(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 905 697**

(51) Int. CI.:

G06F 9/30 (2008.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **28.06.2019 E 19183504 (0)**

(97) Fecha y número de publicación de la concesión europea: **27.10.2021 EP 3608776**

(54) Título: **Sistemas, aparatos y métodos para generar un índice por orden de clasificación y reordenar elementos basándose en el orden de clasificación**

(30) Prioridad:

11.08.2018 US 201862717829 P
27.03.2019 US 201916367186

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
11.04.2022

(73) Titular/es:

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

(72) Inventor/es:

BAUM, DAN;
ZOHAR, RONEN;
MISHRA, ASIT;
SURTI, PRASOONKUMAR;
ELMOUSTAPHA, OULD-AHMED-VALL;
HUGHES, CHRISTOPHER y
HEINECKE, ALEXANDER

(74) Agente/Representante:

LEHMANN NOVO, María Isabel

DESCRIPCIÓN

Sistemas, aparatos y métodos para generar un índice por orden de clasificación y reordenar elementos basándose en el orden de clasificación

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 realizar instrucciones que especifican operaciones de teselas ternarias.

ANTECEDENTES

Las matrices son cada vez más importantes en muchas tareas informáticas, tales como el aprendizaje automático y otros procesamientos de datos en masa. El aprendizaje profundo es una clase de algoritmos de aprendizaje automático. Las arquitecturas de aprendizaje profundo, tales como las redes neuronales profundas, se han aplicado a campos que incluyen la visión por ordenador, el reconocimiento de habla, el procesamiento de lenguaje natural, el reconocimiento de audio, el filtrado de redes sociales, la traducción automática, la bioinformática y el diseño de fármacos.

La inferencia y el entrenamiento, dos herramientas usadas para el aprendizaje profundo, tienden a una aritmética de baja precisión. Maximizar la capacidad de proceso de los cálculos y algoritmos de aprendizaje profundo puede ayudar a satisfacer las necesidades de los procesadores de aprendizaje profundo, por ejemplo, aquellos que realizan un aprendizaje profundo en un centro de datos.

Las instrucciones para realizar instrucciones que especifican operaciones de teselas ternarias sobre tres matrices de origen son útiles en un contexto de aprendizaje automático.

El documento WO2006033056A2 se refiere a un dispositivo y método de microprocesador para operaciones de ordenación aleatoria. Por ejemplo, el dispositivo de microprocesador comprende una arquitectura de procesador de vectores con una unidad de procesador de vectores funcional que comprende unos primeros medios de memoria para almacenar una pluralidad de vectores de índice y medios de procesamiento, estando dispuesta la unidad de procesador de vectores funcional para recibir una instrucción de procesamiento y al menos un vector de entrada a procesar, estando dispuestos dichos primeros medios de memoria para proporcionar a los medios de procesamiento uno de dicha pluralidad de vectores de índice de acuerdo con la instrucción de procesamiento, y estando dispuestos los medios de procesamiento para generar, en respuesta a dicha instrucción, al menos un vector de salida que tiene los elementos del al menos un vector de entrada reorganizados de acuerdo con el vector de índice proporcionado.

SUMARIO

La presente invención se define en las reivindicaciones independientes. Las reivindicaciones dependientes definen realizaciones de la misma.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

La presente invención se ilustra, a modo de ejemplo y no de limitación, en las figuras de los dibujos adjuntos, en los que referencias semejantes 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 matriz;
la **figura 3** ilustra una realización de un sistema que utiliza un acelerador de operaciones matriciales (de teselas);
las **figuras 4 y 5** muestran diferentes realizaciones de cómo se comparte memoria usando un acelerador de operaciones matriciales;
la **figura 6** ilustra una realización de una operación de acumulado de multiplicación matricial usando teselas ("TMMA");
la **figura 7** ilustra una realización de un subconjunto de la ejecución de una iteración de una instrucción de acumulado de multiplicación fusionado encadenado;
la **figura 8** ilustra una realización de un subconjunto de la ejecución de una iteración de una instrucción de acumulado de multiplicación fusionado encadenado;
la **figura 9** ilustra una realización de un subconjunto de la ejecución de una iteración de una instrucción de acumulado de multiplicación fusionado encadenado;
la **figura 10** ilustra una realización de un subconjunto de la ejecución de una iteración de una instrucción de acumulado de multiplicación fusionado encadenado;
la **figura 11** ilustra implementaciones de SIMD de tamaño 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 matriciales;
la **figura 13** ilustra una realización de una canalización de núcleo de procesador que soporta operaciones matriciales usando teselas;

la **figura 14** ilustra una realización de una canalización de núcleo de procesador que soporta operaciones matriciales usando teselas;
la **figura 15** ilustra un ejemplo de una matriz expresada en un formato ordenado por filas y en un formato ordenado por columnas;
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 un 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) a soportar;
las **figuras 20(A)-(D)** ilustran ejemplos de un registro o registros;
la **figura 21** ilustra realizaciones de ejecución de una única instrucción de clasificación de índice y de reordenación;
la **figura 22** ilustra realizaciones de ejecución de una única instrucción de clasificación de índice y una única instrucción de reordenación;
la **figura 23** ilustra realizaciones de una circuitería para generar un índice;
la **figura 24** ilustra realizaciones de una circuitería para generar un índice;
las **figuras 25A-25B** son diagramas de bloques que ilustran un formato de instrucción apto para vectores de carácter genérico y plantillas de instrucción del mismo de acuerdo con realizaciones de la invención;
la **figura 25A** es un diagrama de bloques que ilustra un formato de instrucción apto para vectores de carácter genérico y plantillas de instrucción de clase A del mismo de acuerdo con realizaciones de la invención;
la **figura 25B** es un diagrama de bloques que ilustra el formato de instrucción apto para vectores de carácter genérico y plantillas de instrucción de clase B del mismo de acuerdo con realizaciones de la invención;
la **figura 26A** es un diagrama de bloques que ilustra un formato de instrucción apto para vectores de carácter 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 apto para vectores de carácter específico que constituyen 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 apto para vectores de carácter específico que constituyen 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 apto para vectores de carácter específico que constituyen 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 ilustrativa como una canalización de cambio de nombre de registro, emisión/ejecución fuera de orden 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 cambio de nombre de registro, emisión/ejecución fuera de orden ilustrativo a incluir en un procesador de acuerdo con realizaciones de la invención;
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 (incluyendo otros núcleos del mismo tipo y/o de diferentes tipos) 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 integrada en pastilla y con su subconjunto local de la memoria caché de nivel 2 (L2), de acuerdo con realizaciones de la invención;
la **figura 29B** es una vista ampliada de parte del núcleo de 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 de ordenador 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;
la **figura 35** es un diagrama de bloques que contrapone 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;
la **figura 36** ilustra una realización del procesamiento de una única instrucción;
la **figura 37** ilustra una realización del procesamiento de una única instrucción;
la **figura 38** ilustra una realización del procesamiento de una única instrucción;
la **figura 39** ilustra una realización del procesamiento de una única instrucción;
la **figura 40** ilustra una realización del procesamiento de una única instrucción;
la **figura 41** ilustra una realización del procesamiento de una única instrucción; y
la **figura 42** ilustra una realización del procesamiento de una única instrucción.

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 con detalle circuitos, estructuras y técnicas bien conocidos con el fin de no complicar la comprensión de esta descripción.

Las referencias en la memoria descriptiva a "una realización", "una realización", "una realización de ejemplo" y similares indican que la realización descrita puede incluir un rasgo, estructura o característica particular, pero no puede no ser necesario que cada realización incluya el rasgo, estructura o característica particular. Además, tales expresiones no hacen referencia necesariamente a la misma realización. Además, cuando un rasgo, estructura o característica particular se describe en conexión con una realización, se afirma que está dentro del conocimiento de un experto en la materia afectar a tal rasgo, estructura o característica en conexión con otras realizaciones, ya se describa explícitamente o no.

En muchos procesadores de uso generalizado, el manejo de matrices es una tarea difícil y/o exigente en cuanto a instrucciones. Por ejemplo, las filas de una matriz se podrían colocar en una pluralidad de registros de datos empaquetados (por ejemplo, de SIMD o de vector) y entonces operarse sobre las mismas de forma individual. Por ejemplo, sumar dos matrices 8 x 2 puede requerir una carga o recopilación en cuatro registros de datos empaquetados dependiendo de los tamaños de datos. Entonces se realiza una primera suma de registros de datos empaquetados correspondientes a una primera fila a partir de cada matriz y se realiza una segunda suma de registros de datos empaquetados correspondientes a una segunda fila a partir de cada matriz. Entonces, los registros de datos empaquetados resultantes se dispersan de vuelta a la memoria. Aunque este escenario puede ser aceptable para matrices pequeñas, a menudo no lo es con matrices más grandes.

ANÁLISIS

En el presente documento se describen mecanismos para soportar operaciones matriciales en hardware informático, tales como unidades centrales de procesamiento (CPU), unidades de procesamiento gráfico (GPU) y aceleradores. Las operaciones matriciales utilizan estructuras de datos bidimensionales (2-D) que representan una o más regiones empaquetadas de memoria, tales como registros. De principio a fin 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 una cualquiera de las teselas). De principio a fin de la descripción, se usa vocabulario de matrices (teselas) para indicar operaciones realizadas usando teselas que afectan a una matriz; habitualmente no es relevante si esa matriz es o no más grande que una cualquiera de las teselas.

Se puede actuar sobre cada tesela mediante diferentes operaciones, tales como las que se detallan en el presente documento e incluyen, pero no se limitan a: multiplicación de matrices (teselas), adición de teselas, resta de teselas, diagonal de teselas, puesta a cero de teselas, traspuesta de teselas, producto escalar de teselas, radiodifusión de teselas, radiodifusión de filas de tesela, radiodifusión de columnas de tesela, multiplicación de teselas, multiplicación y acumulación de teselas, movimiento de teselas, etc. Adicionalmente, se puede usar un soporte para operadores tal como el uso de una escala y/o sesgo con estas operaciones o en apoyo de aplicaciones no numéricas en el futuro, por ejemplo, "memoria local" de OpenCL, compresión/descompresión de datos, etc. También se describen en el presente documento instrucciones para realizar operaciones de teselas ternarias (TILETERNOP)

Porciones de almacenamiento (tales como memoria (no volátil y volátil), registros, caché, etc.) se disponen 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). Habitualmente, la dimensión horizontal está relacionada con tamaños de elemento (por ejemplo, 2, 4, 8, 16, 32, 64, 128 bits, etc.). Se pueden soportar múltiples tipos de datos (coma flotante de precisión simple, coma flotante de precisión doble, número entero, etc.).

USO ILUSTRATIVO DE TESELAS CONFIGURADAS

En algunas realizaciones, se pueden configurar parámetros de tesela. Por ejemplo, una tesela dada se puede configurar para proporcionar opciones de tesela. Las opciones de tesela ilustrativas incluyen, pero no se limitan a: 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 de igual tamaño.

La **figura 1A** ilustra una realización de teselas configuradas. Como se muestra, 4 kB de la memoria de aplicación 102 tienen almacenados, en los mismos, 4 títulos de 1 kB, la tesela 0 104, la tesela 1 106, 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. La tesela t0 104 y la tesela t1 106 tienen K filas y N columnas de elementos de 4 bytes (por ejemplo, datos de precisión simple), en donde K es igual a 8 y N = 32. La tesela t2 108 y la tesela t3 110 tienen K filas y N/2 columnas de elementos de 8 bytes (por ejemplo, datos de precisión doble). Debido a que los operandos de precisión doble tienen el doble de anchura que los de precisión simple, esta configuración es coherente con una paleta, usada para proporcionar opciones de tesela, que suministra al menos 4 nombres con un almacenamiento total de al menos 4 kB. Durante el funcionamiento, las teselas se pueden cargar de y almacenar en memoria usando operaciones de carga y de almacenamiento. Dependiendo del esquema de codificación de instrucciones usado, varía la cantidad de memoria de aplicación disponible, así como el tamaño, el número y la configuración de las teselas disponibles.

La **figura 1B** ilustra una realización de teselas configuradas. Como se muestra, 4 kB de la memoria de aplicación 122 tienen almacenados, en los mismos, 2 pares de títulos 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. La tesela t4L 124 y la tesela t4R 126 tienen K filas y N columnas de elementos de 4 bytes (por ejemplo, datos de precisión simple), en donde K es igual a 8 y N es igual a 32. La tesela t5L 128 y la tesela t5R 130 tienen K filas y N/2 columnas de elementos de 8 bytes (por ejemplo, datos de precisión doble). Debido a que los operandos de precisión doble tienen el doble de anchura que los de precisión simple, esta configuración es coherente con una paleta, usada para proporcionar opciones de tesela, que suministra al menos 2 nombres con un almacenamiento total de al menos 4 kB. Las cuatro teselas de la **figura 1A** usan 4 nombres, cada uno nombrando una tesela de 1 kB, mientras que los 2 pares de teselas en la **figura 1B** pueden usar 2 nombres para especificar las teselas emparejadas. En algunas realizaciones, las instrucciones de tesela aceptan un nombre de una tesela emparejada como un operando. Durante el funcionamiento, las teselas se pueden cargar de y almacenar en memoria usando operaciones de carga y de almacenamiento. Dependiendo del esquema de codificación de instrucciones usado, varía la cantidad de memoria de aplicación disponible, así como el tamaño, el número y la configuración de las teselas disponibles.

En algunas realizaciones, se pueden definir parámetros de tesela. Por ejemplo, se usa una "paleta" para proporcionar opciones de tesela. Las opciones ilustrativas incluyen, pero no se limitan a: el número de nombres de tesela, el número de bytes en una fila de almacenamiento, el número de filas y columnas en una tesela, etc. Por ejemplo, una "altura" (número de filas) máxima de una tesela se puede definir como:

Filas máximas de teselas = almacenamiento estructurado / (número de nombres de paleta * número de bytes por fila).

En ese sentido, una aplicación se puede escribir de tal forma que un uso fijo de nombres será capaz de aprovechar los diferentes tamaños de almacenamiento en todas las implementaciones.

La configuración de teselas se hace usando una instrucción de configuración de teselas ("TILECONFIG"), en donde se define un uso de teselas particular en una paleta seleccionada. Esta declaración incluye el número de nombres de tesela 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 coherencia durante la ejecución de una instrucción TILECONFIG para determinar que esta coincide con las restricciones de la entrada de paleta.

TIPOS DE ALMACENAMIENTO DE TESELA ILUSTRATIVOS

La **figura 2** ilustra varios ejemplos de almacenamiento de matriz. En (A), una tesela se almacena en memoria. Como se muestra, cada "fila" consiste en cuatro elementos de datos empaquetados. Para llegar a la siguiente "fila", se usa un valor de zancada. Obsérvese que se pueden almacenar filas de forma consecutiva en memoria. Los accesos a memoria con zancada permiten el acceso de una fila a la siguiente cuando el almacenamiento de teselas no correlaciona la anchura de fila de matriz de memoria subyacente.

Las cargas de tesela desde memoria y los almacenamientos en memoria son habitualmente accesos con zancada desde la memoria de aplicación a filas empaquetadas de datos. Las instrucciones ilustrativas TILELOAD y TILESTORE, u otras referencias de instrucción a la memoria de aplicación como un operando de TESELA en instrucciones de operación de carga son, en algunas realizaciones, reiniciables para manejar (hasta) 2 * filas de fallos de página, excepciones de coma flotante sin enmascarar 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 (una única instrucción, múltiples datos (SIMD) o registros de vector). En este ejemplo, la tesela se superpone sobre tres registros físicos. Habitualmente, se usan registros consecutivos, sin embargo, este no tiene que ser el caso.

En (C), una matriz se almacena en una tesela en almacenamiento no de registro accesible por un circuito de acumulado múltiple fusionado (FMA) usado en operaciones de teselas. Este almacenamiento puede estar dentro de un FMA o ser adyacente al mismo. Adicionalmente, 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 soportados para la arquitectura de TMMA se notifican 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 para cada tesela, el tamaño de elemento para cada tesela y el identificador de paleta. Esta configuración se hace ejecutando la instrucción TILECONFIG.

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

TILECONFIG no solo configura el uso de teselas, sino que también establece una variable de estado que indica que el programa está 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.

Salir de una región de tesela se hace habitualmente con la instrucción TILERELEASEALL. Esta no acepta parámetro alguno e invalida rápidamente todas las teselas (lo que indica que los datos ya no necesitan guardado ni restablecimiento alguno) y borra el estado interno correspondiente a estar en una región de tesela.

En algunas realizaciones, las operaciones de teselas pondrán a cero cualquier fila y cualquier columna más allá de las dimensiones especificadas por la configuración de teselas. Por ejemplo, las operaciones de teselas 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 de teselas también ponen a cero por completo cualquier fila después de las primeras 10 filas configuradas. Cuando se usa una tesela de 1 K con filas de 64 bytes, habría 16 filas, por lo que, en este ejemplo, las últimas 6 filas también se pondrían a cero.

En algunas realizaciones, una instrucción de restablecimiento de contexto (por ejemplo, XRSTOR), cuando se cargan datos, impone que los datos más allá de las filas configuradas para una tesela se mantengan como cero. Si no hay configuración válida alguna, todas las filas se ponen a cero. XRSTOR de datos de tesela 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 debido a que no hay una anchura de elemento asociada con la configuración de teselas.

Un guardado de contexto (por ejemplo, XSAVE) expone toda el área de almacenamiento de TESELA cuando se escribe la misma en memoria. Si XRSTOR cargó datos basura en la parte más a la derecha de una tesela, esos datos serán guardados por XSAVE. XSAVE escribirá ceros para las filas más allá del número especificado para cada tesela.

En algunas realizaciones, las instrucciones de tesela se pueden reiniciar. Las operaciones que acceden a memoria permiten reiniciar después de fallos de página. Las instrucciones computacionales que se ocupan de operaciones de coma flotante también permiten excepciones de coma flotante sin enmascarar, con el enmascaramiento de las excepciones controlado por un registro de control y/o de estado.

Para soportar instrucciones de reinicio después de estos sucesos, las instrucciones almacenan información en los registros de inicio detallados a continuación.

SISTEMAS DE OPERACIONES MATRICIALES (DE TESELAS)

SOPORTE DE HARDWARE ILUSTRATIVO

La **figura 3** ilustra una realización de un sistema que utiliza un acelerador de operaciones matriciales (de teselas). En esta ilustración, un sistema de procesamiento/procesador de anfitrión 301 comunica las órdenes 311 (por ejemplo, operaciones de manipulación de matrices tales como operaciones aritméticas o de manipulación de matrices, u operaciones de carga y de almacenamiento) a un acelerador de operaciones matriciales 307. Sin embargo, esto se muestra de esta forma solo con fines de análisis. Como se detalla más adelante, este acelerador 307 puede ser una parte de un núcleo de procesamiento. Habitualmente, las órdenes 311 que son instrucciones de operador de manipulación de teselas se referirán a teselas como formato de registro-registro ("reg-reg") o de registro-memoria ("reg-mem"). Otras órdenes, tales como TILESTORE, TILELOAD, TILECONFIG, etc., no realizan operaciones de datos sobre una tesela. Las órdenes pueden ser instrucciones descodificadas (por ejemplo, microoperaciones) o

macroinstrucciones para que las maneje el acelerador 307.

