que se coloca relativamente cerca de la circuitería de ejecución y almacena

resultados de datos intermedios. En algún momento, las teselas en la memoria de bloc de notas han de confirmarse en memoria.

Suponiendo que las matrices (teselas) A, B y C ya se han cargado en la memoria de bloc de notas 2102, las matrices (teselas) A, B y C se recuperan de la memoria de bloc de notas 2102 en la tesela A 2104, la tesela B 2106 y la tesela C 2108, respectivamente.

En la adición 2110, se añaden las teselas A y B, y el resultado en la tesela E 2112 ha de rescribirse en la tesela E en la memoria de bloc de notas 2102 (o en un archivo de registro, en una caché o en alguna otra estructura de memoria). La tesela E 2112 se muestra con un borde de línea discontinua para resaltar que es un punto potencial de optimización. En algunas realizaciones, como se divulga en el presente documento y como se ilustra y describe con respecto a la **Figura 21B,** la circuitería de ejecución es para apartar el especificador de tesela de destino, la tesela E, de la instrucción ejecutada en la suma 2110, y en su lugar enrutar los resultados de la operación en 2112 para que se ejecuten en una siguiente operación, multiplicar 2114, en la cadena de instrucciones.

En la multiplicación 2114, el resultado de sumar 2110, o la tesela E, dependiendo de si se apartó la tesela de destino especificada de sumar 2110, se enruta a un motor de procesamiento para multiplicar por un valor (por ejemplo, 3) difundido 2120 desde la memoria de bloc de notas 2102, y el resultado en la tesela F 2116 se rescribirá en la tesela F en la memoria de bloc de notas 2102. La tesela F 2116 se muestra con un borde de línea discontinua para resaltar que es un punto potencial de optimización. En algunas realizaciones, como se divulga en el presente documento y como se ilustra y describe con respecto a la **Figura 21B,** la circuitería de ejecución es para apartar el destino especificado, la tesela F, de multiplicar 2114 y, en lugar de enrutar el resultado de multiplicar 2114 a una siguiente instrucción en la cadena de instrucciones, restar 2118.

En la operación de resta 2118, el resultado de multiplicar 2114, o la tesela F, dependiendo de si se apartó la tesela de destino especificada de multiplicar 2114, se reenvía a un motor de procesamiento como origen de la que se va a restar la tesela C 2108. La operación de resta 2118 representa la última instrucción en la cadena, por lo que la optimización no está disponible, y un resultado, la tesela D 2122, de la operación de resta 2118 ha de almacenarse en 2124 de nuevo en la memoria de bloc de notas 2102 (o en un archivo de registro, en una caché, o en alguna otra estructura de memoria).

La **Figura 21B** es un diagrama de flujo de bloques que ilustra una ejecución ilustrativa de una cadena optimizada de instrucciones de teselas. Como se muestra, la cadena de instrucciones optimizada 2150, como la cadena de instrucciones 2100 de la **Figura 21A,** especifica una secuencia aritmética de operaciones para establecer una tesela de resultado como una función de tres teselas de origen, es decir, resultado = ((A + B) * 3) -C). Una cadena de instrucciones de este tipo podría ejecutarse N veces, generando cada vez un resultado que se va a almacenar en cada elemento de una tesela que tiene M filas y K columnas de elementos. Sin embargo, por simplicidad y para ilustrar la optimización, únicamente se ilustra una instancia de la instrucción, es decir, únicamente una iteración del bucle para generar una tesela.

Para optimizar la cadena de instrucciones de acuerdo con algunas realizaciones, han de apartarse uno o más especificadores de tesela de destino de las instrucciones y, en su lugar, los resultados de esas instrucciones han de enrutarse a instrucciones posteriores en la cadena.

En operación, como se muestra, la cadena de instrucciones 2150 ha de extraerse del almacenamiento como una secuencia de instrucciones (por ejemplo, microoperaciones) o macroinstrucciones, especificando cada una un código de operación aritmético y que tiene un especificador de tesela de destino y al menos un especificador de tesela de origen. Típicamente, las instrucciones en la cadena de instrucciones especifican como teselas de origen y teselas de destino, teselas intermedias almacenadas en la memoria de bloc de notas 2152. En algún momento, las teselas de la memoria de bloc de notas se confirman en memoria.

Suponiendo que las matrices (teselas) A, B y C ya se han cargado en la memoria de bloc de notas 2152, las matrices (teselas) A, B y C se recuperan de la memoria de bloc de notas 2152 en 2154, 2156 y 2158, respectivamente.

En 2160, las teselas A y B se añaden en respuesta a una instrucción que especifica las teselas A y B como teselas de origen, y la tesela E como tesela de destino. En operación, un procesador de SIMD ha de realizar la suma de múltiples elementos vectoriales al mismo tiempo y en paralelo. La circuitería de ejecución optimiza el flujo al dejar de lado el especificador de tesela de destino en 2160 y, en su lugar, enrutar el resultado de la suma a la siguiente instrucción de la cadena, multiplicar 2164. En operación, un procesador ha de realizar la multiplicación 2164 en múltiples elementos vectoriales al mismo tiempo y en paralelo.

DETERMINAR SI Y CUÁNDO APARTAR

De acuerdo con las realizaciones descritas en el presente documento, existen múltiples maneras en las que se puede activar la circuitería de ejecución para optimizar el flujo volviendo a enrutar el resultado de la operación sumar 2160 a la operación multiplicar 2164.

En primer lugar, en algunas realizaciones, la circuitería de ejecución incluye un traductor binario que almacena en memoria intermedia una secuencia de instrucciones y las analiza dinámicamente en tiempo de ejecución para detectar dependencias de datos entre instrucciones. El traductor binario, por ejemplo, puede predecir que un resultado de una operación ejecutada se usará únicamente una vez, en la siguiente instrucción de la cadena, y no se usará de nuevo. Basándose en una predicción de este tipo, la circuitería de ejecución puede optar por apartar una tesela de destino especificada en la memoria de bloc de notas y, en su lugar, enrutar el resultado al siguiente motor de procesamiento que ejecuta la siguiente instrucción en la cadena.

Además, en algunas realizaciones divulgadas, la instrucción sumar 2160 está formateada de acuerdo con un formato de instrucción compatible con la cadena, tal como CHAIN_TILE_OP, que, como se ilustra y describe más adelante con respecto a la **Figura 25A,** especifica una operación aritmética, una tesela de destino, hasta tres teselas de origen, posiblemente una inmediato, y un campo de control de cadena. El formato de la instrucción CHAIN_TILE_OP y otros formatos de instrucción de las realizaciones divulgadas se ilustra y describe con más detalle con respecto a las Figuras 25C-D y las Figuras 26A-D.

Se describen, sin limitación, otras maneras de determinar si existe una cadena y optimizarla. En algunas realizaciones, el campo de control de cadena de CHAIN_TILE_OP incluye un encabezado para identificar una primera instrucción y una o más instrucciones posteriores como parte de la cadena o las instrucciones que se van a optimizar de acuerdo con las realizaciones divulgadas. En algunas realizaciones, un campo de control de cadena sirve para indicar una sugerencia de posición de cadena que es, por ejemplo, una sugerencia de inicio de cadena, una sugerencia intermedia de cadena y una sugerencia de final de cadena para marcar cada instrucción en la cadena. En algunas realizaciones, el campo de control de cadena de cada instrucción en la cadena especifica un índice que indica la posición relativa de la instrucción en la cadena.

Además, en algunas realizaciones divulgadas, una instrucción de inicio de cadena, como se describe e ilustra con más detalle con respecto a la **Figura 25B,** hace que la circuitería de ejecución reconozca que debe seguir una cadena de instrucciones y proporciona algunos detalles de la cadena.

De acuerdo con las realizaciones divulgadas, la circuitería de ejecución, en respuesta a determinar la existencia de una cadena de instrucciones para optimizar, es para seleccionar y configurar dinámicamente una ruta de SIMD de motores de procesamiento para realizar la cadena de instrucciones, para apartar las teselas de destino especificadas de todas menos la última instrucción en la cadena, y para enrutar los resultados de todas menos la última instrucción al siguiente PE que realice una siguiente instrucción en la cadena. En consecuencia, como se muestra, la circuitería de ejecución determina la existencia de una cadena y vuelve a enrutar el resultado de sumar 2160 desde la tesela de destino especificada, la tesela E, al PE que ejecuta la multiplicación 2164. De manera similar, la circuitería de ejecución continúa procesando la cadena volviendo a enrutar el resultado de multiplicar 2164 desde la tesela de destino especificada F al PE que ejecuta la siguiente instrucción en la cadena, restar 2168. En operación, un procesador de SIMD ha de realizar la suma 2160, la multiplicación 2164 y la resta 2168 en múltiples elementos de tesela al mismo tiempo y en paralelo. Al no haber más instrucciones en la cadena después de la resta 2168, la circuitería de ejecución enruta el resultado de la resta 2168 a la tesela de destino especificada D 2172 para que se almacene en 2174 de vuelta a la memoria de bloc de notas 2152.

SELECCIONAR MOTORES DE PROCESAMIENTO (PE) Y CONFIGURACIÓN DE UNA RUTA DE SIMD

Para implementar operaciones de teselas en cadena, las realizaciones descritas seleccionan motores de procesamiento para realizar las operaciones matemáticas en cadena. En algunas realizaciones, un procesador de SIMD incluye una matriz asimétrica de motores de procesamiento (PE). Una matriz de este tipo tiene una diversidad de diferentes tipos de motores de procesamiento, algunos tienen más capacidades de procesamiento, pero consumen más energía o área de encapsulado que otros.

En algunas realizaciones, el procesador de SIMD tiene motores de procesamiento adaptados a las operaciones matemáticas que se realizan. Por ejemplo, algunos motores de procesamiento pueden estar especializados a operaciones unarias, tales como incremento, decremento, raíz cuadrada, negación u operaciones trigonométricas, tales como seno, coseno y tangente, por nombrar algunos ejemplos no limitantes. Algunos PE pueden estar especializados a operaciones binarias, tales como operaciones matemáticas (por ejemplo, sumar, restar, dividir, multiplicar, módulo, etc.) y operaciones lógicas (por ejemplo, AND, OR, NAND, NOR, XOR, XNOR, CMP, etc.), por nombrar algunos ejemplos no limitantes. En algunas realizaciones, el procesador de SIMD tiene una agrupación de motores de procesamiento simétrico para su uso en la implementación de operaciones de teselas en cadena.

En operación, de acuerdo con algunas realizaciones, la circuitería de ejecución selecciona y configura los PE para ejecutar operaciones de teselas en cadena. Por ejemplo, en la realización de la **Figura 21B,** la circuitería de ejecución selecciona PE para procesar múltiples elementos de tesela en paralelo. En algunas realizaciones, la circuitería de ejecución intenta seleccionar y configurar un número mínimo de PE, haciendo coincidir la potencia de procesamiento con los requisitos de procesamiento de las teselas. En algunas realizaciones, se selecciona un PE o conjunto de PE diferente para operar en paralelo en múltiples elementos de una tesela. En algunas realizaciones, cuando se

seleccionan PE para realizar las operaciones de una cadena de instrucciones, la circuitería de ejecución selecciona PE que están dispuestos uno cerca del otro en el encapsulado del procesador, para minimizar las latencias de enrutamiento cuando se ejecuta una cadena de operaciones de SIMD. En algunas realizaciones, cuando se seleccionan PE para realizar las operaciones de una cadena de instrucciones, la circuitería de ejecución ha de seleccionar PE distribuidos uniformemente en varias áreas del encapsulado, para equilibrar y distribuir la utilización de energía incluso a través de toda el área del encapsulado.

Algunas realizaciones divulgadas aprovechan un paradigma de una matriz de motores de procesamiento (PE) asimétricos. En algunas realizaciones, cada uno de los PE en la matriz asimétrica es una unidad aritmético-lógica (ALU) que puede realizar únicamente un número limitado de operaciones. Por ejemplo, el conjunto de PE asimétricos puede incluir algunos PE especializados al desplazamiento, algunos PE especializados a la suma y resta, algunos PE especializados a la conversión de formato, algunos PE especializados a la división, algunos PE especializados a la multiplicación, algunos PE especializados a operaciones unarias, algunos PE especializados a operaciones lógicas, y así sucesivamente, sin limitación. En algunas realizaciones, al menos algunos de los PE pueden ejecutar más de un tipo de operación. En operación, la circuitería de ejecución es para seleccionar y configurar dinámicamente un conjunto de PE para realizar un conjunto de instrucciones en una cadena de instrucciones de SIMD. Se pueden seleccionar diferentes PE para realizar diferentes operaciones, haciendo coincidir el rendimiento requerido con el rendimiento proporcionado, produciendo de esta manera una implementación eficiente en cuanto a energía. Una ventaja de esta matriz asimétrica es que permite una estrecha coincidencia entre la potencia de procesamiento requerida y la potencia gastada: únicamente se usa la potencia de procesamiento necesaria. En algunas realizaciones, la energía se conserva ejecutando una configuración única de un conjunto de PE para realizar diferentes partes de una cadena de operaciones de tesela y permitiendo que esa configuración se reutilice para los elementos de matriz restantes. Como se describe en el presente documento, se puede hacer referencia a una "tesela" y es un caso especial de una matriz.

FLUJOS DE EJECUCIÓN DE PROCESADORES ILUSTRATIVOS

La **Figura 22** es un diagrama de flujo de bloques que ilustra un flujo de ejecución de un procesador que responde a instrucciones de teselas en cadena, de acuerdo con una realización. En 2201, el procesador ha de extraer, usando circuitería de extracción, una pluralidad de instrucciones, cada una de las cuales tiene especificadores de teselas de origen y destino para especificar las respectivas teselas de origen y destino. En 2203, el procesador ha de almacenar en una memoria intermedia, la pluralidad de instrucciones extraídas. La operación 2203 es opcional, según se indica por su borde discontinuo, en la medida en que la pluralidad de instrucciones extraídas pueden almacenarse en memoria intermedia en cualquier otro sitio, o no almacenarse en absoluto. En 2205, el procesador ha de decodificar, usando circuitería de decodificación, la pluralidad de instrucciones extraídas. En 2207, el procesador ha de planificar la ejecución de la pluralidad de instrucciones decodificadas. La operación 2207 es opcional, como lo indica su borde discontinuo, en la medida en que puede producirse en un momento diferente o no producirse en absoluto. En 2209, el procesador ha de ejecutar, usando circuitería de ejecución, la pluralidad de instrucciones decodificadas de la siguiente manera.

En 2211, la circuitería de ejecución del procesador es para identificar la primera y la segunda instrucciones decodificadas que pertenecen a una cadena de instrucciones. La sección titulada "Determinar si apartar y cuándo" describe varias formas en que el procesador ha de identificar la primera y la segunda instrucciones decodificadas que pertenecen a una cadena de instrucciones. Una realización, por ejemplo, implica que el procesador extraiga una instrucción de inicio de cadena antes de extraer la pluralidad de instrucciones en 2201, como se describe e ilustra adicionalmente con respecto a la **Figura 25B.** La instrucción de inicio de cadena es para avisar a la circuitería de ejecución de que ha de seguir una cadena de instrucciones y proporcionar algunos detalles acerca de la cadena.

En 2213, la circuitería de ejecución del procesador es para seleccionar y configurar dinámicamente una ruta de SIMD que comprende un primer y un segundo motores de procesamiento (PE) para ejecutar la primera y la segunda instrucciones decodificadas. En 2215, la circuitería de ejecución del procesador es para configurar dinámicamente una ruta de red desde el primer PE al segundo PE. La operación 2215 es opcional, como se indica por su borde discontinuo, en la medida en que la ruta de red ya se haya podido configurar.

En 2217, de acuerdo con la realización ilustrada, la circuitería de ejecución del procesador ha de apartar el especificador de tesela de destino de la primera instrucción decodificada y, en su lugar, enrutar un resultado de la primera instrucción decodificada desde el primer PE al segundo PE que se usará por el segundo PE para realizar la segunda instrucción decodificada. En algunas otras realizaciones, el especificador de tesela de destino de la primera instrucción ha de ignorarse y, en su lugar, el resultado de la primera instrucción ha de enrutarse a un motor de procesamiento que ejecuta una instrucción posterior en la cadena. En algunas otras realizaciones, el especificador de tesela de destino de la primera instrucción ha de incumplirse y, en su lugar, el resultado de la primera instrucción ha de enrutarse a un motor de procesamiento que ejecuta una instrucción posterior en la cadena. En algunas otras realizaciones, el especificador de tesela de destino de la primera instrucción ha de pasarse por alto y, en su lugar, el resultado de la primera instrucción ha de enrutarse a un motor de procesamiento que ejecuta una instrucción posterior en la cadena. En algunas otras realizaciones, el especificador de tesela de destino de la primera instrucción ha de desobedecerse y, en su lugar, el resultado de la primera instrucción ha de enrutarse a un motor de procesamiento que ejecuta una instrucción posterior en la cadena. En algunas otras realizaciones, el especificador de tesela de destino

de la primera instrucción ha de rechazarse y, en su lugar, el resultado de la primera instrucción ha de enrutarse a un motor de procesamiento que ejecuta una instrucción posterior en la cadena.

En algunas realizaciones, la circuitería de ejecución es para mantener (no mostrado en la **Figura 22)** un registro de instrucciones que han apartado sus especificadores de tesela de destino, y para retroceder y volver a ejecutar uno o más registros de instrucciones según sea necesario.

En 2219, el procesador ha de confirmar un resultado de las instrucciones ejecutadas. La operación 2219 es opcional, como lo indican sus bordes discontinuos, en la medida en que una reescritura de los resultados puede ocurrir en un tiempo diferente, o no ocurrir en absoluto.

La **Figura 23** es un diagrama de flujo de bloques que ilustra la ejecución de instrucciones de teselas en cadena mediante un procesador de una única instrucción y múltiples datos (SIMD), de acuerdo con una realización. Típicamente, el flujo 2300 se realiza mediante circuitería de ejecución tal como la que se detalla en el presente documento, al menos con respecto a las **Figuras 13, 14**, 28A-B y 29A-B.

En 2302, un procesador ha de extraer, decodificar y almacenar en memoria intermedia una o más instrucciones hasta que se haya extraído un conjunto de instrucciones, especificando cada instrucción matrices (teselas) de origen y destino.

En 2304, el procesador prueba un número de condiciones de error, la presencia de cualquiera de las cuales genera un fallo en 2306. Como se muestra, las condiciones de error probadas en 2304 incluyen: 1) si las teselas aún no están configuradas, 2) si el destino y los dos teselas de origen son válidos, 3) si cualquiera de los dos orígenes es un par y 4) cualquiera de los diversos números de filas y columnas está fuera de rango.

Cuando ninguna de las condiciones de fallo probadas en 2304 es verdadera, el procesador en 2308 determina si la primera y la segunda instrucciones decodificadas son parte de una cadena de instrucciones. De lo contrario, el procesador vuelve a 2302 para buscar al menos una instrucción más hasta que se haya extraído el conjunto de instrucciones. Pero, en caso afirmativo, el procesador en 2310 selecciona y configura dinámicamente una ruta de ejecución de SIMD que incluye primer y segundo motores de procesamiento (PE) para ejecutar la primera y la segunda instrucciones decodificadas. En 2312, el procesador ha de apartar el especificador de tesela de destino de la primera instrucción decodificada y, en su lugar, enrutar un resultado de la primera instrucción decodificada desde el primer PE al segundo PE que se usará por el segundo PE cuando se ejecuta la segunda instrucción decodificada. En 2314, el procesador determina si la cadena está completa. De lo contrario, el procesador vuelve a 2302 para buscar al menos una instrucción más hasta que se haya extraído el conjunto de instrucciones. Pero, en caso afirmativo, finaliza el flujo 2300.

IV. PSEUDOCÓDIGO ILUSTRATIVO

La **Figura 24A** es un código nativo ilustrativo que ilustra un bucle de código de SIMD que un procesador va a optimizar, de acuerdo con algunas realizaciones. Como se muestra, el bucle de SIMD en modo nativo 2402 es una función, en concreto, resultado = ((B + C) <<2) + k, que se va a realizar en cada elemento de una tesela M × N.

La **Figura 24B** es un pseudocódigo que ilustra una cadena no optimizada de instrucciones de tesela. Una cadena de instrucciones de este tipo podría ejecutarse M x N veces, por ejemplo, una vez por cada elemento de una tesela que tiene M filas y N columnas de elementos. Sin embargo, por simplicidad y para ilustrar la optimización, únicamente se ilustra una instancia de la instrucción, es decir, únicamente una iteración del código de SIMD ejecutada en cada elemento de bucle. Aquellas instrucciones en el bucle de código de SIMD 2404 cuyos comentarios incluyen un asterisco representan candidatos para optimización, de acuerdo con realizaciones divulgadas en el presente documento, e ilustradas y descritas con respecto a la **Figura 24C.**

La **Figura 24C** es un pseudocódigo que ilustra un flujo de ejecución mediante una circuitería de ejecución de procesador para ejecutar una cadena optimizada de instrucciones de tesela, de acuerdo con algunas realizaciones. Como se muestra, el pseudocódigo 2406 es el mismo que el pseudocódigo no optimizado de la **Figura 24B,** pero sin las instrucciones marcadas con un asterisco. Entonces, en lugar de almacenar la suma de B y C en la tesela A, la circuitería de ejecución aparta la tesela de destino especificada (A) y, en su lugar, envía los resultados de la suma directamente al motor de procesamiento que realiza la instrucción.

El pseudocódigo en la **Figura 24B** y la **Figura 24C** se autodocumenta en virtud de sus comentarios y nombres de variables incluidos.

II. FORMATO O FORMATOS DE INSTRUCCIONES ILUSTRATIVOS

La **Figura 25A** es un diagrama de bloques que ilustra un formato ilustrativo para una instrucción compatible con cadena, de acuerdo con algunas realizaciones. Como se muestra, la instrucción CHAIN_TILE_OP 2500 es una instrucción compatible con cadena que incluye el código de operación 2501, el control de cadena 2502, el destino

2503 y el origen 1 2504. También se muestran varios parámetros opcionales, incluyendo el origen 2 2505, el origen 3 2506, el inmediato 2507, la máscara de escritura 2508 y el tamaño 2509.

El código de operación 2501 especifica una operación de tesela aritmética que se realizará en cada elemento de una tesela. La operación especificada puede ser una operación unaria, una operación binaria (en cuyo caso también ha de especificarse el origen 2 2505) o una operación ternaria (en cuyo caso ha de especificarse el origen 2 2505 y el origen 3 2506). Los ejemplos de operaciones unarias que han de especificarse incluyen incremento, decremento, raíz cuadrada, negación u operaciones trigonométricas, tales como seno, coseno y tangente, por nombrar algunos ejemplos no limitantes. Los ejemplos de operaciones binarias que se especificarán incluyen algunos ejemplos de los cuales, sin limitación, incluyen operaciones matemáticas (por ejemplo, sumar, restar, dividir, multiplicar, módulo, etc.), operaciones lógicas (por ejemplo, AND, OR, NAND, NOR, XOR, XNOR, CMP, etc.), intercambio, umbral y suelo, por nombrar algunos ejemplos no limitativos.

En algunas realizaciones, el control de cadena 2502, como se ilustra y describe al menos con respecto a la **Figura 21B,** es un encabezado para identificar una primera instrucción y una o más instrucciones posteriores como parte de la cadena o las instrucciones que se van a optimizar de acuerdo con las realizaciones divulgadas. En algunas realizaciones, un campo de control de cadena sirve para indicar una sugerencia de posición de cadena que es, por ejemplo, una sugerencia de inicio de cadena, una sugerencia intermedia de cadena y una sugerencia de final de cadena para marcar cada instrucción en la cadena. En algunas realizaciones, el campo de control de cadena de cada instrucción en la cadena especifica un índice que indica la posición relativa de la instrucción en la cadena.

El destino 2503, origen 1 2504, origen 2 2505 y origen 3 2506 son especificadores de teselas para especificar una tesela de destino y teselas de origen. El destino 2503, en algunas realizaciones, es un especificador de tesela de destino para identificar un destino de una instrucción de tesela en cadena. En algunas realizaciones, el destino 2503 es un especificador de tesela de destino que especifica dos o más destinos para un resultado de la instrucción.

Inmediato 2507 es para permitir la especificación de un inmediato en una instrucción. El inmediato 2507 es opcional, como lo indica su borde discontinuo, en el sentido de que no está presente en una implementación del formato compatible con vectores genérico que no soporta un inmediato y no está presente en instrucciones que no usan un inmediato.

La máscara de escritura 2508 es un campo opcional (como lo indica su borde discontinuo) cuyo contenido controla, según la posición del elemento de datos, si esa posición de elemento de datos en la tesela de destino es para reflejar el resultado de la operación. Algunas instrucciones soportan la fusión-máscara de escritura, mientras que otras soportan tanto la fusión como la puesta a cero-máscara de escritura. Cuando se fusiona, las máscaras vectoriales permiten proteger cualquier conjunto de elementos en la tesela de destino de actualizaciones durante la ejecución de cualquier operación; en otra realización, preservar el valor antiguo de cada elemento de la tesela de destino donde el bit de máscara correspondiente tiene un 0. Por el contrario, las máscaras vectoriales de puesta a cero permiten poner a cero cualquier conjunto de elementos en el destino durante la ejecución de cualquier operación. Si no se especifica ninguna máscara de escritura, todos los elementos de destino se tratan como si estuvieran no enmascarados.

El tamaño 2509 es un campo opcional (como lo indica su borde discontinuo) para indicar el tamaño de los elementos de datos de tesela en los que se opera. El tamaño 2509 se muestra como un operando de instrucción separado, pero el asterisco se incluye para indicar que, en algunas realizaciones, ha de incluirse en un código de operación, prefijo o sufijo, "B", "W", "D" y "Q", correspondiente a un tamaño de 1 byte, 2 bytes, 4 bytes u 8 bytes, respectivamente, de cada elemento de tesela. En algunas realizaciones, el tamaño 2509 se incluye en el código de operación, como un prefijo o sufijo, "H", "S", "D", "Q" y "E", correspondientes a niveles de precisión: precisión media (2 bytes), precisión sencilla (4 bytes), precisión doble (8 bytes) y precisión cuádruple (16 bytes), respectivamente, de cada elemento de tesela.

La **Figura 25B** es un diagrama de bloques que ilustra una instrucción de inicio de cadena, de acuerdo con algunas realizaciones. Como se muestra, la instrucción de inicio de cadena 2510 incluye el código de operación 2511, la longitud de cadena 2512, los objetivos de cadena 2513, que incluyen el objetivo 1-objetivo 8 2513A-2513H, respectivamente, objetivos de cadena secundaria opcionales 2514 y operaciones de cadena opcionales 2515. En operación, de acuerdo con algunas realizaciones, un procesador ha de extraer una instrucción de inicio de cadena antes de extraer un conjunto de instrucciones de una cadena de instrucciones. Un inicio de instrucciones de cadena informa al procesador que ha de seguir una cadena de instrucciones y proporciona alguna información acerca de la cadena de instrucciones para informar la selección, configuración y enrutamiento del procesador de una ruta de SIMD de motores de procesamiento para ejecutar la cadena de instrucciones.

El código de operación 2511 hace que un procesador que ejecuta la instrucción reconozca la instrucción como una instrucción de inicio de cadena y puede incluir prefijos o sufijos para especificar aún más el comportamiento de la instrucción. Por ejemplo, el código de operación 2511 puede incluir un prefijo para especificar que la instrucción incluye campos para especificar objetivos de cadena 2513, objetivos de cadena secundaria 2514 y operaciones de cadena 2515. En operación, un procesador, de acuerdo con algunas realizaciones, en respuesta a la instrucción de inicio de

cadena 2510 ha de seleccionar y configurar una ruta de SIMD de motores de procesamiento para ejecutar la cadena de instrucciones.

La longitud de cadena 2512 en algunas realizaciones es inmediata o constante para especificar un número de instrucciones en la cadena de instrucciones. En algunas realizaciones, la longitud de cadena 2512 especifica un registro de propósito general cuyo contenido especifica el número de instrucciones en la cadena de instrucciones.

Los objetivos de cadena 2513 son para especificar un objetivo de al menos la primera instrucción en la cadena de instrucciones. Como se muestra, los objetivos de cadena 2513 han de especificar los objetivos 2513A-2513H para cada una de las primeras siete instrucciones en una cadena de instrucciones que tiene una longitud de ocho (el resultado de las ocho instrucciones en la cadena ha de escribirse en el destino especificado, en lugar de volverse a enrutar a una siguiente instrucción). En algunas realizaciones, el campo 2513 de objetivos de cadena es inmediato, dividido lógicamente en bits para especificar los objetivos de cada una de las instrucciones de la cadena. En algunas realizaciones, los objetivos de cadena 2513 especifican un registro de propósito general que contiene la información de objetivo. En algunas realizaciones, los objetivos de cadena 2513 especifican un registro de vector que contiene la información de objetivo. En algunas realizaciones, los objetivos de cadena 2513 especifican un registro específico del modelo ("MSR") donde se almacena la información de objetivo. En otras realizaciones, los objetivos de cadena 2513 especifican una ubicación de memoria donde se almacena la información de objetivo.

En algunas realizaciones, un campo de objetivos de cadena de 4 bytes 2513 se puede dividir en cuartetos de 4 bits para especificar objetivos para cada una de hasta 8 instrucciones en una cadena de instrucciones. En algunas realizaciones, cada objetivo de cadena 2513A-2513H especifica una posición absoluta de una instrucción de objetivo en la cadena. Como se muestra, el objetivo 1 2513A (de la instrucción 0) es para especificar la instrucción 1; el objetivo 2 2513B es para especificar la instrucción 4; el objetivo 3 2513C; el objetivo 4 2513D es para especificar la instrucción 4; el objetivo 5 2513E es para especificar la instrucción 6; el objetivo 6 2513F es para especificar la instrucción 6; y el objetivo 7 2513G es para especificar la instrucción 7.

En algunas realizaciones, cada uno de los objetivos 1-7, 2513A-G, especifica un desplazamiento relativo de una instrucción objetivo a la que enrutar un resultado. Cada uno de los objetivos 2513A-G podría ser un valor de 2 o 3 bits que especifica una distancia entre una instrucción y su objetivo. Por ejemplo, como se muestra, del objetivo 1 2513A al objetivo 7 2513G se podrían especificar 1, 3, 2, 1, 2, 1, 1, respectivamente.

En algunas realizaciones, los objetivos de cadena secundaria 2514 están formateados como objetivos de cadena 2513 pero especifican un segundo objetivo para cada una de las instrucciones en la cadena de instrucciones. Al igual que con los objetivos de cadena 2513, los objetivos de cadena secundaria 2514 pueden ser un registro inmediato, de propósito general, un registro vectorial, un registro específico de modelo o una ubicación de memoria. Los objetivos de cadena secundaria 2514 son opcionales, como lo indica su borde discontinuo.

En algunas realizaciones, la instrucción de inicio de cadena 2510 incluye operaciones de cadena 2515 para especificar las operaciones a realizar por cada una de la cadena de instrucciones. En operación, un procesador es para usar el campo de operaciones de cadena 2515 para informar su selección, configuración y enrutamiento de los motores de procesamiento a lo largo de la ruta de ejecución de SIMD.

Las Figuras 25C-D son diagramas de bloques que ilustran un formato de instrucción compatible con vectores genérico y las plantillas de instrucción del mismo de acuerdo con algunas realizaciones. Una realización de un formato para una instrucción CHAIN_TILE_OP es CHAIN_TILE_OP (OP, tdest, tsrc1, tsrc2), donde OP especifica una operación aritmética a realizar y tsrcc1 y tsrc2 especifican los orígenes a añadir. En algunas realizaciones, el campo tdest es un valor R/M (tal como 2546 de las Figuras 25C-D), y el campo tsrc1 es el campo de índice de registro de las Figuras 25C-D. En algunas realizaciones, el código de operación OP incluye indicadores, tales como prefijos o sufijos [U, S] [U, S], para indicar si cada uno del primer y segundo orígenes identificados es con signo o sin signo.

En algunas realizaciones, las codificaciones de la instrucción incluyen un operando de direccionamiento de memoria de tipo base de índice de escala (SIB) que identifica indirectamente múltiples ubicaciones de destino indexadas en memoria. En una realización, un operando de memoria de tipo SIB puede incluir una codificación que identifica un registro de dirección base. El contenido del registro de dirección base puede representar una dirección base en la memoria a partir de la que se calculan las direcciones de las ubicaciones de destino particulares en la memoria. Por ejemplo, la dirección base puede ser la dirección de la primera ubicación en un bloque de ubicaciones de destino potenciales para una instrucción vectorial extendida. En una realización, un operando de memoria de tipo SIB puede incluir una codificación que identifica un registro de índice. Cada elemento del registro de índice puede especificar un índice o valor de desplazamiento utilizable para calcular, a partir de la dirección base, una dirección de una ubicación de destino respectiva dentro de un bloque de ubicaciones de destino potenciales. En una realización, un operando de memoria de tipo SIB puede incluir una codificación que especifica un factor de escala que se aplicará a cada valor de índice cuando se calcula una dirección de destino respectiva. Por ejemplo, si un valor de factor de escala de cuatro está codificado en el operando de memoria de tipo SIB, cada valor de índice obtenido de un elemento del registro de índice puede multiplicarse por cuatro y, a continuación, sumarse a la dirección base para calcular una dirección de destino.