En este ejemplo, una interfaz de memoria coherente 303 se acopla al sistema de procesamiento/procesador de anfitrión 301 y al acelerador de operaciones matriciales 307 de tal forma que estos puedan compartir memoria. Las **figuras 4 y 5** muestran diferentes realizaciones de cómo se comparte memoria usando un acelerador de operaciones matriciales. Como se muestra en la **figura 4**, el procesador de anfitrión 401 y la circuitería de acelerador de operaciones matriciales 405 comparten la misma memoria 403. La **figura 5** ilustra una realización en la que el procesador principal 501 y el acelerador de operaciones matriciales 505 no comparten memoria, pero pueden acceder uno a la memoria del otro. Por ejemplo, el procesador 501 puede acceder a la memoria de teselas 507 y utilizar su memoria de anfitrión 503 de forma normal. De forma similar, el acelerador de operaciones matriciales 505 puede acceder a la memoria de anfitrión 503 pero, más habitualmente, usa su propia memoria 507. Obsérvese que estas memorias pueden ser de diferentes tipos.

En algunas realizaciones, el acelerador de operaciones matriciales 307 incluye una pluralidad de los FMA 309 acoplados a las memorias intermedias de datos 305 (en algunas implementaciones, una o más de estas memorias intermedias 305 se almacenan en los FMA de la cuadrícula como se muestra). Las memorias intermedias de datos 305 almacenan teselas cargadas desde memoria y/o teselas a almacenar en memoria (por ejemplo, usando una instrucción de carga de teselas o de almacenamiento de teselas). Las memorias intermedias de datos pueden ser, por ejemplo, una pluralidad de registros. Habitualmente, estos FMA se disponen como una cuadrícula de FMA 309 encadenados que son capaces de leer y de escribir teselas. En este ejemplo, el acelerador de operaciones matriciales 307 es para realizar una operación de multiplicación matricial usando las teselas T0, T1 y T2. Al menos una de las teselas se aloja 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, solo se almacena un subconjunto en la cuadrícula de FMA 309. Como se muestra, T1 está alojado y T0 y T2 no lo están. Obsérvese que A, B y C se refieren a las matrices de estas teselas que pueden ocupar, o no, todo el espacio de la tesela.

La **figura 6** ilustra una realización de una operación de acumulado de multiplicación matricial usando teselas ("TMMA").

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

El vector de origen/destino proviene de una tesela de N filas (la TESELA C 605) y la cuadrícula de los FMA 611 realiza N operaciones de matriz-vector 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 suministra términos de "radiodifusión" a los FMA en cada fase.

Durante el funcionamiento, en algunas realizaciones, los elementos de la matriz B (almacenados en una tesela B 603) se diseminan por toda la cuadrícula rectangular de los FMA. La matriz B (almacenada en la tesela A 601) tiene sus elementos de una fila transpuestos para coincidir con la dimensión de 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 fila siguiente de los FMA (o al resultado final).

La latencia de un único escalón es proporcional a K (la altura de fila de la matriz B) y las TMMA dependientes suelen tener suficientes filas de origen-destino (o bien en una única tesela o bien por toda la tesela) para ocultar esa latencia. Una implementación también puede dividir la dimensión de SIMD (elemento de datos empaquetado) M (la altura de fila de la matriz A) a través de escalones de tiempo, pero esto simplemente cambia la constante por la que se multiplica K. Cuando un programa especifica un K más pequeño que el máximo enumerado por el TMACC, una implementación es libre de implementar esto con "enmascaramiento" o "salidas tempranas".

La latencia de un TMMA completo 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 acumulado de multiplicación fusionado encadenado. En particular, esto ilustra una circuitería de ejecución de una iteración de una posición de elemento de datos empaquetado del destino. En esta realización, el acumulado de multiplicación fusionado encadenado está operando sobre unos orígenes con signo en donde el acumulador tiene 2 veces el tamaño de datos de entrada.

Un primer origen con signo (el origen 1 701) y un segundo origen con signo (el origen 2 703) tienen, 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 (el origen 3 709) tiene dos elementos de datos empaquetados, cada uno de los cuales almacena datos con signo. Los tamaños del primer y el segundo orígenes con signo 701 y 703 son la mitad de los del tercer origen con signo (valor inicial o resultado previo) 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 simple) 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, solo se muestran las dos posiciones de elemento de datos empaquetado más significativas del primer y el segundo orígenes con signo 701 y 703 y la posición de elemento de datos empaquetado más significativa del tercer origen con signo 709. Por supuesto, también se procesarían las otras posiciones de elemento de datos empaquetado.

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

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

Por último, el resultado de la segunda suma o bien se almacena en el destino con signo 715 en una posición de elemento de datos empaquetado que corresponde a la posición de elemento de datos empaquetado usada desde el tercer origen con signo 709 o bien se pasa a la iteración siguiente, si hay una. En algunas realizaciones, se aplica una máscara de escritura a este almacenamiento de tal forma que si se establece una máscara de escritura (bit) correspondiente, tiene lugar el almacenamiento y, si no se establece, no tiene lugar el almacenamiento.

La **figura 8** ilustra una realización de un subconjunto de la ejecución de una iteración de una instrucción de acumulado de multiplicación fusionado encadenado. En particular, esto ilustra una circuitería de ejecución de una iteración de una posición de elemento de datos empaquetado del destino. En esta realización, el acumulado de multiplicación fusionado encadenado está operando sobre unos orígenes con signo en donde el acumulador tiene 2 veces el tamaño de datos de entrada.

Un primer origen con signo (el origen 1 801) y un segundo origen con signo (el origen 2 803) tienen, cada uno, cuatro elementos de datos empaquetados. Cada uno de estos elementos de datos empaquetados almacena datos con signo, tales como datos enteros. Un tercer origen con signo (el origen 3 809) tiene dos elementos de datos empaquetados, cada uno de los cuales almacena datos con signo. Los tamaños del primer y el segundo orígenes con signo 801 y 803 son la mitad de 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 simple), 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, solo se muestran las dos posiciones de elemento de datos empaquetado más significativas del primer y el segundo orígenes con signo 801 y 803 y la posición de elemento de datos empaquetado más significativa del tercer origen con signo 809. Por supuesto, también se procesarían las otras posiciones de elemento de datos empaquetado.

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

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

Una saturación sin signo significa que los valores de salida están limitados a un número sin signo máximo para esa anchura de elemento (todo unos). Una 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)).

El resultado de la comprobación de suma y de saturación se almacena en el resultado con signo 815 en una posición de elemento de datos empaquetado que corresponde a la posición de elemento de datos empaquetado usada desde el tercer origen con signo 809 o bien se pasa a la iteración siguiente, si hay una. En algunas realizaciones, se aplica una máscara de escritura a este almacenamiento de tal forma que si se establece una máscara de escritura (bit) correspondiente, tiene lugar el almacenamiento y, si no se establece, no tiene lugar el almacenamiento.

La **figura 9** ilustra una realización de un subconjunto de la ejecución de una iteración de una instrucción de acumulado de multiplicación fusionado encadenado. En particular, esto ilustra una circuitería de ejecución de una iteración de una posición de elemento de datos empaquetado del destino. En esta realización, el acumulado de multiplicación fusionado encadenado está operando sobre un origen con signo y un origen sin signo en donde el acumulador tiene 4 veces el tamaño de datos de entrada.

Un primer origen con signo (el origen 1 901) y un segundo origen sin signo (el origen 2 903) tienen, cada uno, cuatro elementos de datos empaquetados. Cada uno de estos elementos de datos empaquetados tiene datos, tales como datos de coma flotante o enteros. Un tercer origen con signo (valor inicial o resultado 915) tiene un elemento de datos empaquetado que almacena datos con signo. Los tamaños del primer y el 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 elemento de datos empaquetado más significativas del primer y el segundo orígenes 901 y 903 y la posición de elemento de datos empaquetado más significativa del tercer origen con signo 915. Por supuesto, también se procesarían otras posiciones de elemento de datos empaquetado, de haber alguna.

Como se ilustra, los elementos de datos empaquetados se procesan en cuadrupletes. Por ejemplo, los datos de las posiciones de elemento de datos empaquetado más significativas del primer y el segundo orígenes 901 y 903 se multiplican usando un circuito de multiplicador 905, datos a partir de segundas posiciones de elemento de datos empaquetado más significativas del primer y el segundo orígenes 901 y 903 se multiplican usando un circuito de multiplicador 907, datos a partir de terceras posiciones de elemento de datos empaquetado más significativas del primer y el segundo orígenes 901 y 903 se multiplican usando un circuito de multiplicador 909, y datos a partir de las posiciones de elemento de datos empaquetado menos significativas del primer y el segundo orígenes 901 y 903 se multiplican usando un circuito de multiplicador 911. En algunas realizaciones, los elementos de datos empaquetados con signo del primer origen 901 se amplían con signo y los elementos de datos empaquetados sin signo del segundo origen 903 se amplían con ceros antes de las multiplicaciones.

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

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

Por último, el resultado 919 de la segunda suma o bien se almacena en el destino con signo en una posición de elemento de datos empaquetado que corresponde a la posición de elemento de datos empaquetado usada desde el tercer origen con signo 915 o bien se pasa a la iteración siguiente. En algunas realizaciones, se aplica una máscara de escritura a este almacenamiento de tal forma que si se establece una máscara de escritura (bit) correspondiente, tiene lugar el almacenamiento y, si no se establece, no tiene lugar el almacenamiento.

La **figura 10** ilustra una realización de un subconjunto de la ejecución de una iteración de una instrucción de acumulado de multiplicación fusionado encadenado. En particular, esto ilustra una circuitería de ejecución de una iteración de una posición de elemento de datos empaquetado del destino. En esta realización, el acumulado de multiplicación fusionado encadenado está operando sobre un origen con signo y un origen sin signo en donde el acumulador tiene 4 veces el tamaño de datos de entrada.

Un primer origen con signo (los elementos 1001 del origen con signo 1) y un segundo origen sin signo (los elementos 1003 del origen sin signo 2) tienen, cada uno, cuatro elementos de datos empaquetados. Cada uno de estos elementos

de datos empaquetados almacena datos, tales como datos de coma flotante o enteros. Un tercer origen con signo (resultado inicial o previo 1015) tiene un elemento de datos empaquetado que almacena datos con signo. Los tamaños del primer y el segundo orígenes 1001 y 1003 son una cuarta parte de los del tercer origen con signo (resultado inicial o previo 1015). Por ejemplo, el primer y el segundo orígenes 1001 y 1003 podrían tener elementos de datos empaquetados de 16 bits (por ejemplo, palabra) y el tercer origen con signo 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 elemento de datos empaquetado más significativas del primer y el segundo orígenes 1001 y 1003 y la posición de elemento de datos empaquetado más significativa del tercer origen con signo 1015. Por supuesto, también se procesarían otras posiciones de elemento de datos empaquetado, de haber alguna.

Como se ilustra, los elementos de datos empaquetados se procesan en cuadrupletes. Por ejemplo, los datos de las posiciones de elemento de datos empaquetado más significativas del primer y el segundo orígenes 1001 y 1003 se multiplican usando un circuito de multiplicador 1005, datos a partir de segundas posiciones de elemento de datos empaquetado más significativas del primer y el segundo orígenes 1001 y 1003 se multiplican usando un circuito de multiplicador 1007, datos a partir de terceras posiciones de elemento de datos empaquetado más significativas del primer y el segundo orígenes 1001 y 1003 se multiplican usando un circuito de multiplicador 1009, y datos a partir de las posiciones de elemento de datos empaquetado menos significativas del primer y el segundo orígenes 1001 y 1003 se multiplican usando un circuito de multiplicador 1011. En algunas realizaciones, los elementos de datos empaquetados con signo del primer origen 1001 se amplían con signo y los elementos de datos empaquetados sin signo del segundo origen 1003 se amplían con ceros antes de las multiplicaciones.

En algunas realizaciones, estos circuitos de multiplicador 1005-1011 se reutilizan para otras posiciones de elemento de datos empaquetado. En otras realizaciones, se usan circuitos de multiplicador adicionales de tal modo que los elementos de datos empaquetados se procesen en paralelo. En algunos contextos, se realiza una ejecución en paralelo usando carriles que tienen el tamaño del tercer origen con signo 1015. El resultado de la suma de los resultados de las multiplicaciones se suma a los datos a partir de la posición de elemento de datos empaquetado más significativa del origen con signo 3 1015 usando la circuitería de sumador/saturación 1013.

La circuitería de suma/saturación (acumulador) 1013 conserva un signo de un operando cuando la suma da como resultado un valor que es demasiado grande o demasiado pequeño para una saturación con signo. En particular, una evaluación de saturación tiene lugar sobre el resultado de precisión infinita entre la adición multidireccional y la escritura en el destino. Cuando el acumulador 1013 es de coma flotante y los términos de entrada son enteros, la suma de productos y el valor de entrada de 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 multiplicación y la tercera entrada y se realiza un único redondeo al tipo de acumulador real.

El resultado 1019 de la comprobación de suma y de saturación se almacena en el destino con signo en una posición de elemento de datos empaquetado que corresponde a la posición de elemento de datos empaquetado usada desde el tercer origen con signo 1015 o bien se pasa a la iteración siguiente. En algunas realizaciones, se aplica una máscara de escritura a este almacenamiento de tal forma que si se establece una máscara de escritura (bit) correspondiente, tiene lugar el almacenamiento y, si no se establece, no tiene lugar el almacenamiento.

La **figura 11** ilustra implementaciones de SIMD de tamaño 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 el origen (a los multiplicadores) y los valores de acumulador pueden ser valores con signo o sin signo. Para un acumulador que tiene unos tamaños de entrada 2X (en otras palabras, el valor de entrada de acumulador es el doble del tamaño de los tamaños de elemento de datos empaquetado de los orígenes), la tabla 1101 ilustra diferentes configuraciones. Para orígenes de tamaño de byte, el acumulador usa valores de palabra o de coma flotante de precisión mitad (HPFP) que tienen un tamaño de 16 bits. Para orígenes de tamaño de palabra, el acumulador usa valores de coma flotante de precisión simple (SPFP) o enteros de 32 bits que tienen un tamaño de 32 bits. Para orígenes de tamaño entero de 32 bits o de SPFP, el acumulador usa valores de coma flotante de precisión doble (DPFP) o enteros de 64 que tienen un tamaño de 64 bits.

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

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

Como se ha sugerido anteriormente, una circuitería de operaciones matriciales se puede incluir en un núcleo o como un acelerador externo. La **figura 12** ilustra una realización de un sistema que utiliza una circuitería de operaciones matriciales. En esta ilustración, múltiples entidades se acoplan con una interconexión de anillo 1245.

Una pluralidad de núcleos 1201, 1203, 1205 y 1207 proporcionan un soporte de instrucciones no basado en teselas. En algunas realizaciones, la circuitería de operaciones matriciales 1251 se proporciona en un núcleo 1203 y, en otras realizaciones, las circuiterías de operaciones matriciales 1211 y 1213 son accesibles en la interconexión de anillo 1245.

Adicionalmente, se proporcionan 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 matriciales.

La **figura 13** ilustra una realización de una canalización de núcleo de procesador que soporta operaciones matriciales usando teselas. La circuitería de predicción de bifurcaciones y de descodificación 1303 realiza una predicción de bifurcaciones de instrucciones, una descodificación de instrucciones, y/o ambas, a partir de unas instrucciones almacenadas en el almacenamiento de instrucciones 1301. Por ejemplo, instrucciones detalladas en el presente documento se pueden almacenar en un almacenamiento de instrucciones. En algunas implementaciones, se usa una circuitería separada para la predicción de bifurcaciones y, en algunas realizaciones, al menos algunas instrucciones se descodifican 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 bifurcaciones y de descodificación 1303 se puede implementar usando diversos mecanismos diferentes. Los ejemplos de mecanismos adecuados incluyen, pero no se limitan a, tablas de consulta, implementaciones en hardware, matrices lógicas programables (PLA), memorias de solo lectura (ROM) de microcódigo, etc.

La circuitería de predicción de bifurcaciones y de descodificación 1303 se acopla a la circuitería de asignación/cambio de nombre 1307 que se acopla, en algunas realizaciones, a la circuitería de programador 1309. En algunas realizaciones, estos circuitos proporcionan funcionalidades de cambio de nombre de registro, de asignación de registro y/o de programación realizando uno o más de: 1) cambiar de nombre unos valores de operando lógico a valores de operando físico (por ejemplo, una tabla de alias de registro en algunas realizaciones), 2) asignar unos bits de estado y unos indicadores a la instrucción descodificada, y 3) programar la instrucción descodificada 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 programador 1309 representa cualquier número de programadores diferentes, incluyendo estaciones de reservas, ventana de instrucciones central, etc. La circuitería de programador 1309 se acopla a, o incluye, un 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, unos diferentes de los cuales almacenan uno o más tipos de datos diferentes, tales como entero escalar, coma flotante escalar, entero empaquetado, coma flotante empaquetada, entero vectorial, coma flotante vectorial, estado (por ejemplo, un puntero de instrucción que es la dirección de la siguiente instrucción a ejecutar), teselas, etc. En una realización, el archivo o archivos de registro físico 1315 comprenden una circuitería de registros de vector, una circuitería de registros de máscara de escritura y una circuitería de registros de escalar. Estos circuitos de registro pueden proporcionar registros de vector arquitectónicos, registros de máscara de vector y registros de propósito general. El archivo o archivos de registro físico 1315 son solapados por un circuito de retiro 1317 para ilustrar diversas formas en las que se pueden implementar un cambio de nombre de registro y una ejecución fuera de orden (por ejemplo, usando una memoria intermedia o memorias intermedias de reordenación y un archivo o archivos de registro de retiro; usando un archivo o archivos futuros, una memoria o memorias de historial y un archivo o archivos de registro de retiro; usando correlaciones de registro y una agrupación de registros; etc.). El circuito de retiro 1317 y el archivo o archivos de registro físico 1315 se acoplan a la circuitería de ejecución 1311.

Aunque el cambio de nombre de registros se describe en el contexto de una ejecución fuera de orden, se debería entender que el cambio de nombre de registros se puede usar en una arquitectura en orden. Aunque la realización ilustrada del procesador también puede incluir unidades de memoria caché de instrucciones y de datos separadas y una unidad de memoria caché de L2 compartida, realizaciones alternativas pueden tener una única memoria caché interna tanto para instrucciones como para datos, tal como, por ejemplo, una memoria caché interna de nivel 1 (L1) o múltiples niveles de memoria 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 memoria 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, incluyendo la circuitería de escalares 1321, la circuitería de vectores/SIMD 1323 y la circuitería de operaciones matriciales 1327, así como la circuitería de acceso a memoria 1325. Los circuitos de ejecución 1321, 1323 y 1327 realizan 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). Aunque algunas realizaciones pueden incluir un número de unidades de ejecución dedicadas a funciones o conjuntos de funciones específicos, otras realizaciones pueden incluir solo una unidad de ejecución o múltiples unidades de ejecución que realizan, todas ellas, todas las funciones. La circuitería de escalares 1321 realiza operaciones escalares, la circuitería de vectores/SIMD 1323 realiza operaciones vectoriales/SIMD, y la circuitería de operaciones matriciales 1327 realiza

operaciones matriciales (de teselas) detalladas en el presente documento.

A modo de ejemplo, la arquitectura de núcleo de cambio de nombre de registro, emisión/ejecución fuera de orden ilustrativa puede implementar una canalización como sigue: 1) un circuito de recuperación de instrucciones realiza unas fases de recuperación y de descodificación de longitud; 2) la circuitería de bifurcación y de descodificación 1303 realiza una fase de descodificación; 3) la circuitería de cambio de nombre/asignación 1307 realiza una fase de asignación y una fase de cambio de nombre; 4) la circuitería de programador 1309 realiza una fase de programación; 5) un archivo o archivos de registro físico (acoplados a, o incluidos en, la circuitería de programador 1309 y la circuitería de cambio de nombre/asignación 1307 y una unidad de memoria realizan una fase de lectura de registro/lectura de memoria; la circuitería de ejecución 1311 realiza una fase de ejecución; 6) una unidad de memoria y la unidad o unidades de archivo o archivos de registro físico realizan una fase de escritura no simultánea/escritura en memoria; 7) diversas unidades pueden estar implicadas en la fase de manejo de excepciones; y 8) una unidad de retiro y la unidad o unidades de archivo o archivos de registro físico realizan una fase de confirmación.

El núcleo puede soportar uno o más conjuntos de instrucciones (por ejemplo, el conjunto de instrucciones de x86 (con algunas ampliaciones que se han añadido con versiones más nuevas); el conjunto de instrucciones MIPS de MIPS Technologies de Sunnyvale, CA; el conjunto de instrucciones ARM (con ampliaciones adicionales opcionales tales 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 lógica para soportar una ampliación de conjunto de instrucciones de datos empaquetados (por ejemplo, AVX1, AVX2), permitiendo de ese modo que las operaciones usadas por muchas aplicaciones multimedios se realicen usando datos empaquetados.

Se debería entender que el núcleo puede soportar multiproceso (ejecutar dos o más conjuntos paralelos de operaciones o subprocesos), y puede hacer esto de una diversidad de formas, incluyendo multiproceso segmentado en el tiempo, multiproceso simultáneo (en donde un único núcleo físico proporciona un núcleo lógico para cada uno de los subprocesos para los que ese núcleo físico está sometiendo a multiproceso simultáneamente), o una combinación de los mismos (por ejemplo, recuperación y descodificación segmentadas en el tiempo y multiproceso simultáneo a continuación de lo anterior, tal como en la tecnología Hyperthreading de Intel®).

La **figura 14** ilustra una realización de una canalización de núcleo de procesador que soporta operaciones matriciales usando teselas. La circuitería de predicción de bifurcaciones y de descodificación 1403 realiza una predicción de bifurcaciones de instrucciones, una descodificación de instrucciones, y/o ambas, a partir de unas instrucciones almacenadas en el almacenamiento de instrucciones 1401. Por ejemplo, instrucciones detalladas en el presente documento se pueden almacenar en un almacenamiento de instrucciones. En algunas implementaciones, se usa una circuitería separada para la predicción de bifurcaciones y, en algunas realizaciones, al menos algunas instrucciones se descodifican 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 bifurcaciones y de descodificación 1403 se puede implementar usando diversos mecanismos diferentes. Los ejemplos de mecanismos adecuados incluyen, pero no se limitan a, tablas de consulta, implementaciones en hardware, matrices lógicas programables (PLA), memorias de solo lectura (ROM) de microcódigo, etc.

La circuitería de predicción de bifurcaciones y de descodificación 1403 se acopla a una circuitería de cambio de nombre/asignación 1407 que se acopla, en algunas realizaciones, a la circuitería de programador 1409. En algunas realizaciones, estos circuitos proporcionan funcionalidades de cambio de nombre de registro, de asignación de registro y/o de programación realizando uno o más de: 1) cambiar de nombre unos valores de operando lógico a valores de operando físico (por ejemplo, una tabla de alias de registro en algunas realizaciones), 2) asignar unos bits de estado y unos indicadores a la instrucción descodificada, y 3) programar la instrucción descodificada 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 programador 1409 representa cualquier número de programadores diferentes, incluyendo estaciones de reservas, ventana de instrucciones central, etc. La circuitería de programador 1409 de la unidad o unidades de programador se acopla a, o incluye, un 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, unos diferentes de los cuales almacenan uno o más tipos de datos diferentes, tales como entero escalar, coma flotante escalar, entero empaquetado, coma flotante empaquetada, entero vectorial, coma flotante vectorial, estado (por ejemplo, un puntero de instrucción que es la dirección de la siguiente instrucción a ejecutar), teselas, etc. En una realización, el archivo o archivos de registro físico 1415 comprenden una circuitería de registros de vector, una circuitería de registros de máscara de escritura y una circuitería de registros de escalar. Estos circuitos de registro pueden proporcionar registros de vector arquitectónicos, registros de máscara de vector y registros de propósito general. El archivo o archivos de registro físico 1415 son solapados por un circuito de retiro 1417 para ilustrar diversas formas en las que se pueden implementar un cambio de nombre de registro y una ejecución fuera de orden (por ejemplo, usando una memoria intermedia o memorias intermedias de reordenación y un archivo o archivos de registro de retiro; usando un archivo o archivos futuros, una memoria o memorias de historial y un archivo o archivos de registro de retiro; usando correlaciones de registro y una agrupación de registros; etc.). El circuito de retiro 1417 y el archivo o archivos de registro físico 1415 se acoplan al circuito o circuitos de ejecución 1411.

Aunque el cambio de nombre de registros se describe en el contexto de una ejecución fuera de orden, se debería entender que el cambio de nombre de registros se puede usar en una arquitectura en orden. Aunque la realización ilustrada del procesador también puede incluir unidades de memoria caché de instrucciones y de datos separadas y una unidad de memoria caché de L2 compartida, realizaciones alternativas pueden tener una única memoria caché interna tanto para instrucciones como para datos, tal como, por ejemplo, una memoria caché interna de nivel 1 (L1) o múltiples niveles de memoria 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 memoria caché puede ser externa al núcleo y/o al procesador.

La circuitería de ejecución 1411 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 matriciales (de teselas) detalladas en el presente documento.

A modo de ejemplo, la arquitectura de núcleo de cambio de nombre de registro, emisión/ejecución fuera de orden ilustrativa puede implementar una canalización como sigue: 1) un circuito de recuperación de instrucciones realiza unas fases de recuperación y de descodificación de longitud; 2) la circuitería de bifurcación y de descodificación 1403 realiza una fase de descodificación; 3) la circuitería de cambio de nombre/asignación 1407 realiza una fase de asignación y una fase de cambio de nombre; 4) la circuitería de programador 1409 realiza una fase de programación; 5) un archivo o archivos de registro físico (acoplados a, o incluidos en, la circuitería de programador 1407 y la circuitería de cambio de nombre/asignación 1407 y una unidad de memoria realizan una fase de lectura de registro/lectura de memoria; la circuitería de ejecución 1411 realiza una fase de ejecución; 6) una unidad de memoria y la unidad o unidades de archivo o archivos de registro físico realizan una fase de escritura no simultánea/escritura en memoria; 7) diversas unidades pueden estar implicadas en la fase de manejo de excepciones; y 8) una unidad de retiro y la unidad o unidades de archivo o archivos de registro físico realizan una fase de confirmación.

El núcleo puede soportar uno o más conjuntos de instrucciones (por ejemplo, el conjunto de instrucciones de x86 (con algunas ampliaciones que se han añadido con versiones más nuevas); el conjunto de instrucciones MIPS de MIPS Technologies de Sunnyvale, CA; el conjunto de instrucciones ARM (con ampliaciones adicionales opcionales tales 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 lógica para soportar una ampliación de conjunto de instrucciones de datos empaquetados (por ejemplo, AVX1, AVX2), permitiendo de ese modo que las operaciones usadas por muchas aplicaciones multimedios se realicen usando datos empaquetados.

Se debería entender que el núcleo puede soportar multiproceso (ejecutar dos o más conjuntos paralelos de operaciones o subprocesos), y puede hacer esto de una diversidad de formas, incluyendo multiproceso segmentado en el tiempo, multiproceso simultáneo (en donde un único núcleo físico proporciona un núcleo lógico para cada uno de los subprocesos para los que ese núcleo físico está sometiendo a multiproceso simultáneamente), o una combinación de los mismos (por ejemplo, recuperación y descodificación segmentadas en el tiempo y multiproceso simultáneo a continuación de lo anterior, tal como en la tecnología Hyperthreading de Intel®).

B. DISTRIBUCIÓN

De principio a fin de esta descripción, se expresan datos usando una distribución de datos ordenados por filas. Usuarios de un orden por columnas deberían traducir los términos de acuerdo con su orientación. La **figura 15** ilustra un ejemplo de una matriz expresada en un formato ordenado por filas y en un formato ordenado por columnas. Como se muestra, la matriz A es una matriz 2 x 3. Cuando esta matriz se almacena en un formato ordenado por filas, los elementos de datos de una fila son consecutivos. Cuando esta matriz se almacena en un formato ordenado por columnas, los elementos de datos de una columna son consecutivos. Es una propiedad bien conocida de las matrices que $A^T * B^T = (BA)^T$, en donde el superíndice T significa traspuesta. Leer datos ordenados por columnas como datos ordenados por filas da como resultado que la matriz parezca la matriz transpuesta.

En algunas realizaciones, se utilizan semánticas ordenadas por filas en el hardware, y los datos ordenados por columnas son para intercambiar el orden de operandos, siendo el resultado traspuestas de matriz, pero para lecturas ordenadas por columnas posteriores desde memoria, esta es la matriz no transpuesta, que es la correcta.

Por ejemplo, si hay que multiplicar dos matrices ordenadas por columnas:

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
(3 x 2)	(2 x 3)	(3 x 3)

Las matrices de entrada se almacenarían en una memoria lineal (ordenadas por columnas) como:

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

Leyendo esas matrices como ordenadas por filas con unas dimensiones 2 x 3 y 3 x 2, estas aparecerían como:

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

Intercambiando el orden y multiplicando matricialmente:

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 traspuesta está fuera y se puede almacenar entonces en orden por filas:

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

y se usa en cálculos con un orden por columnas posteriores, es la matriz no transpuesta correcta:

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

USO ILUSTRATIVO

La **figura 16** ilustra un ejemplo de uso de matrices (teselas). En este ejemplo, la matriz C 1601 incluye dos teselas, la matriz A 1603 incluye una tesela y la matriz B 1605 incluye dos teselas. Esta figura muestra un ejemplo del bucle interno de un algoritmo para calcular una multiplicación matricial. En este ejemplo, se usan dos teselas de resultado, tmm0 y tmm1, a partir de la matriz C 1601 para acumular los resultados intermedios. Una tesela a partir de la matriz A 1603 (tmm2) se reutiliza dos veces cuando es multiplicada por dos teselas a partir de la matriz B 1605. 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 para 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 memoria y liberar el uso de teselas.

La **figura 17** ilustra una realización de uso de matrices (teselas). En 1701, se configura un uso de teselas. Por ejemplo, se ejecuta una instrucción TILECONFIG para configurar el uso de teselas, incluyendo establecer un número de filas y columnas por tesela. Habitualmente, al menos una matriz (tesela) se carga desde memoria en 1703. Al menos una operación matricial (de teselas) se realiza en 1705 usando las matrices (teselas). En 1707, al menos una matriz (tesela) se almacena fuera en memoria y una conmutación de contexto puede tener lugar en 1709.

CONFIGURACIÓN ILUSTRATIVA

SOPORTE DE HARDWARE DE CONFIGURACIÓN DE TESELAS

Como se ha analizado anteriormente, habitualmente es necesario configurar el uso de teselas antes de su uso. Por ejemplo, puede que no sea necesario el uso completo de todas las filas y columnas. Configurar estas filas y columnas no solo ahorra energía en algunas realizaciones, sino que la configuración se puede usar para determinar si una operación generará un error. Por ejemplo, habitualmente una multiplicación matricial de la forma (N x M) * (L x N) no funcionará si M y L no son iguales.

Antes de usar matrices que usan teselas, en algunas realizaciones, se ha de configurar un soporte de teselas. Por ejemplo, se configuran cuántas filas y columnas por tesela, teselas que se van a usar, etc. Una instrucción TILECONFIG es, en sí misma, una mejora para un ordenador, debido a que esta prevé soporte para configurar el ordenador para usar un acelerador de matrices (o bien como una parte de un núcleo de procesador o bien como un

dispositivo externo). En particular, una ejecución de la instrucción TILECONFIG hace que una configuración se recupere desde memoria y se aplique a ajustes de matriz (tesela) dentro de un acelerador de matrices.

CONFIGURACIÓN DE USO DE TESELAS

La **figura 18** ilustra un soporte para la configuración del uso de teselas de acuerdo con una realización. Una memoria 1801 contiene la descripción de las matrices (teselas) a soportar 1803.

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

La **figura 19** ilustra una realización de una descripción de las matrices (teselas) 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 un 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 asociadas con este ID como es definido por la configuración.

El byte 1 almacena un valor a almacenar en un registro de "filaInicio" 1903 y el byte 2 almacena un valor a almacenar en un registro de "PInicio" 1905. Para soportar instrucciones de reinicio después de estos sucesos, las instrucciones almacenan información en estos registros. Para soportar instrucciones de reinicio después de sucesos de interrupción tales como los detallados anteriormente, las instrucciones almacenan información en estos registros. El valor de filaInicio indica la fila que debería usarse para un reinicio. El valor de PInicio 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). En general, la posición en la fila (la columna) no es necesaria.

Con la excepción de TILECONFIG y STTILECFG, ejecutar con éxito instrucciones de matrices (teselas) establecerá tanto filaInicio como PInicio a cero.

Cada vez que no se reinicia una instrucción de matrices (teselas) interrumpida, es responsabilidad del software poner a cero los valores de filaInicio y de PInicio. Por ejemplo, los manejadores de excepciones de coma flotante sin enmascarar podrían decidir finalizar la operación en software y cambiar el valor de contador de programa a otra instrucción, habitualmente la instrucción siguiente. En este caso, el controlador de excepciones de software debe poner a cero los valores de filaInicio y de PInicio en la excepción presentada al mismo por el sistema operativo antes de reanudar el programa. Posteriormente, el sistema operativo recargará esos valores usando una instrucción de restablecimiento.

El byte 3 almacena una indicación de pares (1b por tesela) de 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 parámetros de tesela, estos deberían tener el valor cero. Especificar parámetros de tesela para más teselas que el límite de implementación o el límite de paleta da como resultado un error. Las teselas no configuradas se establecen a un estado inicial con 0 filas, 0 columnas.

Por último, la configuración en memoria termina habitualmente con una delineación final tal como todo ceros durante varios bytes consecutivos.

TESELA Y ALMACENAMIENTO DE CONFIGURACIÓN DE TESELAS ILUSTRATIVOS

Las **figuras 20(A)-(D)** ilustran ejemplos de un registro o registros 1819. La **figura 20(A)** ilustra una pluralidad de registros 1819. Como se muestra, cada tesela (TMM0 2001 ... TMMN 2003) tiene un registro separado, almacenando cada registro un tamaño de fila y de columna para esa tesela particular. PInicio y FilaInicio se almacenan en unos registros 2011 y 2013 separados. Se establecen uno o más registros de estado 2015 (por ejemplo, TESELAS_CONFIGURADAS = 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 de TMM0 2021, la configuración de columnas de TMM0 2023, PInicio y FilaInicio se almacenan en unos registros 2011 y 2013 separados. Se establecen uno o más registros

de estado 2015 (por ejemplo, TESELAS_CONFIGURADAS = 1) para indicar que las teselas están configuradas para su uso.

La **figura 20(C)** ilustra un único registro 1819. Como se muestra, este registro almacena las configuraciones de teselas (filas y columnas por tesela) 2031, PInicio 2011 y FilaInicio 2013 se almacenan en un único registro como registros de datos empaquetados. Se establecen uno o más registros de estado 2015 (por ejemplo, TESELAS_CONFIGURADAS = 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 único registro almacena la configuración de las teselas (filas y columnas por tesela) 2031. PInicio y FilaInicio se almacenan en unos registros 2011 y 2013 separados. Se establecen uno o más registros de estado 2015 (por ejemplo, TESELAS_CONFIGURADAS = 1) para indicar que las teselas están configuradas para su uso.

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

GENERACIÓN DE ÍNDICE Y ORIGEN

Realizaciones divulgadas describen instrucciones para realizar una generación de índice por orden de clasificación y reordenar elementos basándose en el índice. Como se detalla a continuación, en algunas realizaciones, la generación y la reordenación de índice se realizan en respuesta a una única instrucción, mientras que, en otras realizaciones, se utilizan al menos dos instrucciones.

En algunas realizaciones, el uso de índices por orden de clasificación y la reordenación pueden aumentar las posibilidades de que rayos vecinos de un trazado de rayos (es decir, los que es más probable que se agrupen en una deformación/vector de SIMD) tomen la misma ruta a través de un espacio físico. Por lo tanto, a medida que se traza su progreso, es probable que estos rayos incidan sobre los mismos objetos, usen las mismas texturas, etc., lo que puede mejorar la localidad de memoria y la divergencia de bifurcación.

La **figura 21** ilustra realizaciones de ejecución de una única instrucción de clasificación de índice y de reordenación. En esta ilustración, la instrucción de clasificación de índice y de reordenación (mnemotécnico de código de operación clasificación-índice-reordenación) incluye campos para identificar una ubicación de destino y una ubicación de origen. El destino y el origen son cada uno para almacenar un vector (contenidos o bien de un registro de vector o bien de una ubicación de memoria). Una o más de las circuiterías 2103 y 2107 mostradas en la figura pueden ser una parte de una circuitería de operaciones matriciales (tal como el que se detalla en el presente documento), una circuitería de vectores/simd (tal como la que se detalla en el presente documento) o un coprocesador (tal como una GPU capaz de realizar una rasterización o trazado de rayos).

El origen 2101 proporciona un vector de elementos de datos a la circuitería de índice 2103. La circuitería de índice 2103 genera un índice 2105 de valores clasificados del origen. La generación del índice 2105 se puede realizar usando varios enfoques, tales como uno basado en comparaciones (tales como usar una o más comparaciones usando mayor que, mayor que o igual a, igual a, menor que y menor que o igual a). En otras figuras se detallan ejemplos de enfoques basados en comparaciones. El tipo o tipos de comparación o comparaciones pueden ser establecidos por el código de operación, en un valor inmediato, etc.

La circuitería de permutación 2107 capta el índice 2105 y permuta los valores del origen 2101 basándose en el índice 2105. Como se muestra, el elemento de datos 0 del origen 2101 tiene un valor de 13. El índice 2105 en el elemento de datos 0 (la posición de elemento de datos correspondiente) tiene un valor de 0, lo que indica a la circuitería de permutación que almacene 13 en el elemento de datos 0 del destino 2109. Los otros valores del origen 2101 se permutan al destino 2109 de acuerdo con el índice. Obsérvese que el orden de bits del origen 2101, el índice 2105 y el 2109 son diferentes en algunas realizaciones.

La **figura 22** ilustra realizaciones de ejecución de una única instrucción de clasificación de índice y una única instrucción de reordenación. En esta ilustración, la instrucción de clasificación de índice (mnemotécnico de código de operación clasificación-índice) incluye campos para identificar una ubicación de destino y una ubicación de origen. El destino y el origen son cada uno para almacenar un vector (contenidos o bien de un registro de vector o bien de una ubicación de memoria). Una o más de las circuiterías 2203 y 2207 mostradas en la figura pueden ser una parte de una circuitería de operaciones matriciales (tal como el que se detalla en el presente documento), una circuitería de vectores/simd (tal como la que se detalla en el presente documento) o un coprocesador (tal como una GPU capaz de realizar una rasterización o trazado de rayos).

El origen 2201 proporciona un vector de elementos de datos a la circuitería de índice 2203. La circuitería de índice 2203 genera un índice de valores clasificados del origen almacenados en el destino 2205. La generación del índice se puede realizar usando varios enfoques, tales como uno basado en comparaciones (tales como usar una o más comparaciones usando mayor que, mayor que o igual a, igual a, menor que y menor que o igual a). En otras figuras se detallan ejemplos de enfoques basados en comparaciones. El tipo o tipos de comparación o comparaciones pueden

ser establecidos por el código de operación, en un valor inmediato, etc.

La instrucción de reordenación (mnemotécnico de código de operación reordenación) incluye campos para identificar una ubicación de destino y dos ubicaciones de origen. El destino y los orígenes son cada uno para almacenar un vector (contenidos o bien de un registro de vector o bien de una ubicación de memoria). Una o más de las circuiterías 2203 y 2207 mostradas en la figura pueden ser una parte de una circuitería de operaciones matriciales (tal como el que se detalla en el presente documento), una circuitería de vectores/simd (tal como la que se detalla en el presente documento) o un coprocesador (tal como una GPU capaz de realizar una rasterización o trazado de rayos).