En una realización, un operando de memoria de tipo SIB de la forma vm32{x, y, z} puede identificar una matriz vectorial de operandos de memoria especificada usando direccionamiento de memoria de tipo SIB. En este ejemplo, la matriz de direcciones de memoria se especifica utilizando un registro base común, un factor de escala constante y un registro de índice vectorial que contiene elementos individuales, cada uno de los cuales es un valor de índice de 32 bits. El registro de índice vectorial puede ser un registro de 128 bits (por ejemplo, XMM) (vm32x), un registro de 256 bits (por ejemplo, YMM) (vm32y) o un registro de 512 bits (por ejemplo, ZMM) (vm32z). En otra realización, un operando de memoria de tipo SIB de la forma vm64{x, y, z} puede identificar una matriz vectorial de operandos de memoria especificada usando direccionamiento de memoria de tipo SIB. En este ejemplo, la matriz de direcciones de memoria se especifica utilizando un registro base común, un factor de escala constante y un registro de índice vectorial que contiene elementos individuales, cada uno de los cuales es un valor de índice de 64 bits. El registro de índice vectorial puede ser un registro de 128 bits (por ejemplo, XMM) (vm64x), un registro de 256 bits (por ejemplo, YMM) (vm64y) o un registro de 512 bits (por ejemplo, ZMM) (vm64z).

V. SISTEMAS, PROCESADORES Y EMULACIÓN ILUSTRATIVOS DETALLADOS

En el presente documento se detallan ejemplos de hardware, software, etc. para ejecutar las instrucciones descritas anteriormente. Por ejemplo, lo que se describe a continuación detalla aspectos de la ejecución de instrucciones, incluyendo diversas etapas de canalización, tal como extraer, decodificar, planificar, ejecutar, retirar, etc.

CONJUNTOS DE INSTRUCCIONES

Un conjunto de instrucciones puede incluir uno o más formatos de instrucciones. Un formato de instrucciones dado puede definir diversos campos (por ejemplo, número de bits, ubicación de los bits) para especificar, entre otras cosas, la operación a realizar (por ejemplo, código de operación) y los operandos en los que se realizará esa operación y/u otro campo o campos de datos (por ejemplo, máscara). Algunos formatos de instrucción se desglosan aún más mediante la definición de plantillas de instrucciones (o subformatos). Por ejemplo, las plantillas de instrucciones de un formato de instrucciones dado pueden definirse para tener diferentes subconjuntos de los campos del formato de instrucciones (los campos incluidos típicamente están en el mismo orden, pero al menos algunos tienen posiciones de bits diferentes porque hay menos campos incluidos) y/o definirse para tener un campo dado interpretado de manera diferente. Por lo tanto, cada instrucción de una ISA se expresa usando un formato de instrucción dado (y, si se define, en una de las plantillas de instrucción de ese formato de instrucción) e incluye campos para especificar la operación y los operandos. Por ejemplo, una instrucción ADD de ejemplo tiene un código de operación específico y un formato de instrucciones que incluye un campo de código de operación para especificar ese código de operación y campos de operando para seleccionar operandos (origen1/destino y origen2); y una aparición de esta instrucción ADD en una secuencia de instrucciones tendrá contenidos específicos en los campos de operandos que seleccionan operandos específicos. Se ha lanzado y/o publicado un conjunto de extensiones de SIMD denominadas Extensiones Vectoriales Avanzadas (AVX) (AVX1 y AVX2) y que usan el esquema de codificación de Extensiones Vectoriales (VEX) (por ejemplo, véase Manual del desarrollador de Software de Arquitecturas Intel® 64 e IA-32, septiembre de 2014; y véase la referencia de programación de extensiones vectoriales avanzadas Intel®, octubre de 2014).

FORMATOS DE INSTRUCCIONES ILUSTRATIVOS

Las realizaciones de la instrucción o instrucciones descritas en el presente documento se pueden realizar en diferentes formatos. Además, a continuación, se detallan sistemas, arquitecturas y canales ilustrativos. Se pueden ejecutar realizaciones de la instrucción o instrucciones en tales sistemas, arquitecturas y canalizaciones, pero no se limitan a las detalladas.

FORMATO DE INSTRUCCIÓN COMPATIBLE CON VECTORES GENÉRICO

Un formato de instrucción compatible con vectores es un formato de instrucción adecuado para instrucciones vectoriales (por ejemplo, hay ciertos campos específicos para operaciones vectoriales). Si bien se describen realizaciones en las que se admiten tanto operaciones vectoriales como operaciones escalares a través del formato de instrucción compatible con vectores, realizaciones alternativas usan únicamente operaciones vectoriales en el formato de instrucción compatible con vectores.

Las **Figuras 25C-25D** son diagramas de bloques que ilustran un formato de instrucción compatible con vectores genérico y plantillas de instrucciones del mismo de acuerdo con realizaciones de la invención. La **Figura 25C** es un diagrama de bloques que ilustra un formato de instrucción compatible con vectores genérico y plantillas de instrucción de clase A del mismo de acuerdo con realizaciones de la invención; mientras que la **Figura 25D** es un diagrama de bloques que ilustra el formato de instrucción compatible con vectores genérico y sus plantillas de instrucción de clase B de acuerdo con realizaciones de la invención. Específicamente, un formato de instrucciones genérico compatible con vectores 2516 para el que se definen plantillas de instrucciones de clase A y clase B, ambas de las cuales incluyen las plantillas de instrucciones sin acceso a memoria 251 y las plantillas de instrucciones de acceso a memoria 2520. El término genérico en el contexto del formato de instrucción compatible con vectores se refiere al formato de instrucciones que no está vinculado a ningún conjunto de instrucciones específico.

Si bien se describirán realizaciones de la invención en las que el formato de instrucción compatible con vectores admite lo siguiente: una longitud (o tamaño) de operando vectorial de 64 bytes con anchuras (o tamaños) de elementos de datos de 32 bits (4 bytes) o 64 bits (8 bytes) (y, por lo tanto, un vector de 64 bytes consiste en 16 elementos de tamaño de palabra doble o, como alternativa, 8 elementos de tamaño de palabra cuádruple); una longitud (o tamaño) de operando vectorial de 64 bytes con anchuras (o tamaños) de elementos de datos de 16 bits (2 bytes) u 8 bits (1 byte); una longitud (o tamaño) de operando vectorial de 32 bytes con anchuras (o tamaños) de elementos de datos de 32 bits (4 bytes), 64 bits (8 bytes), 16 bits (2 bytes) u 8 bits (1 byte); y una longitud (o tamaño) de operando vectorial de 16 bytes con anchuras (o tamaños) de elementos de datos de 32 bits (4 bytes), 64 bits (8 bytes), 16 bits (2 bytes) u 8 bits (1 byte); realizaciones alternativas pueden admitir más, menos y/o diferentes tamaños de operandos vectoriales (por ejemplo, operandos vectoriales de 256 bytes) con más, menos o diferentes anchuras de elementos de datos (por ejemplo, anchuras de elementos de datos de 128 bits (16 bytes)).

Las plantillas de instrucción de clase A en la **Figura 25C** incluyen: 1) dentro de las plantillas de instrucción sin acceso a memoria 2519 se muestra una plantilla de instrucción de operación de tipo de control de redondeo completo sin acceso a memoria 2517 y una plantilla de instrucción de operación de tipo transformada de datos sin acceso a memoria 2518; y 2) dentro de las plantillas de instrucción de acceso a memoria 2520 se muestra una plantilla de instrucción temporal de acceso a memoria 2525 y una plantilla de instrucción no temporal de acceso a memoria 2530. Las plantillas de instrucción de clase B en la **Figura 25D** incluyen: 1) dentro de las plantillas de instrucción sin acceso a memoria 2519 se muestra una plantilla de instrucción de operación de tipo de control de redondeo parcial, control de máscara de escritura, sin acceso a memoria 2522 y una plantilla de instrucción de operación de tipo vsize de control de máscara de escritura sin acceso a memoria 2523; y 2) dentro de las plantillas de instrucción de acceso a memoria 2520 se muestra una plantilla de instrucción de control de máscara de escritura de acceso a memoria 2527.

El formato de instrucción compatible con vectores genérico 2516 incluye los siguientes campos enumerados a continuación en el orden ilustrado en las **Figuras 25C-25D**.

Campo de formato 2540: un valor específico (un valor de identificador de formato de instrucción) en este campo identifica de forma única el formato de instrucción compatible con vectores y, por lo tanto, las apariciones de instrucciones en el formato de instrucción compatible con vectores en los flujos de instrucciones. Como tal, este campo es opcional en el sentido de que no es necesario para un conjunto de instrucciones que únicamente tiene el formato de instrucción compatible con vectores genérico.

Campo de operación de base 2542 - su contenido distingue diferentes operaciones de base.

Campo de índice de registro 2544 - su contenido, directamente o a través de la generación de direcciones, especifica las ubicaciones de los operandos de origen y destino, ya sea en los registros o en la memoria. Estos incluyen un número suficiente de bits para seleccionar N registros de un archivo de registros PxQ (por ejemplo, 32x512, 16x128, 32x1024, 64x1024). Mientras que en una realización N puede ser hasta tres registros de origen y uno de destino, realizaciones alternativas pueden soportar más o menos registros de origen y de destino (por ejemplo, pueden soportar hasta dos orígenes donde uno de estos orígenes también actúa como el destino, pueden soportar hasta tres orígenes donde uno de estos orígenes también actúa como el destino, puede soportar hasta dos orígenes y un destino).

Campo de modificador 2546 - su contenido distingue las ocurrencias de instrucciones en el formato de instrucción de vector genérico que especifican el acceso a memoria de aquellas que no lo hacen; es decir, entre las plantillas de instrucción sin acceso a memoria 2519 y las plantillas de instrucción de acceso a memoria 2520. Las operaciones de acceso a memoria leen y/o escriben en la jerarquía de memoria (especificando, en algunos casos, las direcciones de origen y/o de destino usando valores en registros), mientras que las operaciones sin acceso a memoria no lo hacen (por ejemplo, el origen y los destinos son registros). Si bien en una realización este campo también selecciona entre tres formas diferentes de realizar cálculos de direcciones de memoria, realizaciones alternativas pueden soportar más, menos o diferentes formas de realizar cálculos de direcciones de memoria.

Campo de operación de aumento 2550 - su contenido distingue cuál de una diversidad de operaciones diferentes se realizará además de la operación de base. Este campo es específico del contexto. En una realización de la invención, este campo se divide en un campo de clase 2568, un campo alfa 2552 y un campo beta 2554. El campo de operación de aumento 2550 permite que se realicen grupos comunes de operaciones en una única instrucción en lugar de 2, 3 o 4 instrucciones.

Campo de escala 2560 - su contenido permite escalar el contenido del campo de índice para la generación de direcciones de memoria (por ejemplo, para la generación de direcciones que usa 2^{escala} * índice + base).

Campo de desplazamiento 2562A - su contenido se usa como parte de la generación de direcciones de memoria (por ejemplo, para la generación de direcciones que usa 2^{escala} * índice + base + desplazamiento).

Campo de factor de desplazamiento 2562B (obsérvese que la yuxtaposición del campo de desplazamiento 2562A directamente sobre el campo de factor de desplazamiento 2562B indica que se usa uno u otro) - su contenido se usa

como parte de la generación de direcciones; especifica un factor de desplazamiento a escalar de acuerdo con el tamaño de un acceso a memoria (N) - donde N es el número de bytes en el acceso a memoria (por ejemplo, para la generación de direcciones que usa 2^{escala} * índice + base + desplazamiento escalado). Los bits redundantes de bajo orden se ignoran y, por lo tanto, el contenido del campo del factor de desplazamiento se multiplica por el tamaño total de los operandos de memoria (N) para generar el desplazamiento final que se usará para calcular una dirección efectiva. El valor de N está determinado por el hardware del procesador en tiempo de ejecución basándose en el campo de código de operación completo 2574 (descrito más adelante en el presente documento) y el campo de manipulación de datos 2554C. El campo de desplazamiento 2562A y el campo de factor de desplazamiento 2562B son opcionales en el sentido de que no se usan para las plantillas de instrucciones sin acceso a memoria 2519 y/o diferentes realizaciones pueden implementar únicamente uno o ninguno de los dos.

Campo de anchura de elemento de datos 2564: su contenido distingue cuál de un número de anchuras de elementos de datos se va a usar (en algunas realizaciones, para todas las instrucciones; en otras realizaciones, solo para algunas de las instrucciones). Este campo es opcional en el sentido de que no es necesario si únicamente se soporta una anchura de elemento de datos y/o se soportan anchuras de elementos de datos usando algún aspecto de los códigos de operación.

Campo de máscara de escritura 2570 - su contenido controla, basándose en la posición del elemento de datos, si esa posición del elemento de datos en el operando de vector de destino refleja el resultado de la operación de base y la operación de aumento. Las plantillas de instrucción de clase A soportan el enmascaramiento de escritura de fusión, mientras que las plantillas de instrucción de clase B admiten el enmascaramiento de escritura tanto de fusión como de puesta a cero. Cuando se fusionan, las máscaras vectoriales permiten proteger de actualizaciones cualquier conjunto de elementos en el destino durante la ejecución de cualquier operación (especificada por la operación de base y la operación de aumento); en otra realización, conservando el valor antiguo de cada elemento del destino donde el bit de máscara correspondiente tiene un 0. Por el contrario, cuando las máscaras vectoriales de puesta a cero permiten poner a cero cualquier conjunto de elementos en el destino durante la ejecución de cualquier operación (especificada por la operación de base y la operación de aumento); en una realización, un elemento del destino se establece a 0 cuando el bit de máscara correspondiente tiene un valor 0. Un subconjunto de esta funcionalidad es la capacidad de controlar la longitud del vector de la operación que se realiza (es decir, el intervalo de elementos que se modifican, desde el primero hasta el último); sin embargo, no es necesario que los elementos que se modifican sean consecutivos. Por lo tanto, el campo de máscara de escritura 2570 permite operaciones vectoriales parciales, que incluyen cargas, almacenamientos, aritméticas, lógicas, etc. Si bien se describen realizaciones de la invención en las que el contenido del campo de máscara de escritura 2570 selecciona uno de un número de registros de máscara de escritura que contiene la máscara de escritura que se va a usar (y, por tanto, el contenido del campo de máscara de escritura 2570 identifica indirectamente ese enmascaramiento que se va a realizar), realizaciones alternativas, en su lugar, o adicionales, permiten que el contenido del campo de escritura de máscara 2570 especifique directamente el enmascaramiento que se va a realizar.

Campo inmediato 2572 - su contenido permite la especificación de un inmediato. Este campo es opcional en el sentido de que no está presente en una implementación del formato compatible con vectores genéricos que no soporta inmediato y no está presente en instrucciones que no usan un inmediato.

Campo de clase 2568 - su contenido distingue entre diferentes clases de instrucciones. Con referencia a las Figuras 25C-D, el contenido de este campo selecciona entre instrucciones de clase A y clase B. En las Figuras 25C-D, se usan cuadrados de esquinas redondeadas para indicar que un valor específico está presente en un campo (por ejemplo, clase A 2568A y clase B 2568B para el campo de clase 2568 respectivamente en las Figuras 25C-D).

PLANTILLAS DE INSTRUCCIÓN DE CLASE A

En el caso de las plantillas de instrucciones sin acceso a memoria de clase A 2519, el campo alfa 2552 se interpreta como un campo de RS 2552A, cuyo contenido distingue cuál de los diferentes tipos de operación de aumento va a realizarse (por ejemplo, el redondeo 2552A.1 y la transformación de datos 2552A.2 se especifican respectivamente para las plantillas de instrucciones de operación de tipo de redondeo sin acceso a memoria 2517 y operación de tipo de transformación de datos sin acceso a memoria 2518), mientras que el campo beta 2554 distingue cuál de las operaciones del tipo especificado va a realizarse. En las plantillas de instrucciones sin acceso a memoria 2519, el campo de escala 2560, el campo de desplazamiento 2562A y el campo de escala de desplazamiento 2562B no están presentes.

PLANTILLAS DE INSTRUCCIONES SIN ACCESO A MEMORIA - OPERACIÓN DE TIPO DE CONTROL DE REDONDEO COMPLETO

En la plantilla de instrucción de operación de tipo de control de redondeo completo sin acceso a memoria 2517, el campo beta 2554 se interpreta como un campo de control de redondeo 2554A, cuyo contenido o contenidos proporcionan redondeo estático. Mientras que, en las realizaciones descritas de la invención, el campo de control de redondeo 2554A incluye un campo de supresión de todas las excepciones de coma flotante (SAE) 2556 y un campo de control de operación de redondeo 2558, las realizaciones alternativas pueden soportar la codificación de ambos

conceptos en el mismo campo o solo tener uno o el otro de estos conceptos/campos (por ejemplo, pueden tener solo el campo de control de operación de redondeo 2558).

Campo de SAE 2556 - su contenido distingue si se debe o no deshabilitar el informe de eventos de excepción; cuando el contenido del campo de SAE 2556 indica que la supresión está habilitada, una instrucción dada no informa ningún tipo de bandera de excepción de coma flotante y no genera ningún manejador de excepción de coma flotante.

Campo de control de operación de redondeo 2558: su contenido distingue cuál de un grupo de operaciones de redondeo se realizará (por ejemplo, redondeo hacia arriba, redondeo hacia abajo, redondeo hacia cero y redondeo hacia el valor más cercano). Por tanto, el campo de control de operación de redondeo 2558 permite el cambio del modo de redondeo por instrucción. En una realización de la invención donde un procesador incluye un registro de control para especificar modos de redondeo, el contenido del campo de control de operación de redondeo 2558 anula ese valor de registro.

PLANTILLAS DE INSTRUCCIONES SIN ACCESO A MEMORIA - OPERACIÓN DE TIPO DE TRANSFORMACIÓN DE DATOS

En la plantilla de instrucciones de operación de tipo de transformación de datos sin acceso a memoria 2518, el campo beta 2554 se interpreta como un campo de transformación de datos 2554B, cuyo contenido distingue cuál de un número de transformaciones de datos se va a realizar (por ejemplo, sin transformación de datos, mezcla, difusión).

En el caso de una plantilla de instrucción de acceso a memoria 2520 de clase A, el campo alfa 2552 se interpreta como un campo de sugerencia de desalojo 2552B, cuyo contenido distingue cuál de las sugerencias de desalojo se va a usar (en la **Figura 25C**, temporal 2552B.1 y no temporal 2552B.2 se especifican respectivamente para la plantilla de instrucción de acceso a memoria, temporal 2525 y la plantilla de instrucción de acceso a memoria, no temporal 2530), mientras que el campo beta 2554 se interpreta como un campo de manipulación de datos 2554C, cuyo contenido distingue cuál de un número de operaciones de manipulación de datos (también conocidas como primitivas) se va a realizar (por ejemplo, sin manipulación; difusión; conversión ascendente de un origen y conversión descendente de un destino). Las plantillas de instrucciones de acceso a memoria 2520 incluyen el campo de escala 2560 y, opcionalmente, el campo de desplazamiento 2562A o el campo de escala de desplazamiento 2562B.

Las instrucciones de memoria vectorial realizan cargas de vectores y almacenamientos vectoriales en la memoria, con soporte de conversión. Al igual que con las instrucciones vectoriales normales, las instrucciones de memoria vectorial transfieren datos desde/hacia la memoria en forma de elementos de datos, y los elementos que realmente se transfieren están dictados por el contenido de la máscara vectorial que se selecciona como máscara de escritura.

PLANTILLAS DE INSTRUCCIONES DE ACCESO A MEMORIA - TEMPORAL

Los datos temporales son datos que probablemente se reutilizarán lo suficientemente pronto como para beneficiarse del almacenamiento en caché. Sin embargo, esto es una sugerencia, y diferentes procesadores pueden implementarla de diferentes maneras, incluso ignorando la sugerencia por completo.

PLANTILLAS DE INSTRUCCIONES DE ACCESO A MEMORIA - NO TEMPORAL

Los datos no temporales son datos que es poco probable que se reutilicen lo suficientemente pronto como para beneficiarse del almacenamiento en caché en la caché de 1er nivel y se les debe dar prioridad para la expulsión. Sin embargo, esto es una sugerencia, y diferentes procesadores pueden implementarla de diferentes maneras, incluso ignorando la sugerencia por completo.

PLANTILLAS DE INSTRUCCIONES DE CLASE B

En el caso de las plantillas de instrucciones de clase B, el campo alfa 2552 se interpreta como un campo de control de máscara de escritura (Z) 2552C, cuyo contenido distingue si el enmascaramiento de escritura controlado por el campo de máscara de escritura 2570 debe ser una fusión o una puesta a cero.

En el caso de las plantillas de instrucciones sin acceso a memoria 2519 de clase B, parte del campo beta 2554 se interpreta como un campo de RL 2557A, cuyo contenido distingue cuál de los diferentes tipos de operación de aumento se va a realizar (por ejemplo, redondeo 2557A.1 y la longitud de vector (VSIZE) 2557A.2 se especifican respectivamente para la plantilla de instrucciones de operación de tipo de control de redondeo parcial de control de máscara de escritura sin acceso a memoria 2522 y la plantilla de instrucciones sin de operación de tipo VSIZE de control de máscara de escritura sin acceso a memoria 2523), mientras que el resto del campo beta 2554 distingue cuál de las operaciones del tipo especificado se va a realizar. En las plantillas de instrucciones sin acceso a memoria 2519, el campo de escala 2560, el campo de desplazamiento 2562A y el campo de escala de desplazamiento 2562B no están presentes.

En la plantilla de instrucción de operación sin acceso a memoria, de control de máscara de escritura, tipo de control de redondeo parcial 2522, el resto del campo beta 2554 se interpreta como un campo de operación de redondeo 2559A y el informe de eventos de excepción está inhabilitado (una instrucción dada no informa ninguna clase de bandera de excepción de coma flotante y no genera ningún manejador de excepción de coma flotante).

Campo de control de operación de redondeo 2559A - al igual que el campo de control de operación de redondeo 2558, su contenido distingue cuál de un grupo de operaciones de redondeo realizar (por ejemplo, redondeo hacia arriba, redondeo hacia abajo, redondeo hacia cero y redondeo hacia el valor más cercano). Por tanto, el campo de control de operación de redondeo 2559A permite el cambio del modo de redondeo por instrucción. En una realización de la invención donde un procesador incluye un registro de control para especificar modos de redondeo, el contenido del campo de control de operación de redondeo 2559A anula ese valor de registro.

En la plantilla de instrucciones de operación de tipo VSIZE de control de máscara de escritura sin acceso a memoria 2523, el resto del campo beta 2554 se interpreta como un campo de longitud de vector 2559B, cuyo contenido distingue cuál de un número de longitudes de vector de datos se va a realizar en (por ejemplo, 128, 256 o 512 bytes).

En el caso de una plantilla de instrucción con acceso a memoria 2520 de clase B, parte del campo beta 2554 se interpreta como un campo de difusión 2557B, cuyo contenido distingue si se va a realizar o no la operación de manipulación de datos de tipo difusión, mientras que el resto del campo beta 2554 se interpreta como el campo de longitud vectorial 2559B. Las plantillas de instrucciones de acceso a memoria 2520 incluyen el campo de escala 2560 y, opcionalmente, el campo de desplazamiento 2562A o el campo de escala de desplazamiento 2562B.

Con respecto al formato de instrucción compatible con vectores genéricos 2516, se muestra un campo de código de operación completo 2574 que incluye el campo de formato 2540, el campo de operación de base 2542 y el campo de anchura de elemento de datos 2564. Aunque se muestra una realización donde el campo de código de operación completo 2574 incluye todos estos campos, el campo de código de operación completo 2574 incluye menos de todos estos campos en realizaciones que no los soportan todos. El campo de código de operación completo 2574 proporciona el código de operación (código de operación).

El campo de operación de aumento 2550, el campo de anchura de elemento de datos 2564 y el campo de máscara de escritura 2570 permiten que estas características se especifiquen en una base por instrucción en el formato de instrucciones genérico compatible con vectores.

La combinación del campo de máscara de escritura y el campo de anchura de elemento de datos crea instrucciones de un tipo concreto que permiten que se aplique la máscara basándose en diferentes anchuras de elementos de datos.

Las diversas plantillas de instrucciones que se encuentran dentro de la clase A y la clase B son beneficiosas en diferentes situaciones. En algunas realizaciones de la invención, diferentes procesadores o diferentes núcleos dentro de un procesador pueden soportar solo la clase A, solo la clase B o ambas clases. Por ejemplo, un núcleo en desorden de propósito general de alto rendimiento destinado a computación de propósito general puede soportar únicamente clase B, un núcleo destinado principalmente a gráficos y/o computación científica (rendimiento) puede soportar únicamente clase A, y un núcleo destinado a ambas puede soportar ambas (por supuesto, un núcleo que tiene alguna mezcla de plantillas e instrucciones de ambas clases, pero no todas las plantillas ni instrucciones de ambas clases están dentro del alcance de la invención). Además, un solo procesador puede incluir múltiples núcleos, todos los cuales soportan la misma clase o en los que diferentes núcleos soportan diferentes clases. Por ejemplo, en un procesador con gráficos y núcleos de propósito general separados, uno de los núcleos de gráficos destinado principalmente a gráficos y/o computación científica puede soportar únicamente la clase A, mientras que uno o más de los núcleos de propósito general pueden ser núcleos de propósito general de alto rendimiento con ejecución en desorden y cambio de nombre de registro destinado a computación de propósito general que soportan únicamente clase B. Otro procesador que no tiene un núcleo de gráficos separado, puede incluir uno más núcleos en orden o en desorden de propósito general que soportan tanto clase A como clase B. Por supuesto, las características de una clase también pueden implementarse en la otra clase en diferentes realizaciones de la invención. Los programas escritos en un lenguaje de alto nivel se pondrían (por ejemplo, compilados justo a tiempo o compilados estáticamente) en una diversidad de formas ejecutables diferentes, que incluyen: 1) una forma que tiene únicamente instrucciones de la clase o clases soportadas por el procesador objetivo para su ejecución; o 2) una forma que tiene rutinas alternativas escritas usando diferentes combinaciones de las instrucciones de todas las clases y que tiene un código de flujo de control que selecciona las rutinas a ejecutar basándose en las instrucciones soportadas por el procesador que actualmente está ejecutando el código.

FORMATO DE INSTRUCCIÓN COMPATIBLE CON VECTORES ESPECÍFICO ILUSTRATIVO

La **Figura 26A** es un diagrama de bloques que ilustra un formato de instrucción compatible con vectores específico ilustrativo de acuerdo con realizaciones de la invención. La **Figura 26A** muestra un formato de instrucción compatible con vectores específico 2600 que es específico en el sentido de que especifica la ubicación, el tamaño, la interpretación y el orden de los campos, así como los valores para algunos de esos campos. El formato de instrucción específico compatible con vectores 2600 se puede usar para ampliar el conjunto de instrucciones x86 y, por tanto, algunos de los

campos son similares o iguales a los usados en el conjunto de instrucciones x86 existente y la extensión del mismo (por ejemplo, AVX). Este formato sigue siendo consistente con el campo de codificación de prefijo, el campo de bytes de código de operación real, el campo MOD R/M, el campo SIB, el campo de desplazamiento y los campos inmediatos del conjunto de instrucciones x86 existente con extensiones. Se ilustran los campos de la **Figura 25** en el que se mapean los campos de la **Figura 26A**.

Debe entenderse que, aunque las realizaciones de la invención se describen con referencia al formato de instrucción compatible con vectores específicos 2600 en el contexto del formato de instrucción compatible con vectores genérico 2516 con fines ilustrativos, la invención no se limita al formato de instrucción compatible con vectores específico. 2600 excepto donde se reivindique. Por ejemplo, el formato de instrucción genérico compatible con vectores 2516 contempla una diversidad de tamaños posibles para los diversos campos, mientras que el formato de instrucción específico compatible con vectores 2600 se muestra con campos de tamaños específicos. A modo de ejemplo específico, mientras que el campo de anchura de elemento de datos 2564 se ilustra como un campo de un bit en el formato de instrucción compatible con vectores específico 2600, la invención no está así limitada (es decir, el formato de instrucción compatible con vectores genérico 2516 contempla otros tamaños del campo de anchura de elemento de datos 2564).

El formato de instrucción compatible con vectores genérico 2516 incluye los siguientes campos enumerados a continuación en el orden ilustrado en la **Figura 26A.**

Prefijo EVEX (Bytes 0-3) 2602: está codificado en formato de cuatro bytes.

Campo de formato 2540 (byte EVEX 0, bits [7:0]) - el primer byte (byte de EVEX 0) es el campo de formato 2540 y contiene 0x62 (el valor único usado para distinguir el formato de instrucción compatible con vectores en una realización de la invención).

Del segundo al cuarto bytes (bytes de EVEX 1-3) incluyen un número de campos de bits que proporcionan una capacidad específica.

Campo REX 2605 (byte de EVEX 1, bits [7-5]) - consiste en un campo de bits EVEX.R (byte de EVEX 1, bit [7] - R), campo de bits EVEX.X (byte de EVEX 1, bit [6] - X) y byte de 2557BEX 1, bit[5] - B). Los campos de bits EVEX.R, EVEX.X y EVEX.B proporcionan la misma funcionalidad que los campos de bits VEX correspondientes y se codifican en forma de complemento a 1, es decir, ZMM0 se codifica como 1111B, ZMM15 se codifica como 0000B. Otros campos de las instrucciones codifican los tres bits inferiores de los índices de registro como es conocido en la técnica (rrr, xxx y bbb), de modo que Rrrr, Xxxx y Bbbb pueden formarse sumando EVEX.R, EVEX.X, y EVEX.B.

REX' 2610A: esta es la primera parte del campo REX' 2610 y es el campo de bits EVEX.R' (byte de EVEX 1, bit [4] - R') que se usa para codificar los 16 superiores o los 16 inferiores del conjunto extendido de 32 registros. En una realización de la invención, este bit, junto con otros como se indica a continuación, se almacena en formato de bits invertidos para distinguir (en el bien conocido modo x86 de 32 bits) de la instrucción BOUND, cuyo byte de código de operación real es 62, pero no acepta en el campo MOD R/M (descrito a continuación) el valor de 11 en el campo MOD; realizaciones alternativas de la invención no almacenan este y los otros bits indicados a continuación en el formato invertido. Se usa un valor de 1 para codificar los 16 registros inferiores. En otras palabras, R'Rrrr se forma combinando EVEX.R', EVEX.R y el otro RRR de otros campos.

Campo de mapa de código de operación 2615 (byte 1 de EVEX, bits [3:0] - mmmm) - su contenido codifica un byte de código de operación inicial implícito (0F, 0F 38 o 0F 3).

Campo de anchura de elemento de datos 2564 (byte de EVEX 2, bit [7] - W) - se representa mediante la notación EVEX.W. EVEX.W se usa para definir la granularidad (tamaño) del tipo de datos (ya sean elementos de datos de 32 bits o elementos de datos de 64 bits).

EVEX.vvvv 2620 (byte de EVEX 2, bits [6:3]-vvvv)- la función de EVEX.vvvv puede incluir lo siguiente: 1) EVEX.vvvv codifica el primer operando de registro de origen, especificado en forma invertida (complemento a 1) y es válido para instrucciones con 2 o más operandos de origen; 2) EVEX.vvvv codifica el operando de registro de destino, especificado en forma de complemento a 1 para ciertos desplazamientos de vector; o 3) EVEX.vvvv no codifica ningún operando, el campo está reservado y debe contener 1111b. Por tanto, el campo 2620 de EVEX.vvvv codifica los 4 bits de orden inferior del primer especificador de registro de origen almacenado en forma invertida (complemento a 1). Dependiendo de la instrucción, se usa un campo de bits EVEX adicional diferente para extender el tamaño del especificador a 32 registros.

EVEX.U 2568 campo de Clase (byte EVEX 2, bit [2]-U) - Si EVEX.U = 0, indica clase A o EVEX.U0; si EVEX.U = 1 indica clase B o EVEX.U1.