La circuitería de permutación 2207 capta el origen 2211 que era el índice almacenado en el destino 2205 y permuta los valores del origen 2246 (los mismos valores del origen 2201, o el mismo origen que el origen 2201) basándose en el índice. Como se muestra, el elemento de datos 0 del origen 2246 tiene un valor de 13. El índice en el elemento de datos 0 (la posición de elemento de datos correspondiente) tiene un valor de 0, lo que indica a la circuitería de permutación que almacene 13 en el elemento de datos 0 del destino 2209. Los otros valores del origen 2246 se permutan al destino 2209 de acuerdo con el índice. Obsérvese que el orden de bits del origen u orígenes 2201, 2221, 2246; y los destinos 2205 y 2209 son diferentes en algunas realizaciones.

La **figura 23** ilustra realizaciones de una circuitería para generar un índice. Como se muestra, la circuitería de comparación 2303 es para comparar cada elemento de la entrada 2301 (origen de las dos figuras previas) consigo mismo y con otros elementos de la entrada 2301. En algunas realizaciones, la circuitería de comparación 2303 es una cuadrícula de ALU canalizadas. La comparación puede ser uno de muchos tipos, incluyendo, pero sin limitarse a: mayor que, mayor que o igual a, igual a, menor que y menor o igual a. En este ejemplo, las comparaciones son todas mayores que. Cuando es verdadero, el resultado es un 1 y, cuando es falso, el resultado es un 0. Los resultados de cada comparación se suman para generar un valor de índice almacenado en el índice 2305. Como se muestra, la posición de elemento de datos 0 almacena un 13 y 13 no es mayor que ninguno de los elementos y, por lo tanto, cada comparación es falsa y la suma de cuatro 0 es 0, y se almacena 0 como el índice para esa posición de elemento de datos. Los tipos de comparaciones pueden ser establecidos por el código de operación, en un valor inmediato, etc.

La **figura 24** ilustra realizaciones de una circuitería para generar un índice. Como se muestra, la circuitería de comparación 2403 es para comparar cada elemento de la entrada 2401 (origen de las dos figuras previas) consigo mismo y con otros elementos de la entrada 2401. En algunas realizaciones, la circuitería de comparación 2403 es una cuadrícula de ALU canalizadas. La comparación puede ser uno de muchos tipos, incluyendo, pero sin limitarse a: mayor que, mayor que o igual a, igual a, menor que y menor o igual a.

En este ejemplo, se realiza una primera comparación de mayor que (véanse las líneas de trazo discontinuo verticales). Cuando es verdadero, el resultado es un 1 y, cuando es falso, el resultado es un 0. Los resultados de cada comparación se suman para generar un valor de índice almacenado en el índice 2405. Como se muestra, la posición de elemento de datos 0 almacena un 13 y 13 no es mayor que ninguno de los elementos y, por lo tanto, cada comparación es falsa y la suma de cuatro 0 es 0, y se almacena 0 como el índice para esa posición de elemento de datos.

Desafortunadamente, puede haber ocasiones en las que dos o más sumas son iguales y se hacen comparaciones adicionales (mostradas en líneas de trazo discontinuo horizontales y, en este ejemplo, mayor que igual a). Para cualesquiera dos elementos que se clasifican, hay dos comparaciones. En una de esas pruebas de este ejemplo, hay una prueba para elemento1 > elemento2 y, en el otro caso, se somete a prueba elemento2 >= elemento1. En un empate, uno de estos será verdadero y el otro será falso.

Como se muestra, hay dos 17. La columna para el 17 más a la izquierda verá una prueba '>' fallida contra el otro 17, mientras que el 17 más a la derecha verá un '>=' con éxito contra el otro 17. Los tipos de comparaciones pueden ser establecidos por el código de operación, en un valor inmediato, etc.

Métodos ilustrativos

La **figura 36** ilustra una realización del procesamiento de una única instrucción. El procesamiento puede ser realizado, en su totalidad o en parte, por una o más de una circuitería de operaciones matriciales (tal como el que se detalla en el presente documento), una circuitería de vectores/simd (tal como la que se detalla en el presente documento) o un coprocesador (tal como una GPU capaz de realizar una rasterización o trazado de rayos).

En 3601, una circuitería de recuperación recupera una instrucción. La instrucción incluye un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice usando una o más comparaciones del elemento consigo mismo y con otros elementos del vector de origen, y permutando valores de los elementos del vector de origen basándose en el índice y almacenando los valores permutados.

En 3603, la circuitería de descodificación descodifica la instrucción recuperada. La instrucción descodificada se usa

para configurar una circuitería de ejecución para ejecutar la instrucción descodificada para realizar los actos detallados anteriormente. En algunas realizaciones, la descodificación incluye generar una o más microoperaciones. Obsérvese que, debido a que esta instrucción no ha existido antes, no hay circuitería de descodificación alguna capaz de descodificar esta instrucción de una forma que permita que la circuitería de ejecución ejecute las acciones anteriores de forma apropiada.

La ejecución de la instrucción descodificada se programa (según sea necesario) 3605, que es una etapa opcional (como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

En 3607, la instrucción descodificada (tal como microoperaciones) se ejecuta usando una circuitería de ejecución para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice usando una o más comparaciones del elemento consigo mismo y con otros elementos del vector de origen, y permutando valores de los elementos del vector de origen basándose en el índice y almacenando los valores permutados en el vector de destino.

En algunas realizaciones, la instrucción ejecutada se confirma o se retira en 3609, que es una etapa opcional (como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

La **figura 37** ilustra una realización del procesamiento de una única instrucción. El procesamiento puede ser realizado, en su totalidad o en parte, por una o más de una circuitería de operaciones matriciales (tal como el que se detalla en el presente documento), una circuitería de vectores/simd (tal como la que se detalla en el presente documento) o un coprocesador (tal como una GPU capaz de realizar una rasterización o trazado de rayos).

En 3701, una circuitería de recuperación recupera una instrucción. La instrucción incluye un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice, y permutando los valores de los elementos del vector de origen en el vector de destino basándose en los valores de índice para los elementos.

En 3703, la circuitería de descodificación descodifica la instrucción recuperada. La instrucción descodificada se usa para configurar una circuitería de ejecución para ejecutar la instrucción descodificada para realizar los actos detallados anteriormente. En algunas realizaciones, la descodificación incluye generar una o más microoperaciones. Obsérvese que, debido a que esta instrucción no ha existido antes, no hay circuitería de descodificación alguna capaz de descodificar esta instrucción de una forma que permita que la circuitería de ejecución ejecute las acciones anteriores de forma apropiada.

La ejecución de la instrucción descodificada se programa (según sea necesario) 3705, que es una etapa opcional (como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

En 3707, la instrucción descodificada (tal como microoperaciones) se ejecuta usando una circuitería de ejecución para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice, y permutando los valores de los elementos del vector de origen en el vector de destino basándose en los valores de índice para los elementos.

En algunas realizaciones, la instrucción ejecutada se confirma o se retira en 3709, que es una etapa opcional (como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

La **figura 38** ilustra una realización del procesamiento de una única instrucción. El procesamiento puede ser realizado, en su totalidad o en parte, por una o más de una circuitería de operaciones matriciales (tal como el que se detalla en el presente documento), una circuitería de vectores/SIMD (tal como la que se detalla en el presente documento) o un coprocesador (tal como una GPU capaz de realizar una rasterización o trazado de rayos).

En 3801, una circuitería de recuperación recupera una instrucción. La instrucción incluye un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para indexar valores del vector de origen y almacenar un resultado de la indexación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice usando una o más comparaciones.

En 3803, la circuitería de descodificación descodifica la instrucción recuperada. La instrucción descodificada se usa para configurar una circuitería de ejecución para ejecutar la instrucción descodificada para realizar los actos detallados anteriormente. En algunas realizaciones, la descodificación incluye generar una o más microoperaciones. Obsérvese

que, debido a que esta instrucción no ha existido antes, no hay circuitería de descodificación alguna capaz de descodificar esta instrucción de una forma que permita que la circuitería de ejecución ejecute las acciones anteriores de forma apropiada.

La ejecución de la instrucción descodificada se programa (según sea necesario) 3805, que es una etapa opcional (como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

En 3807, la instrucción descodificada (tal como microoperaciones) se ejecuta usando una circuitería de ejecución para indexar valores del vector de origen y almacenar un resultado de la indexación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice usando una o más comparaciones.

En algunas realizaciones, la instrucción ejecutada se confirma o se retira en 3809, que es una etapa opcional (como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

La **figura 39** ilustra una realización del procesamiento de una única instrucción. El procesamiento puede ser realizado, en su totalidad o en parte, por una o más de una circuitería de operaciones matriciales (tal como el que se detalla en el presente documento), una circuitería de vectores/SIMD (tal como la que se detalla en el presente documento) o un coprocesador (tal como una GPU capaz de realizar una rasterización o trazado de rayos).

En 3901, una circuitería de recuperación recupera una instrucción. La instrucción incluye un primer campo para identificar una ubicación de un vector de origen, un segundo campo de operando para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para indexar valores del vector de origen y almacenar un resultado de la indexación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice.

En 3903, la circuitería de descodificación descodifica la instrucción recuperada. La instrucción descodificada se usa para configurar una circuitería de ejecución para ejecutar la instrucción descodificada para realizar los actos detallados anteriormente. En algunas realizaciones, la descodificación incluye generar una o más microoperaciones. Obsérvese que, debido a que esta instrucción no ha existido antes, no hay circuitería de descodificación alguna capaz de descodificar esta instrucción de una forma que permita que la circuitería de ejecución ejecute las acciones anteriores de forma apropiada.

La ejecución de la instrucción descodificada se programa (según sea necesario) 3905, que es una etapa opcional (como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

En 3907, la instrucción descodificada (tal como microoperaciones) se ejecuta usando una circuitería de ejecución para indexar valores del vector de origen y almacenar un resultado de la indexación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice.

En algunas realizaciones, la instrucción ejecutada se confirma o se retira en 3909, que es una etapa opcional (como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

La **figura 40** ilustra una realización del procesamiento de una única instrucción. El procesamiento puede ser realizado, en su totalidad o en parte, por una o más de una circuitería de operaciones matriciales (tal como el que se detalla en el presente documento), una circuitería de vectores/SIMD (tal como la que se detalla en el presente documento) o un coprocesador (tal como una GPU capaz de realizar una rasterización o trazado de rayos).

En 4001, una circuitería de recuperación recupera una instrucción. La instrucción incluye un primer campo para identificar una ubicación de un primer vector de origen, un segundo campo para identificar una ubicación de un vector de destino, un tercer campo para identificar una ubicación de un segundo vector de origen y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para permutar los valores de elementos del primer vector de origen basándose en un índice almacenado en el segundo vector de origen y almacenar el resultado de la permutación en el vector de destino.

En 4003, la circuitería de descodificación descodifica la instrucción recuperada. La instrucción descodificada se usa para configurar una circuitería de ejecución para ejecutar la instrucción descodificada para realizar los actos detallados anteriormente. En algunas realizaciones, la descodificación incluye generar una o más microoperaciones. Obsérvese que, debido a que esta instrucción no ha existido antes, no hay circuitería de descodificación alguna capaz de descodificar esta instrucción de una forma que permita que la circuitería de ejecución ejecute las acciones anteriores de forma apropiada.

La ejecución de la instrucción descodificada se programa (según sea necesario) 4005, que es una etapa opcional

(como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

En 4007, la instrucción descodificada (tal como microoperaciones) se ejecuta usando una circuitería de ejecución para permutar los valores de elementos del primer vector de origen basándose en un índice almacenado en el segundo vector de origen y almacenar el resultado de la permutación en el vector de destino.

En algunas realizaciones, la instrucción ejecutada se confirma o se retira en 4009, que es una etapa opcional (como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

La **figura 41** ilustra una realización del procesamiento de una única instrucción. El procesamiento puede ser realizado, en su totalidad o en parte, por una o más de una circuitería de operaciones matriciales (tal como el que se detalla en el presente documento), una circuitería de vectores/SIMD (tal como la que se detalla en el presente documento) o un coprocesador (tal como una GPU capaz de realizar una rasterización o trazado de rayos).

En 4101, una circuitería de recuperación recupera una instrucción. La instrucción incluye un primer campo para identificar una ubicación de un primer vector de origen, un segundo campo para identificar una ubicación de un primer vector de origen, un tercer campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada posición de elemento del primer y el segundo vectores de origen, un valor de índice usando una o más comparaciones de los elementos, y permutando y almacenando los valores de los elementos basándose en los valores de índice para los elementos en el vector de destino.

En 4103, la circuitería de descodificación descodifica la instrucción recuperada. La instrucción descodificada se usa para configurar una circuitería de ejecución para ejecutar la instrucción descodificada para realizar los actos detallados anteriormente. En algunas realizaciones, la descodificación incluye generar una o más microoperaciones. Obsérvese que, debido a que esta instrucción no ha existido antes, no hay circuitería de descodificación alguna capaz de descodificar esta instrucción de una forma que permita que la circuitería de ejecución ejecute las acciones anteriores de forma apropiada.

La ejecución de la instrucción descodificada se programa (según sea necesario) 4105, que es una etapa opcional (como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

En 4107, la instrucción descodificada (tal como microoperaciones) se ejecuta usando una circuitería de ejecución para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada posición de elemento del primer y el segundo vectores de origen, un valor de índice usando una o más comparaciones de los elementos, y permutando y almacenando los valores de los elementos basándose en los valores de índice para los elementos en el vector de destino.

En algunas realizaciones, la instrucción ejecutada se confirma o se retira en 4109, que es una etapa opcional (como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

La **figura 42** ilustra una realización del procesamiento de una única instrucción. El procesamiento puede ser realizado, en su totalidad o en parte, por una o más de una circuitería de operaciones matriciales (tal como el que se detalla en el presente documento), una circuitería de vectores/SIMD (tal como la que se detalla en el presente documento) o un coprocesador (tal como una GPU capaz de realizar una rasterización o trazado de rayos).

En 4201, una circuitería de recuperación recupera una instrucción. La instrucción incluye un primer campo para identificar una ubicación de un primer vector de origen, un segundo campo para identificar una ubicación de un primer vector de origen, un tercer campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para, para cada posición de elemento del primer y el segundo vectores de origen, generar un valor de índice usando una o más comparaciones de los elementos y almacenar el valor de índice generado en una posición de elemento correspondiente del vector de destino.

En 4203, la circuitería de descodificación descodifica la instrucción recuperada. La instrucción descodificada se usa para configurar una circuitería de ejecución para ejecutar la instrucción descodificada para realizar los actos detallados anteriormente. En algunas realizaciones, la descodificación incluye generar una o más microoperaciones. Obsérvese que, debido a que esta instrucción no ha existido antes, no hay circuitería de descodificación alguna capaz de descodificar esta instrucción de una forma que permita que la circuitería de ejecución ejecute las acciones anteriores de forma apropiada.

La ejecución de la instrucción descodificada se programa (según sea necesario) 4205, que es una etapa opcional

(como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

En 4207, la instrucción descodificada (tal como microoperaciones) se ejecuta usando una circuitería de ejecución para, para cada posición de elemento del primer y el segundo vectores de origen, generar un valor de índice usando una o más comparaciones de los elementos, y almacenar el valor de índice generado en una posición de elemento correspondiente del vector de destino.

En algunas realizaciones, la instrucción ejecutada se confirma o se retira en 4209, que es una etapa opcional (como es indicado por su borde de trazo discontinuo) en la medida en la que esta puede tener lugar en un momento diferente, o no tener lugar en absoluto.

En algunas realizaciones, las instrucciones de clasificación de índice y/o de clasificación de índice y de permutación realizan comparaciones del orden ordinal de elementos u otros órdenes para un vector V de tamaño n. Por ejemplo,

```
For (i=0;i<n;i++)
Res[i] = 0
For (j=0;j<n;j++)

Res[i] +=(cmp_a(v[i],v[j]) && cmp_b(i,j))
```

En algunas realizaciones, las instrucciones de clasificación de índice y/o de clasificación de índice y de permutación generan una máscara de comparación de forma bit a bit (es decir, res[i] |= (f_cmp(v[i],v[j]) && o_cmp(i,g)) ? 1 : 0) << j).

En algunas realizaciones, las instrucciones de clasificación de índice y/o de clasificación de índice y de permutación resaltan cierta dependencia de res[i - 1], usando la misma para resolver casos como "devuelve el índice del primer elemento más grande que un elemento en otro vector (es decir, res[-1] = const; res[i] = r_fnc(res[i-1]) && ((f_cmp(v[i],v[j]) && o_cmp(i,g)) ? 1 : 0)).

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 fases de canalización, tales como recuperar, descodificar, programar, ejecutar, retirar, etc.

Conjuntos de instrucciones

Un conjunto de instrucciones puede incluir uno o más formatos de instrucción. Un formato de instrucción dado puede definir diversos campos (por ejemplo, número de bits, ubicación de bits) para especificar, entre otras cosas, la operación a realizar (por ejemplo, código de operación) y el operando u operandos en los que esa operación se va a realizar y/u otro campo o campos de datos (por ejemplo, máscara). Algunos formatos de instrucción se descomponen adicionalmente a través de la definición de plantillas de instrucción (o subformatos). Por ejemplo, se puede definir que las plantillas de instrucción de un formato de instrucción dado tengan diferentes subconjuntos de los campos del formato de instrucción (los campos incluidos están habitualmente en el mismo orden, pero al menos algunos tienen posiciones de bits diferentes debido a que hay menos campos incluidos) y/o se puede definir que tengan un campo dado interpretado de forma diferente. Por lo tanto, cada instrucción de una ISA se expresa usando un formato de instrucción dado (y, si se define, en una dada 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 ilustrativa tiene un código de operación específico y un formato de instrucción que incluye un campo de código de operación para especificar ese código de operación y campos de operando para seleccionar operandos (origen 1/destino y origen 2); y una aparición de esta instrucción ADD en un flujo de instrucciones tendrá contenidos específicos en los campos de operando que seleccionan operandos específicos. Se ha lanzado y/o publicado un conjunto de ampliaciones de SIMD denominadas Ampliaciones de Vectores Avanzadas (AVX) (AVX1 y AVX2) y que usan el esquema de codificación de Ampliaciones de Vectores (VEX) (por ejemplo, véase el documento Manual para desarrolladores de software para arquitecturas Intel® 64 e IA-32, septiembre de 2014; y véase el documento *Intel® Advanced Vector Extensions Programming Reference*, octubre de 2014).

Formatos de instrucción ilustrativos

Se pueden materializar realizaciones de la instrucción o instrucciones descritas en el presente documento en diferentes formatos. Adicionalmente, a continuación se detallan sistemas, arquitecturas y canalizaciones 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 apto para vectores de carácter genérico

Un formato de instrucción apto para vectores es un formato de instrucción adecuado para instrucciones de vector (por ejemplo, hay ciertos campos específicos de operaciones vectoriales). Aunque se describen realizaciones en las que se soportan operaciones tanto vectoriales como escalares a través del formato de instrucción apto para vectores, realizaciones alternativas usan solo operaciones vectoriales en el formato de instrucción apto para vectores.

Las **figuras 25A-25B** son diagramas de bloques que ilustran un formato de instrucción apto para vectores de carácter genérico y plantillas de instrucción del mismo de acuerdo con realizaciones de la invención. la **figura 25A** es un diagrama de bloques que ilustra un formato de instrucción apto para vectores de carácter genérico y plantillas de instrucción de clase A del mismo de acuerdo con realizaciones de la invención; mientras que la **figura 25B** es un diagrama de bloques que ilustra el formato de instrucción apto para vectores de carácter genérico y plantillas de instrucción de clase B del mismo de acuerdo con realizaciones de la invención. Específicamente, un formato de instrucción apto para vectores de carácter genérico 2500 para el que se definen plantillas de instrucción de clase A y de clase B, ambas de las cuales incluyen plantillas de instrucción sin acceso a memoria 2505 y plantillas de instrucción de acceso a memoria 2520. El término genérico en el contexto del formato de instrucción apto para vectores se refiere a que el formato de instrucción no está vinculado a conjunto de instrucciones específico alguno.

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

Las plantillas de instrucción de clase A en la **figura 25A** incluyen: 1) dentro de las plantillas de instrucción sin acceso a memoria 2505 se muestra una plantilla de instrucción de operación de tipo control de redondeo completo sin acceso a memoria 2510 y una plantilla de instrucción de operación de tipo transformada de datos sin acceso a memoria 2515; 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 25B** incluyen: 1) dentro de las plantillas de instrucción sin acceso a memoria 2505 se muestra una plantilla de instrucción de operación de tipo control de redondeo parcial de control de máscara de escritura sin acceso a memoria 2512 y una plantilla de instrucción de operación de tipo vsize de control de máscara de escritura sin acceso a memoria 2517; 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 apto para vectores de carácter genérico 2500 incluye los campos siguientes enumerados a continuación en el orden ilustrado en las **figuras 25A-25B**.

Campo de formato 2540 - un valor específico (un valor de identificador de formato de instrucción) en este campo identifica de forma exclusiva el formato de instrucción apto para vectores y, por lo tanto, apariciones de instrucciones en el formato de instrucción apto para vectores en flujos de instrucciones. En ese sentido, este campo es opcional en el sentido de que no es necesario para un conjunto de instrucciones que solo tiene el formato de instrucción apto para vectores de carácter genérico.

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

Campo de índice de registro 2544 - su contenido, directamente o a través de una generación de direcciones, especifica las ubicaciones de los operandos de origen y de destino, ya estén estos en registros o en memoria. Estos incluyen un número suficiente de bits para seleccionar N registros de un archivo de registro P x Q (por ejemplo, 32 x 512, 16 x 128, 32 x 1024, 64 x 1024). Aunque, en una realización, N puede ser hasta tres orígenes y un registro de destino, realizaciones alternativas pueden soportar más o menos orígenes y registros de destino (por ejemplo, pueden soportar hasta dos orígenes en donde uno de estos orígenes también actúa como el destino, pueden soportar hasta tres orígenes en 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 apariciones de instrucciones en el formato de instrucción de vector genérico que especifican un acceso a memoria de aquellas que no lo hacen; es decir, entre las plantillas de instrucción sin acceso a memoria 2505 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). Aunque, en una realización, este campo también selecciona entre tres formas diferentes de realizar cálculos de dirección de memoria, realizaciones alternativas pueden soportar más,

menos o diferentes formas de realizar cálculos de dirección de memoria.

Campo de operación de aumento 2550 - su contenido distingue cuál de una diversidad de operaciones diferentes se va a realizar además de la operación 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 en 2, 3 o 4 instrucciones.

Campo de escala 2560 - su contenido permite ajustar a escala el contenido del campo de índice para una generación de direcciones de memoria (por ejemplo, para una 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 una 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 una generación de direcciones; este especifica un factor de desplazamiento que se va a ajustar a escala con el tamaño de un acceso a memoria (N) - en donde N es el número de bytes en el acceso a memoria (por ejemplo, para una generación de direcciones que usa 2^{escala} * índice + base + desplazamiento ajustado a escala). Los bits de orden bajo redundantes se ignoran y, por lo tanto, el contenido del campo de factor de desplazamiento es multiplicado por el tamaño total de operandos de memoria (N) con el fin de generar el desplazamiento final a usar en el cálculo de una dirección eficaz. El valor de N es determinado por el hardware de 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 estos no se usan para las plantillas de instrucción sin acceso a memoria 2505 y/o diferentes realizaciones pueden implementar solo uno o ninguno de los dos.

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

Campo de máscara de escritura 2570 - su contenido controla, de una forma por posición de elemento de datos, si esa posición de elemento de datos en el operando de vector de destino refleja el resultado de la operación base y la operación de aumento. Las plantillas de instrucción de clase A soportan un enmascaramiento de escritura de combinación, mientras que las plantillas de instrucción de clase B soportan un enmascaramiento de escritura tanto de combinación como de puesta a cero. Cuando se combina, las máscaras de vector permiten proteger de actualizaciones cualquier conjunto de elementos en el destino durante la ejecución de cualquier operación (especificada por la operación base y la operación de aumento); en otra realización, conservando el valor antiguo de cada elemento del destino en donde el bit de máscara correspondiente tiene un 0. En contraposición, cuando se pone a cero, las máscaras de vector permiten poner a cero cualquier conjunto de elementos en el destino durante la ejecución de cualquier operación (especificada por la operación 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 de vector de la operación que se realiza (es decir, el rango 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, incluyendo cargas, almacenamientos, aritméticas, lógicas, etc. Aunque 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 a usar (y, por lo tanto, el contenido del campo de máscara de escritura 2570 identifica indirectamente el enmascaramiento a realizar), realizaciones alternativas permiten, en su lugar o adicionalmente, que el contenido del campo de escritura de máscara 2570 especifique directamente el enmascaramiento a realizar.

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

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

Plantillas de instrucción de clase A

En el caso de las plantillas de instrucción sin acceso a memoria 2505 de clase A, el campo alfa 2552 se interpreta como un campo RS 2552A, cuyo contenido distingue cuál de los diferentes tipos de operación de aumento se va a realizar (por ejemplo, el redondeo 2552A.1 y la transformada de datos 2552A.2 se especifican, respectivamente, para las plantillas de instrucción de operación de tipo redondeo sin acceso a memoria 2510 y de operación de tipo transformada de datos sin acceso a memoria 2515), mientras que el campo beta 2554 distingue cuál de las operaciones del tipo especificado se va a realizar. En las plantillas de instrucción sin acceso a memoria 2505, no están presentes el campo de escala 2560, el campo de desplazamiento 2562A y el campo de escala de desplazamiento 2562B.

Plantillas de instrucción sin acceso a memoria - operación de tipo control de redondeo completo

En la plantilla de instrucción de operación de tipo control de redondeo completo sin acceso a memoria 2510, el campo beta 2554 se interpreta como un campo de control de redondeo 2554A, cuyo contenido o contenidos proporcionan un redondeo estático. Aunque, 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, realizaciones alternativas pueden soportar pueden codificar ambos de estos conceptos en el mismo campo o solo tener uno u 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 deshabilita o no la notificación de sucesos de excepción; cuando el contenido del campo de SAE 2556 indica que se habilita una supresión, una instrucción dada no notifica tipo alguno de indicador de excepción de coma flotante y no genera manejador de excepciones de coma flotante alguno.

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 más cercano). Por lo tanto, el campo de control de operación de redondeo 2558 permite el cambio del modo de redondeo de una forma por instrucción. En una realización de la invención en la que un procesador incluye un registro de control para especificar modos de redondeo, el contenido del campo de control de operación de redondeo 2550 reemplaza ese valor de registro.

Plantillas de instrucción sin acceso a memoria - operación de tipo transformada de datos

En la plantilla de instrucción de la operación de tipo transformada de datos sin acceso a memoria 2515, el campo beta 2554 se interpreta como un campo de transformada de datos 2554B, cuyo contenido distingue cuál de un número de transformadas de datos se va a realizar (por ejemplo, sin transformada de datos, alineación, radiodifusión).

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 expulsión 2552B, cuyo contenido distingue cuál de las sugerencias de expulsión se va a usar (en la **figura 25A**, se especifican temporal 2552B.1 y no temporal 2552B.2, respectivamente, para la plantilla de instrucción temporal de acceso a memoria 2525 y la plantilla de instrucción no temporal de acceso a memoria 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; radiodifusión; conversión ascendente de un origen; y conversión descendente de un destino). Las plantillas de instrucción 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 de vectores realizan cargas de vector desde y almacenamientos de vector en memoria, con soporte de conversión. Al igual que con las instrucciones de vector regulares, las instrucciones de memoria de vector transfieren datos desde/hacia la memoria de una forma elemento de datos a elemento de datos, con los elementos que realmente se transfieren dictados por los contenidos de la máscara de vector que se selecciona como máscara de escritura.

Plantillas de instrucción de acceso a memoria - temporal

Los datos temporales son datos que es probable que se reutilicen lo bastante pronto como para beneficiarse de un almacenamiento en memoria caché. Sin embargo, esto es una sugerencia, y diferentes procesadores pueden implementarlo de diferentes formas, incluso ignorando la sugerencia por completo.

Plantillas de instrucción de acceso a memoria - no temporal

Los datos no temporales son datos que es poco probable que se reutilicen lo bastante pronto como para beneficiarse de un almacenamiento en memoria caché en la memoria caché de primer nivel y se les debería dar prioridad para la expulsión. Sin embargo, esto es una sugerencia, y diferentes procesadores pueden implementarlo de diferentes formas, incluso ignorando la sugerencia por completo.

Plantillas de instrucción de clase B

En el caso de las plantillas de instrucción 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 debería ser una combinación o una puesta a cero.

En el caso de las plantillas de instrucción sin acceso a memoria 2505 de clase B, parte del campo beta 2554 se interpreta como un campo RL 2557A, cuyo contenido distingue cuál de los diferentes tipos de operación de aumento se va a realizar (por ejemplo, el redondeo 2557A.1 y la longitud de vector (VSIZE) 2557A.2 se especifican, respectivamente, para la plantilla de instrucción de operación de tipo control de redondeo parcial de control de máscara de escritura sin acceso a memoria 2512 y la plantilla de instrucción de operación de tipo VSIZE de control de máscara de escritura sin acceso a memoria 2517), mientras que el resto del campo beta 2554 distingue cuál de las operaciones del tipo especificado se va a realizar. En las plantillas de instrucción sin acceso a memoria 2505, no están presentes el campo de escala 2560, el campo de desplazamiento 2562A y el campo de escala de desplazamiento 2562B.

En la plantilla de instrucción de operación de tipo control de redondeo parcial de control de máscara de escritura sin acceso a memoria 2510, el resto del campo beta 2554 se interpreta como un campo de operación de redondeo 2559A y se deshabilita la notificación de sucesos de excepción (una instrucción dada no notifica tipo alguno de indicador de excepción de coma flotante y no genera manejador de excepciones de coma flotante alguno).

Campo de control de operación de redondeo 2559A - exactamente 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 más cercano). Por lo tanto, el campo de control de operación de redondeo 2559A permite el cambio del modo de redondeo de una forma por instrucción. En una realización de la invención en la que un procesador incluye un registro de control para especificar modos de redondeo, el contenido del campo de control de operación de redondeo 2550 reemplaza ese valor de registro.

En la plantilla de instrucción de la operación de tipo VSIZE de control de máscara de escritura sin acceso a memoria 2517, el resto del campo beta 2554 se interpreta como un campo de longitud de vector 2559B, cuyo contenido distingue sobre cuál de un número de longitudes de vector de datos se va a realizar (por ejemplo, 128, 256 o 512 bytes).

En el caso de una plantilla de instrucción de acceso a memoria 2520 de clase B, parte del campo beta 2554 se interpreta como un campo de radiodifusión 2557B, cuyo contenido distingue si se va a realizar o no la operación de manipulación de datos de tipo radiodifusión, mientras que el resto del campo beta 2554 se interpreta como el campo de longitud de vector 2559B. Las plantillas de instrucción 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 apto para vectores de carácter genérico 2500, se muestra un campo de código de operación completo 2574 que incluye el campo de formato 2540, el campo de operación base 2542 y el campo de anchura de elemento de datos 2564. Aunque se muestra una realización en la que 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 soportan la totalidad de los mismos. El campo de código de operación completo 2574 proporciona el código de operación (código de op.).

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 de una forma por instrucción en el formato de instrucción apto para vectores de carácter genérico.

La combinación de campo de máscara de escritura y campo de anchura de elemento de datos crea instrucciones con tipo ya que permiten aplicar la máscara basándose en diferentes anchuras de elemento de datos.

Las diversas plantillas de instrucción halladas 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 fuera de orden de propósito general de desempeño alto destinado a computación de propósito general puede soportar solo la clase B, un núcleo destinado principalmente a gráficos y/o computación científica (de capacidad de proceso) puede soportar solo la clase A, y un núcleo destinado y ambos puede soportar ambas (por supuesto, un núcleo que tiene alguna combinación de plantillas e instrucciones a partir de ambas clases, pero no todas las plantillas e instrucciones a partir de ambas clases, está dentro del alcance de la invención). Además, un único 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 solo la clase A, mientras que uno o más de los núcleos de propósito general pueden ser núcleos de propósito general de desempeño alto con ejecución fuera de orden y cambio de nombre de registro destinados a computación de propósito general que soportan solo la clase B. Otro procesador que no tiene un núcleo de gráficos separado puede incluir uno más núcleos en orden o fuera de orden de propósito general que soportan tanto la clase A como la clase B. Por supuesto, las características a partir de una clase también se pueden implementar en la otra clase en diferentes realizaciones de la invención. 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 solo instrucciones de la clase o clases soportadas por el procesador de destino para su ejecución; o 2) una forma que tiene rutinas alternativas escritas usando diferentes combinaciones de 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 apto para vectores de carácter específico ilustrativo

La **figura 26A** es un diagrama de bloques que ilustra un formato de instrucción apto para vectores de carácter específico ilustrativo de acuerdo con realizaciones de la invención. La **figura 26A** muestra un formato de instrucción apto para vectores de carácter específico 2600 que es específico en el sentido de que el mismo especifica la ubicación, el tamaño, la interpretación y el orden de los campos, así como valores para algunos de esos campos. El formato de instrucción apto para vectores de carácter específico 2600 se puede usar para ampliar el conjunto de instrucciones de x86 y, por lo tanto, algunos de los campos son similares o iguales a los usados en el conjunto de instrucciones de x86 existente y la ampliación del mismo (por ejemplo, AVX). Este formato sigue siendo coherente 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 de x86 existente con ampliaciones. Se ilustra los campos a partir de la **figura 25** con los que se correlacionan los campos a partir de la **figura 26A**.

Se debería entender que, aunque se describen realizaciones de la invención con referencia al formato de instrucción apto para vectores de carácter específico 2600 en el contexto del formato de instrucción apto para vectores de carácter genérico 2500 con fines ilustrativos, la invención no se limita al formato de instrucción apto para vectores de carácter específico 2600 excepto en donde se reivindique. Por ejemplo, el formato de instrucción apto para vectores de carácter genérico 2500 contempla una diversidad de tamaños posibles para los diversos campos, mientras que el formato de instrucción apto para vectores de carácter específico 2600 se muestra como que tiene 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 apto para vectores de carácter específico 2600, la invención no se limita a ello (es decir, el formato de instrucción apto para vectores de carácter genérico 2500 contempla otros tamaños del campo de anchura de elemento de datos 2564).

El formato de instrucción apto para vectores de carácter genérico 2500 incluye los campos siguientes enumerados a continuación en el orden ilustrado en la **figura 26A**.

Prefijo EVEX (bytes 0-3) 2602 - se codifica en un formato de cuatro bytes.

Campo de formato 2540 (byte de EVEX 0, bits [7:0]) - el primer byte (byte de EVEX 0) es el campo de formato 2540 y contiene 0x62 (el valor exclusivo usado para distinguir el formato de instrucción apto para 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 2557B (byte de EVEX 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 usando la 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 se conoce en la técnica (rrr, xxx y bbb), de tal forma que Rrrr, Xxxx y Bbbb se pueden formar sumando EVEX.R, EVEX.X y EVEX.B.

Campo REX' 2510 - esta es la primera parte del campo REX' 2510 y es el campo de bits EVEX.R' (byte 1 de EVEX, bit [4] - R') que se usa para codificar o bien los 16 superiores o bien los 16 inferiores del conjunto de 32 registros ampliado. En una realización de la invención, este bit, junto con otros como se indica a continuación, se almacena en un formato invertido en bits para distinguirlo (en el bien conocido modo de 32 bits de x86) 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 a partir de otros campos.

Campo de correlación de código de operación 2615 (byte de EVEX 1, 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 (o bien elementos de datos de 32 bits o bien elementos de datos de 64 bits).

EVEX.vvvv 2620 (byte de EVEX 2, bits [6:3] - vvvv) - el papel de EVEX.vvvv puede incluir lo siguiente: 1) EVEX.vvvv codifica el primer operando de registro de origen, especificado de 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 operando alguno, el campo está reservado y debería contener 1111b. Por lo tanto, el campo EVEX.vvvv 2620 codifica los 4 bits de orden inferior del primer especificador de registro de origen almacenado de forma invertida (complemento a 1). Dependiendo de la instrucción, se usa un campo de bits EVEX diferente adicional para ampliar el tamaño de especificador a 32 registros.

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

Campo de codificación de prefijo 2625 (byte de EVEX 2, bits [1:0] - pp) - proporciona bits adicionales para el campo de operación base. Además de proporcionar soporte para las instrucciones SSE heredadas en el formato de prefijo EVEX, este también tiene la ventaja de compactar el prefijo SIMD (en lugar de requerir un byte para expresar el prefijo SIMD, el prefijo EVEX requiere solo 2 bits). En una realización, para soportar instrucciones de SSE heredadas que usan un prefijo SIMD (66H, F2H, F3H) tanto en el formato heredado como en el formato de prefijo EVEX, estos prefijos SIMD heredados se codifican en el campo de codificación de prefijo SIMD; y, en tiempo de ejecución, se expanden al prefijo SIMD heredado antes de proporcionarse a la PLA del descodificador (de tal modo que la PLA pueda ejecutar tanto el formato heredado como el de EVEX de estas instrucciones heredadas sin modificaciones). Aunque instrucciones más nuevas podrían usar el contenido del campo de codificación de prefijo EVEX directamente como una ampliación de código de operación, ciertas realizaciones se expanden de una forma similar por razones de coherencia, pero permiten que diferentes significados sean especificados por estos prefijos SIMD heredados. Una realización alternativa puede rediseñar la PLA para soportar las codificaciones de prefijo SIMD de 2 bits y, por lo tanto, no requieren 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 ha descrito previamente, 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 ha descrito previamente, este campo es específico del contexto.

Campo REX' 2510 - este es el resto del campo REX' y es el campo de bits EVEX.V' (byte 3 de EVEX, bit [3] - V') que se puede usar para codificar o bien los 16 superiores o bien los 16 inferiores del conjunto de 32 registros ampliado. Este bit se almacena en un formato invertido en bits. 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 ha descrito previamente. 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 alguna para la instrucción particular (esto se puede implementar de una diversidad de formas, incluyendo el uso de una máscara de escritura cableada físicamente a todo unos o hardware que sortea el hardware de enmascaramiento).

El campo de código de operación real 2630 (byte 4) también se conoce como el 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 ha descrito previamente, el contenido del campo MOD 2642 distingue entre operaciones de acceso a memoria y sin acceso a memoria. El papel del campo Reg 2644 se puede resumir en dos situaciones: codificar o bien el operando de registro de destino o un operando de registro de origen o tratarse como una ampliación de código de operación y no usarse para codificar operando de instrucción alguno. El papel del campo R/M 2646 puede incluir lo siguiente: codificar el operando de instrucción que hace referencia a una dirección de memoria o codificar o bien el operando de registro de destino o bien un operando de registro de origen.

Byte de Escala, Índice, Base (SIB) (byte 6) - como se ha descrito previamente, el contenido del campo de escala 2550 se usa para una generación de direcciones de memoria. SIB.xxx 2654 y SIB.bbb 2656 - se ha hecho referencia previamente a los contenidos de estos campos con respecto a los índices de registro Xxxx y Bbbb.