Campo de codificación de prefijo 2625 (byte EVEX 2, bits [1:0]-pp) - proporciona bits adicionales para el campo de operación de base. Además de proporcionar soporte para las instrucciones SSE heredadas en el formato de prefijo

EVEX, esto también tiene la ventaja de compactar el prefijo de SIMD (en lugar de requerir un byte para expresar el prefijo de SIMD, el prefijo de EVEX requiere solo 2 bits). En una realización, para soportar instrucciones SSE heredadas que usan un prefijo de SIMD (66H, F2H, F3H) tanto en el formato heredado como en el formato de prefijo EVEX, estos prefijos de SIMD heredados se codifican en el campo de codificación de prefijo de SIMD; y en el tiempo de ejecución se expanden en el prefijo de SIMD heredado antes de proporcionarse a la PLA del decodificador (para que la PLA pueda ejecutar tanto el formato heredado como el EVEX de estas instrucciones heredadas sin modificaciones). Aunque las instrucciones más nuevas podrían usar el contenido del campo de codificación del prefijo EVEX directamente como una extensión del código de operación, ciertas realizaciones se expanden de manera similar para mantener la consistencia, pero permiten que estos prefijos de SIMD heredados especifiquen diferentes significados. Una realización alternativa puede rediseñar la PLA para soportar las codificaciones de prefijo de SIMD de 2 bits y, por lo tanto, no requerir la expansión.

Campo alfa 2552 (byte de EVEX 3, bit [7] - EH; también conocido como EVEX.EH, EVEX.rs, EVEX.RL, EVEX.control de máscara de escritura y EVEX.N; también ilustrado con α) - como se describió anteriormente, este campo es específico del contexto.

Campo beta 2554 (byte de EVEX 3, bits [6:4]-SSS, también conocido como EVEX.s_{2-0}, EVEX.r_{2-0}, EVEX.rr1, EVEX.LL0, EVEX.LLB; también ilustrado con βββ) - como se describió anteriormente, este campo es específico del contexto.

REX' 2610B - este es el resto del campo REX' 2610 y es el campo de bits EVEX.V' (Byte de EVEX 3, bit [3] - V') que se puede usar para codificar los 16 superiores o los 16 inferiores del conjunto de 32 registros extendido. Este bit se almacena en formato de bits invertidos. Se usa un valor de 1 para codificar los 16 registros inferiores. En otras palabras, V'VVVV se forma combinando EVEX.V', EVEX.vvvv.

Campo de máscara de escritura 2570 (byte de EVEX 3, bits [2:0]-kkk) - su contenido especifica el índice de un registro en los registros de máscara de escritura como se describió anteriormente. En una realización de la invención, el valor específico EVEX.kkk=000 tiene un comportamiento especial que implica que no se usa máscara de escritura para la instrucción en particular (esto puede implementarse de diversas formas, incluido el uso de una máscara de escritura predeterminada a todos unos o hardware que pasa por alto el hardware de enmascaramiento).

El campo de código de operación real 2630 (byte 4) también se conoce como byte de código de operación. Parte del código de operación se especifica en este campo.

El campo MOD R/M 2640 (byte 5) incluye el campo MOD 2642, el campo Reg 2644 y el campo R/M 2646. Como se describió anteriormente, el contenido del campo MOD 2642 distingue entre operaciones de acceso a memoria y operaciones sin acceso a memoria. La función del campo Reg 2644 se puede resumir en dos situaciones: codificar el operando de registro de destino o un operando de registro de origen, o tratarse como una extensión de código de operación y no usarse para codificar un operando de instrucción. La función del campo R/M 2646 puede incluir lo siguiente: codificar el operando de instrucción que hace referencia a una dirección de memoria, o codificar el operando del registro de destino o un operando del registro de origen.

Byte de escala, índice, base (SIB) (Byte 6) - como se describió anteriormente, el contenido del campo de escala 2560 se usa para la generación de direcciones de memoria. SIB.xxx 2654 y SIB.bbb 2656 - el contenido de estos campos ha sido mencionado anteriormente con respecto a los índices de registro Xxxx y Bbbb.

Campo de desplazamiento 2562A (Bytes 7-10) - cuando el campo MOD 2642 contiene 10, los bytes 7-10 son el campo de desplazamiento 2562A, y funciona igual que el desplazamiento de 32 bits heredado (disp32) y funciona con granularidad de byte.

Campo de factor de desplazamiento 2562B (Byte 7) - cuando el campo MOD 2642 contiene 01, el byte 7 es el campo de factor de desplazamiento 2562B. La ubicación de este campo es la misma que la del desplazamiento de 8 bits (disp8) del conjunto de instrucciones x86 heredado, que funciona con granularidad de bytes. Dado que disp8 es un signo extendido, solo puede direccionar entre -128 y 127 desplazamientos de bytes; en términos de líneas de caché de 64 bytes, disp8 usa 8 bits que se pueden establecer en solo cuatro valores realmente útiles: -128, -64, 0 y 64; dado que a menudo se necesita un intervalo mayor, se usa disp32; sin embargo, disp32 requiere 4 bytes. A diferencia de disp8 y disp32, el campo de factor de desplazamiento 2562B es una reinterpretación de disp8; cuando se usa el campo de factor de desplazamiento 2562B, el desplazamiento real se determina por el contenido del campo de factor de desplazamiento multiplicado por el tamaño del acceso de operando de memoria (N). Este tipo de desplazamiento se denomina disp8*N. Esto reduce la longitud de instrucción promedio (un único byte de lo usado para el desplazamiento, pero con un intervalo mucho mayor). Tal desplazamiento comprimido se basa en la suposición de que el desplazamiento efectivo es un múltiplo de la granularidad del acceso a memoria y, por lo tanto, no es necesario codificar los bits de bajo orden redundantes del desplazamiento de dirección. En otras palabras, el campo de factor de desplazamiento 2562B sustituye a desplazamiento de 8 bits del conjunto de instrucciones x86 heredado. Por tanto, el campo de factor de desplazamiento 2562B se codifica de la misma manera que un desplazamiento de 8 bits del conjunto de instrucciones x86 (por lo que no hay cambios en las reglas de codificación ModRM/SIB) con la única

excepción de que disp8 se sobrecargue a disp8*N. En otras palabras, no hay cambios en las reglas de codificación o longitudes de codificación, sino únicamente en la interpretación del valor de desplazamiento por el hardware (que necesita escalar el desplazamiento por el tamaño del operando de memoria para obtener un desplazamiento de dirección byte a byte). El campo inmediato 2572 funciona como se ha descrito anteriormente.

CAMPO DE CÓDIGO DE OPCIÓN COMPLETO

La **Figura 26B** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico 2600 que componen el campo de código de operación completo 2574 de acuerdo con una realización de la invención. Específicamente, el campo de código de operación completo 2574 incluye el campo de formato 2540, el campo de operación base 2542 y el campo de anchura de elemento de datos (W) 2564. El campo de operación base 2542 incluye el campo de codificación de prefijo 2625, el campo de correlación de código de operación 2615 y el campo de código de operación real 2630.

CAMPO DE ÍNDICE DE REGISTRO

La **Figura 26C** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico 2600 que componen el campo de índice de registro 2544 de acuerdo con una realización de la invención. Específicamente, el campo de índice de registro 2544 incluye el campo REX 2605, el campo REX' 2610, el campo MODR/M.reg 2644, el campo MODR/M.r/m 2646, el campo VVVV 2620, el campo xxx 2654 y el campo bbb 2656.

CAMPO DE OPERACIÓN DE AUMENTO

La **Figura 26D** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico 2600 que componen el campo de operación de aumento 2550 de acuerdo con una realización de la invención. Cuando el campo de clase (U) 2568 contiene 0, significa EVEX.U0 (clase A 2568A); cuando contiene 1, significa EVEX.U1 (clase B 2568B). Cuando U=0 y el campo MOD 2642 contiene 11 (lo que significa una operación sin acceso a memoria), el campo alfa 2552 (byte de EVEX 3, bit [7] - EH) se interpreta como el campo de RS 2552A. Cuando el campo de RS 2552A contiene un 1 (redondeo 2552A.1), el campo beta 2554 (byte de EVEX 3, bits [6:4]-SSS) se interpreta como el campo de control de redondeo 2554A. El campo de control de redondeo 2554A incluye un campo de SAE de un bit 2556 y un campo de control de operación de redondeo de dos bits 2558. Cuando el campo rs 2552A contiene un 0 (transformación de datos 2552A.2), el campo beta 2554 (byte de EVEX 3, bits [6:4]-SSS) se interpreta como un campo de transformación de datos de tres bits 2554B. Cuando U=0 y el campo MOD 2642 contiene 00, 01 o 10 (lo que significa una operación de acceso a memoria), el campo alfa 2552 (byte de EVEX 3, bit [7] - EH) se interpreta como el campo de sugerencia de desalojo (EH) 2552B y el campo beta 2554 (byte de EVEX 3, bits [6:4]-SSS) se interpreta como un campo de manipulación de datos de tres bits 2554C.

Cuando U=1, el campo alfa 2552 (byte de EVEX 3, bit [7] - EH) se interpreta como el campo de control de máscara de escritura (Z) 2552C. Cuando U=1 y el campo MOD 2642 contiene 11 (lo que significa una operación sin acceso a memoria), parte del campo beta 2554 (byte de EVEX 3, bit [4]- S_0) se interpreta como el campo de RL 2557A; cuando contiene un 1 (redondeo 2557A.1) el resto del campo beta 2554 (byte de EVEX 3, bit [6-5]- S_{2-1}) se interpreta como el campo de operación de redondeo 2559A, mientras que cuando el campo de RL 2557A contiene un 0 (VSIZE 2557.A2) el resto del campo beta 2554 (byte de EVEX 3, bit [6-5]- S_{2-1}) se interpreta como el campo de longitud de vector 2559B (byte de EVEX 3, bit [6-5]- L_{1-0}). Cuando U=1 y el campo MOD 2642 contiene 00, 01 o 10 (lo que significa una operación de acceso a memoria), el campo beta 2554 (byte de EVEX 3, bits [6:4] - SSS) se interpreta como el campo de longitud de vector 2559B (byte de EVEX 3, bit [6-5] - L_{1-0}) y el campo de radiodifusión 2557B (byte de EVEX 3, bit [4] - B).

ARQUITECTURA DE REGISTRO ILUSTRATIVA

La **Figura 27** es un diagrama de bloques de una arquitectura de registro 2700 de acuerdo con una realización de la invención. En la realización ilustrada, hay 32 registros vectoriales 2710 que tienen 512 bits de anchura; estos registros se referencian como zmm0 a zmm31. Los 256 bits de orden inferior de los 16 registros zmm inferiores se superponen en los registros ymm0-16. Los 128 bits de orden inferior de los 16 registros zmm inferiores (los 128 bits de orden inferior de los registros ymm) se superponen en los registros xmm0-15. El formato de instrucciones compatible con vectores específico 2600 opera en este archivo de registro superpuesto, como se ilustra en las siguientes tablas.

Longitud de vector ajustable	Clase	Operaciones	Registros
Plantillas de instrucción que no incluyen el campo de longitud de vector 2559B	A (Figura 25C; U=0)	2517, 2518, 2525, 2530	registros zmm (la longitud de vector es de 64 bytes)

Longitud de vector ajustable	Clase	Operaciones	Registros
	B (Figura 25D; U=1)	2522	registros zmm (la longitud de vector es de 64 bytes)
Plantillas de instrucciones que incluyen el campo de longitud de vector 2559B	B (Figura 25D; U=1)	2523, 2527	registros zmm, ymm o xmm (la longitud del vector es de 64 bytes, 32 bytes o 16 bytes) dependiendo del campo de longitud de vector 2559B

En otras palabras, el campo de longitud de vector 2559B selecciona entre una longitud máxima y una o más longitudes más cortas, donde cada una de tales longitudes más cortas es la mitad de la longitud de la longitud precedente; y las plantillas de instrucciones sin el campo de longitud de vector 2559B operan en la longitud de vector máxima. Además, en una realización, las plantillas de instrucciones de clase B del formato de instrucciones específico compatible con vectores 2600 operan en datos de coma flotante de precisión sencilla/doble escalar o empaquetados y datos de número entero escalar o empaquetados. Las operaciones escalares son operaciones realizadas en la posición de elemento de datos de orden más bajo en un registro zmm/ymm/xmm; las posiciones de los elementos de datos de orden superior se dejan igual que antes de la instrucción o se ponen a cero dependiendo de la realización.

Registros de máscara de escritura 2715 - en la realización ilustrada, hay 8 registros de máscara de escritura (k0 a k7), cada uno de 64 bits de tamaño. En una realización alternativa, los registros de máscara de escritura 2715 tienen un tamaño de 16 bits. Como se ha descrito anteriormente, en una realización de la invención, el registro de máscara vectorial k0 no se puede usar como máscara de escritura; cuando se usa la codificación que normalmente indicaría k0 para una máscara de escritura, selecciona una máscara de escritura programada de 0xFFFF, deshabilitando efectivamente la máscara de escritura para esa instrucción.

Registros de propósito general 2725 - en la realización ilustrada, hay dieciséis registros de propósito general de 64 bits que se usan junto con los modos de direccionamiento x86 existentes para direccionar operandos de memoria. A estos registros se hace referencia con los nombres RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP y R8 a R15.

Archivo de registro de pila de coma flotante escalar (pila x87) 2745, en el que se asigna un alias al archivo de registro plano de número entero empaquetado MMX 2750, en la realización ilustrada, la pila x87 es una pila de ocho elementos usada para realizar operaciones escalares de coma flotante en datos de coma flotante de 32/64/80 bits usando la extensión del conjunto de instrucciones x87; mientras que los registros MMX se usan para realizar operaciones con datos de números enteros empaquetados de 64 bits, así como para contener operandos para algunas operaciones realizadas entre los registros MMX y XMM.

Realizaciones alternativas de la invención pueden usar registros más anchos o más estrechos. Además, las realizaciones alternativas de la invención pueden usar más, menos o diferentes registros y archivos de registro.

ARQUITECTURAS DE NÚCLEO, PROCESADORES Y ARQUITECTURAS INFORMÁTICAS ILUSTRATIVOS

Los núcleos de procesador se pueden implementar de diferentes maneras, para diferentes propósitos y en diferentes procesadores. Por ejemplo, las implementaciones de tales núcleos pueden incluir: 1) un núcleo en orden de propósito general destinado a computación de propósito general; 2) un núcleo en desorden de propósito general de alto rendimiento destinado a computación de propósito general; 3) un núcleo de propósito especial destinado principalmente a gráficos y/o computación científica (rendimiento). Las implementaciones de diferentes procesadores pueden incluir: 1) una CPU que incluye uno o más núcleos en orden de propósito general destinados a la computación de propósito general y/o uno o más núcleos en desorden de propósito general destinados a la computación de propósito general; y 2) un coprocesador que incluye uno o más núcleos de propósito especial destinados principalmente a gráficos y/o temas científicos (rendimiento). Tales procesadores diferentes conducen a diferentes arquitecturas de sistemas informáticos, que pueden incluir: 1) el coprocesador en un chip separado de la CPU; 2) el coprocesador en un encapsulado separado en el mismo paquete que una CPU; 3) el coprocesador en el mismo encapsulado que una CPU (en cuyo caso, un coprocesador de este tipo en ocasiones se denomina lógica de propósito especial, tal como gráficos integrados y/o lógica científica (rendimiento), o núcleos de propósito especial); y 4) un sistema en un chip que puede incluir en el mismo encapsulado la CPU descrita (en ocasiones denominada como el núcleo o núcleos de aplicación o el procesador o procesadores de aplicación), el coprocesador descrito anteriormente y funcionalidad adicional. A continuación, se describen arquitecturas de núcleo ilustrativas, seguidas de descripciones de procesadores y arquitecturas informáticas ilustrativas.

ARQUITECTURAS DE NÚCLEO ILUSTRATIVAS

DIAGRAMA DE BLOQUES DE NÚCLEO EN ORDEN Y EN DESORDEN

La **Figura 28A** es un diagrama de bloques que ilustra tanto una canalización en orden ilustrativa como una canalización de emisión/ejecución en desorden y cambio de nombre de registro ilustrativa de acuerdo con realizaciones de la invención. La **Figura 28B** es un diagrama de bloques que ilustra tanto una realización ilustrativa de un núcleo de arquitectura en orden como un núcleo de arquitectura de emisión/ejecución en desorden de cambio de nombre de registro ilustrativo que se va a incluir en un procesador de acuerdo con realizaciones de la invención. Los cuadros con líneas continuas de las Figuras 28A-B ilustran la canalización en orden y el núcleo en orden, mientras que la adición opcional de los cuadros con líneas discontinuas ilustra la canalización y núcleo de emisión/ejecución en desorden de cambio de nombre de registro. Dado que el aspecto en orden es un subconjunto del aspecto en desorden, se describirá el aspecto en desorden.

En la **Figura 28A,** una canalización de procesador 2800 incluye una etapa de recuperación 2802, una etapa de decodificación de longitud 2804, una etapa de decodificación 2806, una etapa de asignación 2808, una etapa de cambio de nombre 2810, una etapa de planificación (también conocida como despacho o emisión) 2812, una etapa de lectura de registro/lectura de memoria 2814, una etapa de ejecución 2816, una etapa de escritura diferida/escritura de memoria 2818, una etapa de manejo de excepciones 2822 y una etapa de confirmación 2824.

La **Figura 28B** muestra el núcleo de procesador 2890 que incluye una unidad de sección de entrada 2830 acoplada a una unidad de motor de ejecución 2850, y ambas están acopladas a una unidad de memoria 2870. El núcleo 2890 puede ser un núcleo de computación de conjunto de instrucciones reducido (RISC), un núcleo de computación de conjunto de instrucciones complejo (CISC), un núcleo de palabras de instrucción muy largas (VLIW) o un tipo de núcleo híbrido o alternativo. Como otra opción más, el núcleo 2890 puede ser un núcleo de propósito especial, tal como, por ejemplo, un núcleo de red o comunicación, un motor de compresión, un núcleo de coprocesador, un núcleo de unidad de procesamiento de gráficos informáticos de propósito general (GPGPU), un núcleo de gráficos o similar.

La unidad de extremo frontal 2830 incluye una unidad de predicción de bifurcación 2832 acoplada a una unidad de caché de instrucciones 2834, que está acoplada a una memoria intermedia de traducción adelantada (TLB) de instrucciones 2836, que está acoplada a una unidad de extracción de instrucciones 2838, que está acoplada para enviar la instrucción extraída 2839 a una unidad de decodificación 2840. La unidad de decodificación 2840 (o decodificador) puede decodificar instrucciones y generar como salida una o más instrucciones decodificadas 2841, o microoperaciones, puntos de entrada de microcódigo, microinstrucciones, otras instrucciones u otras señales de control, que se decodifican a partir de, o que de otro modo reflejan o se derivan de las instrucciones originales. La unidad de decodificación 2840 puede implementarse usando diversos mecanismos diferentes. Ejemplos de mecanismos adecuados incluyen, pero no se limitan a tablas de consulta, implementaciones de hardware, matrices lógicas programables (PLA), memorias de solo lectura (ROM) de microcódigo, etc. En una realización, el núcleo 2890 incluye una ROM de microcódigo u otro medio que almacena microcódigo para determinadas macroinstrucciones (por ejemplo, en la unidad de decodificación 2840 o de otro modo dentro de la unidad de extremo frontal 2830). La unidad de decodificación 2840 está acoplada a una unidad de cambio de nombre/asignación 2852 en la unidad de motor de ejecución 2850.

La unidad de motor de ejecución 2850 incluye la unidad de cambio de nombre/asignación 2852 acoplada a una unidad de retiro 2854 y un conjunto de una o más unidades de planificación 2856. La unidad o unidades de planificación 2856 representan cualquier número de planificadores diferentes, incluyendo estaciones de reserva, ventana de instrucción central, etc. La unidad o unidades de planificación 2856 están acopladas a la unidad o unidades de archivo o archivos de registro físico 2858. Cada una de las unidades de archivo o archivos de registro físico 2858 representa uno o más archivos de registro físico, diferentes de los que almacenan uno o más tipos de datos diferentes, tales como entero escalar, coma flotante escalar, entero empaquetado, coma flotante empaquetado, entero vectorial, coma flotante vectorial, estado (por ejemplo, un puntero de instrucción que es la dirección de la siguiente instrucción a ejecutar), etc. En una realización, la unidad de archivo o archivos de registros físicos 2858 comprende una unidad de registros vectoriales, una unidad de registros de máscara y una unidad de registros escalares. Estas unidades de registro pueden proporcionar registros vectoriales arquitectónicos, registros de máscara vectorial y registros de propósito general. La unidad o unidades de archivo o archivos de registro físico 2858 se superponen con la unidad de retiro 2854 para ilustrar diversas formas en las que se puede implementar el cambio de nombre de registro y la ejecución en desorden (por ejemplo, usando una memoria o memorias intermedias de reordenación y un archivo o archivos de registro de retiro; usando un archivo o archivos futuros, una memoria o memorias intermedias de historial y un archivo o archivos de registro de retiro; usando mapas de registros y una agrupación de registros; etc.). La unidad de retiro 2854 y la unidad o unidades de archivo o archivos de registro físico 2858 se acoplan a la agrupación o agrupaciones de ejecución 2860. La agrupación o agrupaciones de ejecución 2860 incluye un conjunto de una o más unidades de ejecución 2862 y un conjunto de una o más unidades de acceso a memoria 2864. Las unidades de ejecución 2862 pueden realizar diversas operaciones (por ejemplo, desplazamientos, suma, resta, multiplicación) y sobre diversos tipos de datos (por ejemplo, coma flotante escalar, entero empaquetado, coma flotante empaquetada, entero vectorial, coma flotante vectorial). Si bien algunas realizaciones pueden incluir un número de unidades de ejecución especializadas a funciones específicas o conjuntos de funciones, otras realizaciones pueden incluir únicamente una unidad de ejecución o múltiples unidades de ejecución que realizan todas las funciones. La unidad o unidades de planificador 2856, la unidad o unidades de archivo o archivos de registro físico 2858 y la agrupación o agrupación de ejecución 2860 se muestran como posiblemente varios porque ciertas realizaciones crean canalizaciones separadas

para ciertos tipos de datos/operaciones (por ejemplo, una canalización de número entero escalar, una canalización de coma flotante escalar/número entero empaquetado/coma flotante empaquetado/número entero vectorial/coma flotante vectorial y/o una canalización de acceso a memoria, cada una de las cuales tiene su propia unidad de planificador, unidad de archivo o archivos de registro físico y/o agrupación de ejecución - y en el caso de una canalización de acceso a memoria separada, se implementan ciertas realizaciones en las que únicamente la agrupación de ejecución de esta canalización tiene la unidad o unidades de acceso a memoria 2864). También se debe entender que, cuando se usan canalizaciones separadas, una o más de estas canalizaciones pueden ser de emisión/ejecución en desorden y el resto en orden.

El conjunto de unidades con acceso a memoria 2864 está acoplado a la unidad de memoria 2870, que incluye una unidad TLB de datos 2872 acoplada a una unidad de caché de datos 2874 acoplada a una unidad de caché de nivel 2 (L2) 2876. En una realización ilustrativa, las unidades con acceso a memoria 2864 puede incluir una unidad de carga, una unidad de dirección de almacenamiento y una unidad de datos de almacenamiento, cada una de las cuales está acoplada a la unidad TLB de datos 2872 en la unidad de memoria 2870. La unidad de caché de instrucciones 2834 se acopla además a una unidad de caché de nivel 2 (L2) 2876 en la unidad de memoria 2870. La unidad de caché L2 2876 está acoplada a uno o más niveles de caché y eventualmente a una memoria principal.

A modo de ejemplo, la arquitectura central ilustrativa de cambio de nombre de registro, emisión/ejecución desordenada puede implementar la canalización 2800 de la siguiente manera: 1) la recuperación de instrucciones 2838 realiza las etapas de recuperación y decodificación de longitud 2802 y 2804; 2) la unidad de decodificación 2840 realiza la etapa de decodificación 2806; 3) la unidad de cambio de nombre/asignación 2852 realiza la etapa de asignación 2808 y la etapa de cambio de nombre 2810; 4) la o las unidades de planificación 2856 realizan la etapa de planificación 2812; 5) la o las unidades de archivos de registro físico 2858 y la unidad de memoria 2870 realizan la etapa 2814 de lectura de registro/lectura de memoria; el grupo de ejecución 2860 realiza la etapa de ejecución 2816; 6) la unidad de memoria 2870 y la o las unidades de archivos de registro físico 2858 realizan la etapa 2818 de reescritura/escritura en memoria; 7) varias unidades pueden estar involucradas en la etapa de manejo de excepciones 2822; y 8) la unidad de retiro 2854 y la o las unidades de archivos de registro físico 2858 realizan la etapa de confirmación 2824.

El núcleo 2890 puede soportar uno o más conjuntos de instrucciones (por ejemplo, el conjunto de instrucciones x86 (con algunas extensiones que se han añadido con versiones más recientes); el conjunto de instrucciones MIPS de MIPS Technologies de Sunnyvale, CA; el conjunto de instrucciones ARM (con extensiones opcionales adicionales tal como NEON) de ARM Holdings de Sunnyvale, CA), incluyendo la instrucción o instrucciones descritas en el presente documento. En una realización, el núcleo 2890 incluye una lógica para soportar una extensión del conjunto de instrucciones de datos empaquetados (por ejemplo, AVX1, AVX2), permitiendo de este modo que las operaciones usadas por muchas aplicaciones multimedia se realicen usando datos empaquetados.

Debe entenderse que el núcleo puede soportar hilos múltiples (ejecutar dos o más conjuntos paralelos de operaciones o hilos), y puede hacerlo de diversas maneras, incluyendo hilos múltiples en intervalos de tiempo, hilos múltiples simultáneos (donde un único núcleo físico proporciona un núcleo lógico para cada uno de los hilos que el núcleo físico está subprocesando de forma múltiple simultáneamente), o una combinación de los mismos (por ejemplo, recuperación y decodificación en cortes de tiempo e hilos múltiples simultáneos a partir de entonces, como en la tecnología Intel® Hyperthreading).

Si bien el cambio de nombre de registros se describe en el contexto de la ejecución en desorden, se debe entender que el cambio de nombre de registros se puede usar en una arquitectura en orden. Aunque la realización ilustrada del procesador también incluye unidades de caché de instrucciones y de datos 2834/2874 separadas y una unidad de caché de L2 2876 compartida, realizaciones alternativas pueden tener una única caché interna tanto para instrucciones como para datos, tal como, por ejemplo, una caché interna de nivel 1 (L1) o múltiples niveles de caché interna. En algunas realizaciones, el sistema puede incluir una combinación de una memoria caché interna y una memoria caché externa que es externa al núcleo y/o al procesador. Como alternativa, toda la caché puede ser externa al núcleo y/o al procesador.

ARQUITECTURA DE NÚCLEO EN ORDEN ILUSTRATIVA ESPECÍFICA

Las Figuras 29A-B ilustran un diagrama de bloques de una arquitectura de núcleo en orden, ilustrativa más específica, cuyo núcleo sería uno de varios bloques lógicos (que incluyen otros núcleos del mismo tipo y/o tipos diferentes) en un chip. Los bloques lógicos se comunican a través de una red de interconexión de ancho de banda alto (por ejemplo, una red en anillo) con alguna lógica de función fija, interfaces de E/S de memoria y otra lógica de E/S necesaria, dependiendo de la aplicación.

La **Figura 29A** es un diagrama de bloques de un único núcleo de procesador, junto con su conexión a la red de interconexión en chip 2902 y con su subconjunto local de la caché de nivel 2 (L2) 2904, de acuerdo con realizaciones de la invención. En una realización, un decodificador de instrucciones 2900 soporta el conjunto de instrucciones x86 con una extensión del conjunto de instrucciones de datos empaquetados. Una caché L1 2906 permite accesos de baja latencia a la memoria caché en las unidades escalares y vectoriales. Mientras que en una realización (para simplificar el diseño), una unidad escalar 2908 y una unidad vectorial 2910 usan conjuntos de registros separados

(respectivamente, registros escalares 2912 y registros vectoriales 2914) y los datos transferidos entre ellas se escriben en la memoria y, a continuación, se vuelven a leer desde una caché de nivel 1 (L1) 2906, realizaciones alternativas de la invención pueden usar un enfoque diferente (por ejemplo, usar un único conjunto de registros o incluir una ruta de comunicación que permite que los datos se transfieran entre los dos archivos de registro sin tener que escribirse ni volverse a leer).

El subconjunto local de la caché de L2 2904 es parte de una caché de L2 global que se divide en subconjuntos locales separados, uno por núcleo de procesador. Cada núcleo de procesador tiene una ruta de acceso directo a su propio subconjunto local de la caché de L2 2904. Los datos leídos por un núcleo de procesador se almacenan en su subconjunto de caché de L2 2904 y se puede acceder a los mismos rápidamente, en paralelo con que otros núcleos de procesador accedan a sus propios subconjuntos de caché de L2 locales. Los datos escritos por un núcleo de procesador se almacenan en su propio subconjunto de memoria caché de L2 2904 y se eliminan de otros subconjuntos, si es necesario. La red en anillo garantiza la coherencia de los datos compartidos. La red en anillo es bidireccional para permitir que agentes tales como núcleos de procesador, caché L2 y otros bloques lógicos se comuniquen entre sí dentro del chip. Cada ruta de datos del anillo tiene 1012 bits de anchura por dirección.

La **Figura 29B** es una vista ampliada de parte del núcleo del procesador en la **Figura 29A** de acuerdo con realizaciones de la invención. La **Figura 29B** incluye una caché de datos L1 2906A parte de la caché L1 2904, así como más detalles con respecto a la unidad vectorial 2910 y los registros vectoriales 2914. Específicamente, la unidad vectorial 2910 es una unidad de procesamiento vectorial (VPU) de anchura 16 (véase la ALU 2928 de anchura 16), que ejecuta una o más de instrucciones de números enteros, flotantes de precisión simple y flotantes de precisión doble. La VPU soporta el mezclado de las entradas de registro con la unidad de mezcla 2920, la conversión numérica con las unidades de conversión numérica 2922A-B y la replicación con la unidad de replicación 2924 en la entrada de memoria. Los registros de máscara de escritura 2926 permiten predecir escrituras vectoriales resultantes.

La **Figura 30** es un diagrama de bloques de un procesador 3000 que puede tener más de un núcleo, puede tener un controlador de memoria integrado y puede tener gráficos integrados de acuerdo con realizaciones de la invención. Los recuadros con líneas continuas en la **Figura 30** ilustran un procesador 3000 con un único núcleo 3002A, un agente de sistema 3010, un conjunto de una o más unidades de controlador de bus 3016, mientras que, la adición opcional de los recuadros con líneas discontinuas ilustra un procesador alternativo 3000 con múltiples núcleos 3002A-N, un conjunto de una o más unidad o unidades de controlador de memoria integrado 3014 en la unidad de agente de sistema 3010 y lógica de propósito especial 3008.

Por tanto, diferentes implementaciones del procesador 3000 pueden incluir: 1) una CPU con la lógica de propósito especial 3008 que es una lógica (que puede incluir uno o más núcleos) de gráficos y/o temas científicos integrados (rendimiento), y siendo los núcleos 3002A-N uno o más núcleos de propósito general (por ejemplo, núcleos en orden de propósito general, núcleos en desorden de propósito general, una combinación de los dos); 2) un coprocesador con núcleos 3002A-N que son un gran número de núcleos de propósito especial destinados principalmente a gráficos y/o temas científicos (rendimiento); y 3) un coprocesador con los núcleos 3002A-N que son un gran número de núcleos en orden de propósito general. Por tanto, el procesador 3000 puede ser un procesador de propósito general, coprocesador o procesador de propósito especial, tal como, por ejemplo, un procesador de red o comunicación, motor de compresión, procesador de gráficos, GPGPU (unidad de procesamiento de gráficos de propósito general), un coprocesador de alto rendimiento de muchos núcleos integrados (MIC) (que incluye 30 o más núcleos), procesador integrado o similar. El procesador puede implementarse en uno o más chips. El procesador 3000 puede ser parte y/o puede implementarse en uno o más sustratos usando cualquiera de un número de tecnologías de proceso, tales como, por ejemplo, BiCMOS, CMOS o NMOS.

La jerarquía de memoria incluye uno o más niveles de caché dentro de los núcleos, un conjunto o una o más unidades de caché compartidas 3006 y memoria externa (no mostrada) acoplada al conjunto de unidades de controlador de memoria integrado 3014. El conjunto de unidades de caché compartida 3006 puede incluir una o más caché de nivel medio, tales como de nivel 2 (L2), nivel 3 (L3), nivel 4 (L4) u otros niveles de caché, una caché de último nivel (LLC) y/o combinaciones de las mismas. Si bien en una realización una unidad de interconexión basada en anillo 3012 interconecta la lógica de gráficos integrada 3008 (la lógica de gráficos integrada 3008 es un ejemplo y también se denomina en el presente documento lógica de propósito especial), el conjunto de unidades de caché compartida 3006 y la unidad de agente del sistema 3010/unidad o unidades de controlador de memoria integrada 3014, realizaciones alternativas pueden usar cualquier número de técnicas bien conocidas para interconectar dichas unidades. En una realización, se mantiene la coherencia entre una o más unidades de caché 3006 y núcleos 3002-A-N.

En algunas realizaciones, uno o más de los núcleos 3002A-N son aptos para múltiples subprocesos. El agente de sistema 3010 incluye aquellos componentes que coordinan y operan los núcleos 3002A-N. La unidad de agente de sistema 3010 puede incluir, por ejemplo, una unidad de control de energía (PCU) y una unidad de visualización. La PCU puede ser o incluir la lógica y los componentes necesarios para regular el estado de energía de los núcleos 3002A-N y la lógica de gráficos integrados 3008. La unidad de visualización es para controlar una o más pantallas conectadas externamente.

Los núcleos 3002A-N pueden ser homogéneos o heterogéneos en términos de conjunto de instrucciones de arquitectura; es decir, dos o más de los núcleos 3002A-N pueden ser capaces de ejecutar el mismo conjunto de instrucciones, mientras que otros pueden ser capaces de ejecutar solo un subconjunto de ese conjunto de instrucciones o un conjunto de instrucciones diferente.

ARQUITECTURAS INFORMÁTICAS ILUSTRATIVAS

Las **Figuras 31-34** son diagramas de bloques de arquitecturas informáticas ilustrativas. Otros diseños y configuraciones de sistema conocidos en la técnica para portátiles, equipos de sobremesa, PC portátiles, asistentes digitales personales, estaciones de trabajo de ingeniería, servidores, dispositivos de red, concentradores de red, conmutadores, procesadores integrados, procesadores de señales digitales (DSP), dispositivos de gráficos, dispositivos de videojuegos, decodificadores de salón, microcontroladores, teléfonos móviles, reproductores multimedia portátiles, dispositivos de mano y diversos otros dispositivos electrónicos, también son adecuados. En general, son generalmente adecuados una gran diversidad de sistemas o dispositivos electrónicos que pueden incorporar un procesador y/u otra lógica de ejecución como se divulga en el presente documento.

Haciendo referencia ahora a la **Figura 31,** se muestra un diagrama de bloques de un sistema 3100 de acuerdo con una realización de la presente invención. El sistema 3100 puede incluir uno o más procesadores 3110, 3115, que están acoplados a un concentrador de controlador 3120. En una realización, el concentrador de controlador 3120 incluye un concentrador de controlador de memoria gráfica (GMCH) 3190 y un concentrador de entrada/salida (IOH) 3150 (que puede estar en chips separados); el GMCH 3190 incluye controladores de memoria y gráficos a los que están acoplados la memoria 3140 y un coprocesador 3145; el IOH 3150 acopla los dispositivos de entrada/salida (E/S) 3160 al GMCH 3190. Como alternativa, uno o ambos controladores de memoria y gráficos están integrados dentro del procesador (como se describe en el presente documento), la memoria 3140 y el coprocesador 3145 están acoplados directamente al procesador 3110, y al concentrador de controlador 3120 en un único chip con el IOH 3150.

La naturaleza opcional de los procesadores adicionales 3115 se indica en la **Figura 31** con líneas discontinuas. Cada procesador 3110, 3115 puede incluir uno o más de los núcleos de procesamiento descritos en el presente documento y puede ser alguna versión del procesador 3000.

La memoria 3140 puede ser, por ejemplo, una memoria dinámica de acceso aleatorio (DRAM), una memoria de cambio de fase (PCM) o una combinación de las dos. Para al menos una realización, el concentrador de controlador 3120 se comunica con el o los procesadores 3110, 3115 a través de un bus multipunto, tal como un bus frontal (FSB), una interfaz punto a punto tal como QuickPath Interconnect (QPI) o una conexión similar 3195.

En una realización, el coprocesador 3145 es un procesador de propósito especial, tal como, por ejemplo, un procesador MIC de alto rendimiento, un procesador de red o comunicación, motor de compresión, procesador de gráficos, GPGPU, procesador integrado o similar. En una realización, el concentrador de controlador 3120 puede incluir un acelerador de gráficos integrado.

Puede haber varias diferencias entre los recursos físicos 3110, 3115 en lo que respecta a un espectro de métricas de mérito, que incluyen características arquitectónicas, microarquitectónicas, térmicas, de consumo de energía y similares.

En una realización, el procesador 3110 ejecuta instrucciones que controlan operaciones de procesamiento de datos de un tipo general. Incrustadas dentro de las instrucciones puede haber instrucciones de coprocesador. El procesador 3110 reconoce estas instrucciones de coprocesador como de un tipo que debería ser ejecutado por el coprocesador adjunto 3145. En consecuencia, el procesador 3110 emite estas instrucciones de coprocesador (o señales de control que representan instrucciones de coprocesador) en un bus del coprocesador u otra interconexión, al coprocesador 3145. Los coprocesadores 3145 aceptan y ejecutan las instrucciones de coprocesador recibidas.

Con referencia ahora a la **Figura 32**, se muestra un diagrama de bloques de un primer sistema 3200 ilustrativo más específico de acuerdo con una realización de la presente invención. Como se muestra en la **Figura 32**, el sistema multiprocesador 3200 es un sistema de interconexión punto a punto e incluye un primer procesador 3270 y un segundo procesador 3280 acoplados a través de una interconexión punto a punto 3250. Cada uno de los procesadores 3270 y 3280 puede ser alguna versión del procesador 3000. En una realización de la invención, los procesadores 3270 y 3280 son respectivamente los procesadores 3110 y 3115, mientras que el coprocesador 3238 es el coprocesador 3145. En otra realización, los procesadores 3270 y 3280 son respectivamente el procesador 3110 y el coprocesador 3145.

Los procesadores 3270 y 3280 se muestran incluyendo las unidades de controlador de memoria integrada (IMC) 3272 y 3282, respectivamente. El procesador 3270 también incluye como parte de sus unidades de controlador de bus, interfaces punto a punto (P-P) 3276 y 3278; de manera similar, el segundo procesador 3280 incluye interfaces P-P 3286 y 3288. Los procesadores 3270, 3280 pueden intercambiar información a través de una interfaz punto a punto (P-P) 3250 usando circuitos de interfaz P-P 3278, 3288. Como se muestra en la **Figura 32**, los IMC 3272 y 3282 acoplan los procesadores a las respectivas memorias, en concreto, una memoria 3232 y una memoria 3234, que pueden ser porciones de la memoria principal conectadas localmente a los respectivos procesadores.

Cada uno de los procesadores 3270, 3280 puede intercambiar información con un conjunto de chips 3290 a través de interfaces P-P 3252, 3254 individuales usando circuitos de interfaz punto a punto 3276, 3294, 3286, 3298. El conjunto de chips 3290 puede opcionalmente intercambiar información con el coprocesador 3238 a través de una interfaz de alto rendimiento 3292. En una realización, el coprocesador 3238 es un procesador de propósito especial, tal como, por ejemplo, un procesador MIC de alto rendimiento, un procesador de red o comunicación, motor de compresión, procesador de gráficos, GPGPU, procesador integrado o similar.

Se puede incluir una memoria caché compartida (no mostrada) en uno cualquiera de los procesadores o fuera de ambos procesadores, pero conectada con los procesadores por medio de una interconexión P-P, de tal forma que la información de memoria caché local de uno cualquiera de los procesadores, o de ambos, se pueda almacenar en la memoria caché compartida si un procesador pasa a un modo de bajo consumo.

El conjunto de chips 3290 se puede acoplar a un primer bus 3216 a través de una interfaz 3296. En una realización, el primer bus 3216 puede ser un bus de interconexión de componentes periféricos (PCI), o un bus tal como un bus PCI Express u otro bus de interconexión de E/S de tercera generación, aunque el alcance de la presente invención no está así limitado.

Como se muestra en la **Figura 32**, diversos dispositivos de E/S 3214 pueden acoplarse al primer bus 3216, junto con un puente de bus 3218 que acopla el primer bus 3216 a un segundo bus 3220. En una realización, uno o más procesadores adicionales 3215, tales como coprocesadores, procesadores MIC de alto rendimiento, GPGPU, aceleradores (tales como, por ejemplo, aceleradores de gráficos o unidades de procesamiento de señales digitales (DSP)), matrices de puertas programables en campo o cualquier otro procesador, están acoplados al primer bus 3216. En una realización, el segundo bus 3220 puede ser un bus de recuento bajo de pines (LPC). Se pueden acoplar diversos dispositivos a un segundo bus 3220 que incluye, por ejemplo, un teclado y/o ratón 3222, dispositivos de comunicaciones 3227 y una unidad de almacenamiento 3228 tal como una unidad de disco u otro dispositivo de almacenamiento masivo que puede incluir instrucciones/códigos y datos 3230, en una realización. Además, se puede acoplar una E/S de audio 3224 al segundo bus 3220. Obsérvese que son posibles otras arquitecturas. Por ejemplo, en lugar de la arquitectura de punto a punto de la **Figura 32**, un sistema puede implementar un bus multipunto u otra arquitectura de este tipo.

Con referencia ahora a la **Figura 33**, se muestra un diagrama de bloques de un segundo sistema 3300 ilustrativo más específico de acuerdo con una realización de la presente invención. Elementos similares en las **Figuras 32 y 33** llevan números de referencia similares, y ciertos aspectos de la **Figura 32** se han omitido de la **Figura 33** para evitar complicar otros aspectos de la **Figura 33**.

La **Figura 33** ilustra que los procesadores 3270, 3280 pueden incluir memoria integrada y lógica de control de E/S ("CL") 3272 y 3282, respectivamente. Por tanto, la CL 3272, 3282 incluye unidades de controlador de memoria integrado e incluye lógica de control de E/S. La **Figura 33** ilustra que no solo las memorias 3232, 3234 están acopladas a la CL 3272, 3282, sino que también los dispositivos de E/S 3314 también están acoplados a la lógica de control 3272, 3282. Los dispositivos de E/S heredados 3315 están acoplados al conjunto de chips 3290.

Con referencia ahora a la **Figura 34**, se muestra un diagrama de bloques de un SoC 3400 de acuerdo con una realización de la presente invención. Elementos similares en la **Figura 30** llevan los mismos números de referencia. Además, los recuadros con línea discontinua son características opcionales en los SoC más avanzados. En la **Figura 34**, una unidad o unidades de interconexión 3402 están acopladas a: un procesador de aplicaciones 3410 que incluye un conjunto de uno o más núcleos 3002A-N, que incluyen una o más unidades de caché 3004A-N y una unidad o unidades de caché compartida 3006; una unidad de agente de sistema 3010; una unidad o unidades de controlador de bus 3016; una unidad o unidades de controlador de memoria integrado 3014; un conjunto o uno o más coprocesadores 3420 que pueden incluir lógica de gráficos integrada, un procesador de imágenes, un procesador de audio y un procesador de vídeo; una unidad de memoria de acceso aleatorio estática (SRAM) 3430; una unidad de acceso directo a memoria (DMA) 3432; y una unidad de visualización 3440 para acoplarse a una o más pantallas externas. En una realización, el coprocesador o coprocesadores 3420 incluyen un procesador de propósito especial, tal como, por ejemplo, un procesador de red o comunicación, motor de compresión, GPGPU, un procesador MIC de alto rendimiento, procesador integrado o similar.

Las realizaciones de los mecanismos divulgados en el presente documento pueden implementarse en hardware, software, firmware o una combinación de tales enfoques de implementación. Las realizaciones de la invención pueden implementarse como programas informáticos o código de programa que se ejecuta en sistemas programables que comprenden al menos un procesador, un sistema de almacenamiento (que incluye memoria y/o elementos de almacenamiento volátiles y no volátiles), al menos un dispositivo de entrada y al menos un dispositivo de salida.

El código de programa, tal como el código 3230 ilustrado en la **Figura 32**, se puede aplicar a las instrucciones de entrada para realizar las funciones descritas en el presente documento y generar información de salida. La información de salida puede aplicarse a uno o más dispositivos de salida, de forma conocida. Para los propósitos de esta solicitud, un sistema de procesamiento incluye cualquier sistema que tenga un procesador, tal como, por ejemplo; un procesador

de señales digitales (DSP), un microcontrolador, un circuito integrado específico de la aplicación (ASIC) o un microprocesador.

El código de programa se puede implementar en un lenguaje de programación orientado a objetos o procedimental de alto nivel para comunicarse con un sistema de procesamiento. El código de programa también se puede implementar en lenguaje ensamblador o máquina, si se desea. De hecho, los mecanismos descritos en el presente documento no están limitados en su alcance a ningún lenguaje de programación particular. En cualquier caso, el lenguaje puede ser un lenguaje compilado o interpretado.

Uno o más aspectos de al menos una realización pueden implementarse mediante instrucciones representativas almacenadas en un medio legible por máquina que representa diversa lógica dentro del procesador que, cuando las lee una máquina, hace que la máquina fabrique lógica para realizar las técnicas descritas en el presente documento. Tales representaciones, conocidas como "núcleos de IP", pueden almacenarse en un medio legible por máquina tangible y suministrarse a diversos clientes o instalaciones de fabricación para cargarlas en las máquinas de fabricación que realmente hacen la lógica o el procesador.

Tales medios de almacenamiento legibles por máquina pueden incluir, sin limitación, disposiciones tangibles no transitorias de artículos fabricados o formados por una máquina o dispositivo, que incluyen medios de almacenamiento tales como discos duros, cualquier otro tipo de disco, incluyendo disquetes, discos ópticos memorias de solo lectura de disco compacto (CD-ROM), discos compactos reescribibles (CD-RW) y discos magneto-ópticos, dispositivos de semiconductores tales como memorias de solo lectura (ROM), memorias de acceso aleatorio (RAM) tales como memorias de acceso aleatorio dinámicas (DRAM), memorias de acceso aleatorio estáticas (SRAM), memorias de solo lectura programables y borrables (EPROM), memorias flash, memorias de solo lectura programables y borrables eléctricamente (EEPROM), memorias de cambio de fase (PCM), tarjetas magnéticas u ópticas, o cualquier otro tipo de medio adecuado para almacenamiento de instrucciones electrónicas.

En consecuencia, las realizaciones de la invención también incluyen medios legibles por máquina, tangibles, no transitorios que contienen instrucciones o datos de diseño, tales como el lenguaje de descripción de hardware (HDL), que define estructuras, circuitos, aparatos, procesadores y/o características de sistema descritos en el presente documento. Tales realizaciones también pueden denominarse productos de programa.

EMULACIÓN (INCLUYENDO TRADUCCIÓN BINARIA, TRANSFORMACIÓN DE CÓDIGOS, ETC.)

En algunos casos, se puede usar un convertidor de instrucciones para convertir una instrucción de un conjunto de instrucciones de origen a un conjunto de instrucciones objetivo. Por ejemplo, el convertidor de instrucciones puede traducir (por ejemplo, usando traducción binaria estática, traducción binaria dinámica que incluye compilación dinámica), transformar, emular o convertir de otro modo una instrucción en una o más instrucciones para que las procese el núcleo. El convertidor de instrucciones puede implementarse en software, hardware, firmware o una combinación de los mismos. El convertidor de instrucciones puede estar en el procesador, fuera del procesador o en parte dentro y fuera del procesador.

La **Figura 35** es un diagrama de bloques que contrasta el uso de un convertidor de instrucciones de software para convertir instrucciones binarias en un conjunto de instrucciones de origen en instrucciones binarias en un conjunto de instrucciones de destino de acuerdo con realizaciones de la invención. En la realización ilustrada, el convertidor de instrucciones es un convertidor de instrucciones de software, aunque, como alternativa, el convertidor de instrucciones puede implementarse en software, firmware, hardware o diversas combinaciones de los mismos. La **Figura 35** muestra que un programa en un lenguaje de alto nivel 3502 puede compilarse usando un compilador x86 3504 para generar código binario x86 3506 que puede ejecutarse de forma nativa mediante un procesador con al menos un núcleo de conjunto de instrucciones x86 3516. El procesador con al menos un núcleo de conjunto de instrucciones x86 3516 representa cualquier procesador que puede realizar sustancialmente las mismas funciones que un procesador Intel con al menos un núcleo de conjunto de instrucciones x86 mediante la ejecución compatible o el procesamiento de otro modo (1) de una porción sustancial del conjunto de instrucciones del núcleo del conjunto de instrucciones Intel x86 o (2) de versiones de código objeto de aplicaciones u otro software destinado a ejecutarse en un procesador Intel con al menos un núcleo del conjunto de instrucciones x86, para conseguir sustancialmente el mismo resultado que un procesador Intel con al menos un núcleo del conjunto de instrucciones x86. El compilador x86 3504 representa un compilador que puede funcionar para generar código binario x86 3506 (por ejemplo, código objeto) que puede, con o sin procesamiento de vinculación adicional, ejecutarse en el procesador con al menos un núcleo de conjunto de instrucciones x86 3516. De manera similar, la **Figura 35** muestra que el programa en el lenguaje de alto nivel 3502 puede compilarse usando un compilador de conjunto de instrucciones alternativo 3508 para generar un código binario de conjunto de instrucciones alternativo 3510 que puede ejecutarse de forma nativa por un procesador sin al menos un núcleo de conjunto de instrucciones x86 3514 (por ejemplo, un procesador con núcleos que ejecutan el conjunto de instrucciones MIPS de MIPS Technologies de Sunnyvale, CA y/o que ejecutan el conjunto de instrucciones ARM de ARM Holdings de Sunnyvale, CA). El convertidor de instrucciones 3512 se usa para convertir el código binario x86 3506 en un código que el procesador puede ejecutar de forma nativa sin un núcleo de conjunto de instrucciones x86 3514. No es probable que este código convertido sea el mismo que el código binario del conjunto de instrucciones alternativo 3510 porque es difícil hacer un convertidor de instrucciones apto para esto; sin embargo, el código

convertido conseguirá la operación general y estará compuesto por instrucciones del conjunto de instrucciones alternativo. Por tanto, el convertidor de instrucciones 3512 representa software, firmware, hardware o una combinación de los mismos que, mediante emulación, simulación o cualquier otro proceso, permite que un procesador u otro dispositivo electrónico que no tiene un procesador o núcleo de conjunto de instrucciones x86 ejecute el código binario x86 3506.

REIVINDICACIONES

1. Un procesador que comprende:
circuitería de extracción para extraer una pluralidad de instrucciones;
circuitería de decodificación (1303, 1403) para decodificar la pluralidad de instrucciones extraídas; y
circuitería de ejecución (1311), que responde a la pluralidad de instrucciones decodificadas para:
identificar una primera y una segunda instrucciones decodificadas que pertenecen a una cadena de instrucciones;
seleccionar una ruta que comprende un primer y un segundo motores de procesamiento (PE) para ejecutar la primera y la segunda instrucciones decodificadas; y
enrutar un resultado de la primera instrucción decodificada desde el primer PE al segundo PE para usarse por el segundo PE para realizar la segunda instrucción decodificada; **caracterizado por que**
cada una de las instrucciones tiene especificadores de tesela de origen (2504, 2505, 2506) y destino (2503) para especificar teselas de origen y destino respectivas, representando una tesela una estructura de datos bidimensional en una porción de almacenamiento que tiene elementos de datos dispuestos en filas y columnas;
la ruta es una ruta de SIMD y se selecciona y configura dinámicamente;
el especificador de tesela de destino (2503) de la primera instrucción decodificada se aparta y el resultado de la primera instrucción decodificada se enruta en su lugar desde el primer PE al segundo PE; y
la circuitería de ejecución (1311) es para establecer un bit sucio en una configuración de tesela de la tesela de destino especificada de la primera instrucción cuando se aparta el especificador de tesela de destino de la primera instrucción decodificada, y provocar un fallo si una instrucción posterior lee la tesela sucia.

2. El procesador de la reivindicación 1, en donde la pluralidad de instrucciones comprende al menos dos instrucciones, y en donde cada instrucción de la cadena de instrucciones ha de comprender, además, un campo de control de cadena para provocar que la circuitería de ejecución (1311) determine que la instrucción es parte de una cadena; y
en donde el campo de control de cadena es para indicar una sugerencia de posición de cadena que comprende una de una sugerencia de inicio de cadena, una sugerencia intermedia de cadena y una sugerencia de final de cadena para marcar una última instrucción en la cadena, y en donde el campo de control de cadena es para provocar que la circuitería de ejecución (1311) determine la existencia de la cadena, seleccionar y configurar dinámicamente la ruta de SIMD para realizar la cadena de instrucciones, apartar los especificadores de tesela de destino (2503) de todas menos la última instrucción en la cadena y, en su lugar, enrutar los resultados de todas menos la última instrucción a un siguiente PE que realice una siguiente instrucción en la cadena.

3. El procesador de la reivindicación 1, en donde la primera y la segunda instrucciones decodificadas especifican diferentes códigos de operación para especificar una primera y una segunda operaciones aritméticas diferentes, y en donde el primer y el segundo motores de procesamiento seleccionados han de tener una funcionalidad limitada, especializándose en la primera operación aritmética y la segunda operación aritmética, respectivamente.

4. El procesador de la reivindicación 1, en donde una primera instrucción de la cadena de instrucciones comprende, además, un campo de control de cadena, comprendiendo el campo de control de cadena una cabecera de cadena para identificar la primera instrucción y una o más instrucciones posteriores como parte de la cadena o las instrucciones.

5. El procesador de una cualquiera de las reivindicaciones 1-4, en donde cada una de las teselas de origen y destino especificadas comprenden M filas y N columnas de elementos y únicamente han de contener elementos válidos en una fila.

6. El procesador de una cualquiera de las reivindicaciones 1-5, en donde la circuitería de ejecución (1311) es, además, para mantener un registro de instrucciones que han apartado sus especificadores de tesela de destino (2503), y para retroceder y volver a ejecutar una o más de las instrucciones registradas según sea necesario.

7. Un método realizado por un procesador, comprendiendo el método:
extraer (2201), usando circuitería de extracción, una pluralidad de instrucciones;
decodificar (2205), usando circuitería de decodificación (1303, 1403), la pluralidad de instrucciones extraídas; y
ejecutar (2209), usando circuitería de ejecución (1311), la pluralidad de instrucciones decodificadas, para:
identificar (2211, 2308) una primera y una segunda instrucciones decodificadas que pertenecen a una cadena de instrucciones;
seleccionar una ruta que comprende un primer y un segundo motores de procesamiento (PE) para ejecutar la primera y la segunda instrucciones decodificadas; y
enrutar un resultado de la primera instrucción decodificada desde el primer PE al segundo PE para usarse por el segundo PE para realizar la segunda instrucción decodificada; **caracterizado por que**
cada una de las instrucciones tiene especificadores de tesela de origen (2504, 2505, 2506) y destino (2503) para especificar teselas de origen y destino respectivas, representando una tesela una estructura de datos bidimensional en una porción de almacenamiento que tiene elementos de datos dispuestos en filas y columnas;
la ruta es una ruta de SIMD y se selecciona y configura dinámicamente;
el especificador de tesela de destino (2503) de la primera instrucción decodificada se aparta y el resultado de la primera instrucción decodificada se enruta en su lugar desde el primer PE al segundo PE; y

la circuitería de ejecución (1311) es para establecer un bit sucio en una configuración de tesela de la tesela de destino especificada de la primera instrucción cuando se aparta el especificador de tesela de destino de la primera instrucción decodificada, y provocar un fallo si una instrucción posterior lee la tesela sucia.

8. El método de la reivindicación 7, en donde la pluralidad de instrucciones comprende al menos dos instrucciones, y en donde cada instrucción de la cadena de instrucciones ha de comprender, además, un campo de control de cadena para provocar que la circuitería de ejecución (1311) determine que la instrucción es parte de una cadena; y
en donde el campo de control es para indicar una sugerencia de posición de cadena que comprende una de una sugerencia de inicio de cadena, una sugerencia intermedia de cadena y una sugerencia de final de cadena para marcar una última instrucción en la cadena, y en donde el campo de control de cadena es para provocar que la circuitería de ejecución (1311) determine la existencia de la cadena de instrucciones, seleccionar y configurar dinámicamente la ruta de SIMD para realizar la cadena de instrucciones, apartar los especificadores de tesela de destino (2503) de todas menos la última instrucción en la cadena y, en su lugar, enrutar los resultados de todas menos la última instrucción a un siguiente PE que realice una siguiente instrucción en la cadena.

9. El método de la reivindicación 7, en donde la primera y la segunda instrucciones decodificadas especifican diferentes códigos de operación para especificar una primera y una segunda operaciones aritméticas diferentes, y en donde el primer y el segundo motores de procesamiento seleccionados han de tener una funcionalidad limitada, especializándose en la primera operación aritmética y la segunda operación aritmética, respectivamente.

10. El método de una cualquiera de las reivindicaciones 7-9, en donde una primera instrucción de la cadena de instrucciones comprende, además, un campo de control de cadena, comprendiendo el campo de control de cadena una cabecera de cadena para identificar la primera instrucción y una o más instrucciones posteriores como parte de la cadena o las instrucciones.

11. El método de una cualquiera de las reivindicaciones 7-10, en donde cada una de las teselas de origen y destino especificadas comprenden M filas y N columnas de elementos y únicamente han de contener elementos válidos en una fila.

12. Almacenamiento legible por máquina que incluye instrucciones legibles por máquina que, cuando se ejecutan mediante un procesador de acuerdo con cualquiera de las reivindicaciones 1-6, ejecutan un método de acuerdo con cualquiera de las reivindicaciones 7-11.

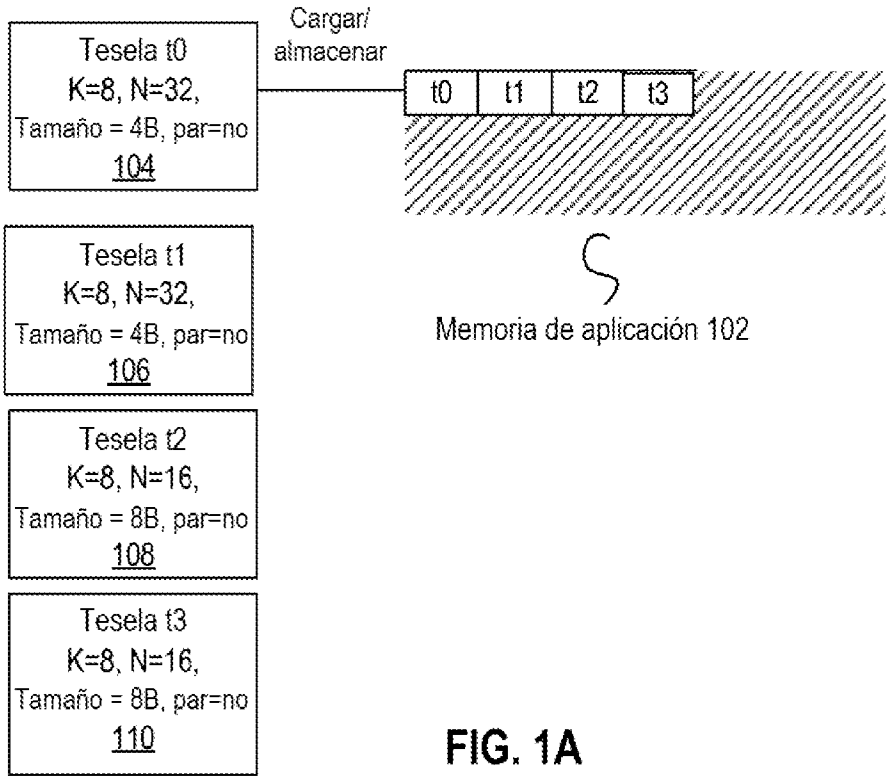

Tesela t0
K=8, N=32,
Tamaño = 4B, par=no
104
Cargar/
almacenar
t0
t1
t2
t3
Memoria de aplicación 102
Tesela t1
K=8, N=32,
Tamaño = 4B, par=no
106
Tesela t2
K=8, N=16,
Tamaño = 8B, par=no
108
Tesela t3
K=8, N=16,
Tamaño = 8B, par=no
110

FIG. 1A

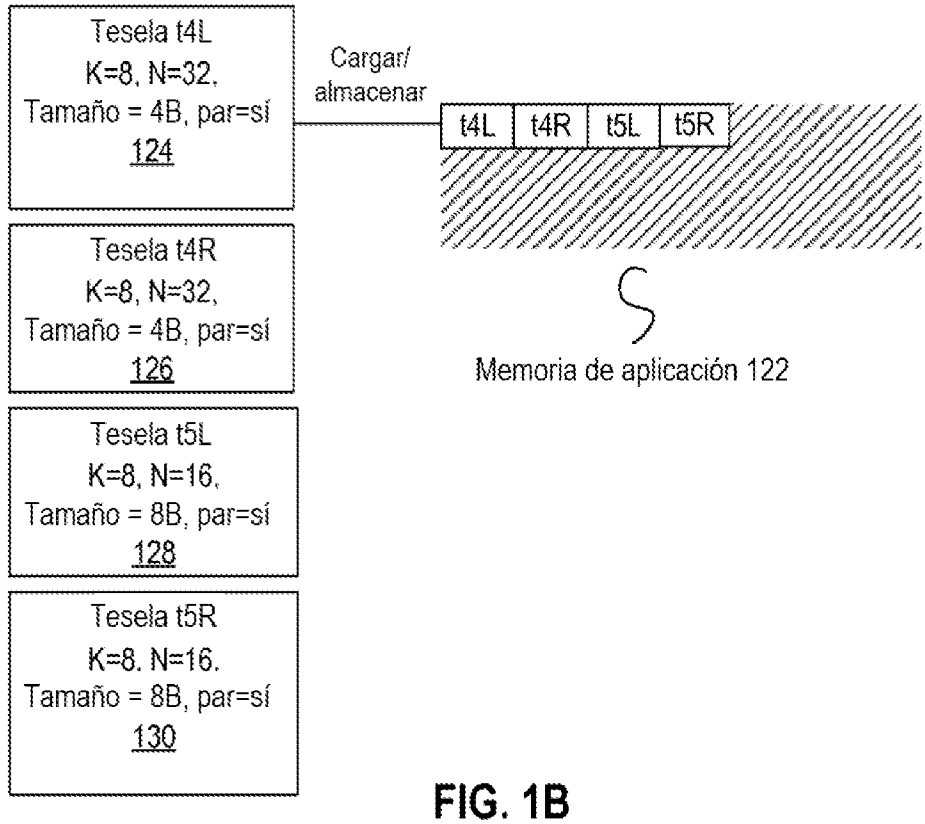

Tesela t4L
K=8, N=32,
Tamaño = 4B, par=sí
124
Cargar/
almacenar
t4L
t4R
t5L
t5R
Memoria de aplicación 122
Tesela t4R
K=8, N=32,
Tamaño = 4B, par=sí
126
Tesela t5L
K=8, N=16,
Tamaño = 8B, par=sí
128
Tesela t5R
K=8, N=16,
Tamaño = 8B, par=sí
130

FIG. 1B

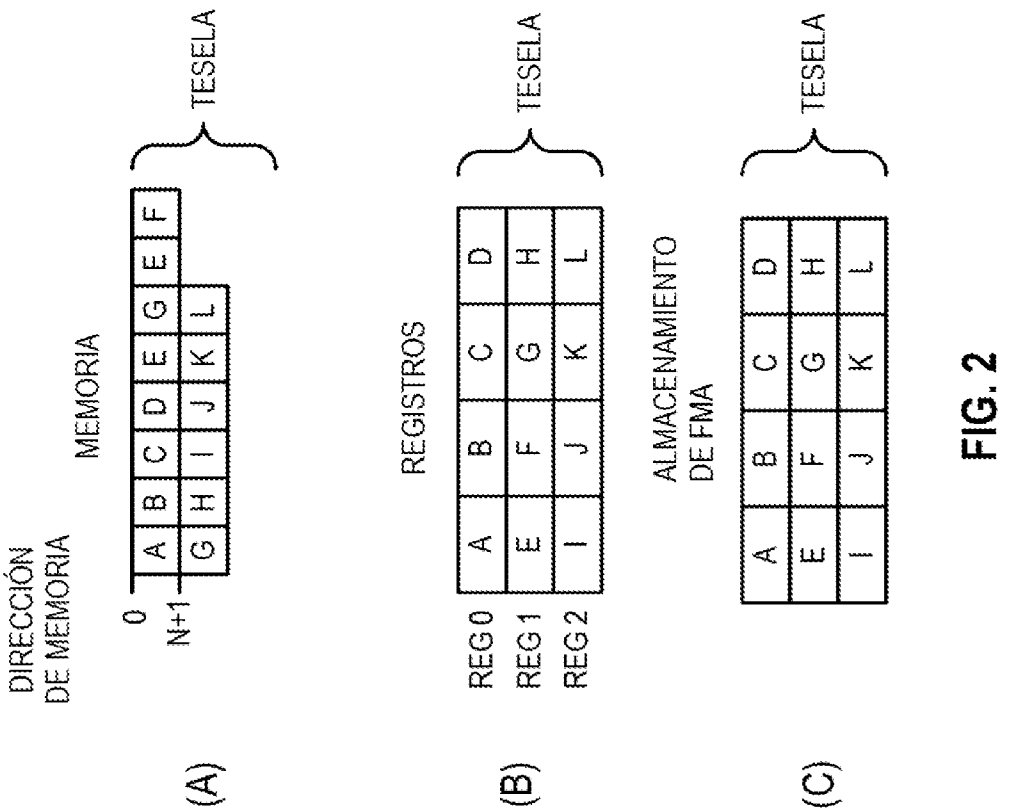

DIRECCIÓN DE MEMORIA
MEMORIA
0
N+1
A B C D E G E F
G H I J K L
TESELA
(A)
REGISTROS
REG 0
REG 1
REG 2
A B C D
E F G H
I J K L
TESELA
(B)
ALMACENAMIENTO DE FMA
A B C D
E F G H
I J K L
TESELA
(C)