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

Campo de factor de desplazamiento 2562B (byte 7) - cuando el campo MOD 2642 contiene un 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 de conjunto de instrucciones de x86 heredado (disp8), que funciona con una granularidad de bytes. Debido a que disp8 se amplía con signo, este solo puede abordar desplazamientos entre -128 y 127 bytes; en términos de líneas de memoria caché de 64 bytes, disp8 usa 8 bits que se pueden establecer a solo cuatro valores realmente útiles -128, -64, 0 y 64; debido a que a menudo se necesita un rango 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 es determinado 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 (se usa un único byte para el desplazamiento, pero con un rango mucho mayor). Tal desplazamiento comprimido supone que el desplazamiento eficaz es un múltiplo de la granularidad del acceso a memoria y, por lo tanto, no es necesario codificar los bits de orden bajo redundantes del desplazamiento de dirección. En otras palabras, el campo de factor de desplazamiento 2562B sustituye el desplazamiento de 8 bits de conjunto de instrucciones de x86 heredado. Por lo tanto, el campo de factor de desplazamiento 2562B se codifica de la misma forma que un desplazamiento de 8 bits de conjunto de instrucciones de x86 (por lo que no hay cambio alguno en las reglas de codificación de ModRM/SIB), con la única excepción de que disp8 se sobrecarga a disp8 * N. En otras palabras, no hay cambio alguno en las reglas de codificación o en las longitudes de codificación, sino solo en la interpretación del valor de desplazamiento por hardware (que necesita ajustar a escala el desplazamiento con el tamaño del operando de memoria para obtener un desplazamiento de dirección byte a byte). El campo de valor inmediato 2572 funciona como se ha descrito previamente.

Campo de código de operación completo

La **figura 26B** es un diagrama de bloques que ilustra los campos del formato de instrucción apto para vectores de carácter específico 2600 que constituyen 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 (W) de elemento de datos 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 apto para vectores de carácter específico 2600 que constituyen 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/Mr/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 apto para vectores de carácter específico 2600 que constituyen el campo de operación de aumento 2550 de acuerdo con una realización de la invención. Cuando el campo de clase (U) 2568 contiene un 0, ello significa EVEX.U0 (clase A 2568A); cuando este contiene un 1, ello significa EVEX.U1 (clase B 2568B). Cuando U = 0 y el campo MOD 2642 contiene un 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 rs 2552A. Cuando el campo 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 2556 de un bit y un campo de operación de redondeo 2558 de dos bits. Cuando el campo rs 2552A contiene un 0 (transformada de datos 2552A.2), el campo beta 2554 (byte de EVEX 3, bits [6:4] - SSS) se interpreta como un campo de transformada de datos 2554A de tres bits. 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, bits [7] - EH) se interpreta como el campo de sugerencia de expulsión (EH) 2552B y el campo beta 2554 (byte de EVEX 3, bits [6:4] - SSS) se interpreta como un campo de manipulación de datos 2554C de tres bits.

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 un 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 RL 2557A; cuando este 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 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 de vector 2710 que tienen una anchura de 512 bits; estos registros se referencian como zmm0 a zmm31. Los 256 bits de orden inferior de los 16 registros zmm inferiores se superponen sobre 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 sobre los registros xmm0-15. El formato de instrucción apto para vectores de carácter específico 2600 opera sobre estos archivos de registro superpuestos, como se ilustra en las tablas a continuación.

Longitud de vector ajustable	Clase	Operaciones	Registros
Plantillas de instrucción que no incluyen el campo de longitud de vector 2559B	A (la **figura 25A**; U = 0)	2510, 2515, 2525, 2530	registros zmm (la longitud de vector es de 64 bytes)
	B (la **figura 25B**; U = 1)	2512	registros zmm (la longitud de vector es de 64 bytes)
Plantillas de instrucción que sí incluyen el campo de longitud de vector 2559B	B (la **figura 25B**; U = 1)	2517, 2527	registros zmm, ymm o xmm (la longitud de 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, en 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 sobre la longitud de vector máxima. Además, en una realización, las plantillas de instrucción de clase B del formato de instrucción apto para vectores de carácter específico 2600 operan sobre datos de coma flotante de precisión simple/doble escalares o empaquetados y datos enteros escalares o empaquetados. Las operaciones escalares son operaciones realizadas sobre la posición de elemento de datos del orden más bajo en un registro zmm/ymm/xmm; las posiciones de elemento de datos de orden superior o bien se dejan igual que como estaban antes de la instrucción o bien 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 (de k0 a k7), cada uno de un tamaño de 64 bits. En una realización alternativa, los registros de máscara de escritura 2715 tienen un tamaño de 16 bits. Como se ha descrito previamente, en una realización de la invención, el registro de máscara de vector k0 no se puede usar como una máscara de escritura; cuando la codificación que normalmente indicaría k0 se usa para una máscara de escritura, este selecciona una máscara de escritura cableada físicamente de 0xFFFF, deshabilitando en la práctica el enmascaramiento 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 de x86 existentes para direccionar operandos de memoria. A estos registros se hace referencia mediante los nombres RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP y de R8 a R15.

Archivo de registro de pila de coma flotante escalar (pila de x87) 2745, en el que se establece un alias para el archivo de registro plano de entero empaquetado MMX 2750 - en la realización ilustrada, la pila de x87 es una pila de ocho elementos usada para realizar operaciones de coma flotante escalar sobre datos de coma flotante de 32/64/80 bits usando la ampliación del conjunto de instrucciones de x87; mientras que los registros MMX se usan para realizar operaciones sobre datos 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. Adicionalmente, realizaciones alternativas de la invención pueden usar más, menos o diferentes archivos de registro y registros.

Arquitecturas de núcleo, procesadores y arquitecturas de ordenador ilustrativas

Los núcleos de procesador se pueden implementar de diferentes formas, para diferentes fines 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 fuera de orden de propósito general de desempeño alto destinado a computación de propósito general; 3) un núcleo de propósito especial destinado principalmente a gráficos y/o computación científica (de capacidad de proceso). Las implementaciones de diferentes procesadores pueden incluir: 1) una CPU que incluye uno o más núcleos en orden de propósito general destinados a computación de propósito general y/o uno o más núcleos fuera de orden de propósito general destinados a computación de propósito general; y 2) un coprocesador que incluye uno o más núcleos de propósito especial destinados principalmente a cálculos gráficos y/o científicos (de capacidad de proceso). Tales procesadores diferentes conducen a diferentes arquitecturas de sistema informático, que pueden incluir: 1) el coprocesador en un chip separado de la CPU; 2) el coprocesador en una pastilla separada en el mismo encapsulado que una CPU; 3) el coprocesador en la misma pastilla que una CPU (en cuyo caso, un coprocesador de este tipo se denomina, a veces, lógica de propósito especial, tal como gráficos integrados y/o lógica científica (de capacidad de proceso), o como

núcleos de propósito especial); y 4) un sistema en un chip que puede incluir, en la misma pastilla, la CPU descrita (denominada, a veces, el núcleo o núcleos de aplicación o procesador o procesadores de aplicación), el coprocesador descrito anteriormente y una funcionalidad adicional. A continuación, se describen arquitecturas de núcleo ilustrativas, seguidas de descripciones de procesadores y arquitecturas de ordenador ilustrativas.

Arquitecturas de núcleo ilustrativas

Diagrama de bloques de núcleo en orden y fuera de orden

La **figura 28A** es un diagrama de bloques que ilustra tanto una canalización en orden ilustrativa como una canalización de cambio de nombre de registro, emisión/ejecución fuera de orden 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 cambio de nombre de registro, emisión/ejecución fuera de orden ilustrativo a incluir en un procesador de acuerdo con realizaciones de la invención. Los recuadros con líneas de trazo continuo en las **figuras 28A-B** ilustran la canalización en orden y el núcleo en orden, mientras que la adición opcional de los recuadros con líneas de trazo discontinuo ilustra la canalización y el núcleo de cambio de nombre de registro, emisión/ejecución fuera de orden. Debido a que el aspecto en orden es un subconjunto del aspecto fuera de orden, se describirá el aspecto fuera de orden.

En la **figura 28A**, una canalización de procesador 2800 incluye una fase de recuperación 2802, una fase de descodificación de longitud 2804, una fase de descodificación 2806, una fase de asignación 2808, una fase de cambio de nombre 2810, una fase de programación (también conocida como despacho o emisión) 2812, una fase de lectura de registro/lectura de memoria 2814, una fase de ejecución 2816, una fase de escritura no simultánea/escritura en memoria 2818, una fase de manejo de excepciones 2822 y una fase de confirmación 2824.

La **figura 28B** muestra el núcleo de procesador 2890 que incluye una unidad de extremo frontal 2830 acoplada a una unidad de motor de ejecución 2850, y ambas se acoplan a una unidad de memoria 2870. El núcleo 2890 puede ser un núcleo de informática de conjunto de instrucciones reducido (RISC), un núcleo de informática 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 de 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 similares.

La unidad de extremo frontal 2830 incluye una unidad de predicción de bifurcaciones 2832 acoplada a una unidad de memoria caché de instrucciones 2834, que se acopla a una memoria intermedia de traducción adelantada (TLB) de instrucciones 2836, que se acopla a una unidad de recuperación de instrucciones 2838, que se acopla a una unidad de descodificación 2840. La unidad de descodificación 2840 (o descodificador) puede descodificar instrucciones y generar como una salida una o más microoperaciones, puntos de entrada de microcódigo, microinstrucciones, otras instrucciones u otras señales de control, que se descodifican a partir de, o que reflejan de otro modo, o se derivan de, las instrucciones originales. La unidad de descodificación 2840 se puede implementar usando diversos mecanismos diferentes. Los ejemplos de mecanismos adecuados incluyen, pero no se limitan a, tablas de consulta, implementaciones en 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 ciertas macroinstrucciones (por ejemplo, en la unidad de descodificación 2840 o, de lo contrario, dentro de la unidad de extremo frontal 2830). La unidad de descodificación 2840 se acopla 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 unidad o más unidades de programador 2856. La unidad o unidades de programador 2856 representan cualquier número de programadores diferentes, incluyendo estaciones de reservas, ventana de instrucciones central, etc. La unidad o unidades de programador 2856 se acoplan 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, unos diferentes de los cuales almacenan uno o más tipos de datos diferentes, tales como entero escalar, coma flotante escalar, entero empaquetado, coma flotante empaquetada, 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 registro físico 2858 comprende una unidad de registros de vector, una unidad de registros de máscara de escritura y una unidad de registros de escalar. Estas unidades de registro pueden proporcionar registros de vector arquitectónicos, registros de máscara de vector y registros de propósito general. La unidad o unidades de archivo o archivos de registro físico 2858 son solapadas por la unidad de retiro 2854 para ilustrar diversas formas en las que se pueden implementar un cambio de nombre de registro y una ejecución fuera de orden (por ejemplo, usando una memoria intermedia o memorias intermedias de reordenación y un archivo o archivos de registro de retiro; usando un archivo o archivos futuros, una memoria o memorias de historial y un archivo o archivos de registro de retiro; usando correlaciones de registro 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). Aunque algunas realizaciones pueden incluir un número de unidades de ejecución dedicadas a funciones o conjuntos de funciones específicos, otras realizaciones pueden incluir solo una unidad de ejecución o múltiples unidades de ejecución que realizan, todas ellas, todas las funciones. La unidad o unidades de programador 2856, la unidad o unidades de archivo o archivos de registro físico 2858 y la agrupación o agrupaciones de ejecución 2860 se muestran como que son posiblemente una pluralidad debido a que ciertas realizaciones crean canalizaciones separadas para ciertos tipos de datos/operaciones (por ejemplo, una canalización de entero escalar, una canalización de coma flotante escalar/entero empaquetado/coma flotante empaquetada/entero vectorial/coma flotante vectorial y/o una canalización de acceso a memoria que tienen, cada una, sus propias unidad de programador, 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 solo la agrupación de ejecución de esta canalización tiene la unidad o unidades de acceso a memoria 2864). También se debería entender que, cuando se usan canalizaciones separadas, una o más de estas canalizaciones pueden ser de emisión/ejecución fuera de orden y, el resto, en orden.

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

A modo de ejemplo, la arquitectura de núcleo de cambio de nombre de registro, emisión/ejecución fuera de orden ilustrativa puede implementar la canalización 2800 como sigue: 1) la recuperación de instrucciones 2838 realiza las fases de recuperación y de descodificación de longitud 2802 y 2804; 2) la unidad de descodificación 2840 realiza la fase de descodificación 2806; 3) la unidad de cambio de nombre/asignación 2852 realiza la fase de asignación 2808 y la fase de cambio de nombre 2810; 4) la unidad o unidades de programador 2856 realizan la fase de programación 2812; 5) la unidad o unidades de archivo o archivos de registro físico 2858 y la unidad de memoria 2870 realizan la fase de lectura de registro/lectura de memoria 2814; la agrupación de ejecución 2860 realiza la fase de ejecución 2816; 6) la unidad de memoria 2870 y la unidad o unidades de archivo o archivos de registro físico 2858 realizan la fase de escritura no simultánea/escritura en memoria 2818; 7) diversas unidades pueden estar implicadas en la fase de manejo de excepciones 2822; y 8) la unidad de retiro 2854 y la unidad o unidades de archivo o archivos de registro físico 2858 realizan la fase de confirmación 2824.

El núcleo 2890 puede soportar uno o más conjuntos de instrucciones (por ejemplo, el conjunto de instrucciones de x86 (con algunas ampliaciones que se han añadido con versiones más nuevas); el conjunto de instrucciones MIPS de MIPS Technologies de Sunnyvale, CA; el conjunto de instrucciones ARM (con ampliaciones adicionales opcionales tales 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 lógica para soportar una ampliación de conjunto de instrucciones de datos empaquetados (por ejemplo, AVX1, AVX2), permitiendo de ese modo que las operaciones usadas por muchas aplicaciones multimedios se realicen usando datos empaquetados.

Se debería entender que el núcleo puede soportar multiproceso (ejecutar dos o más conjuntos paralelos de operaciones o subprocesos), y puede hacer esto de una diversidad de formas, incluyendo multiproceso segmentado en el tiempo, multiproceso simultáneo (en donde un único núcleo físico proporciona un núcleo lógico para cada uno de los subprocesos para los que ese núcleo físico está sometiendo a multiproceso simultáneamente), o una combinación de los mismos (por ejemplo, recuperación y descodificación segmentadas en el tiempo y multiproceso simultáneo a continuación de lo anterior, tal como en la tecnología Hyperthreading de Intel®).

Aunque el cambio de nombre de registros se describe en el contexto de una ejecución fuera de orden, se debería 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 memoria caché de instrucciones y de datos 2834/2874 separadas y una unidad de memoria caché de L2 2876 compartida, realizaciones alternativas pueden tener una única memoria caché interna tanto para instrucciones como para datos, tal como, por ejemplo, una memoria caché interna de nivel 1 (L1) o múltiples niveles de memoria 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 memoria 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 (incluyendo otros núcleos del mismo tipo y/o de diferentes tipos) 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 integrada en pastilla 2902 y con su subconjunto local de la memoria caché de nivel 2 (L2) 2904, de acuerdo con realizaciones de la invención. En una realización, un descodificador de instrucciones 2900 soporta el conjunto de instrucciones de x86 con una ampliación de conjunto de instrucciones de datos empaquetados. Una memoria caché de L1 2906 permite accesos de latencia baja a memoria caché a las unidades de escalares y de vectores. Aunque en una realización (para simplificar el diseño), una unidad de escalares 2908 y una unidad de vectores 2910 usan conjuntos de registros separados (respectivamente, los registros de escalar 2912 y los registros de vector 2914) y los datos transferidos entre los mismos se escriben en memoria y entonces se vuelven a leer desde una memoria 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 permita que se transfieran datos entre los dos archivos de registro sin que se escriban y se vuelvan a leer).

El subconjunto local de la memoria caché de L2 2904 es parte de una memoria 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 memoria caché de L2 2904. Los datos leídos por un núcleo de procesador se almacenan en su subconjunto de memoria 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 memoria 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 asegura una coherencia para datos compartidos. La red en anillo es bidireccional para permitir que agentes tales como núcleos de procesador, memorias caché de L2 y otros bloques lógicos se comuniquen entre sí dentro del chip. Cada ruta de datos de anillo tiene 1012 bits de anchura por dirección.

La **figura 29B** es una vista ampliada de parte del núcleo de procesador en la **figura 29A** de acuerdo con realizaciones de la invención. La **figura 29B** incluye una memoria caché de datos de L1 2906A parte de la memoria caché de L1 2904, así como más detalles con respecto a la unidad de vectores 2910 y los registros de vector 2914. Específicamente, la unidad de vectores 2910 es una unidad de procesamiento de vectores (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 la alineación de las entradas de registro con la unidad de alineación 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 afirmar 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 de trazo continuo 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 de trazo discontinuo ilustra un procesador 3000 alternativo con múltiples núcleos 3002A-N, un conjunto de una unidad o más unidades de controlador de memoria integrado 3014 en la unidad de agente de sistema 3010 y la lógica de propósito especial 3008.

Por lo tanto, diferentes implementaciones del procesador 3000 pueden incluir: 1) una CPU con la lógica de propósito especial 3008 que es una lógica de gráficos y/o científica (de capacidad de proceso) integrada (que puede incluir uno o más núcleos), y los núcleos 3002A-N que son uno o más núcleos de propósito general (por ejemplo, núcleos en orden de propósito general, núcleos fuera de orden de propósito general, una combinación de los dos); 2) un coprocesador con los núcleos 3002A-N que son un gran número de núcleos de propósito especial destinados principalmente a cálculos gráficos y/o científicos (de capacidad de proceso); y 3) un coprocesador con los núcleos 3002A-N que son un gran número de núcleos en orden de propósito general. Por lo tanto, el procesador 3000 puede ser un procesador de propósito general, un coprocesador o un procesador de propósito especial, tal como, por ejemplo, un procesador de red o de comunicación, un motor de compresión, un procesador de gráficos, una GPGPU (unidad de procesamiento de gráficos de propósito general), un coprocesador de muchos núcleos integrados (MIC) de capacidad de proceso alta (que incluye 30 o más núcleos), un procesador integrado o similar. El procesador se puede implementar en uno o más chips. El procesador 3000 puede ser una parte y/o se puede implementar sobre 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 memoria caché dentro de los núcleos, un conjunto o una o más unidades de memoria caché compartidas 3006 y memoria externa (no mostrada) acoplada al conjunto de unidades de controlador de memoria integrado 3014. El conjunto de unidades de memoria caché compartidas 3006 puede incluir una o más memorias caché de nivel medio, tales como de nivel 2 (L2), de nivel 3 (L3), de nivel 4 (L4) o de otros niveles de memoria caché, una memoria caché de último nivel (LLC) y/o combinaciones de las mismas. Aunque, en una realización, una unidad de interconexión basada en anillo 3012 interconecta la lógica de gráficos integrados 3008 (la lógica de gráficos integrados 3008 es un ejemplo y también se denomina en el presente documento lógica de propósito

especial), el conjunto de unidades de memoria caché compartidas 3006 y la unidad de agente de sistema 3010/unidad o unidades de controlador de memoria integrado 3014, realizaciones alternativas pueden usar cualquier número de técnicas bien conocidas para interconectar tales unidades. En una realización, se mantiene la coherencia entre una o más unidades de memoria caché 3006 y los núcleos 3002-AN.

En algunas realizaciones, uno o más de los núcleos 3002A-N son capaces de multiproceso. El agente de sistema 3010 incluye aquellos componentes que coordinan y hacen funcionar los núcleos 3002A-N. La unidad de agente de sistema 3010 puede incluir, por ejemplo, una unidad de control de alimentación (PCU) y una unidad de visualización. La PCU puede ser o incluir una lógica y unos componentes necesarios para regular el estado de alimentación de los núcleos 3002A-N y la lógica de gráficos integrada 3008. La unidad de visualización es para accionar uno o más visualizadores conectados externamente.

Los núcleos 3002A-N pueden ser homogéneos o heterogéneos en términos del 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 de ordenador ilustrativas

Las **figuras 31-34** son diagramas de bloques de arquitecturas de ordenador ilustrativas. También son adecuados otros diseños y configuraciones de sistema conocidos en las materias para portátiles, equipos de sobremesa, PC portátiles, asistentes digitales personales, estaciones de trabajo para ingeniería, servidores, dispositivos de red, concentradores de red, conmutadores, procesadores integrados, procesadores de señales digitales (DSP), dispositivos de gráficos, dispositivos de videojuegos, descodificadores de salón, microcontroladores, teléfonos celulares, reproductores de medios portátiles, dispositivos de mano y diversos otros dispositivos electrónicos. En general, son generalmente adecuados una enorme diversidad de sistemas o dispositivos electrónicos capaces de incorporar un procesador y/u otra lógica de ejecución como se divulga en el presente documento.

Haciendo referencia a continuación a la **figura 31**, se muestra un diagrama de bloques 3100 de un sistema de acuerdo con una realización de la presente invención. El sistema 3100 puede incluir uno o más procesadores 3110, 3115, que se acoplan a un concentrador de controlador 3120. En una realización, el concentrador de controlador 3120 incluye un concentrador de controlador de memoria de gráficos (GMCH) 3190 y un Concentrador de Entrada/Salida (IOH) 3150 (que pueden estar en chips separados); el GMCH 3190 incluye controladores de memoria y de gráficos a los que se acoplan 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 de los controladores de memoria y de gráficos están integrados dentro del procesador (como se describe en el presente documento), la memoria 3140 y el coprocesador 3145 se acoplan directamente al procesador 3110, y el 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 de trazo discontinuo. 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 de acceso aleatorio dinámica (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 procesador o procesadores 3110, 3115 a través de un bus multipunto, tal como un bus de lado frontal (FSB), una interfaz de 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 de MIC de capacidad de proceso alta, un procesador de red o de comunicación, un motor de compresión, un procesador de gráficos, una GPGPU, un procesador integrado o similar. En una realización, el concentrador de controlador 3120 puede incluir un acelerador de gráficos integrado.

Puede haber una diversidad de diferencias entre los recursos físicos 3110, 3115 en términos de un espectro de métricas de méritos 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. Puede haber instrucciones de coprocesador integradas dentro de las instrucciones. El procesador 3110 reconoce estas instrucciones de coprocesador como de un tipo que debería ser ejecutado por el coprocesador 3145 adjunto. 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.

Haciendo referencia a continuación 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 de múltiples procesadores 3200 es un sistema de interconexión de punto a punto e incluye un primer

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

Cada uno de los procesadores 3270, 3280 puede intercambiar información con un conjunto de chips 3290 a través de las interfaces P-P 3252, 3254 individuales usando los circuitos de interfaz de 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 desempeño alto 3292. En una realización, el coprocesador 3238 es un procesador de propósito especial, tal como, por ejemplo, un procesador de MIC de capacidad de proceso alta, un procesador de red o de comunicación, un motor de compresión, un procesador de gráficos, una GPGPU, un 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 a través 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 puede almacenar en la memoria caché compartida si un procesador se coloca en 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 se limita a ello.

Como se muestra en la **figura 32**, diversos dispositivos de E/S 3214 se pueden acoplar 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, un procesador o más procesadores 3215 adicionales, tales como coprocesadores, procesadores de MIC de capacidad de proceso alta, unas 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, se acoplan al primer bus 3216. En una realización, el segundo bus 3220 puede ser un bus de recuento de pines bajo (LPC). Diversos dispositivos se pueden acoplar a un segundo bus 3220 incluyendo, por ejemplo, un teclado y/o un ratón 3222, unos dispositivos de comunicación 3227 y una unidad de almacenamiento 3228, tal como una unidad de disco u otro dispositivo de almacenamiento masivo que puede incluir instrucciones/código y datos 3230, en una realización. Además, una E/S de audio 3224 se puede acoplar 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.

Haciendo referencia a continuación 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 semejantes en las **figuras 32 y 33** llevan números de referencia semejantes, y ciertos aspectos de la **figura 32** se han omitido de la **figura 33** con el fin de evitar complicar otros aspectos de la **figura 33**.

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

Haciendo referencia a continuación a la **figura 34**, se muestra un SoC 3400 de un sistema de acuerdo con una realización de la presente invención. Elementos semejantes en la **figura 30** llevan números de referencia semejantes. Además, los recuadros con líneas de trazo discontinuo son características opcionales en SoC más avanzados. En la **figura 34**, una unidad o unidades de interconexión 3402 se acoplan a: un procesador de aplicaciones 3410 que incluye un conjunto de uno o más núcleos 3002A-N, que incluyen las unidades de memoria caché 3004A-N y la unidad o unidades de memoria 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 a memoria directo (DMA) 3432; y una unidad de visualización 3440 para acoplar a uno o más visualizadores externos. En una realización, el coprocesador o coprocesadores 3420 incluyen un procesador de propósito especial,

tal como, por ejemplo, un procesador de red o de comunicación, un motor de compresión, una GPGPU, un procesador de MIC de capacidad de proceso alta, un procesador integrado o similar.

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

Un código de programa, tal como el código 3230 ilustrado en la **figura 32**, se puede aplicar a unas instrucciones de entrada para realizar las funciones descritas en el presente documento y generar una información de salida. La información de salida se puede aplicar a uno o más dispositivos de salida, de una forma conocida. Para los fines 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 de aplicación específica (ASIC) o un microprocesador.

El código de programa se puede implementar en un lenguaje de programación orientado a objetos o de procedimientos 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 lenguaje de programación particular alguno. En cualquier caso, el lenguaje puede ser un lenguaje compilado o interpretado.

Uno o más aspectos de al menos una realización se pueden implementar mediante instrucciones representativas almacenadas en un medio legible por máquina que representa diversas lógicas dentro del procesador, que, cuando son leídas por una máquina, hacen que la máquina fabrique una lógica para realizar las técnicas descritas en el presente documento. Tales representaciones, conocidas como "núcleos de IP", se pueden almacenar 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, incluyendo medios de almacenamiento tales como discos duros, cualquier otro tipo de disco, incluyendo disquetes, discos ópticos, discos compactos - memorias de solo lectura (CD-ROM), discos compactos regrabables (CD-RW) y discos magnetoópticos, 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), memoria de cambio de fase (PCM), tarjetas magnéticas u ópticas, o cualquier otro tipo de medio adecuado para almacenar instrucciones electrónicas.

En consecuencia, realizaciones de la invención también incluyen medios legibles por máquina tangibles no transitorios que contienen instrucciones o que contienen datos de diseño, tales como Lenguaje de Descripción de Hardware (HDL), que definen estructuras, circuitos, aparatos, procesadores y/o características de sistema descritos en el presente documento. Tales realizaciones también se pueden denominar productos de programa.

Emulación (incluyendo traducción binaria, transformación de código, 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 de destino. 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 a procesar por el núcleo. El convertidor de instrucciones se puede implementar en software, hardware, firmware o una combinación de los mismos. El convertidor de instrucciones puede estar en un procesador, fuera de un procesador o en parte dentro y en parte fuera de un procesador.

La **figura 35** es un diagrama de bloques que contrapone 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 se puede implementar en software, firmware, hardware o diversas combinaciones de los mismos. La **figura 35** muestra que un programa en un lenguaje de alto nivel 3502 se puede compilar usando un compilador de x86 3504 para generar un código binario de x86 3506 que puede ser ejecutado de forma nativa por un procesador con al menos un núcleo de conjunto de instrucciones de x86 3516. El procesador con al menos un núcleo de conjunto de instrucciones de x86 3516 representa cualquier procesador que pueda realizar sustancialmente las mismas funciones que un procesador de Intel con al menos un núcleo de conjunto de instrucciones de x86 ejecutando o procesando de otro modo, de forma compatible, (1) una porción sustancial del conjunto de instrucciones del núcleo de conjunto de instrucciones de x86 de Intel o (2) versiones de código objeto de aplicaciones u otro software destinado a ejecutarse en un procesador de

Intel con al menos un núcleo de conjunto de instrucciones de x86, con el fin de lograr sustancialmente el mismo resultado que un procesador de Intel con al menos un núcleo de conjunto de instrucciones de x86. El compilador de x86 3504 representa un compilador que se puede hacer funcionar para generar un código binario de x86 3506 (por ejemplo, código objeto) que, con o sin un procesamiento de vinculación adicional, se puede ejecutar en el procesador con al menos un núcleo de conjunto de instrucciones de x86 3516. De forma similar, la **figura 35** muestra que el programa en el lenguaje de alto nivel 3502 se puede compilar usando un compilador de conjunto de instrucciones alternativo 3508 para generar un código binario de conjunto de instrucciones alternativo 3510 que puede ser ejecutado de forma nativa por un procesador sin al menos un núcleo de conjunto de instrucciones de 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 de x86 3506 en un código que puede ser ejecutado de forma nativa por el procesador sin un núcleo de conjunto de instrucciones de x86 3514. No es probable que este código convertido sea el mismo que el código binario de conjunto de instrucciones alternativo 3510 debido a que es difícil hacer un convertidor de instrucciones capaz de esto; sin embargo, el código convertido logrará el funcionamiento general y estará compuesto por instrucciones a partir del conjunto de instrucciones alternativo. Por lo tanto, el convertidor de instrucciones 3512 representa software, firmware, hardware o una combinación de los mismos que, a través de 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 de x86 ejecute el código binario de x86 3506.

Los siguientes ejemplos de procesadores, métodos y sistemas son útiles para comprender la presente invención como se ha descrito anteriormente y como se define por las reivindicaciones. Los ejemplos 21 a 140 no cubiertos por las reivindicaciones se deberían considerar como presentados solo con fines ilustrativos

El ejemplo 1 se refiere a un procesador que comprende: una circuitería de descodificación para descodificar una instrucción, incluyendo la instrucción un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice usando una o más comparaciones del propio elemento y con otros elementos de datos del vector de origen, y permutando los valores de los elementos del vector de origen basándose en los valores de índice para los elementos; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

El ejemplo 2 se refiere al procesador como se describe en el ejemplo 1, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 3 se refiere al procesador como se describe en cualquiera de los ejemplos 1-2, en donde el procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 4 se refiere al procesador como se describe en cualquiera de los ejemplos 1-3, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

El ejemplo 5 se refiere al procesador como se describe en cualquiera de los ejemplos 1-3, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 6 se refiere al procesador como se describe en cualquiera de los ejemplos 1-5, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 7 se refiere al procesador como se describe en cualquiera de los ejemplos 1-6, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 8 se refiere al procesador como se describe en cualquiera de los ejemplos 1-7, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 9 se refiere al procesador como se describe en cualquiera de los ejemplos 1-7, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 10 se refiere a un método que comprende: descodificar una instrucción, incluyendo la instrucción un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice usando una o más comparaciones del propio elemento y con otros elementos de datos del vector de origen, y permutando los valores de los elementos del

vector de origen basándose en los valores de índice para los elementos; y ejecutar la instrucción descodificada usando una circuitería de ejecución como es indicado por el código de operación.

El ejemplo 11 se refiere al método según se describe en el ejemplo 10, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 12 se refiere al método como se describe en cualquiera de los ejemplos 10-11, en donde el procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 13 se refiere al método como se describe en cualquiera de los ejemplos 10-12, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

El ejemplo 14 se refiere al método como se describe en cualquiera de los ejemplos 10-12, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 15 se refiere al método como se describe en cualquiera de los ejemplos 10-14, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 16 se refiere al método como se describe en cualquiera de los ejemplos 10-15, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 17 se refiere al método como se describe en cualquiera de los ejemplos 10-16, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 18 se refiere al método como se describe en cualquiera de los ejemplos 10-16, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 19 se refiere al medio no transitorio para almacenar una instrucción que, cuando es procesada por un aparato, ha de hacer que se realice el método según se describe en cualquiera de los ejemplos 10-18.

El ejemplo 20 se refiere a un sistema que comprende: una memoria para almacenar una aplicación de trazado de rayos; y un procesador acoplado a la memoria. El procesador comprende: una circuitería de descodificación para descodificar una instrucción de la aplicación de trazado de rayos, incluyendo la instrucción un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice usando una o más comparaciones del propio elemento y con otros elementos de datos del vector de origen, y permutando los valores de los elementos del vector de origen basándose en los valores de índice para los elementos; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

El ejemplo 21 se refiere a un procesador que comprende: una circuitería de descodificación para descodificar una instrucción, incluyendo la instrucción un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice, y permutando los valores de los elementos del vector de origen en el vector de destino basándose en los valores de índice para el elemento; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

El ejemplo 22 se refiere al procesador como se describe en el ejemplo 21, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 23 se refiere al procesador como se describe en cualquiera de los ejemplos 21-22, en donde el procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 24 se refiere al procesador como se describe en cualquiera de los ejemplos 21-23, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

El ejemplo 25 se refiere al procesador como se describe en cualquiera de los ejemplos 21-23, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 26 se refiere al procesador como se describe en cualquiera de los ejemplos 21-25, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 27 se refiere al procesador como se describe en cualquiera de los ejemplos 21-26, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 28 se refiere al procesador como se describe en cualquiera de los ejemplos 21-27, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 29 se refiere al procesador como se describe en cualquiera de los ejemplos 21-27, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 30 se refiere a un método que comprende: descodificar una instrucción, incluyendo la instrucción un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice, y permutando los valores de los elementos del vector de origen en el vector de destino basándose en los valores de índice para el elemento; y ejecutar la instrucción descodificada usando una circuitería de ejecución como es indicado por el código de operación.

El ejemplo 31 se refiere al método según se describe en el ejemplo 30, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 32 se refiere al método como se describe en cualquiera de los ejemplos 30-31, en donde el procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 33 se refiere al método como se describe en cualquiera de los ejemplos 30-32, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

El ejemplo 34 se refiere al método como se describe en cualquiera de los ejemplos 30-32, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 35 se refiere al método como se describe en cualquiera de los ejemplos 30-34, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 36 se refiere al método como se describe en cualquiera de los ejemplos 30-35, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 37 se refiere al método como se describe en cualquiera de los ejemplos 30-36, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 38 se refiere al método como se describe en cualquiera de los ejemplos 30-36, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 39 se refiere al medio no transitorio para almacenar una instrucción que, cuando es procesada por un aparato, ha de hacer que se realice el método según se describe en cualquiera de los ejemplos 30-38.

El ejemplo 40 se refiere a un sistema que comprende: una memoria para almacenar una aplicación de trazado de rayos; y un procesador acoplado a la memoria. El procesador comprende: una circuitería de descodificación para descodificar una instrucción de la aplicación de trazado de rayos, incluyendo la instrucción un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice, y permutando los valores de los elementos del vector de origen en el vector de destino basándose en los valores de índice para el elemento; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

El ejemplo 41 se refiere a un procesador que comprende: una circuitería de descodificación para descodificar una

instrucción, incluyendo la instrucción un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para indexar valores del vector de origen y almacenar un resultado de la indexación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice usando una o más comparaciones; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

El ejemplo 42 se refiere al procesador como se describe en el ejemplo 41, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 43 se refiere al procesador como se describe en cualquiera de los ejemplos 41-42, en donde el procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 44 se refiere al procesador como se describe en cualquiera de los ejemplos 41-43, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

El ejemplo 45 se refiere al procesador como se describe en cualquiera de los ejemplos 41-43, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 46 se refiere al procesador como se describe en cualquiera de los ejemplos 41-45, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 47 se refiere al procesador como se describe en cualquiera de los ejemplos 41-46, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 48 se refiere al procesador como se describe en cualquiera de los ejemplos 41-47, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 49 se refiere al procesador como se describe en cualquiera de los ejemplos 41-47, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 50 se refiere a un método que comprende: descodificar una instrucción, incluyendo la instrucción un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para indexar valores del vector de origen y almacenar un resultado de la indexación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice usando una o más comparaciones; y ejecutar la instrucción descodificada usando una circuitería de ejecución como es indicado por el código de operación.

El ejemplo 51 se refiere al método según se describe en el ejemplo 50, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 52 se refiere al método como se describe en cualquiera de los ejemplos 50-51, en donde el procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 53 se refiere al método como se describe en cualquiera de los ejemplos 50-52, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

El ejemplo 54 se refiere al método como se describe en cualquiera de los ejemplos 50-52, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 55 se refiere al método como se describe en cualquiera de los ejemplos 50-54, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 56 se refiere al método como se describe en cualquiera de los ejemplos 50-55, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 57 se refiere al método como se describe en cualquiera de los ejemplos 50-56, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 58 se refiere al método como se describe en cualquiera de los ejemplos 50-56, en donde la circuitería de

ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 59 se refiere al medio no transitorio para almacenar una instrucción que, cuando es procesada por un aparato, ha de hacer que se realice el método según se describe en cualquiera de los ejemplos 50-58.

El ejemplo 60 se refiere a un sistema que comprende: una memoria para almacenar una aplicación de trazado de rayos; y un procesador acoplado a la memoria. El procesador comprende: una circuitería de descodificación para descodificar una instrucción de la aplicación de trazado de rayos, incluyendo la instrucción un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para indexar valores del vector de origen y almacenar un resultado de la indexación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice usando una o más comparaciones; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

El ejemplo 61 se refiere a un procesador que comprende: una circuitería de descodificación para descodificar una instrucción, incluyendo la instrucción un primer campo de operando para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para indexar valores del vector de origen y almacenar un resultado de la indexación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

El ejemplo 62 se refiere al procesador como se describe en el ejemplo 61, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 63 se refiere al procesador como se describe en cualquiera de los ejemplos 61-62, en donde el procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 64 se refiere al procesador como se describe en cualquiera de los ejemplos 61-63, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

El ejemplo 65 se refiere al procesador como se describe en cualquiera de los ejemplos 61-63, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 66 se refiere al procesador como se describe en cualquiera de los ejemplos 61-65, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 67 se refiere al procesador como se describe en cualquiera de los ejemplos 61-66, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 68 se refiere al procesador como se describe en cualquiera de los ejemplos 61-67, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 69 se refiere al procesador como se describe en cualquiera de los ejemplos 61-67, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 70 se refiere a un método que comprende: descodificar una instrucción, incluyendo la instrucción un primer campo de operando para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para indexar valores del vector de origen y almacenar un resultado de la indexación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice; y ejecutar la instrucción descodificada usando una circuitería de ejecución como es indicado por el código de operación.

El ejemplo 71 se refiere al método según se describe en el ejemplo 70, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 72 se refiere al método como se describe en cualquiera de los ejemplos 70-71, en donde el procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 73 se refiere al método como se describe en cualquiera de los ejemplos 70-72, en donde las ubicaciones

del vector de destino y los vectores de origen son registros de vector.

El ejemplo 74 se refiere al método como se describe en cualquiera de los ejemplos 70-72, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 75 se refiere al método como se describe en cualquiera de los ejemplos 70-74, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 76 se refiere al método como se describe en cualquiera de los ejemplos 70-75, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 77 se refiere al método como se describe en cualquiera de los ejemplos 70-76, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 78 se refiere al método como se describe en cualquiera de los ejemplos 70-76, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 79 se refiere al medio no transitorio para almacenar una instrucción que, cuando es procesada por un aparato, ha de hacer que se realice el método según se describe en cualquiera de los ejemplos 70-78.