FIG. 2

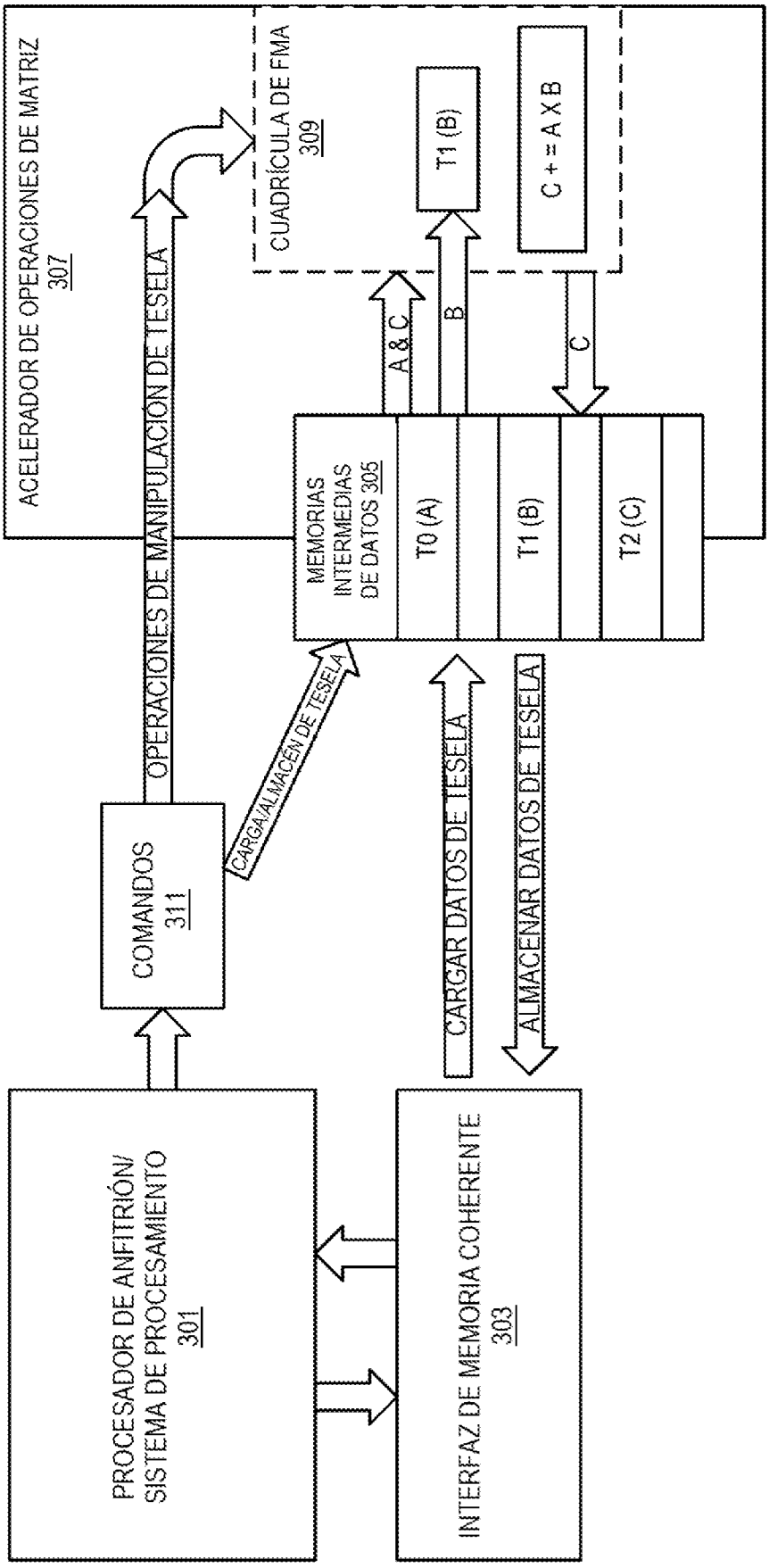

ACELERADOR DE OPERACIONES DE MATRIZ
307
CUADRÍCULA DE FMA
309
T1 (B)
C + = A X B
OPERACIONES DE MANIPULACIÓN DE TESELA
A & C
B
C
MEMORIAS INTERMEDIAS DE DATOS 305
T0 (A)
T1 (B)
T2 (C)
CARGA/ALMACÉN DE TESELA
COMANDOS
311
CARGAR DATOS DE TESELA
ALMACENAR DATOS DE TESELA
PROCESADOR DE ANFITRIÓN/ SISTEMA DE PROCESAMIENTO
301
INTERFAZ DE MEMORIA COHERENTE
303

FIG. 3

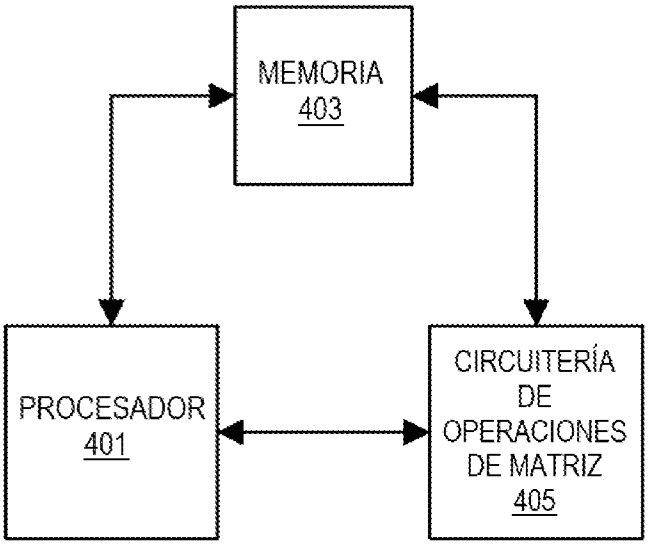

MEMORIA
403
PROCESADOR
401
CIRCUITERÍA
DE
OPERACIONES
DE MATRIZ
405

FIG. 4

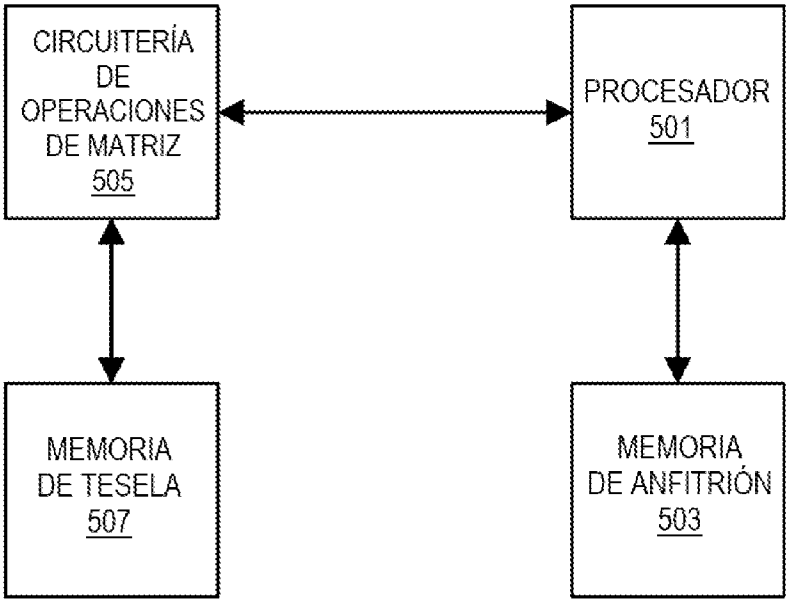

CIRCUITERÍA
DE
OPERACIONES
DE MATRIZ
505
PROCESADOR
501
MEMORIA
DE TESELA
507
MEMORIA
DE ANFITRIÓN
503

FIG. 5

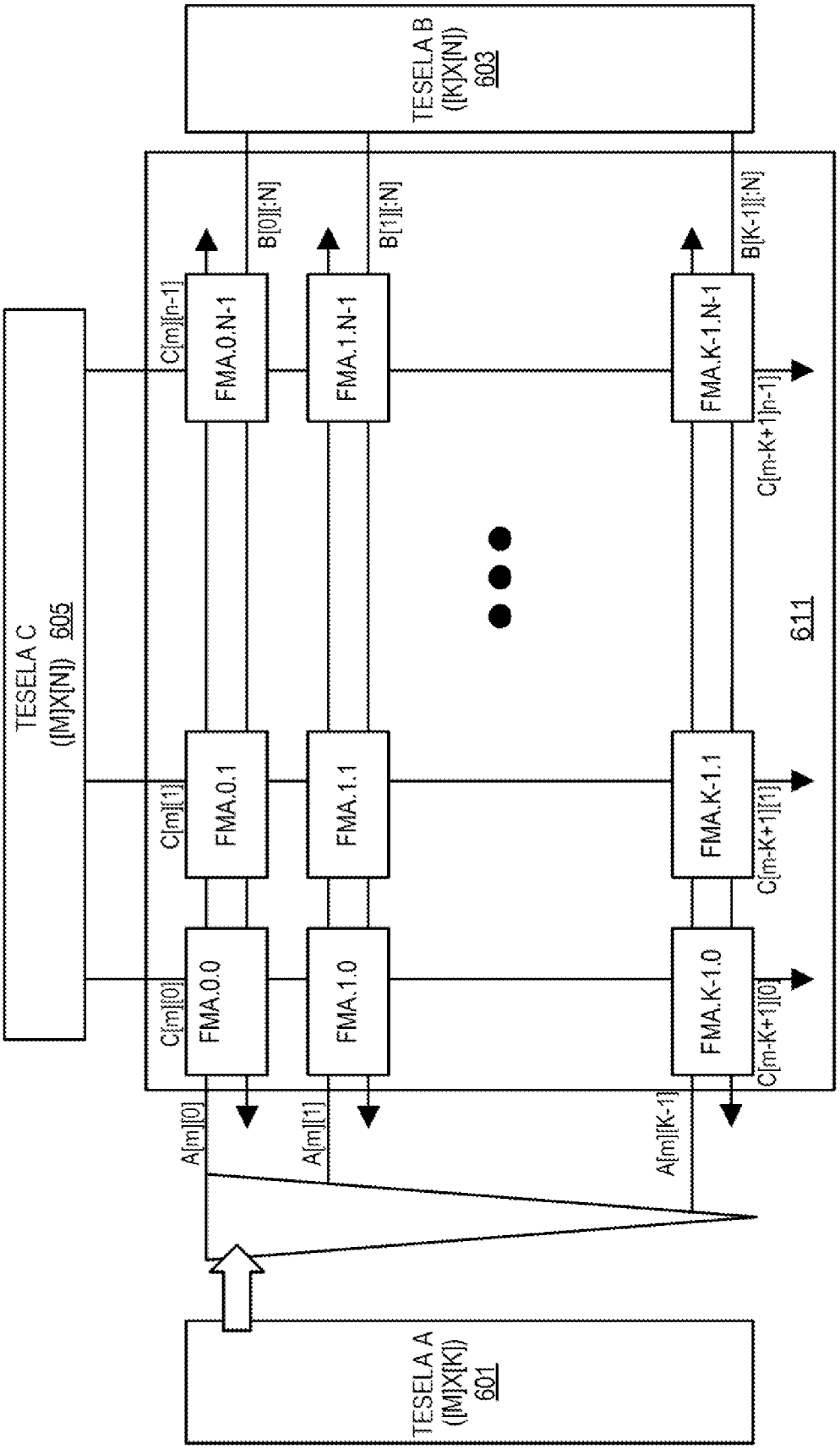

TESELA A
([M]X[K])
601
TESELA B
([K]X[N])
603
TESELA C
([M]X[N]) 605
611
A[m][0]
A[m][1]
A[m][K-1]
B[0][:N]
B[1][:N]
B[K-1][:N]
C[m][0]
C[m][1]
C[m][n-1]
FMA.0.0
FMA.1.0
FMA.K-1.0
FMA.0.1
FMA.1.1
FMA.K-1.1
FMA.0.N-1
FMA.1.N-1
FMA.K-1.N-1
C[m-K+1][0]
C[m-K+1][1]
C[m-K+1][n-1]

FIG. 6

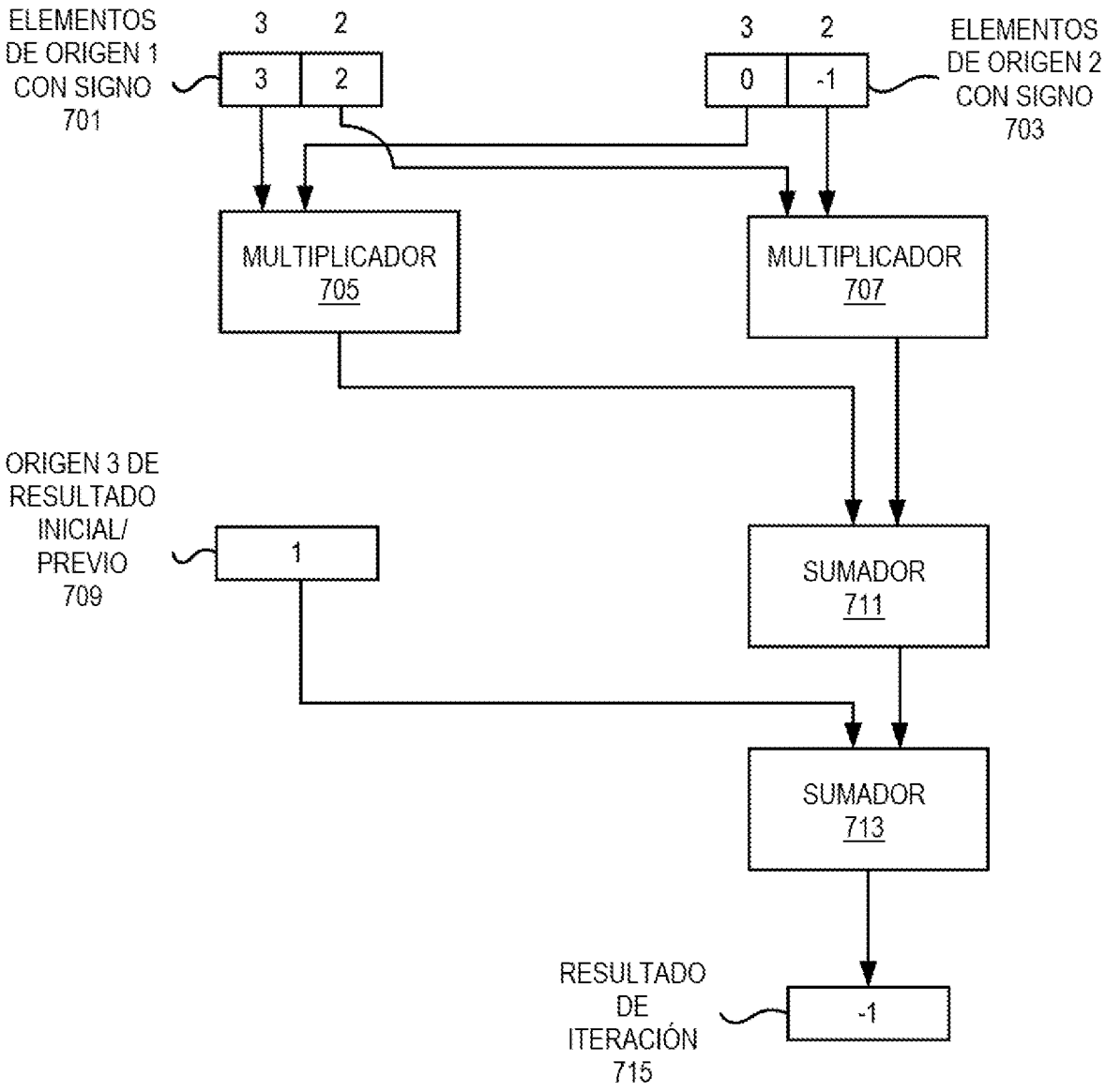

ELEMENTOS DE ORIGEN 1 CON SIGNO 701
3
2
3
2
3
2
0
-1
ELEMENTOS DE ORIGEN 2 CON SIGNO 703
MULTIPLICADOR 705
MULTIPLICADOR 707
ORIGEN 3 DE RESULTADO INICIAL/ PREVIO 709
1
SUMADOR 711
SUMADOR 713
RESULTADO DE ITERACIÓN 715
-1

FIG. 7

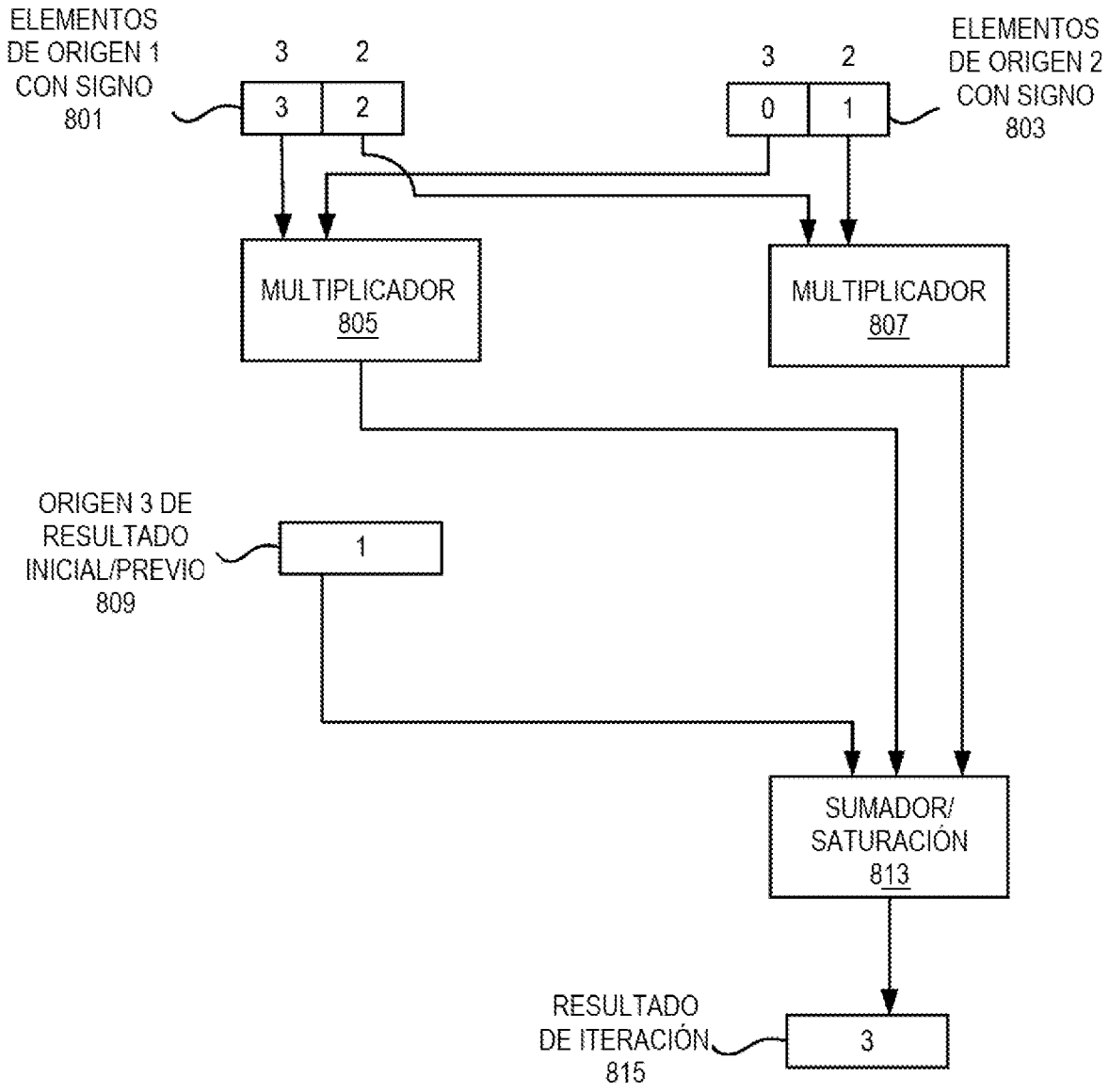

ELEMENTOS DE ORIGEN 1 CON SIGNO 801
3
2
3
2
3
2
0
1
ELEMENTOS DE ORIGEN 2 CON SIGNO 803
MULTIPLICADOR 805
MULTIPLICADOR 807
ORIGEN 3 DE RESULTADO INICIAL/PREVIO 809
1
SUMADOR/ SATURACIÓN 813
RESULTADO DE ITERACIÓN 815
3

FIG. 8

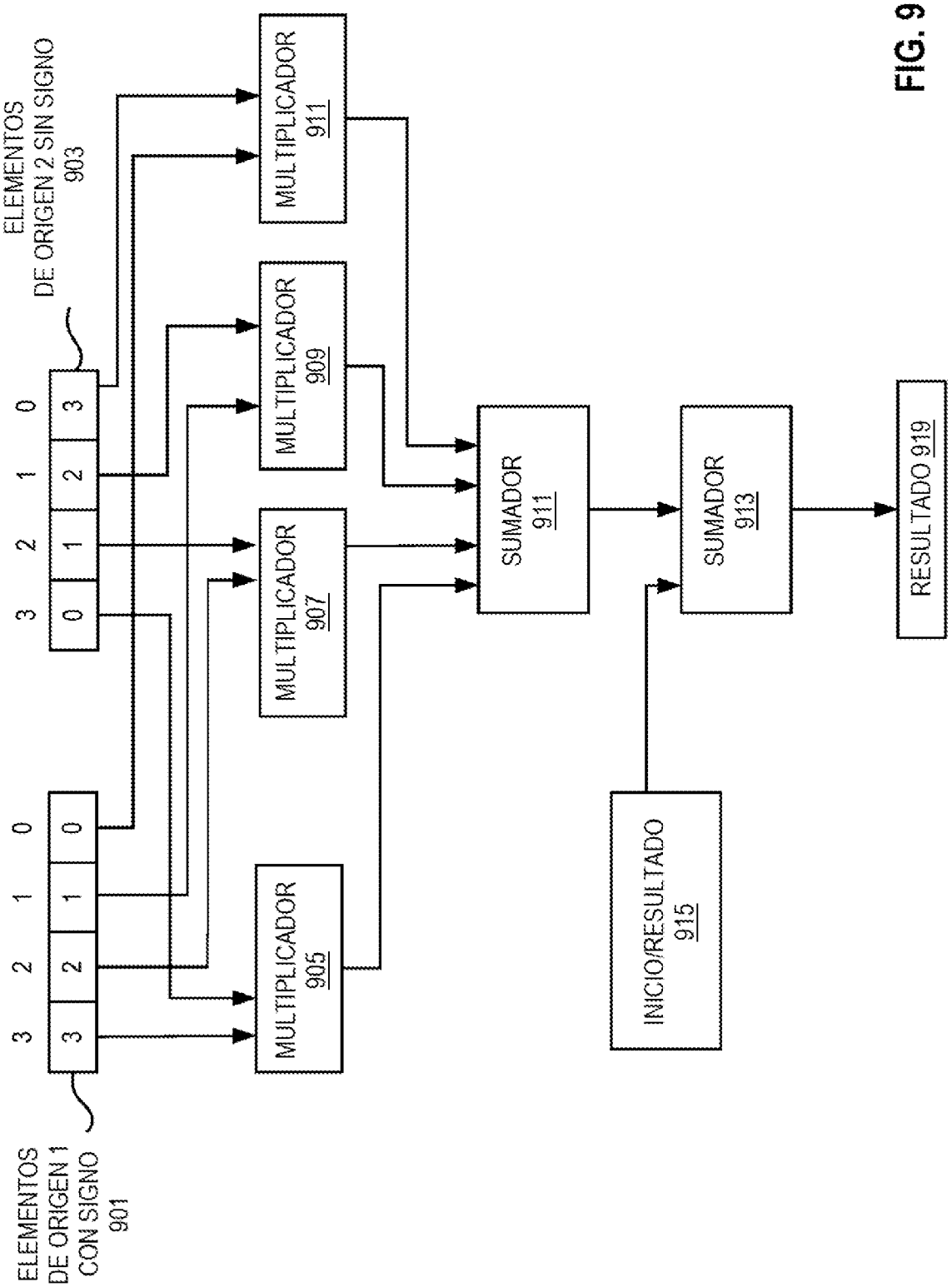

ELEMENTOS DE ORIGEN 1 CON SIGNO 901
3 2 1 0
3 2 1 0
ELEMENTOS DE ORIGEN 2 SIN SIGNO 903
3 2 1 0
0 1 2 3
MULTIPLICADOR 905
MULTIPLICADOR 907
MULTIPLICADOR 909
MULTIPLICADOR 911
SUMADOR 911
SUMADOR 913
INICIO/RESULTADO 915
RESULTADO 919

FIG. 9

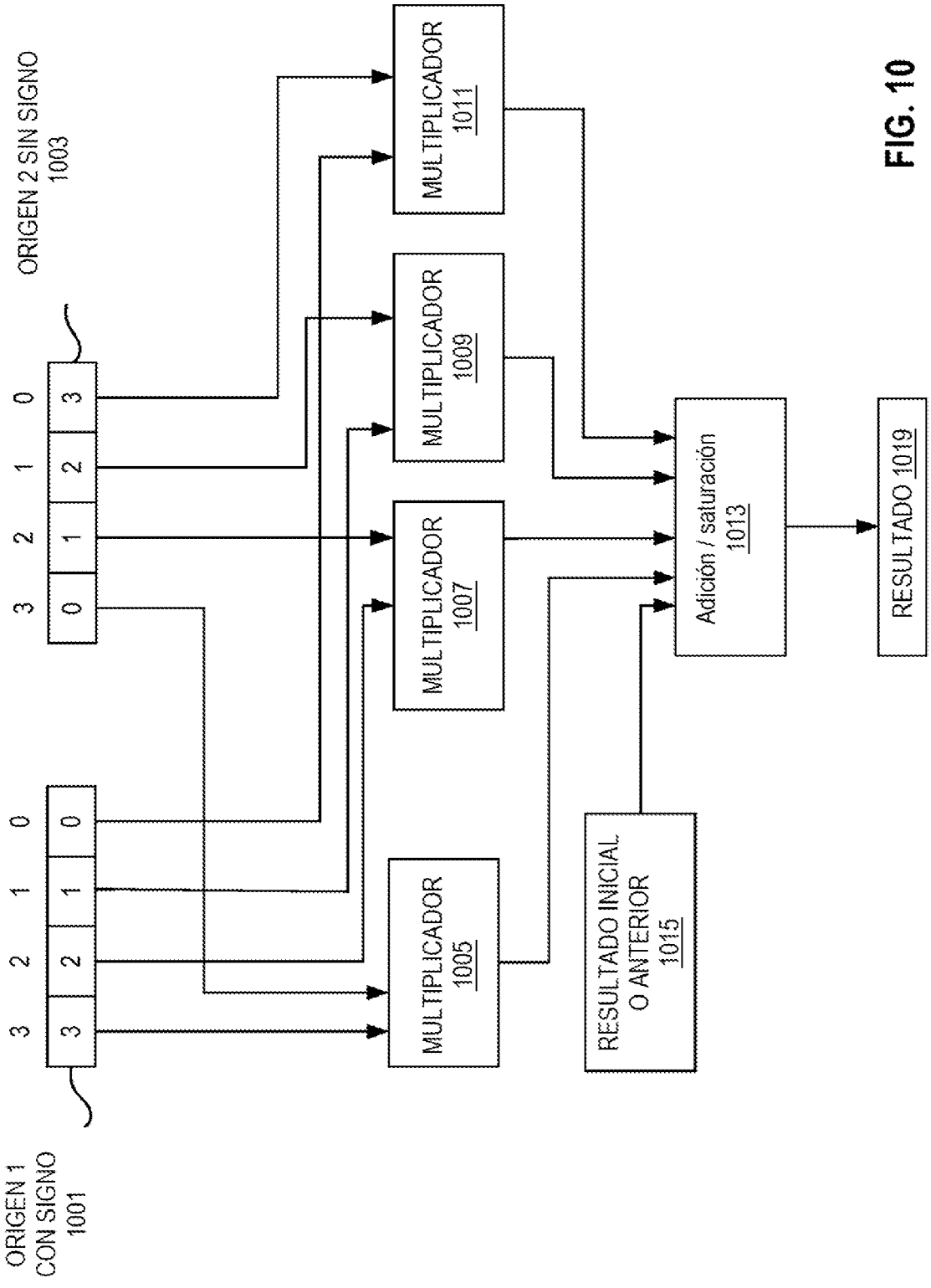

ORIGEN 1 CON SIGNO 1001
ORIGEN 2 SIN SIGNO 1003
3 2 1 0
3 2 1 0
MULTIPLICADOR 1005
MULTIPLICADOR 1007
MULTIPLICADOR 1009
MULTIPLICADOR 1011
RESULTADO INICIAL O ANTERIOR 1015
Adición / saturación 1013
RESULTADO 1019

FIG. 10

TAMAÑOS DE ENTRADA DE ACUMULADOR 2X 1101

ORÍGENES	BITS	ACUMULADOR	BITS
BYTE	8	PALABRA/HPFP	16
PALABRA	16	INT32/SPFP	32
SPFP/INT32	32	INT64/DPFP	64

TAMAÑOS DE ENTRADA DE ACUMULADOR 4X 11013

ORÍGENES	BITS	ACUMULADOR	BITS
BYTE	8	INT32/SPFP	32
PALABRA	16	INT64/DPFP	64

TAMAÑOS DE ENTRADA DE ACUMULADOR 8X 1105

ORÍGENES	BITS	ACUMULADOR	BITS
BYTE	8	INT64/DPFP	64

FIG. 11

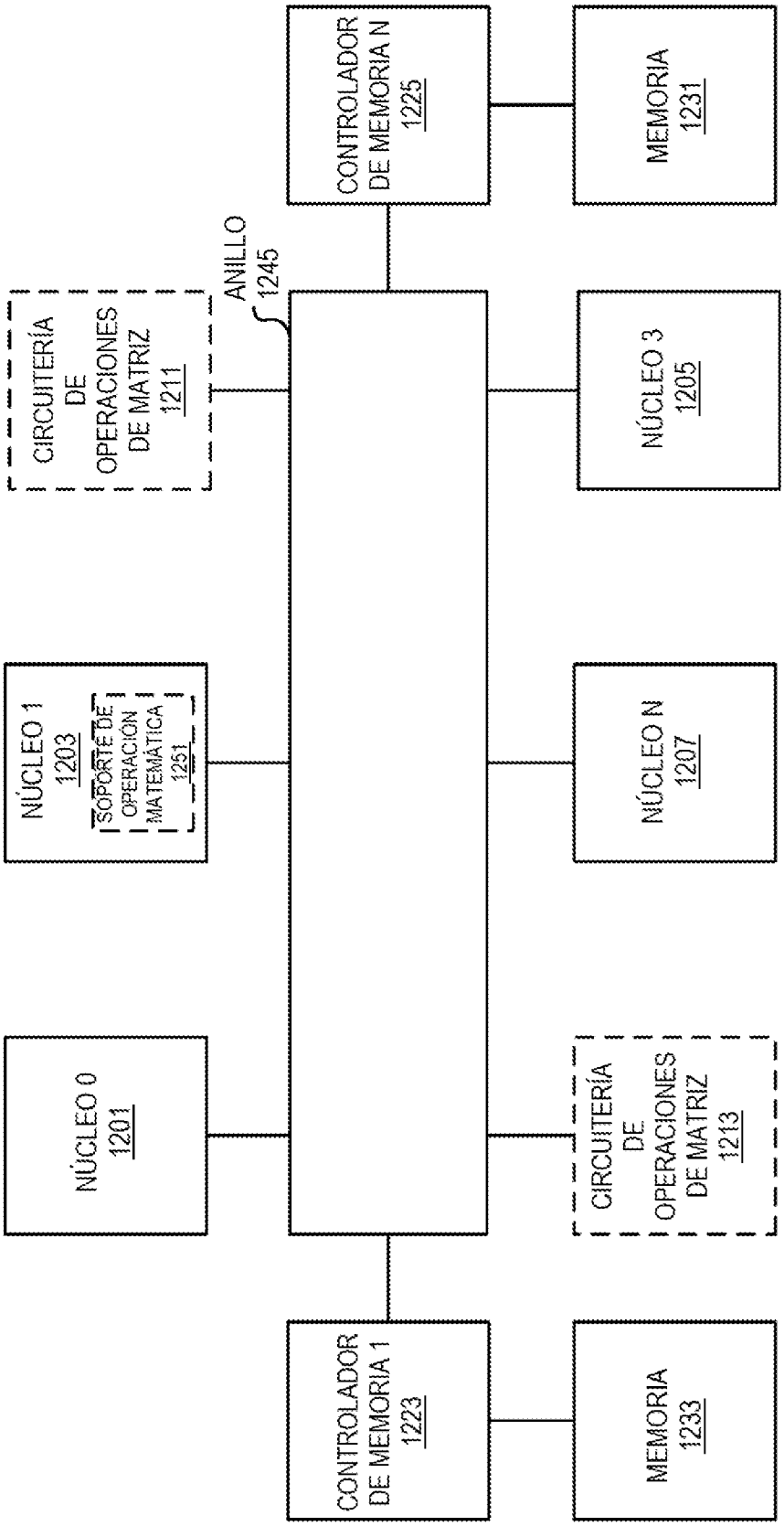

CONTROLADOR DE MEMORIA N 1225
MEMORIA 1231
ANILLO 1245
CIRCUITERÍA DE OPERACIONES DE MATRIZ 1211
NÚCLEO 3 1205
NÚCLEO 1 1203
SOPORTE DE OPERACIÓN MATEMÁTICA 1251
NÚCLEO N 1207
NÚCLEO 0 1201
CIRCUITERÍA DE OPERACIONES DE MATRIZ 1213
CONTROLADOR DE MEMORIA 1 1223
MEMORIA 1233

FIG. 12

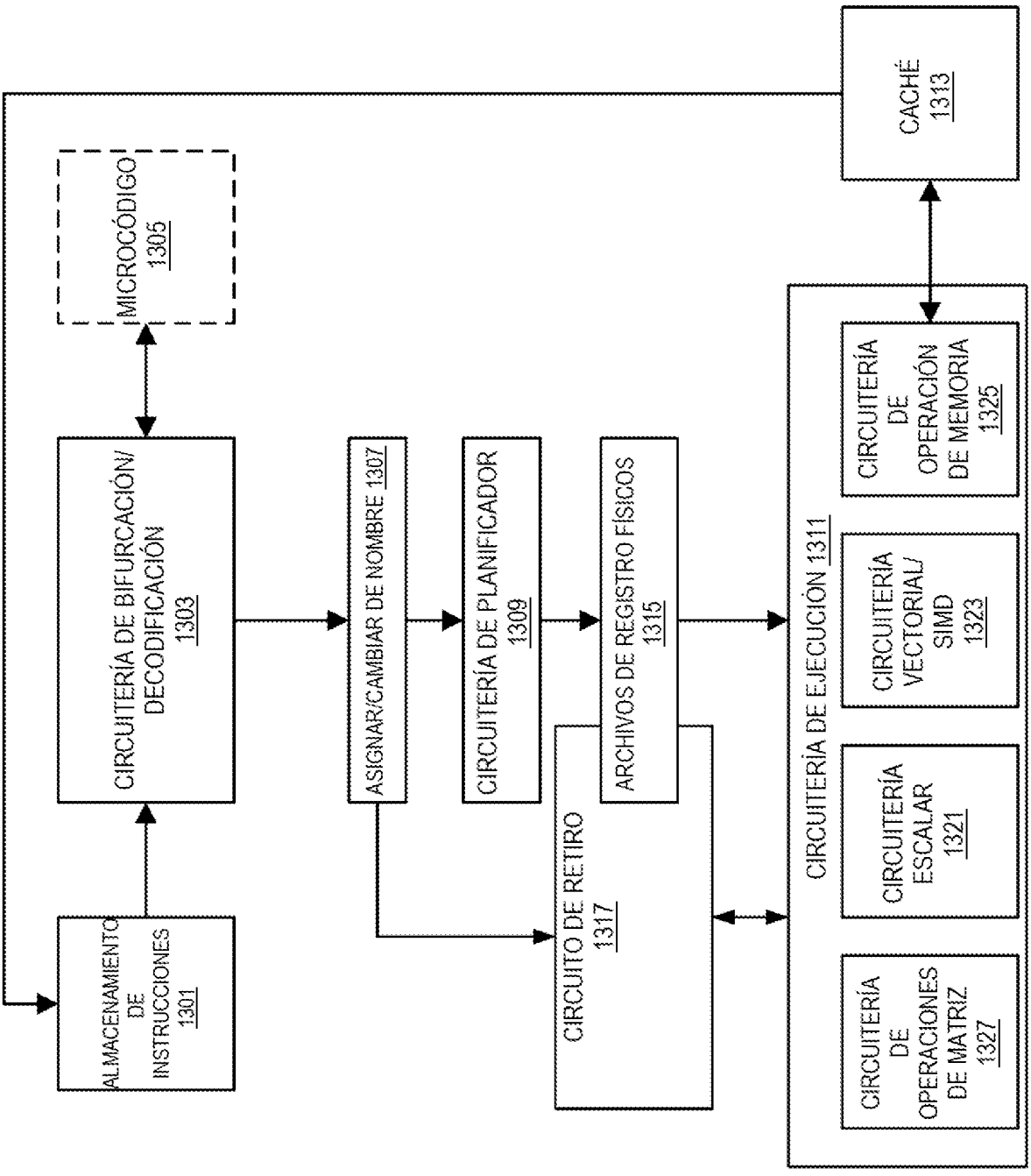
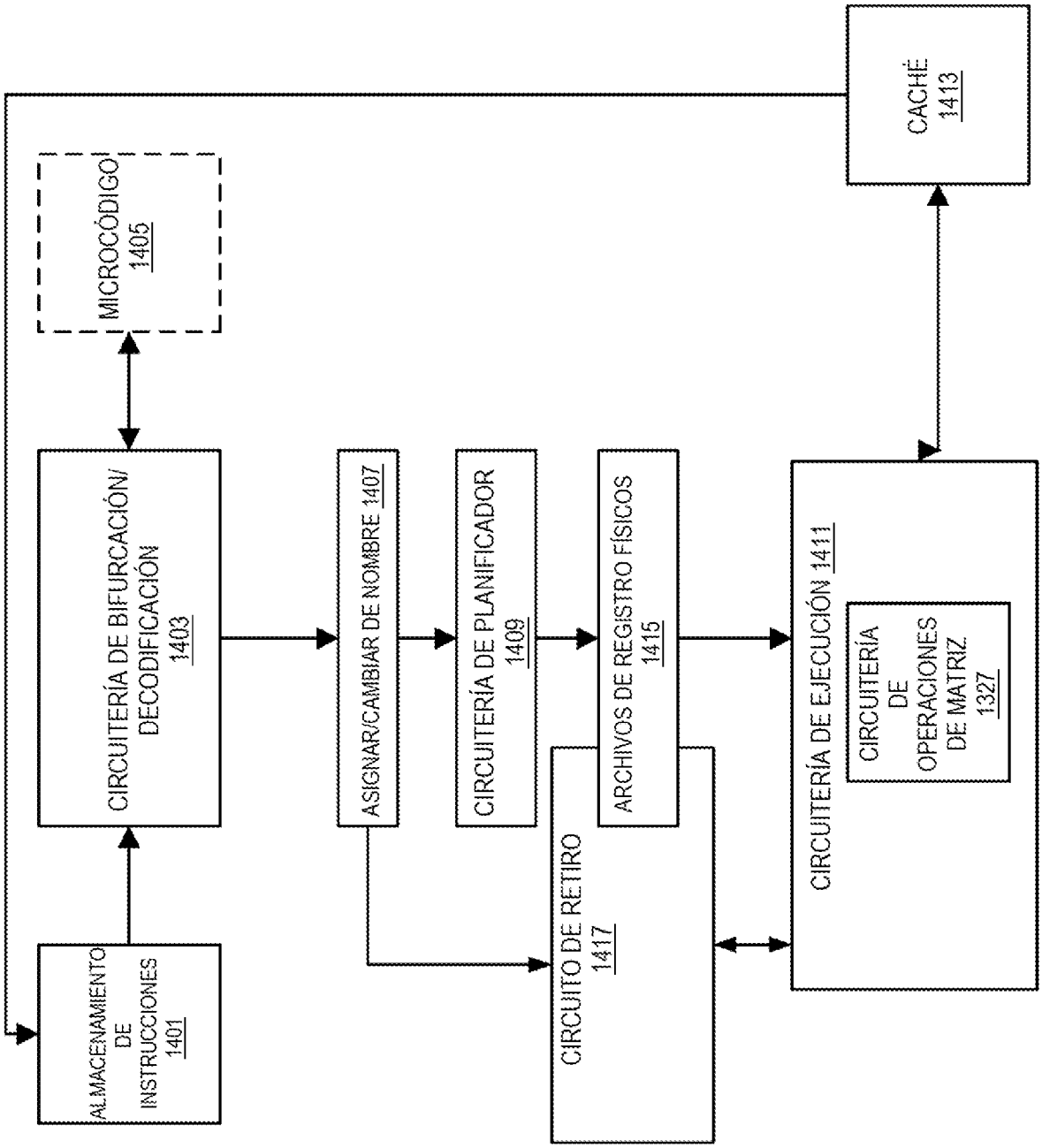

ALMACENAMIENTO DE INSTRUCCIONES 1301
CIRCUITERÍA DE BIFURCACIÓN/ DECODIFICACIÓN 1303
MICROCÓDIGO 1305
ASIGNAR/CAMBIAR DE NOMBRE 1307
CIRCUITERÍA DE PLANIFICADOR 1309
ARCHIVOS DE REGISTRO FÍSICOS 1315
CIRCUITO DE RETIRO 1317
CACHÉ 1313
CIRCUITERÍA DE EJECUCIÓN 1311
CIRCUITERÍA DE OPERACIÓN DE MEMORIA 1325
CIRCUITERÍA VECTORIAL/ SIMD 1323
CIRCUITERÍA ESCALAR 1321
CIRCUITERÍA DE OPERACIONES DE MATRIZ 1327

FIG. 13

FIG. 14


ALMACENAMIENTO DE INSTRUCCIONES 1401
MICROCÓDIGO 1405
CACHÉ 1413
CIRCUITERÍA DE BIFURCACIÓN/ DECODIFICACIÓN 1403
ASIGNAR/CAMBIAR DE NOMBRE 1407
CIRCUITERÍA DE PLANIFICADOR 1409
ARCHIVOS DE REGISTRO FÍSICOS 1415
CIRCUITO DE RETIRO 1417
CIRCUITERÍA DE EJECUCIÓN 1411
CIRCUITERÍA DE OPERACIONES DE MATRIZ 1327

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix}$$

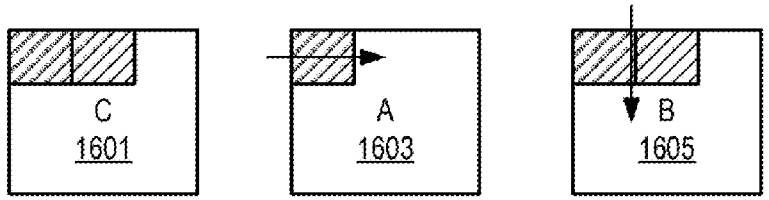
DIRECCIÓN	VALOR
0	A_{11}
1	A_{12}
2	A_{13}
3	A_{21}
4	A_{22}
5	A_{23}

FILA PRINCIPAL

DIRECCIÓN	VALOR
0	A_{11}
1	A_{21}
2	A_{12}
3	A_{22}
4	A_{13}
5	A_{23}

COLUMNA PRINCIPAL

FIG. 15




```
TILECONFIG [RAX]
//SE SUPONE ALGUNOS BUCLES EXTERIORES CONTROLANDO EL TESELADO DE CACHÉ (NO MOSTRADO)
{
TILELOAD TMM0, RSI+RDI // SRCDST, RSI APUNTA A C, RDI TIENE
TILELOAD TMM1, RSI+RDI+N // SEGUNDA TESELA DE C, DESENROLLÁNDOSE EN LA DIMENSIÓN DE SIMD N
MOV KK, 0
LOOP:
TILELOAD TMM2, R8+R9 // SRC2 SE ENCUENTRA EN CARGA DE PASO DE A, REUTILIZADA PARA INSTRUCCIÓN 2 TMMA.
TILELOAD TMM3, R10+R11 // SRC1 SE ENCUENTRA EN CARGA DE PASO DE B
TMMAPS TMM0, TMM2, TMM3 // ACTUALIZAR TESELA IZQUIERDA DE C
TILELOAD TMM3, R10+R11+N // SRC1 CARGADO CON B DESDE LA SIGUIENTE TESELA MÁS A LA DERECHA
TMMAPS TMM1, TMM2, TMM3 // ACTUALIZAR TESELA DERECHA DE C
ADD R8, K //ACTUALIZAR PUNTEROS POR CONSTANTES CONOCIDAS FUERA DE BUCLE
ADD R10, K*R11
ADD KK, K
CMP KK, LIMIT
JNE LOOP
TILESTORE RSI+RDI, TMM0 // ACTUALIZAR LA MATRIZ C EN MEMORIA
TILESTORE RSI+RDI+M, TMM1
} // FIN DE BUCLE EXTERNO
TILERELEASE // DEVOLVER TESELAS A ESTADO INICIAL
```

FIG. 16

ES 2 988 858 T3

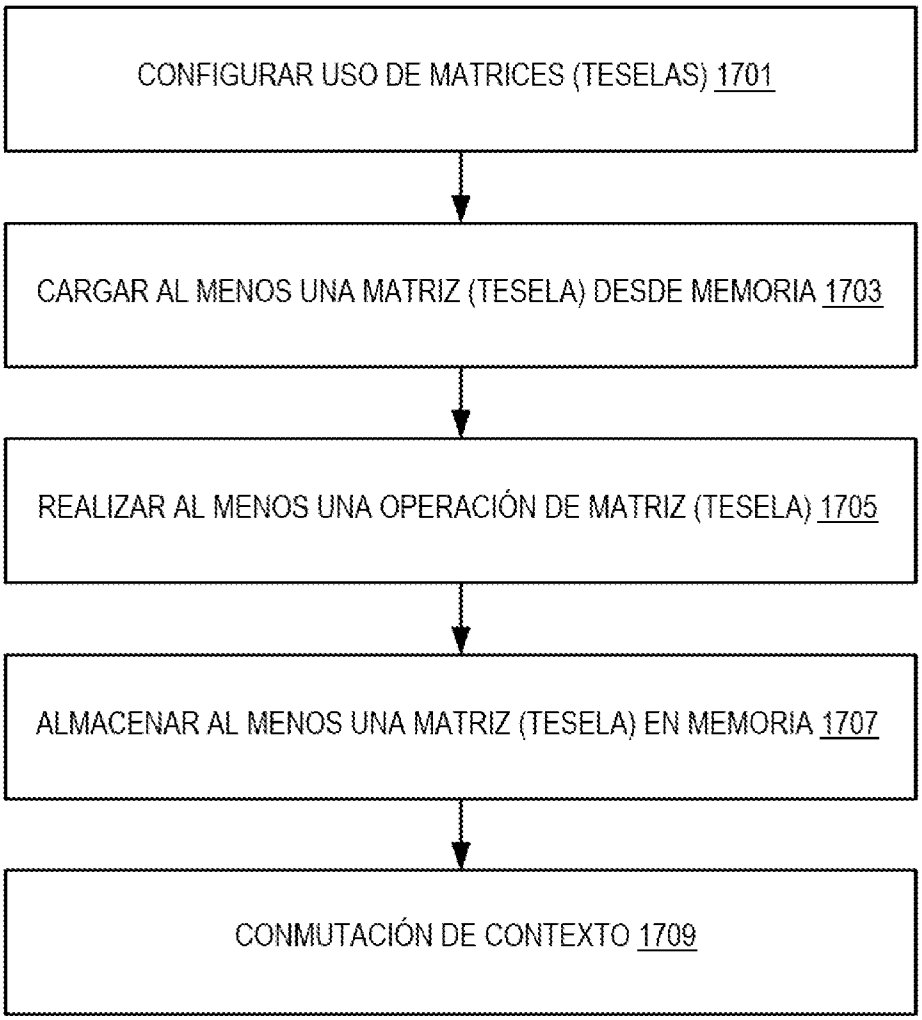

CONFIGURAR USO DE MATRICES (TESELAS) 1701
CARGAR AL MENOS UNA MATRIZ (TESELA) DESDE MEMORIA 1703
REALIZAR AL MENOS UNA OPERACIÓN DE MATRIZ (TESELA) 1705
ALMACENAR AL MENOS UNA MATRIZ (TESELA) EN MEMORIA 1707
CONMUTACIÓN DE CONTEXTO 1709

FIG. 17

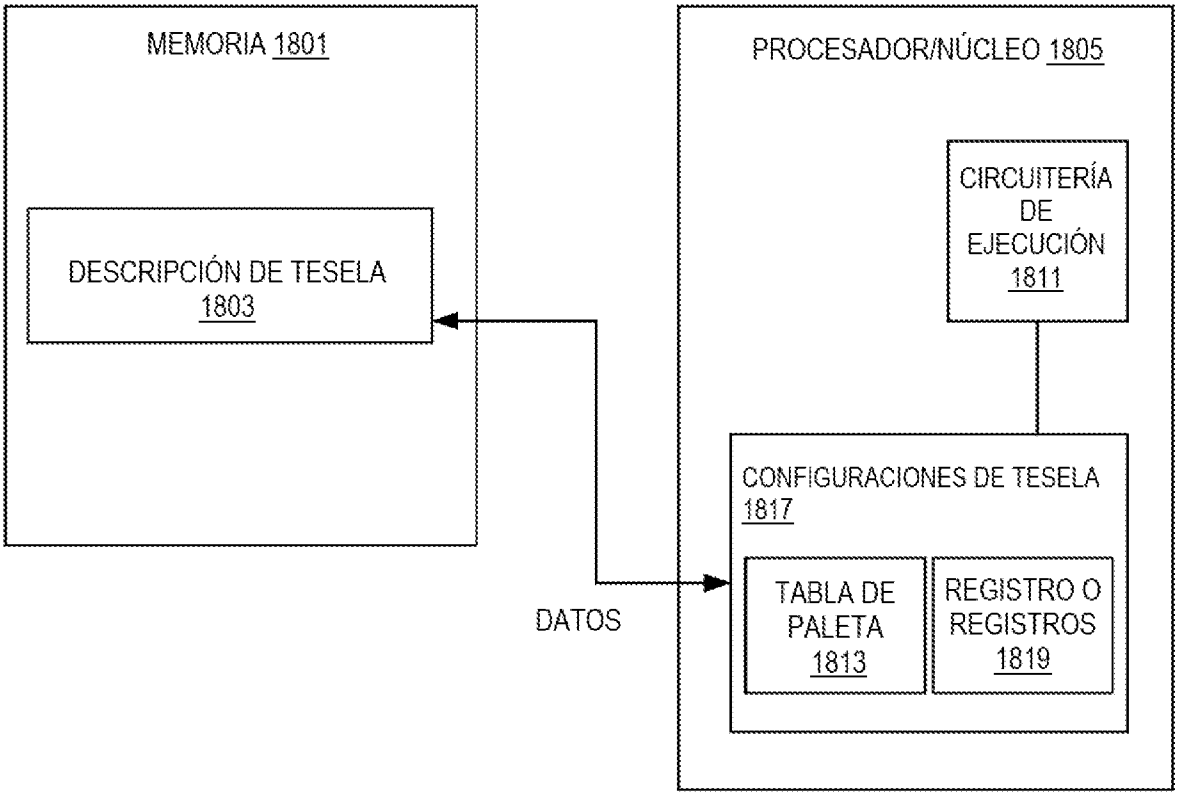

MEMORIA 1801
DESCRIPCIÓN DE TESELA 1803
DATOS
PROCESADOR/NÚCLEO 1805
CIRCUITERÍA DE EJECUCIÓN 1811
CONFIGURACIONES DE TESELA 1817
TABLA DE PALETA 1813
REGISTRO O REGISTROS 1819

FIG. 18

ID DE PALETA 1901	STARTM 1903
STARTP 1905	INDICADORES DE PAR 1907
0	0
0	0

. . .

0	0
TMM0 FILAS 1913	TMM0 COLUMNAS 1915
TMM1 FILAS	TMM1 COLUMNAS
▪ ▪ ▪	
TMM15 FILAS	TMM15 COLUMNAS
0	

FIG. 19

FIG. 20(A)

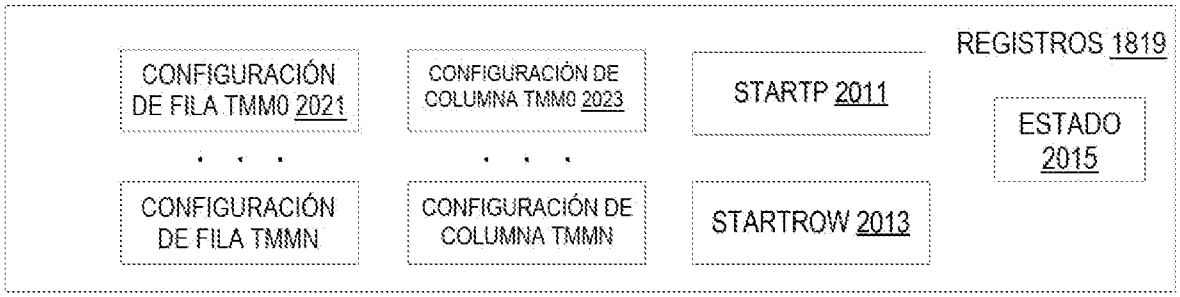


FIG. 20(B)

FIG. 20(C)

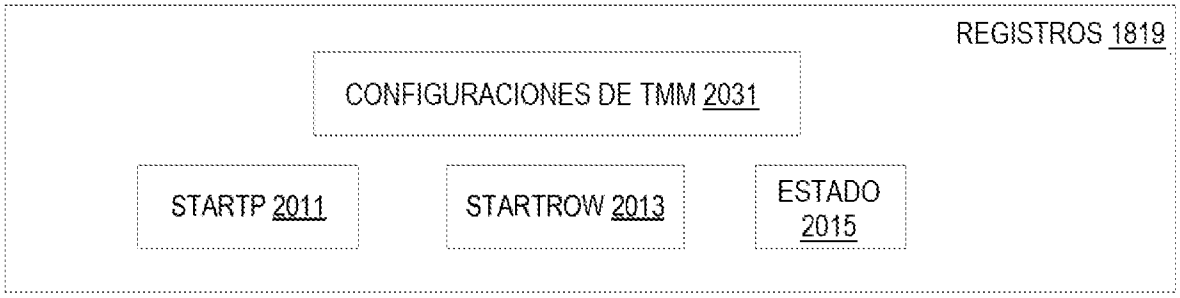


FIG. 20(D)

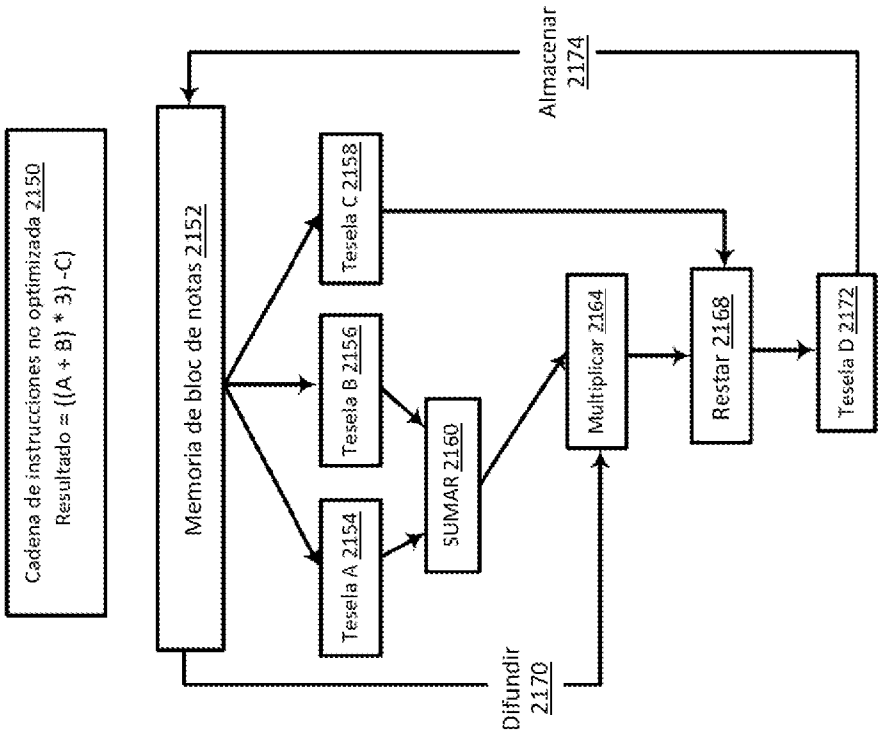


FIG. 21B

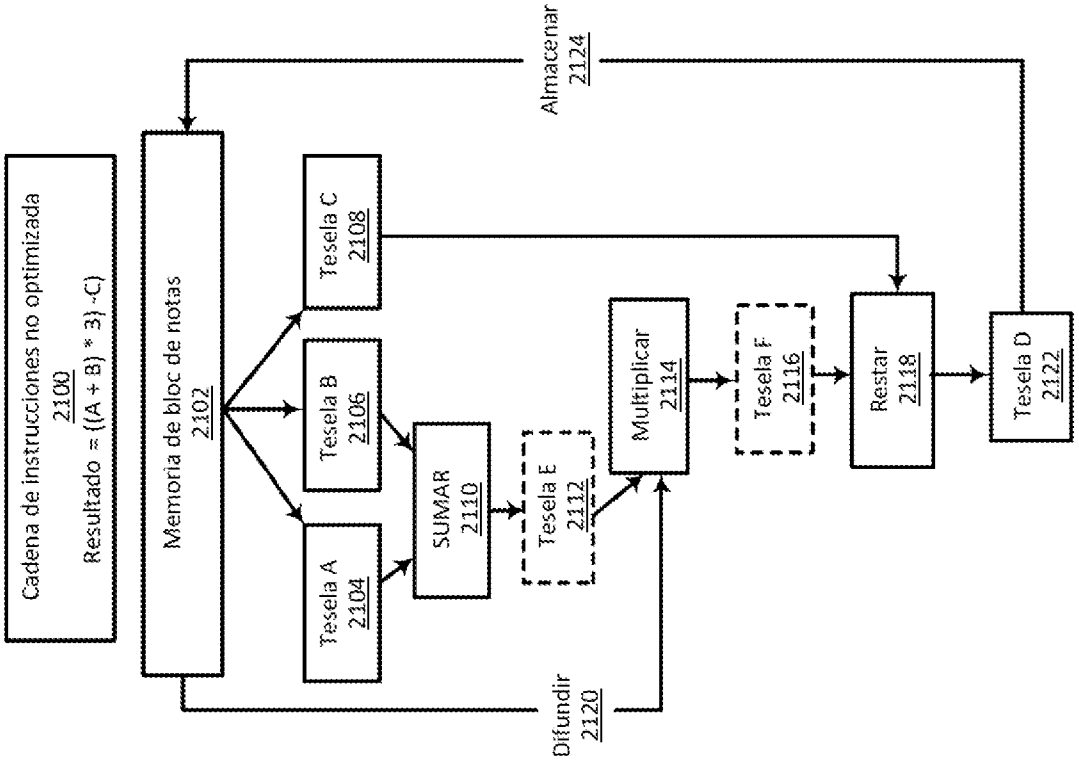


FIG. 21A

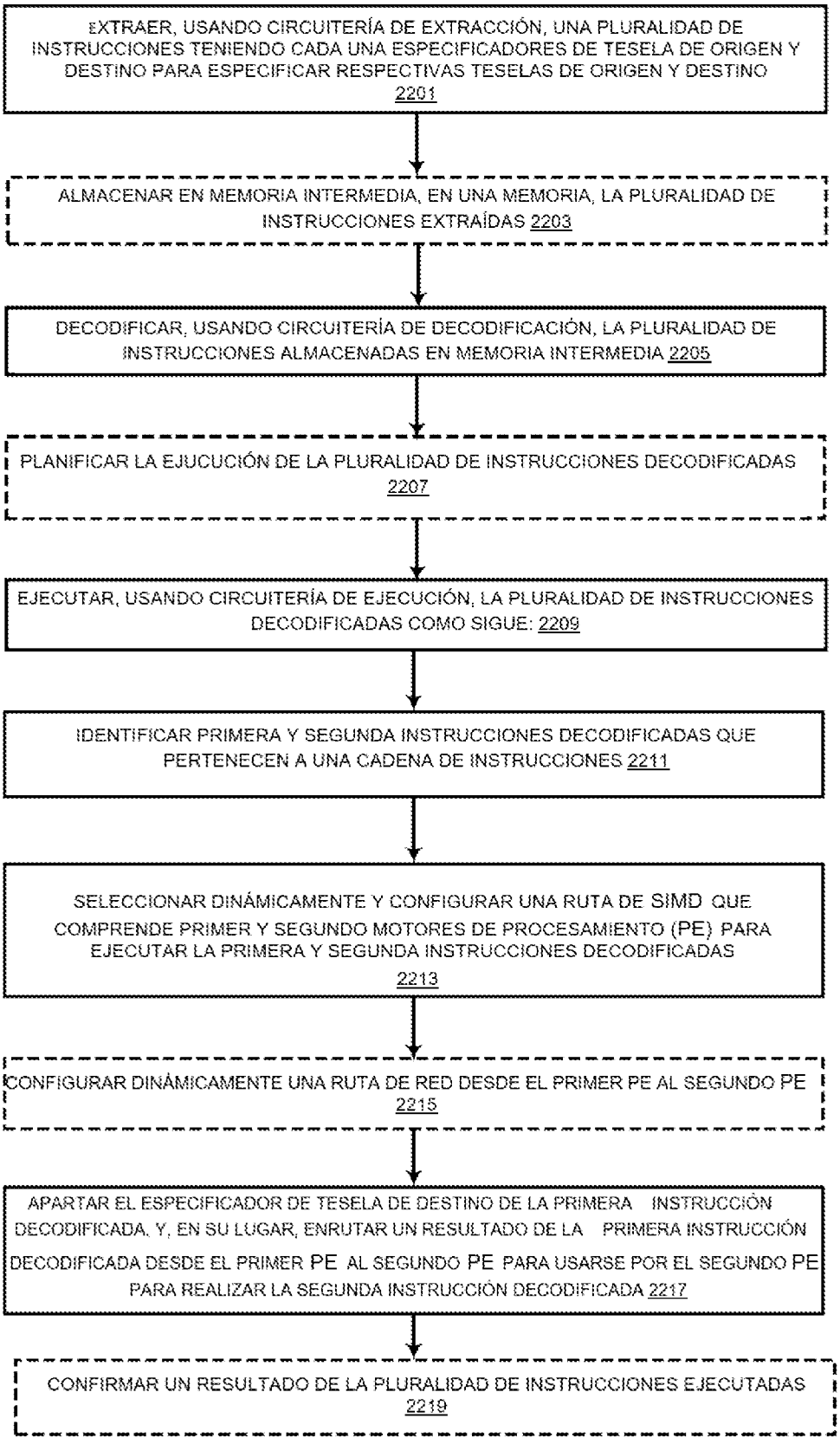

EXTRAER, USANDO CIRCUITERÍA DE EXTRACCIÓN, UNA PLURALIDAD DE INSTRUCCIONES TENIENDO CADA UNA ESPECIFICADORES DE TESELA DE ORIGEN Y DESTINO PARA ESPECIFICAR RESPECTIVAS TESELAS DE ORIGEN Y DESTINO 2201
ALMACENAR EN MEMORIA INTERMEDIA, EN UNA MEMORIA, LA PLURALIDAD DE INSTRUCCIONES EXTRAÍDAS 2203
DECODIFICAR, USANDO CIRCUITERÍA DE DECODIFICACIÓN, LA PLURALIDAD DE INSTRUCCIONES ALMACENADAS EN MEMORIA INTERMEDIA 2205
PLANIFICAR LA EJUCUCIÓN DE LA PLURALIDAD DE INSTRUCCIONES DECODIFICADAS 2207
EJECUTAR, USANDO CIRCUITERÍA DE EJECUCIÓN, LA PLURALIDAD DE INSTRUCCIONES DECODIFICADAS COMO SIGUE: 2209
IDENTIFICAR PRIMERA Y SEGUNDA INSTRUCCIONES DECODIFICADAS QUE PERTENECEN A UNA CADENA DE INSTRUCCIONES 2211
SELECCIONAR DINÁMICAMENTE Y CONFIGURAR UNA RUTA DE SIMD QUE COMPRENDE PRIMER Y SEGUNDO MOTORES DE PROCESAMIENTO (PE) PARA EJECUTAR LA PRIMERA Y SEGUNDA INSTRUCCIONES DECODIFICADAS 2213
CONFIGURAR DINÁMICAMENTE UNA RUTA DE RED DESDE EL PRIMER PE AL SEGUNDO PE 2215
APARTAR EL ESPECIFICADOR DE TESELA DE DESTINO DE LA PRIMERA INSTRUCCIÓN DECODIFICADA, Y, EN SU LUGAR, ENRUTAR UN RESULTADO DE LA PRIMERA INSTRUCCIÓN DECODIFICADA DESDE EL PRIMER PE AL SEGUNDO PE PARA USARSE POR EL SEGUNDO PE PARA REALIZAR LA SEGUNDA INSTRUCCIÓN DECODIFICADA 2217
CONFIRMAR UN RESULTADO DE LA PLURALIDAD DE INSTRUCCIONES EJECUTADAS 2219

FIG. 22

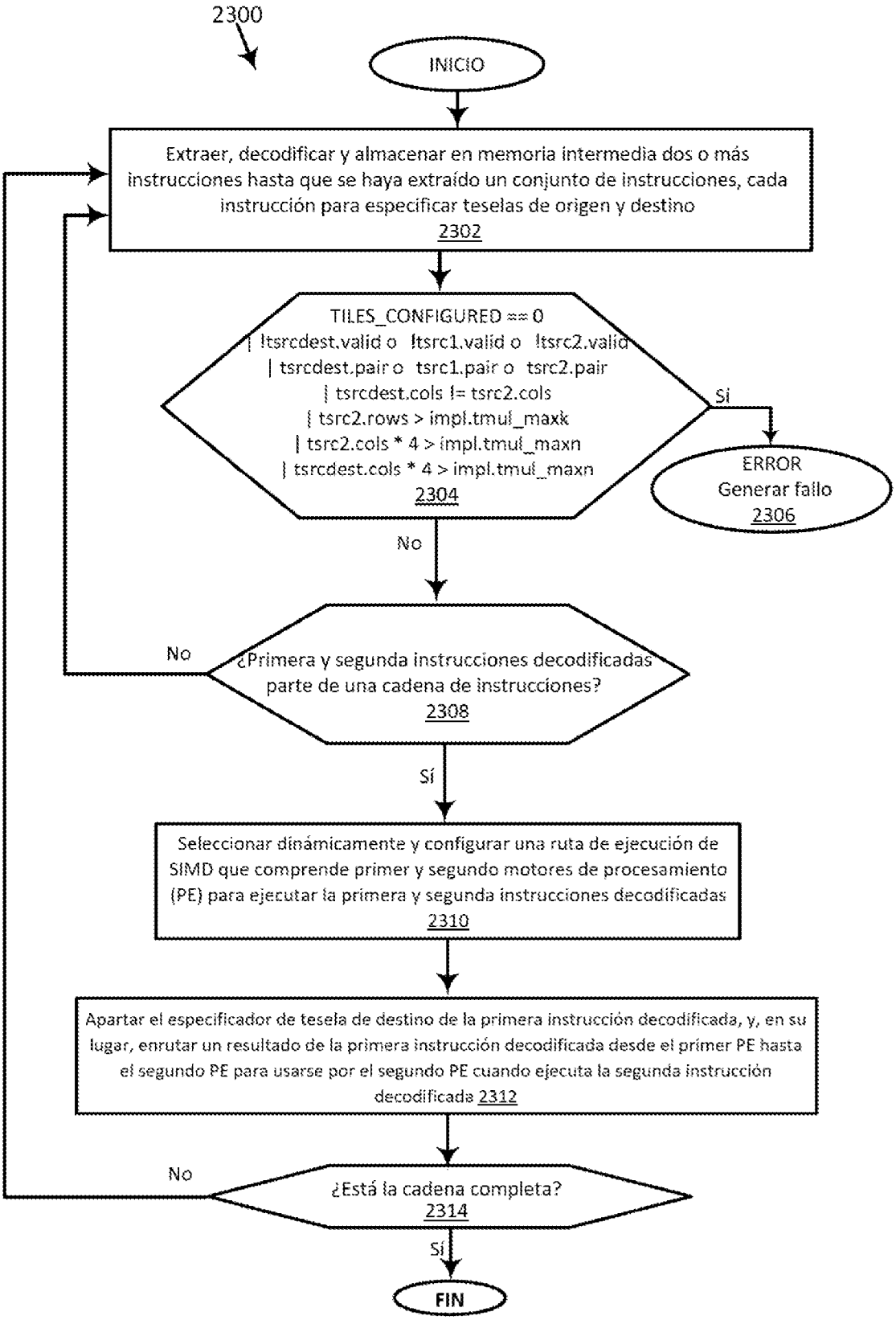

2300
INICIO
Extraer, decodificar y almacenar en memoria intermedia dos o más instrucciones hasta que se haya extraído un conjunto de instrucciones, cada instrucción para especificar teselas de origen y destino
2302
TILES_CONFIGURED == 0
| !tsrcdest.valid o !tsrc1.valid o !tsrc2.valid
| tsrcdest.pair o tsrc1.pair o tsrc2.pair
| tsrcdest.cols != tsrc2.cols
| tsrc2.rows > impl.tmul_maxk
| tsrc2.cols * 4 > impl.tmul_maxn
| tsrcdest.cols * 4 > impl.tmul_maxn
2304
Sí
ERROR
Generar fallo
2306
No
No
¿Primera y segunda instrucciones decodificadas parte de una cadena de instrucciones?
2308
Sí
Seleccionar dinámicamente y configurar una ruta de ejecución de SIMD que comprende primer y segundo motores de procesamiento (PE) para ejecutar la primera y segunda instrucciones decodificadas
2310
Apartar el especificador de tesela de destino de la primera instrucción decodificada, y, en su lugar, enrutar un resultado de la primera instrucción decodificada desde el primer PE hasta el segundo PE para usarse por el segundo PE cuando ejecuta la segunda instrucción decodificada 2312
No
¿Está la cadena completa?
2314
Sí
FIN

FIG. 23

Bucle SIMD en código nativo 2402

```
for (m=0; m<max_M; m++) {
      for (n=0; n<max_N; n++) {
            A[m][n] = ((B[m][n] + C[m][n]) << 2) + k;
      }
}
```

FIG. 24A

Bucle de código de SIMD no optimizado 2404

```
{
 tileload tmm0, [rax]          // cargar siguiente segmento de B
 tileload tmm1, [rbx]          // cargar siguiente segmento de C
 vaddps tmm2, tmm0, tmm1       // añadir segmento B a segmento C
 tilestore [rcx], tmm2;        // * almacenar resultado en A
tileload (tmm0, [rcx]          // * cargar A
 vshl tmm2, tmm2, 2;           // desplazar segmento A por valor inmediato de 2
 tilestore [rcx], tmm2;        // * almacenar resultado en A
tileload (tmm0, [rcx];         // * cargar A
 vaddps tmm2, tmm2, k;         // añadir valor inmediato de k a A
 tilestore [rcx], tmm2;        // almacenar segmento de resultados en A
}
```

FIG. 24B

Bucle de código de SIMD optimizado 2406

```
{
tileload tmm0, [rax]          // cargar siguiente segmento de B
tileload tmm1, [rbx]          // cargar siguiente segmento de C
vaddps tmm2, tmm0, tmm1 // Añadir segmento B a segmento C, almacenar en temporal (inicio de cadena)
vshl tmm2, tmm2, 2;           // desplazar resultado temporal por valor inmediato de 2
vaddps tmm2, tmm2, k;         // añadir valor inmediato de k – (fin de cadena)
tilestore [rcx], tmm2;        // almacenar segmento de resultados en A
}
```

FIG. 24C

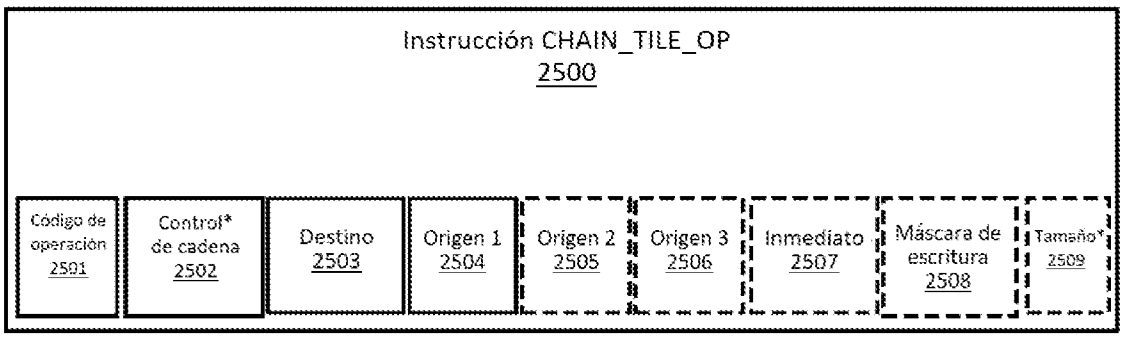

Instrucción CHAIN_TILE_OP
2500
Código de operación 2501
Control* de cadena 2502
Destino 2503
Origen 1 2504
Origen 2 2505
Origen 3 2506
Inmediato 2507
Máscara de escritura 2508
Tamaño* 2509

FIG. 25A

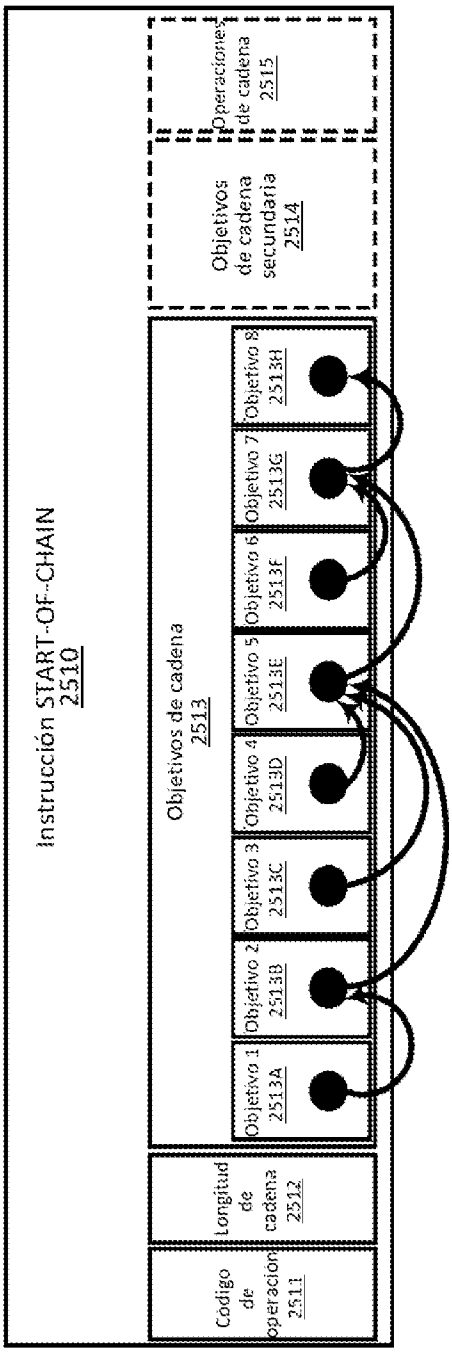

Instrucción START-OF-CHAIN
2510
Código de operación 2511
Longitud de cadena 2512
Objetivos de cadena 2513
Objetivo 1 2513A
Objetivo 2 2513B
Objetivo 3 2513C
Objetivo 4 2513D
Objetivo 5 2513E
Objetivo 6 2513F
Objetivo 7 2513G
Objetivo 8 2513H
Objetivos de cadena secundaria 2514
Operaciones de cadena 2515

FIG. 25B

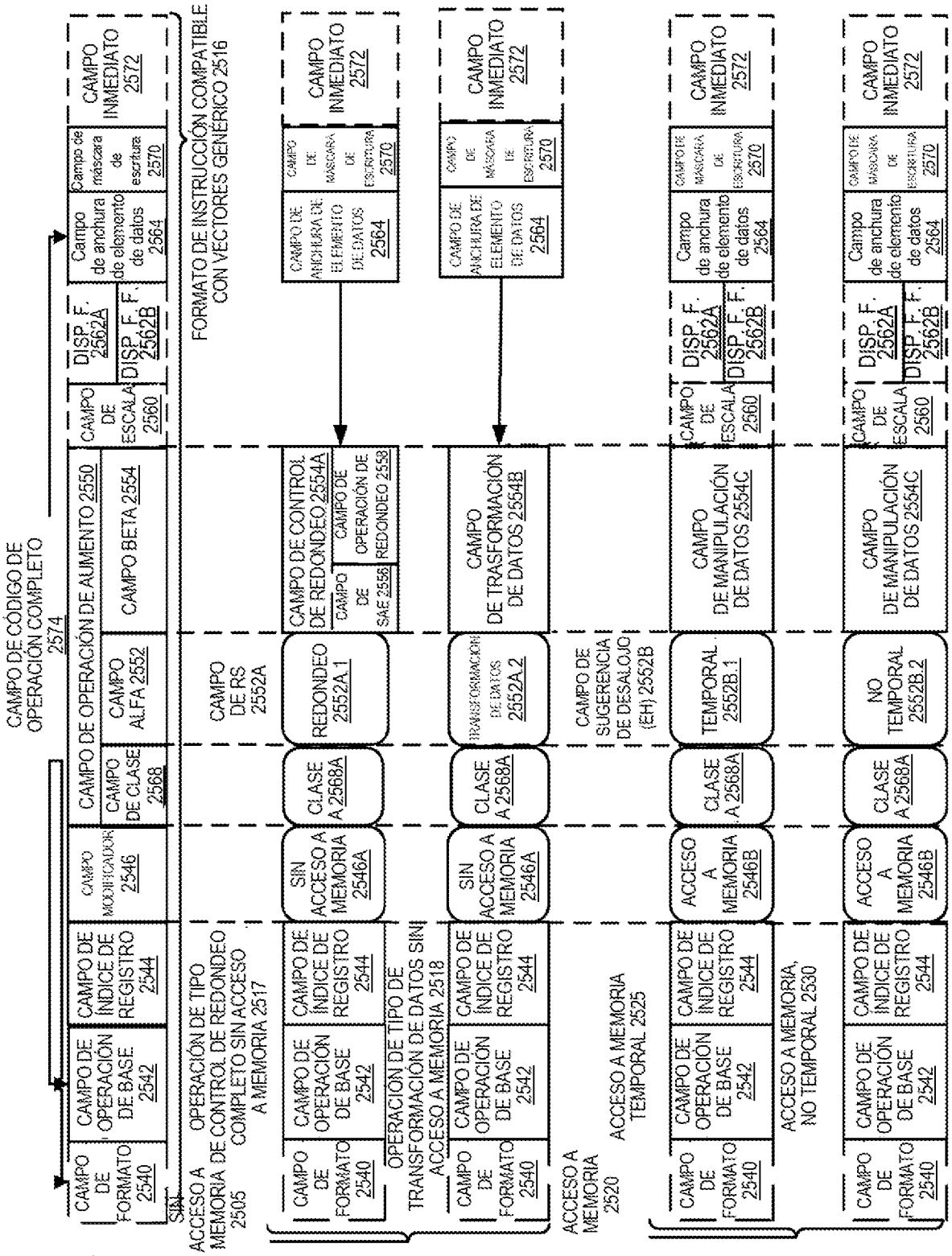
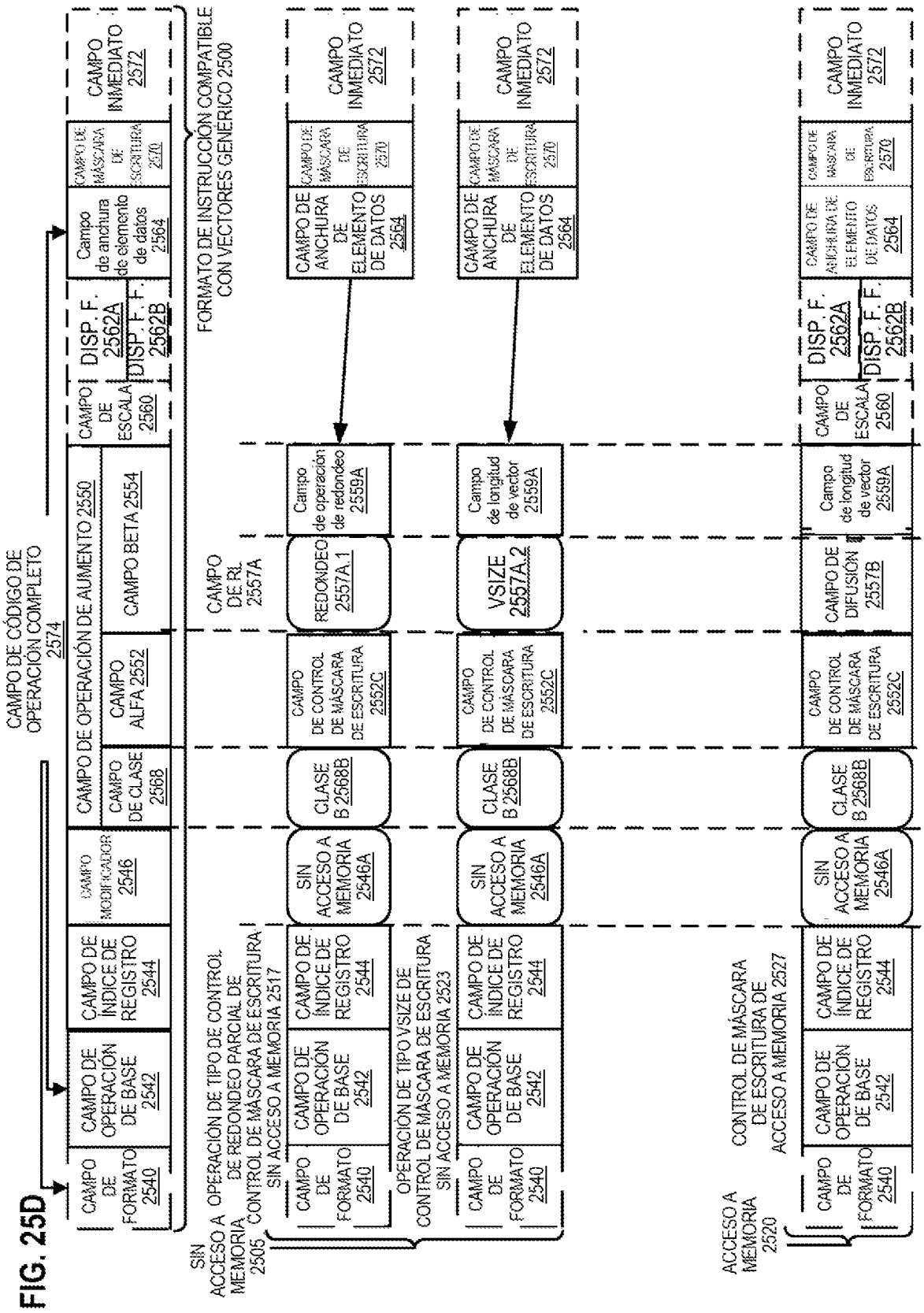

CAMPO DE CÓDIGO DE OPERACIÓN COMPLETO 2574
FORMATO DE INSTRUCCIÓN COMPATIBLE CON VECTORES GENÉRICO 2516
CAMPO DE FORMATO 2540
CAMPO DE OPERACIÓN DE BASE 2542
CAMPO DE ÍNDICE DE REGISTRO 2544
CAMPO MODIFICADOR 2546
CAMPO DE OPERACIÓN DE AUMENTO 2550
CAMPO DE CLASE 2568
CAMPO ALFA 2552
CAMPO BETA 2554
CAMPO DE ESCALA 2560
DISP. F. 2562A
DISP. F. F. 2562B
Campo de anchura de elemento de datos 2564
Campo de máscara de escritura 2570
CAMPO INMEDIATO 2572
SIN ACCESO A MEMORIA 2546A
ACCESO A MEMORIA 2546B
CLASE A 2568A
NO TEMPORAL 2552B.2
TEMPORAL 2552B.1
REDONDEO 2552A.1
TRANSFORMACIÓN DE DATOS 2552A.2
CAMPO DE RS 2552A
CAMPO DE CONTROL DE REDONDEO 2554A
CAMPO DE OPERACIÓN DE REDONDEO 2558
CAMPO DE SAE 2556
CAMPO DE TRASFORMACIÓN DE DATOS 2554B
CAMPO DE MANIPULACIÓN DE DATOS 2554C
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 2564
CAMPO DE MÁSCARA DE ESCRITURA 2570
CAMPO DE SUGERENCIA DE DESALOJO (EH) 2552B
ACCESO A MEMORIA 2520
ACCESO A MEMORIA TEMPORAL 2525
ACCESO A MEMORIA, NO TEMPORAL 2530
OPERACIÓN DE TIPO DE TRANSFORMACIÓN DE DATOS SIN ACCESO A MEMORIA 2518
OPERACIÓN DE TIPO DE CONTROL DE REDONDEO SIN ACCESO A MEMORIA 2517
SIN ACCESO A MEMORIA 2505

FIG.25C

FIG. 25D


CAMPO DE CÓDIGO DE OPERACIÓN COMPLETO 2574
FORMATO DE INSTRUCCIÓN COMPATIBLE CON VECTORES GENÉRICO 2500
CAMPO DE FORMATO 2540
CAMPO DE OPERACIÓN DE BASE 2542
CAMPO DE ÍNDICE DE REGISTRO 2544
CAMPO MODIFICADOR 2546
CAMPO DE OPERACIÓN DE AUMENTO 2550
CAMPO DE CLASE 2568
CAMPO ALFA 2552
CAMPO BETA 2554
CAMPO DE ESCALA 2560
DISP. F. 2562A
DISP. F. F. 2562B
Campo de anchura de elemento de datos 2564
CAMPO DE MÁSCARA DE ESCRITURA 2570
CAMPO INMEDIATO 2572
ACCESO A MEMORIA 2520
SIN ACCESO A MEMORIA 2505
OPERACIÓN DE TIPO DE CONTROL DE REDONDEO PARCIAL DE CONTROL DE MÁSCARA DE ESCRITURA SIN ACCESO A MEMORIA 2517
OPERACIÓN DE TIPO VSIZE DE CONTROL DE MASCARA DE ESCRITURA SIN ACCESO A MEMORIA 2523
CONTROL DE MÁSCARA DE ESCRITURA DE ACCESO A MEMORIA 2527
SIN ACCESO A MEMORIA 2546A
CLASE B 2568B
CAMPO DE CONTROL DE MÁSCARA DE ESCRITURA 2552C
CAMPO DE RL 2557A
REDONDEO 2557A.1
VSIZE 2557A.2
Campo de operación de redondeo 2559A
Campo de longitud de vector 2559A
CAMPO DE DIFUSIÓN 2557B
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 2564

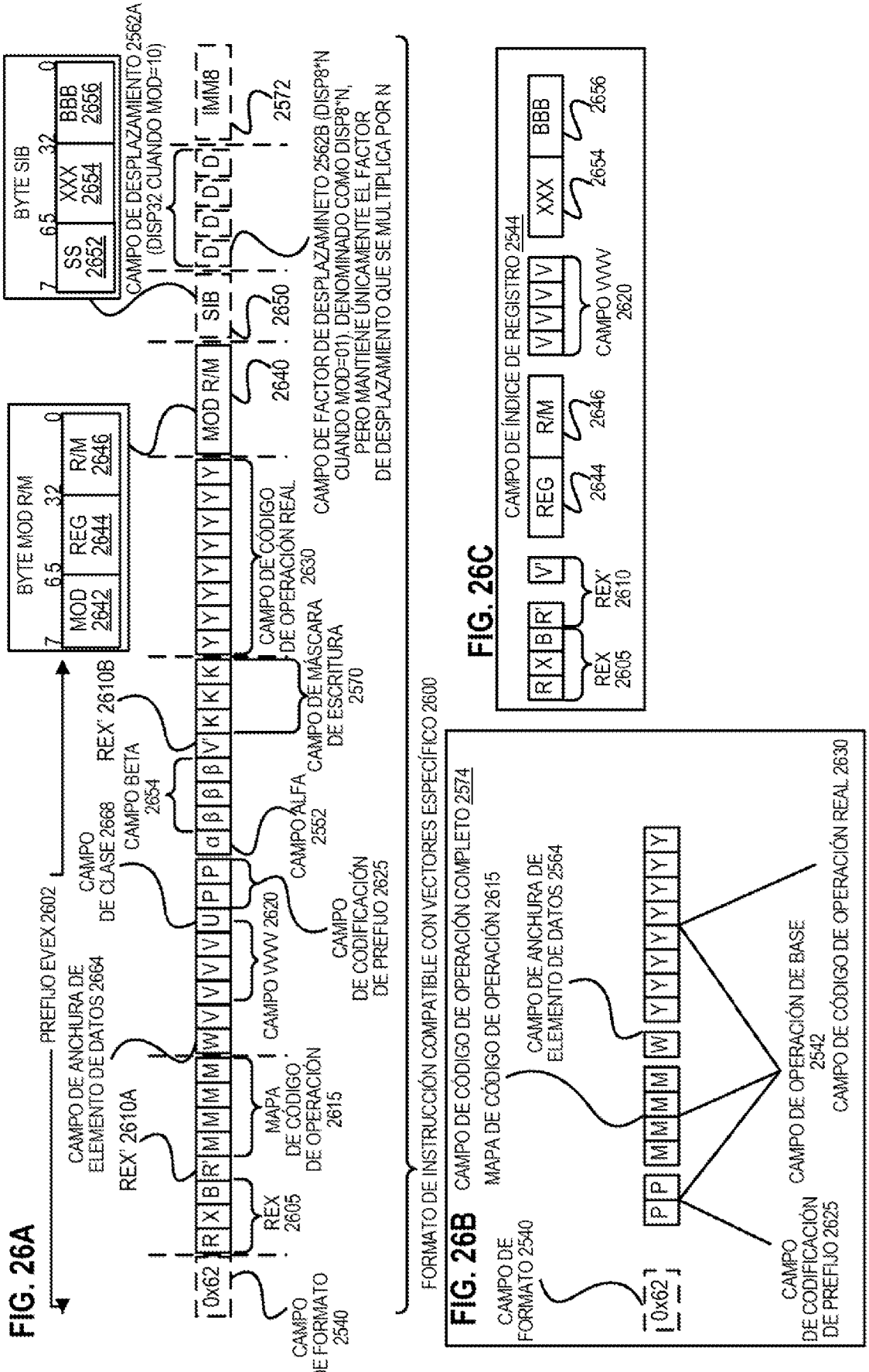


FIG. 26A
PREFIJO EVEX 2602
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 2664
CAMPO DE CLASE 2668
REX' 2610B
CAMPO BETA 2654
CAMPO ALFA 2562
CAMPO DE MÁSCARA DE ESCRITURA 2570
CAMPO DE CÓDIGO DE OPERACIÓN REAL 2630
CAMPO DE FACTOR DE DESPLAZAMIENTO 2562B (DISP8*N CUANDO MOD=01), DENOMINADO COMO DISP8*N, PERO MANTIENE ÚNICAMENTE EL FACTOR DE DESPLAZAMIENTO QUE SE MULTIPLICA POR N
BYTE MOD R/M
MOD 2642
REG 2644
R/M 2646
BYTE SIB
SS 2652
XXX 2654
BBB 2656
CAMPO DE DESPLAZAMIENTO 2562A (DISP32 CUANDO MOD=10)
REX' 2610A
REX 2605
MAPA DE CÓDIGO DE OPERACIÓN 2615
CAMPO VVVV 2620
CAMPO DE CODIFICACIÓN DE PREFIJO 2625
CAMPO DE FORMATO 2540
0x62
MOD R/M
SIB
IMM8
2572
2650
2640
FORMATO DE INSTRUCCIÓN COMPATIBLE CON VECTORES ESPECÍFICO 2600
FIG. 26B
CAMPO DE CÓDIGO DE OPERACIÓN COMPLETO 2574
MAPA DE CÓDIGO DE OPERACIÓN 2615
CAMPO DE FORMATO 2540
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 2564
CAMPO DE CODIFICACIÓN DE PREFIJO 2625
CAMPO DE OPERACIÓN DE BASE 2542
CAMPO DE CÓDIGO DE OPERACIÓN REAL 2630
FIG. 26C
CAMPO DE ÍNDICE DE REGISTRO 2544
REG 2644
R/M 2646
CAMPO VVVV 2620
REX 2605
REX' 2610
XXX 2654
BBB 2656

FIG. 26D

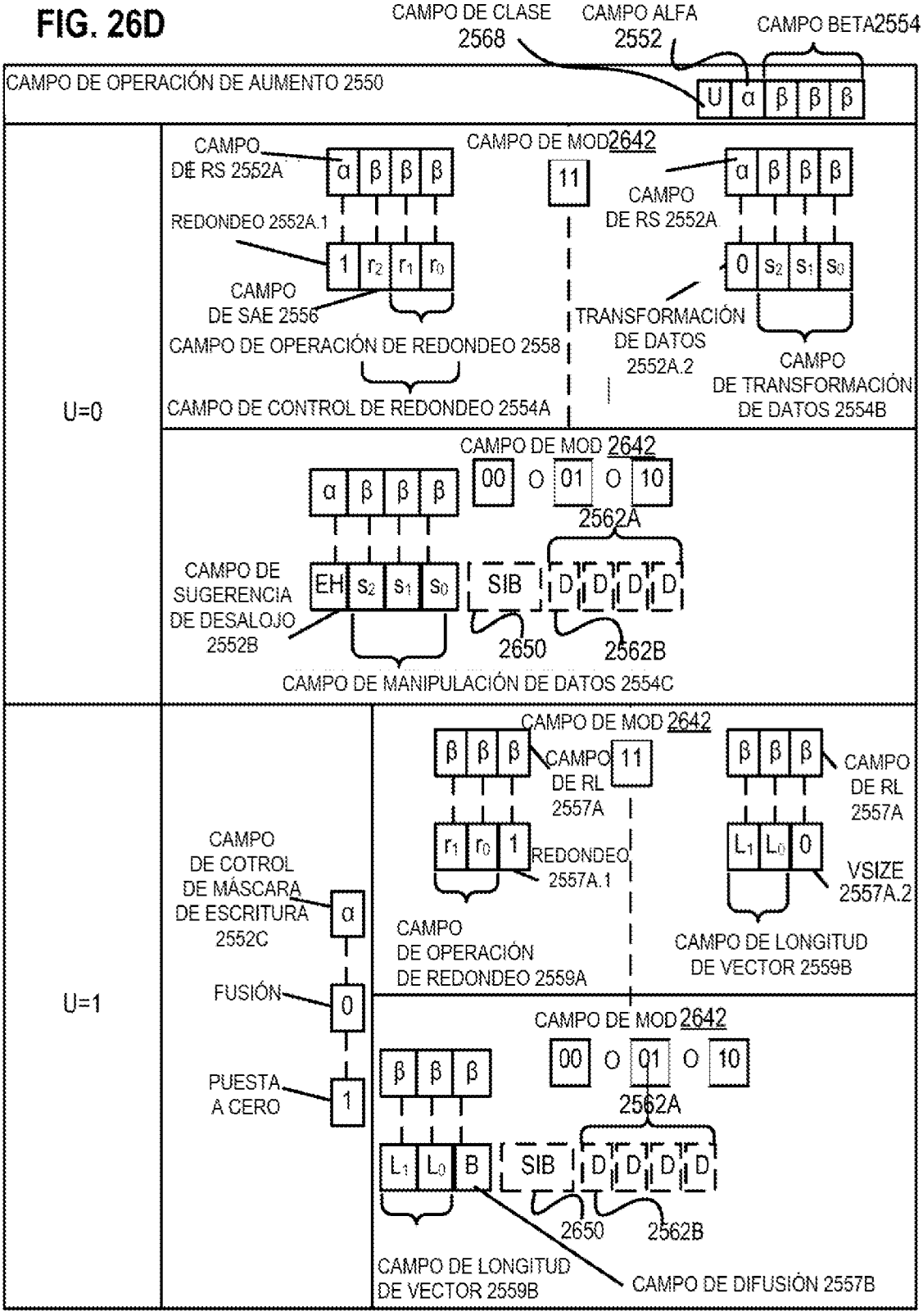

CAMPO DE CLASE 2568
CAMPO ALFA 2552
CAMPO BETA2554
CAMPO DE OPERACIÓN DE AUMENTO 2550
U α β β β
U=0
CAMPO DE RS 2552A
α β β β
REDONDEO 2552A.1
1 r2 r1 r0
CAMPO DE SAE 2556
CAMPO DE OPERACIÓN DE REDONDEO 2558
CAMPO DE CONTROL DE REDONDEO 2554A
CAMPO DE MOD2642
11
CAMPO DE RS 2552A
α β β β
0 s2 s1 s0
TRANSFORMACIÓN DE DATOS 2552A.2
CAMPO DE TRANSFORMACIÓN DE DATOS 2554B
CAMPO DE MOD 2642
00 O 01 O 10
2562A
α β β β
EH s2 s1 s0
SIB D D D D
CAMPO DE SUGERENCIA DE DESALOJO 2552B
2650
2562B
CAMPO DE MANIPULACIÓN DE DATOS 2554C
U=1
CAMPO DE COTROL DE MÁSCARA DE ESCRITURA 2552C
α
FUSIÓN
0
PUESTA A CERO
1
CAMPO DE MOD 2642
β β β
CAMPO DE RL 2557A
11
r1 r0 1
REDONDEO 2557A.1
CAMPO DE OPERACIÓN DE REDONDEO 2559A
β β β
CAMPO DE RL 2557A
L1 L0 0
VSIZE 2557A.2
CAMPO DE LONGITUD DE VECTOR 2559B
CAMPO DE MOD 2642
00 O 01 O 10
2562A
β β β
L1 L0 B
SIB D D D D
2650
2562B
CAMPO DE LONGITUD DE VECTOR 2559B
CAMPO DE DIFUSIÓN 2557B

FIG. 27

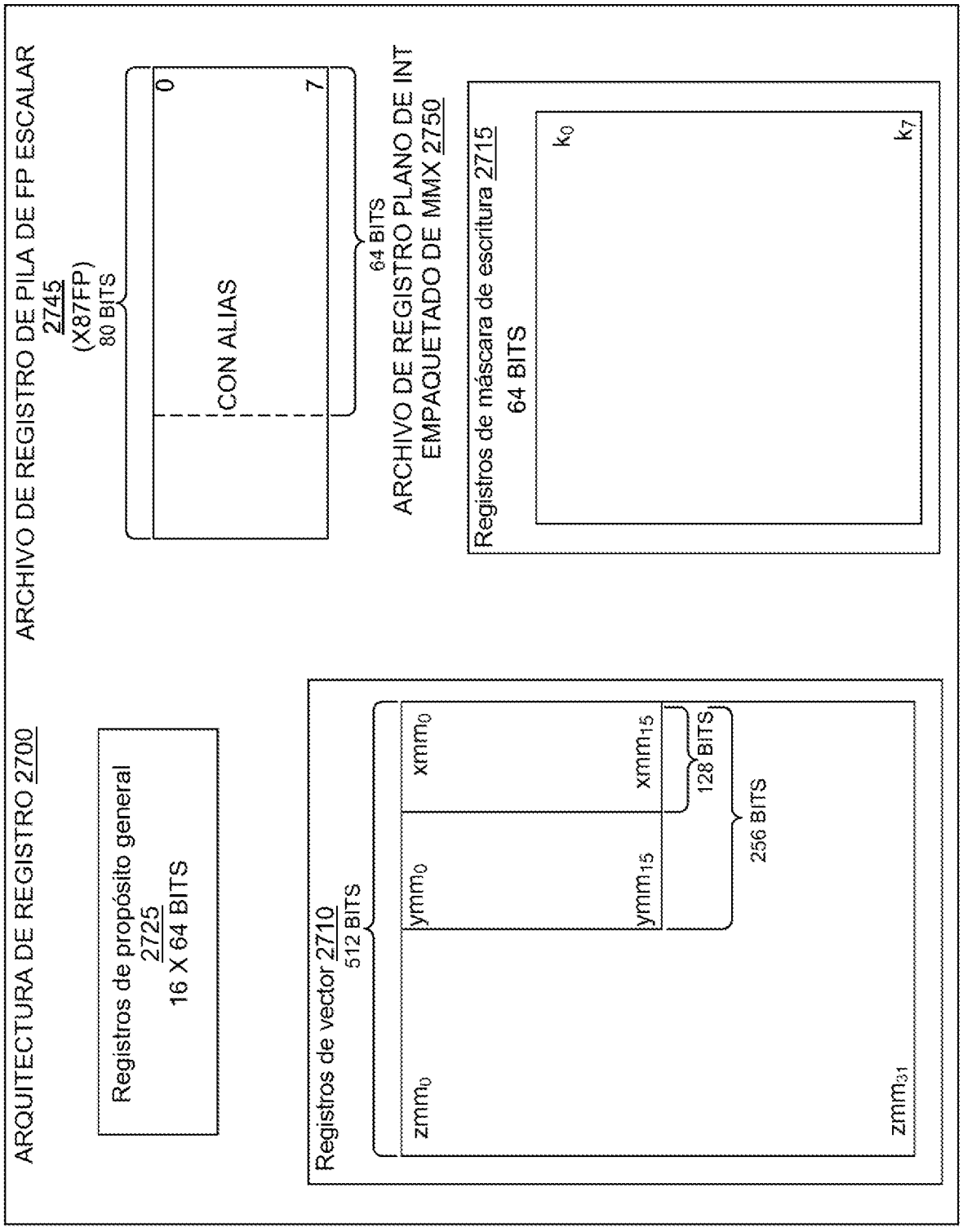

ARQUITECTURA DE REGISTRO 2700
Registros de propósito general 2725
16 X 64 BITS
Registros de vector 2710
512 BITS
zmm0
ymm0
xmm0
ymm15
xmm15
128 BITS
256 BITS
zmm31
ARCHIVO DE REGISTRO DE PILA DE FP ESCALAR 2745
(X87FP)
80 BITS
0
CON ALIAS
7
64 BITS
ARCHIVO DE REGISTRO PLANO DE INT EMPAQUETADO DE MMX 2750
Registros de máscara de escritura 2715
64 BITS
k0
k7

FIG. 28A

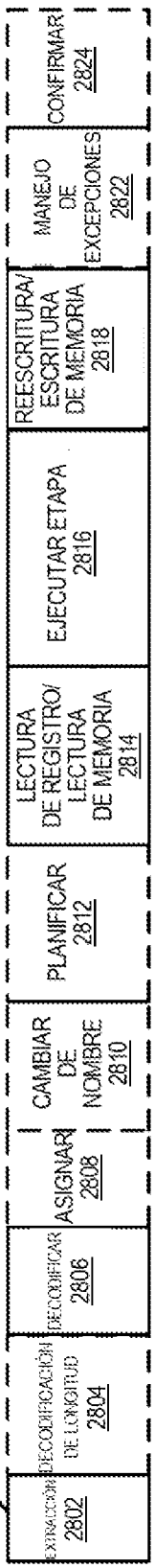


FIG. 28B

FIG. 29A

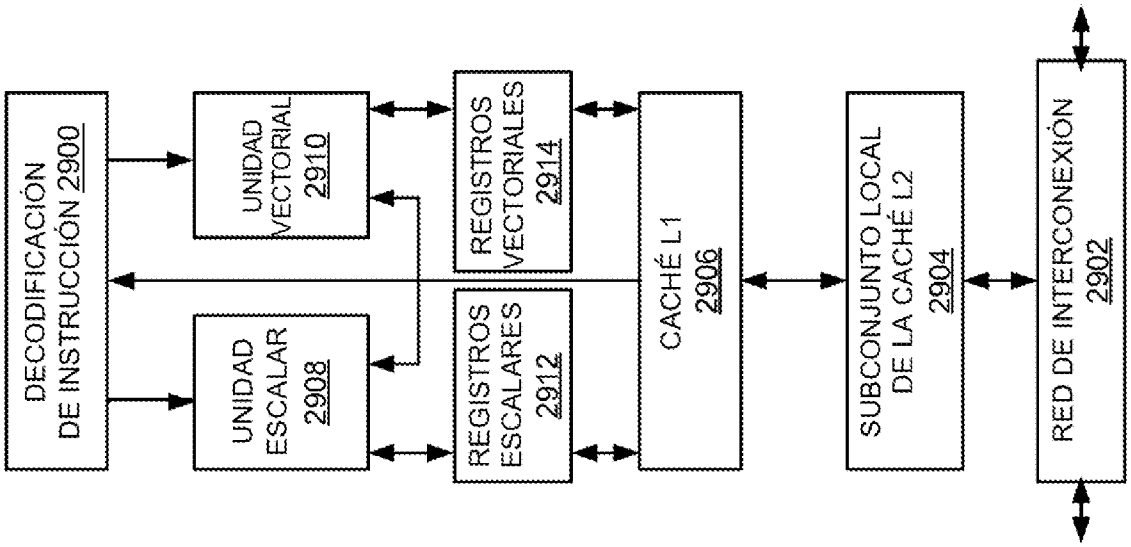
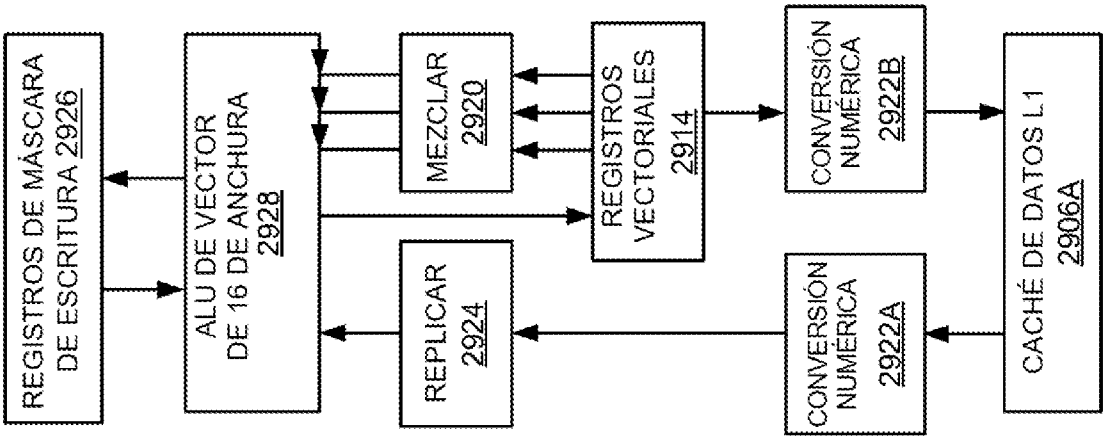


FIG. 29B



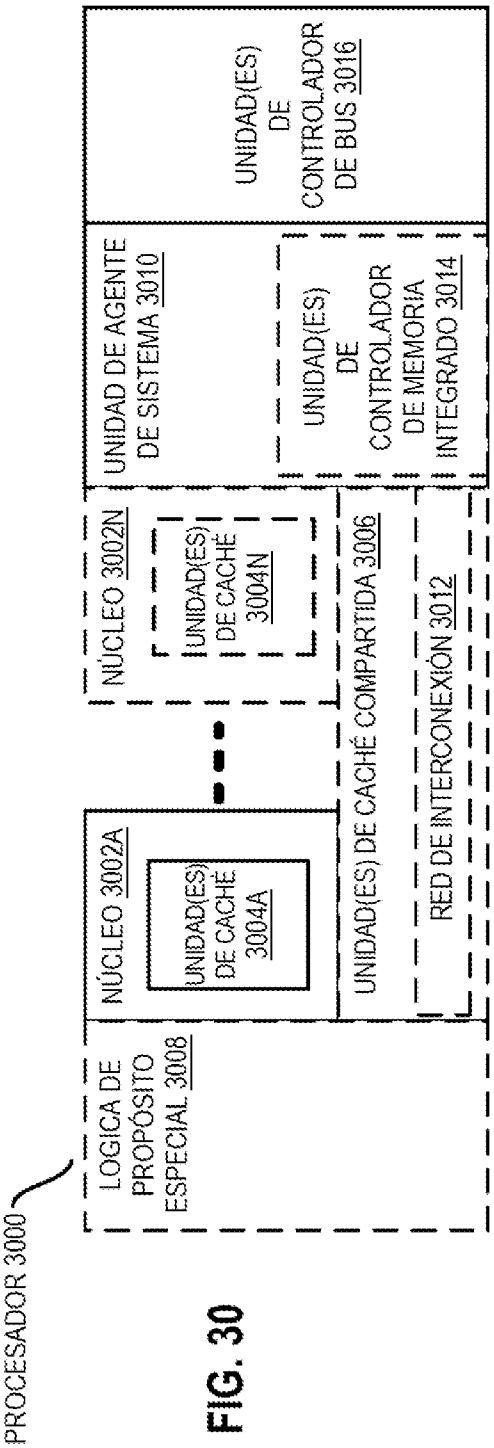

PROCESADOR 3000
LOGICA DE PROPÓSITO ESPECIAL 3008
NÚCLEO 3002A
UNIDAD(ES) DE CACHÉ 3004A
NÚCLEO 3002N
UNIDAD(ES) DE CACHÉ 3004N
UNIDAD(ES) DE CACHÉ COMPARTIDA 3006
RED DE INTERCONEXIÓN 3012
UNIDAD DE AGENTE DE SISTEMA 3010
UNIDAD(ES) DE CONTROLADOR DE MEMORIA INTEGRADO 3014
UNIDAD(ES) DE CONTROLADOR DE BUS 3016

FIG. 30

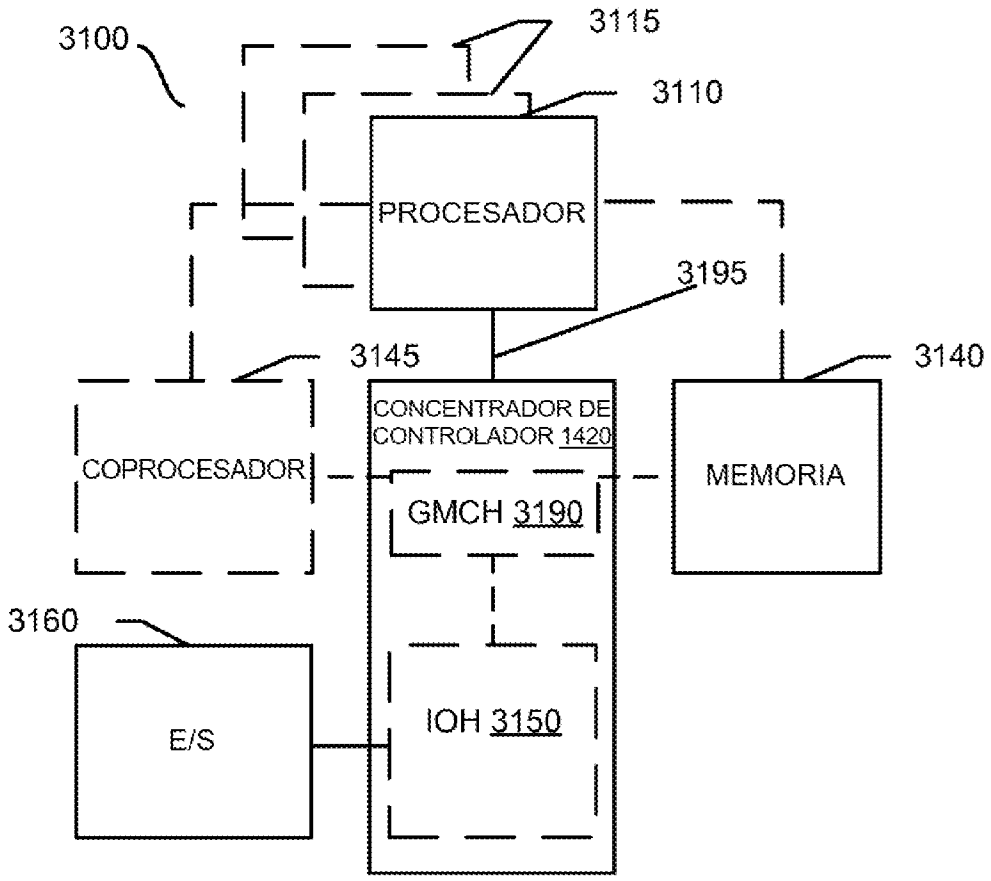

3100
3115
3110
PROCESADOR
3195
3145
3140
CONCENTRADOR DE CONTROLADOR 1420
COPROCESADOR
MEMORIA
GMCH 3190
3160
E/S
IOH 3150

FIG. 31

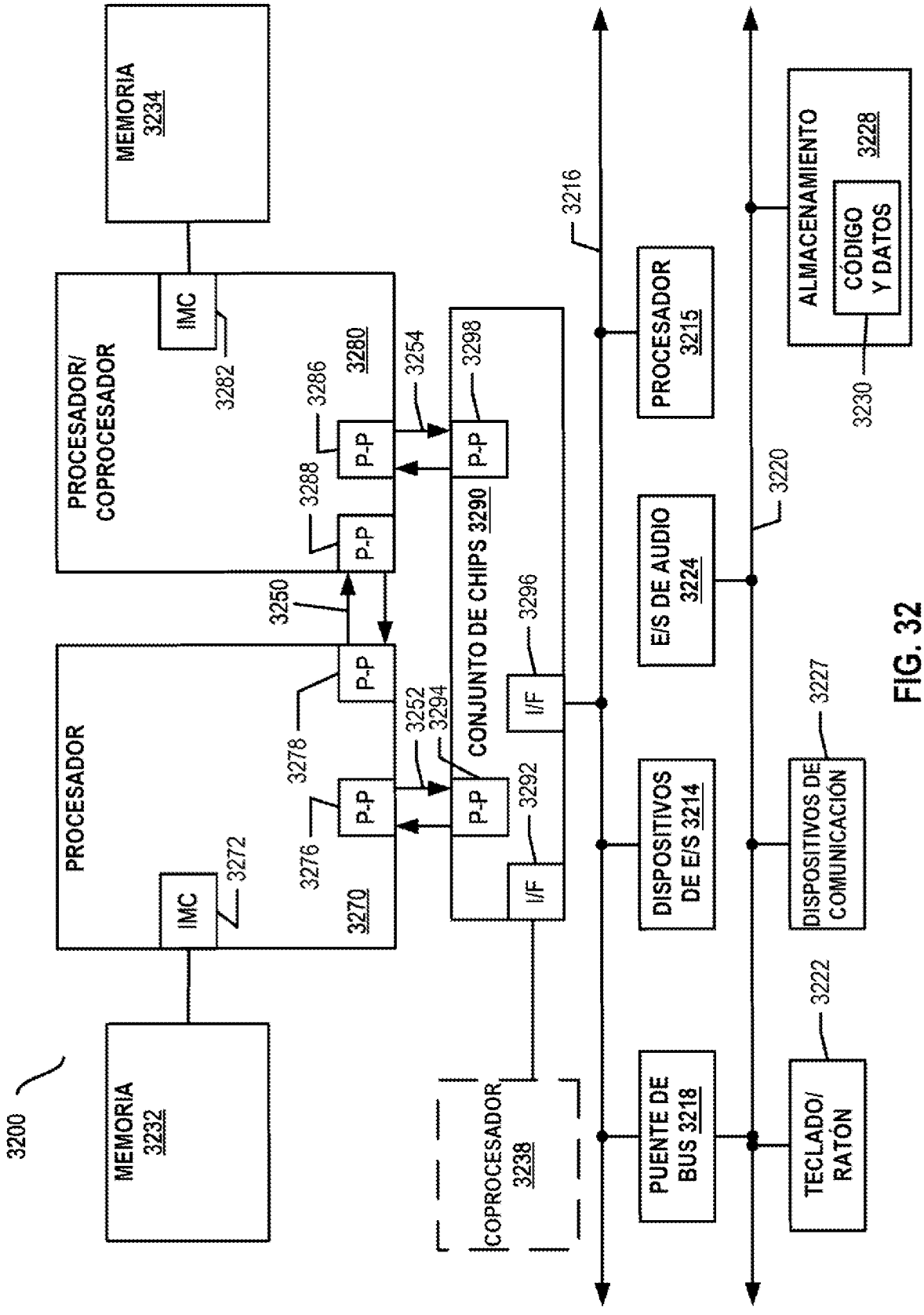

3200
MEMORIA 3232
MEMORIA 3234
PROCESADOR
IMC 3272
PROCESADOR/ COPROCESADOR
IMC 3282
3270
3280
P-P
3276
3278
3286
3288
3250
3252
3254
3294
3298
3296
CONJUNTO DE CHIPS 3290
I/F
3292
COPROCESADOR 3238
3216
PROCESADOR 3215
E/S DE AUDIO 3224
DISPOSITIVOS DE E/S 3214
PUENTE DE BUS 3218
3220
ALMACENAMIENTO
CÓDIGO Y DATOS 3228
3230
3227
DISPOSITIVOS DE COMUNICACIÓN
3222
TECLADO/ RATÓN

FIG. 32

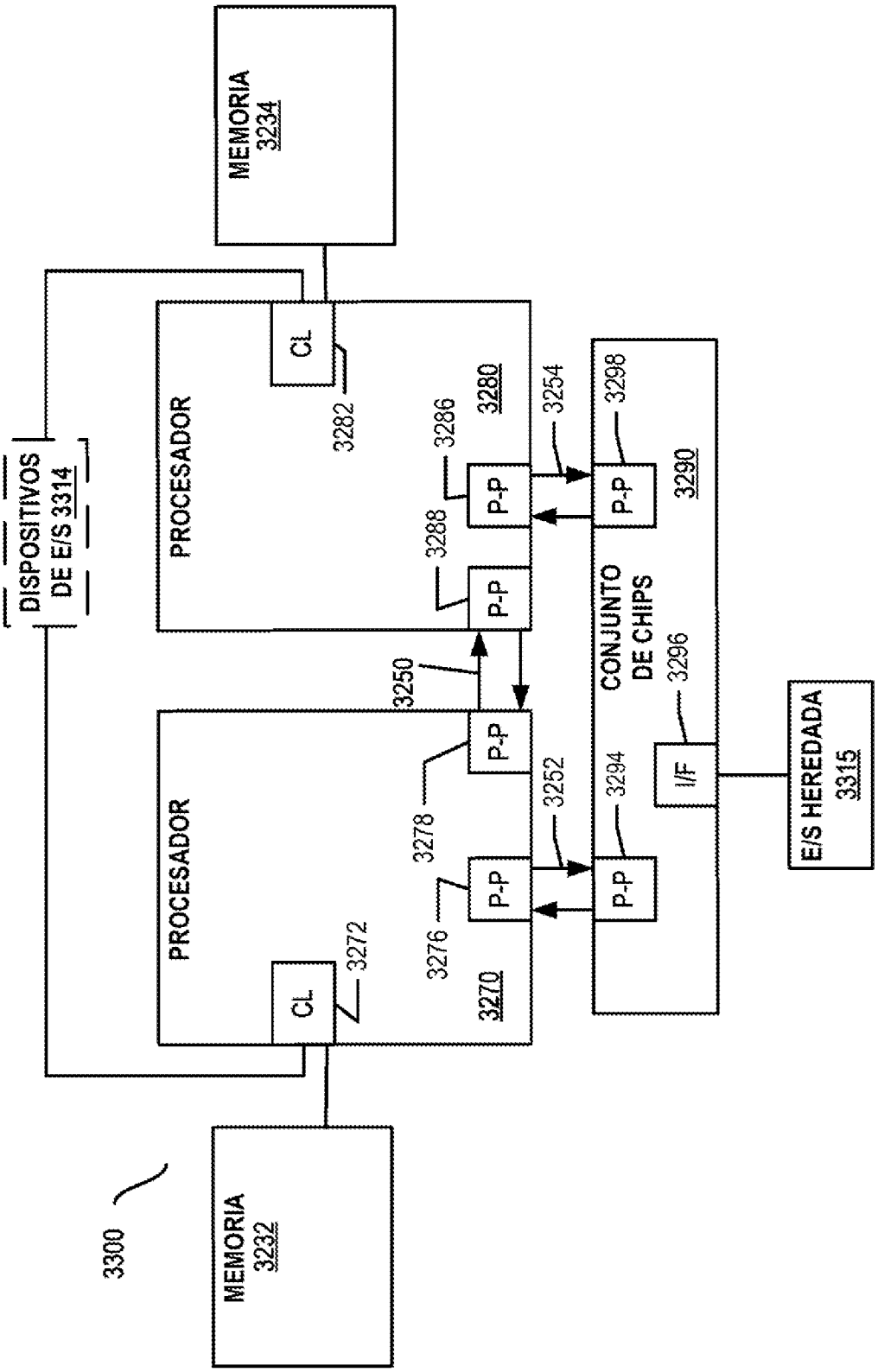

3300
MEMORIA
3232
MEMORIA
3234
DISPOSITIVOS
DE E/S 3314
PROCESADOR
CL
3272
3270
PROCESADOR
CL
3282
3280
P-P
3276
P-P
3278
P-P
3286
P-P
3288
3250
3252
3254
CONJUNTO
DE CHIPS
3290
P-P
3294
P-P
3298
I/F
3296
E/S HEREDADA
3315

FIG. 33

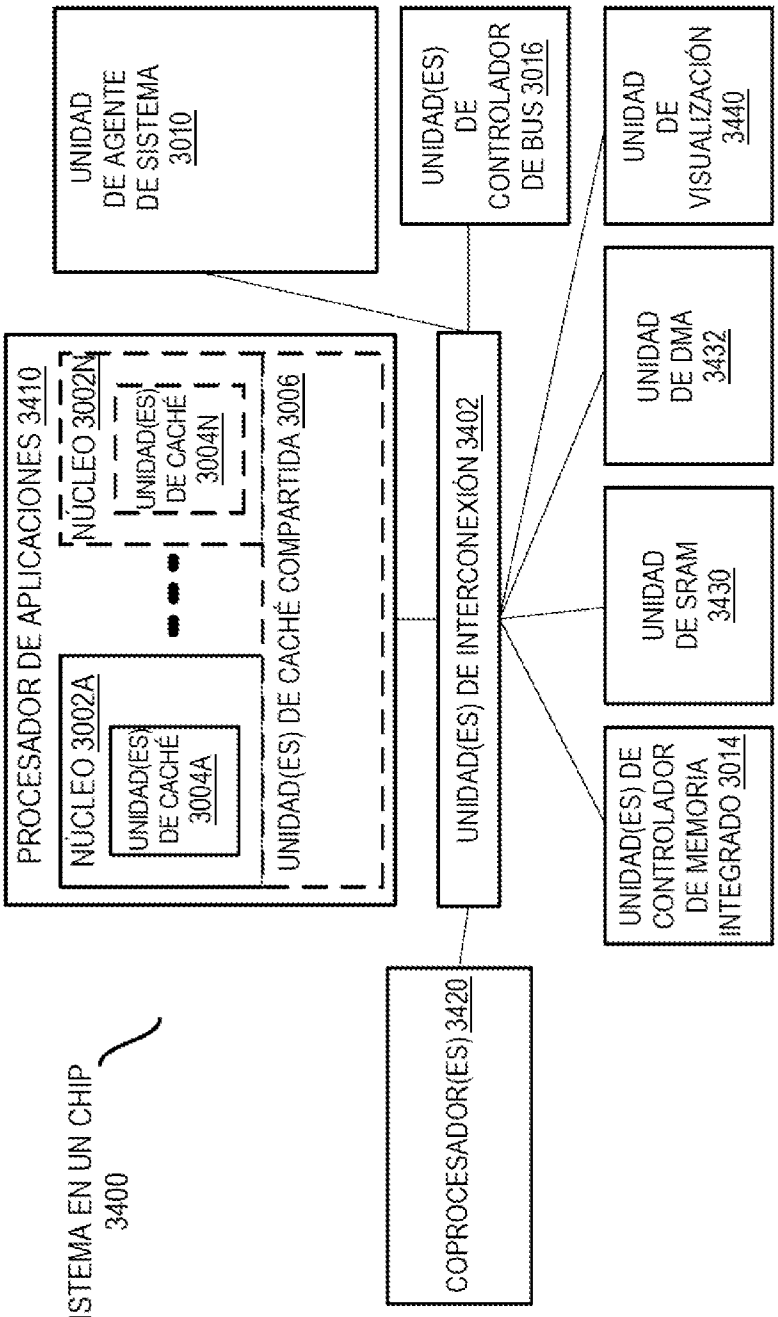

SISTEMA EN UN CHIP
3400
PROCESADOR DE APLICACIONES 3410
NÚCLEO 3002A
UNIDAD(ES) DE CACHÉ 3004A
NÚCLEO 3002N
UNIDAD(ES) DE CACHÉ 3004N
UNIDAD(ES) DE CACHÉ COMPARTIDA 3006
UNIDAD DE AGENTE DE SISTEMA 3010
UNIDAD(ES) DE CONTROLADOR DE BUS 3016
UNIDAD DE VISUALIZACIÓN 3440
UNIDAD DE DMA 3432
UNIDAD DE SRAM 3430
UNIDAD(ES) DE CONTROLADOR DE MEMORIA INTEGRADO 3014
UNIDAD(ES) DE INTERCONEXIÓN 3402
COPROCESADOR(ES) 3420

FIG. 34

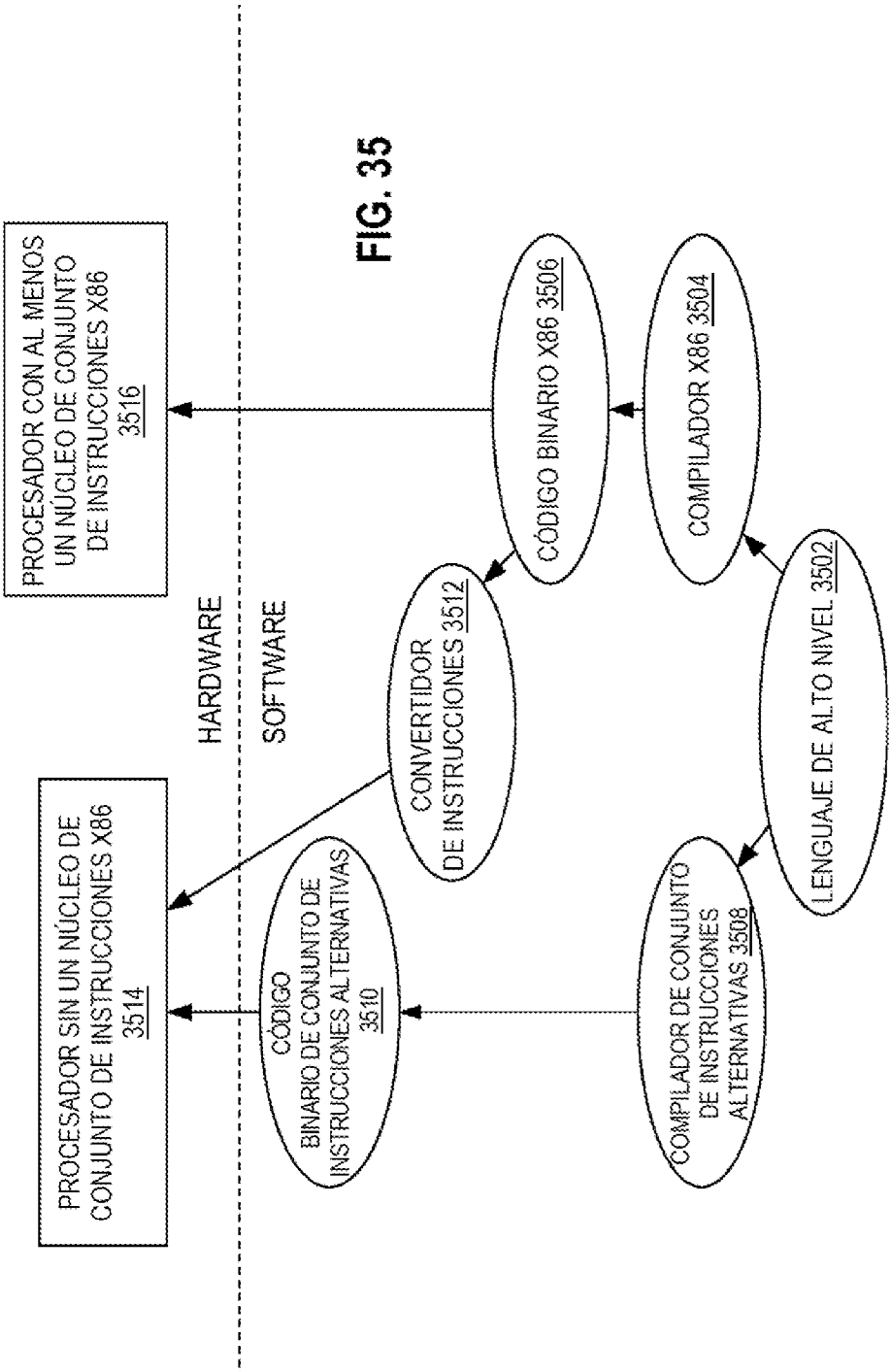

PROCESADOR CON AL MENOS UN NÚCLEO DE CONJUNTO DE INSTRUCCIONES X86 3516
HARDWARE
SOFTWARE
PROCESADOR SIN UN NÚCLEO DE CONJUNTO DE INSTRUCCIONES X86 3514
CÓDIGO BINARIO DE CONJUNTO DE INSTRUCCIONES ALTERNATIVAS 3510
CONVERTIDOR DE INSTRUCCIONES 3512
CÓDIGO BINARIO X86 3506
COMPILADOR X86 3504
COMPILADOR DE CONJUNTO DE INSTRUCCIONES ALTERNATIVAS 3508
LENGUAJE DE ALTO NIVEL 3502

FIG. 35