El ejemplo 80 se refiere a un sistema que comprende: una memoria para almacenar una aplicación de trazado de rayos; y un procesador acoplado a la memoria. El procesador comprende: una circuitería de descodificación para descodificar una instrucción de la aplicación de trazado de rayos, incluyendo la instrucción un primer campo de operando para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para indexar valores del vector de origen y almacenar un resultado de la indexación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

El ejemplo 81 se refiere a un procesador que comprende: una circuitería de descodificación para descodificar una instrucción, incluyendo la instrucción un primer campo para identificar una ubicación de un primer vector de origen, un segundo campo para identificar una ubicación de un vector de destino, un tercer campo para identificar una ubicación de un segundo vector de origen y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para permutar los valores de elementos del primer vector de origen basándose en un índice almacenado en el segundo vector de origen y almacenar el resultado de la permutación en el vector de destino; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

El ejemplo 82 se refiere al procesador como se describe en el ejemplo 81, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 83 se refiere al procesador como se describe en cualquiera de los ejemplos 81-82, en donde el procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 84 se refiere al procesador como se describe en cualquiera de los ejemplos 81-83, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

El ejemplo 85 se refiere al procesador como se describe en cualquiera de los ejemplos 81-83, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 86 se refiere al procesador como se describe en cualquiera de los ejemplos 81-85, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 87 se refiere al procesador como se describe en cualquiera de los ejemplos 81-86, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 88 se refiere al procesador como se describe en cualquiera de los ejemplos 81-87, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 89 se refiere al procesador como se describe en cualquiera de los ejemplos 81-87, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de

procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 90 se refiere a un método que comprende: descodificar una instrucción, incluyendo la instrucción un primer campo para identificar una ubicación de un primer vector de origen, un segundo campo para identificar una ubicación de un vector de destino, un tercer campo para identificar una ubicación de un segundo vector de origen y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para permutar los valores de elementos del primer vector de origen basándose en un índice almacenado en el segundo vector de origen y almacenar el resultado de la permutación en el vector de destino; y ejecutar la instrucción descodificada usando una circuitería de ejecución como es indicado por el código de operación.

El ejemplo 91 se refiere al método según se describe en el ejemplo 90, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 92 se refiere al método como se describe en cualquiera de los ejemplos 90-91, en donde el procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 93 se refiere al método como se describe en cualquiera de los ejemplos 90-92, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

El ejemplo 94 se refiere al método como se describe en cualquiera de los ejemplos 90-92, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 95 se refiere al método como se describe en cualquiera de los ejemplos 90-94, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 96 se refiere al método como se describe en cualquiera de los ejemplos 90-95, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 97 se refiere al método como se describe en cualquiera de los ejemplos 90-96, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 98 se refiere al método como se describe en cualquiera de los ejemplos 90-96, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 99 se refiere al medio no transitorio para almacenar una instrucción que, cuando es procesada por un aparato, ha de hacer que se realice el método según se describe en cualquiera de los ejemplos 90-98.

El ejemplo 100 se refiere a un sistema que comprende: una memoria para almacenar una aplicación de trazado de rayos; y un procesador acoplado a la memoria. Comprendiendo el procesador: una circuitería de descodificación para descodificar una instrucción de la aplicación de trazado de rayos, incluyendo la instrucción un primer campo para identificar una ubicación de un primer vector de origen, un segundo campo para identificar una ubicación de un vector de destino, un tercer campo para identificar una ubicación de un segundo vector de origen y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para permutar los valores de elementos del primer vector de origen basándose en un índice almacenado en el segundo vector de origen y almacenar el resultado de la permutación en el vector de destino; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

El ejemplo 101 se refiere a un procesador que comprende: una circuitería de descodificación para descodificar una instrucción, incluyendo la instrucción un primer campo para identificar una ubicación de un primer vector de origen, un segundo campo para identificar una ubicación de un primer vector de origen, un tercer campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada posición de elemento del primer y el segundo vectores de origen, un valor de índice usando una o más comparaciones de los elementos, y permutando y almacenando los valores de los elementos basándose en los valores de índice para los elementos en el vector de destino; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

El ejemplo 102 se refiere al procesador como se describe en el ejemplo 101, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 103 se refiere al procesador como se describe en cualquiera de los ejemplos 101-102, en donde el

procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 104 se refiere al procesador como se describe en cualquiera de los ejemplos 101-103, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

El ejemplo 105 se refiere al procesador como se describe en cualquiera de los ejemplos 101-103, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 106 se refiere al procesador como se describe en cualquiera de los ejemplos 101-105, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 107 se refiere al procesador como se describe en cualquiera de los ejemplos 101-106, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 108 se refiere al procesador como se describe en cualquiera de los ejemplos 101-107, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 109 se refiere al procesador como se describe en cualquiera de los ejemplos 101-107, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 110 se refiere a un método que comprende: descodificar una instrucción, incluyendo la instrucción un primer campo para identificar una ubicación de un primer vector de origen, un segundo campo para identificar una ubicación de un primer vector de origen, un tercer campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada posición de elemento del primer y el segundo vectores de origen, un valor de índice usando una o más comparaciones de los elementos, y permutando y almacenando los valores de los elementos basándose en los valores de índice para los elementos en el vector de destino; y ejecutar la instrucción descodificada usando una circuitería de ejecución como es indicado por el código de operación.

El ejemplo 111 se refiere al método según se describe en el ejemplo 110, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 112 se refiere al método como se describe en cualquiera de los ejemplos 110-111, en donde el procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 113 se refiere al método como se describe en cualquiera de los ejemplos 110-112, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

El ejemplo 114 se refiere al método como se describe en cualquiera de los ejemplos 110-112, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 115 se refiere al método como se describe en cualquiera de los ejemplos 110-114, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 116 se refiere al método como se describe en cualquiera de los ejemplos 110-115, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 117 se refiere al método como se describe en cualquiera de los ejemplos 110-116, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 118 se refiere al método como se describe en cualquiera de los ejemplos 110-116, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 119 se refiere al medio no transitorio para almacenar una instrucción que, cuando es procesada por un aparato, ha de hacer que se realice el método según se describe en cualquiera de los ejemplos 110-118.

El ejemplo 120 se refiere a un sistema que comprende: una memoria para almacenar una aplicación de trazado de

rayos; y un procesador acoplado a la memoria. El procesador comprende: una circuitería de descodificación para descodificar una instrucción de la aplicación de trazado de rayos, incluyendo la instrucción un primer campo para identificar una ubicación de un primer vector de origen, un segundo campo para identificar una ubicación de un primer vector de origen, un tercer campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada posición de elemento del primer y el segundo vectores de origen, un valor de índice usando una o más comparaciones de los elementos, y permutando y almacenando los valores de los elementos basándose en los valores de índice para los elementos en el vector de destino; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

El ejemplo 121 se refiere a un procesador que comprende: una circuitería de descodificación para descodificar una instrucción, la instrucción para un primer campo para identificar una ubicación de un primer vector de origen, un segundo campo para identificar una ubicación de un primer vector de origen, un tercer campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para, para cada posición de elemento del primer y el segundo vectores de origen, generar un valor de índice usando una o más comparaciones de los elementos y almacenar el valor de índice generado en una posición de elemento correspondiente del vector de destino; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

El ejemplo 122 se refiere al procesador como se describe en el ejemplo 121, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 123 se refiere al procesador como se describe en cualquiera de los ejemplos 121-122, en donde el procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 124 se refiere al procesador como se describe en cualquiera de los ejemplos 121-123, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

El ejemplo 125 se refiere al procesador como se describe en cualquiera de los ejemplos 121-123, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 126 se refiere al procesador como se describe en cualquiera de los ejemplos 121-125, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 127 se refiere al procesador como se describe en cualquiera de los ejemplos 121-126, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 128 se refiere al procesador como se describe en cualquiera de los ejemplos 121-127, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 129 se refiere al procesador como se describe en cualquiera de los ejemplos 121-127, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 130 se refiere a un método que comprende: descodificar una instrucción, incluyendo la instrucción un primer campo para identificar una ubicación de un primer vector de origen, un segundo campo para identificar una ubicación de un primer vector de origen, un tercer campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para, para cada posición de elemento del primer y el segundo vectores de origen, generar un valor de índice usando una o más comparaciones de los elementos y almacenar el valor de índice generado en una posición de elemento correspondiente del vector de destino; y ejecutar la instrucción descodificada usando una circuitería de ejecución como es indicado por el código de operación.

El ejemplo 131 se refiere al método según se describe en el ejemplo 130, en donde la instrucción es una parte de una aplicación de trazado de rayos.

El ejemplo 132 se refiere al método como se describe en cualquiera de los ejemplos 130-131, en donde el procesador es una unidad de procesamiento de gráficos (GPU) que soporta un trazado de rayos.

El ejemplo 133 se refiere al método como se describe en cualquiera de los ejemplos 130-132, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

El ejemplo 134 se refiere al método como se describe en cualquiera de los ejemplos 130-132, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

El ejemplo 135 se refiere al método como se describe en cualquiera de los ejemplos 130-134, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

El ejemplo 136 se refiere al método como se describe en cualquiera de los ejemplos 130-135, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

El ejemplo 137 se refiere al método como se describe en cualquiera de los ejemplos 130-136, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales.

El ejemplo 138 se refiere al método como se describe en cualquiera de los ejemplos 130-136, en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

El ejemplo 139 se refiere al medio no transitorio para almacenar una instrucción que, cuando es procesada por un aparato, ha de hacer que se realice el método según se describe en cualquiera de los ejemplos 130-138.

El ejemplo 140 se refiere a un sistema que comprende: una memoria para almacenar una aplicación de trazado de rayos; y un procesador acoplado a la memoria. Comprendiendo el procesador: una circuitería de descodificación para descodificar una instrucción de la aplicación de trazado de rayos, incluyendo la instrucción un primer campo para identificar una ubicación de un primer vector de origen, un segundo campo para identificar una ubicación de un primer vector de origen, un tercer campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para, para cada posición de elemento del primer y el segundo vectores de origen, generar un valor de índice usando una o más comparaciones de los elementos y almacenar el valor de índice generado en una posición de elemento correspondiente del vector de destino; y una circuitería de ejecución para ejecutar la instrucción descodificada como es indicado por el código de operación.

REIVINDICACIONES

1. Un procesador (1390) que comprende:
una circuitería de descodificación (1303) para descodificar una instrucción, incluyendo la instrucción un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución (1311) que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice usando una o más comparaciones del propio elemento y con otros elementos de datos del vector de origen, y permutando los valores de los elementos del vector de origen basándose en los valores de índice para los elementos; y
una circuitería de ejecución (1311) para ejecutar la instrucción descodificada como es indicado por el código de operación,
en donde la circuitería de ejecución (1311) comprende una circuitería de operaciones matriciales (2301) para generar el índice y una circuitería de procesamiento de vectores (2303) para permutar y almacenar los valores de los elementos basándose en los valores de índice.

2. El procesador (1390) de la reivindicación 1, en donde la instrucción es una parte de una aplicación de trazado de rayos.

3. El procesador (1390) de la reivindicación 2, en donde el procesador (1390) es una unidad de procesamiento de gráficos, GPU, que soporta un trazado de rayos.

4. El procesador (1390) de cualquiera de las reivindicaciones 1-3, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

5. El procesador (1390) de cualquiera de las reivindicaciones 1-3, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

6. El procesador (1390) de cualquiera de las reivindicaciones 1-5, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

7. El procesador (1390) de cualquiera de las reivindicaciones 1-6, en donde, para desempatar resultados de comparación, la circuitería de ejecución (1311) ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

8. Un método en un procesador, comprendiendo el método:
descodificar, por una circuitería de descodificación del procesador, una instrucción (3603), incluyendo la instrucción un primer campo para identificar una ubicación de un vector de origen, un segundo campo para identificar una ubicación de un vector de destino y un código de operación para indicar a una circuitería de ejecución que ejecute la instrucción descodificada para clasificar valores del vector de origen y almacenar un resultado de la clasificación en el vector de destino generando, por cada elemento del vector de origen, un valor de índice usando una o más comparaciones del propio elemento y con otros elementos de datos del vector de origen, y permutando los valores de los elementos del vector de origen basándose en los valores de índice para los elementos; y
ejecutar, por la circuitería de ejecución del procesador, la instrucción descodificada como es indicado por el código de operación (3607),
en donde la circuitería de ejecución comprende una circuitería de operaciones matriciales para generar el índice y una circuitería de procesamiento de vectores para permutar y almacenar los valores de los elementos basándose en los valores de índice.

9. El método de la reivindicación 8, en donde la instrucción es una parte de una aplicación de trazado de rayos.

10. El método de la reivindicación de la reivindicación 9, en donde el procesador es una unidad de procesamiento de gráficos, GPU, que soporta un trazado de rayos.

11. El método de cualquiera de las reivindicaciones 8-10, en donde las ubicaciones del vector de destino y los vectores de origen son registros de vector.

12. El método de cualquiera de las reivindicaciones 8-10, en donde la ubicación del vector de destino es un registro de vector y la ubicación del vector de origen es al menos una ubicación en memoria.

13. El método de cualquiera de las reivindicaciones 8-12, en donde un tipo de al menos una de las comparaciones es uno de igual a, mayor que, mayor que o igual a, menor que y menor que o igual a.

14. El método de cualquiera de las reivindicaciones 8-13, en donde, para desempatar resultados de comparación, la circuitería de ejecución ha de realizar una primera comparación entre elementos y una segunda comparación entre elementos.

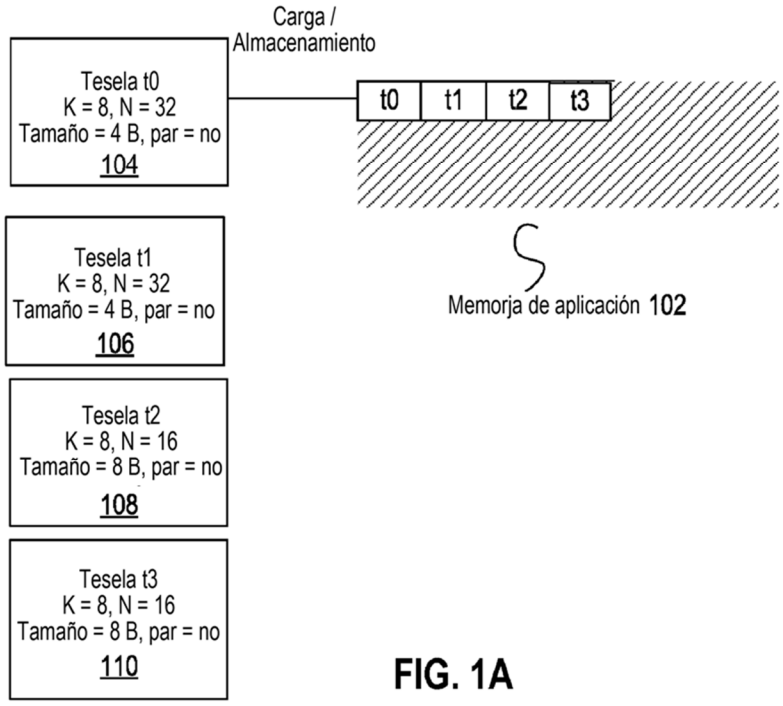

Carga /
Almacenamiento
Tesela t0
K = 8, N = 32
Tamaño = 4 B, par = no
104
t0
t1
t2
t3
Tesela t1
K = 8, N = 32
Tamaño = 4 B, par = no
106
Memorja de aplicación 102
Tesela t2
K = 8, N = 16
Tamaño = 8 B, par = no
108
Tesela t3
K = 8, N = 16
Tamaño = 8 B, par = no
110

FIG. 1A

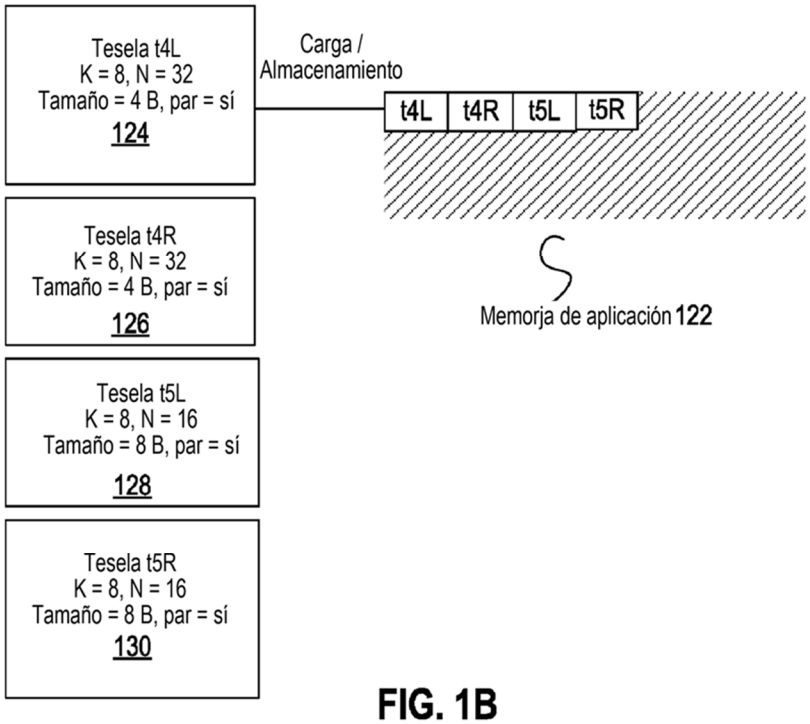

Tesela t4L
K = 8, N = 32
Tamaño = 4 B, par = sí
124
Carga /
Almacenamiento
t4L
t4R
t5L
t5R
Tesela t4R
K = 8, N = 32
Tamaño = 4 B, par = sí
126
Memorja de aplicación 122
Tesela t5L
K = 8, N = 16
Tamaño = 8 B, par = sí
128
Tesela t5R
K = 8, N = 16
Tamaño = 8 B, par = sí
130

FIG. 1B

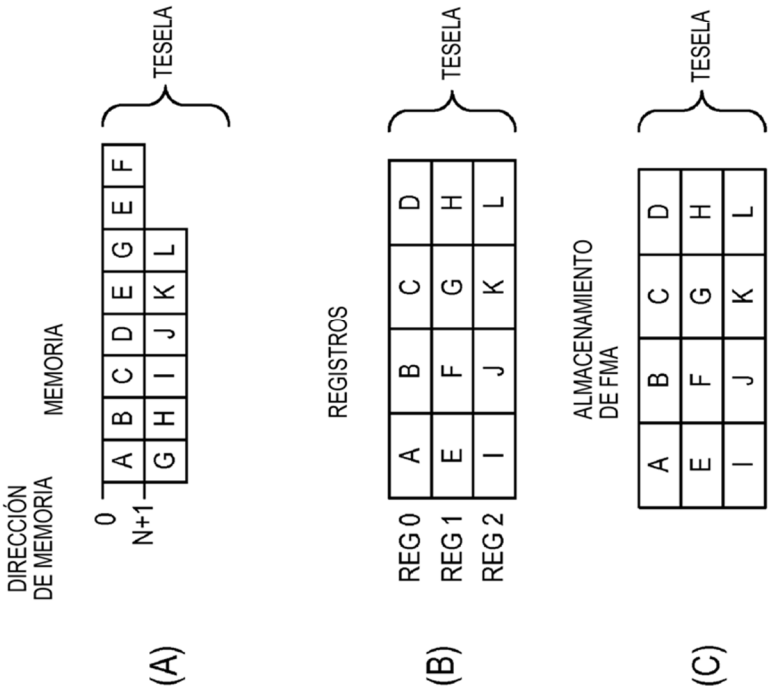

DIRECCIÓN DE MEMORIA
MEMORIA
0
N+1
A B C D E G E F
G H I J K L
TESELA
(A)
REGISTROS
REG 0
REG 1
REG 2
A B C D
E F G H
I J K L
TESELA
(B)
ALMACENAMIENTO DE FMA
A B C D
E F G H
I J K L
TESELA
(C)

FIG. 2

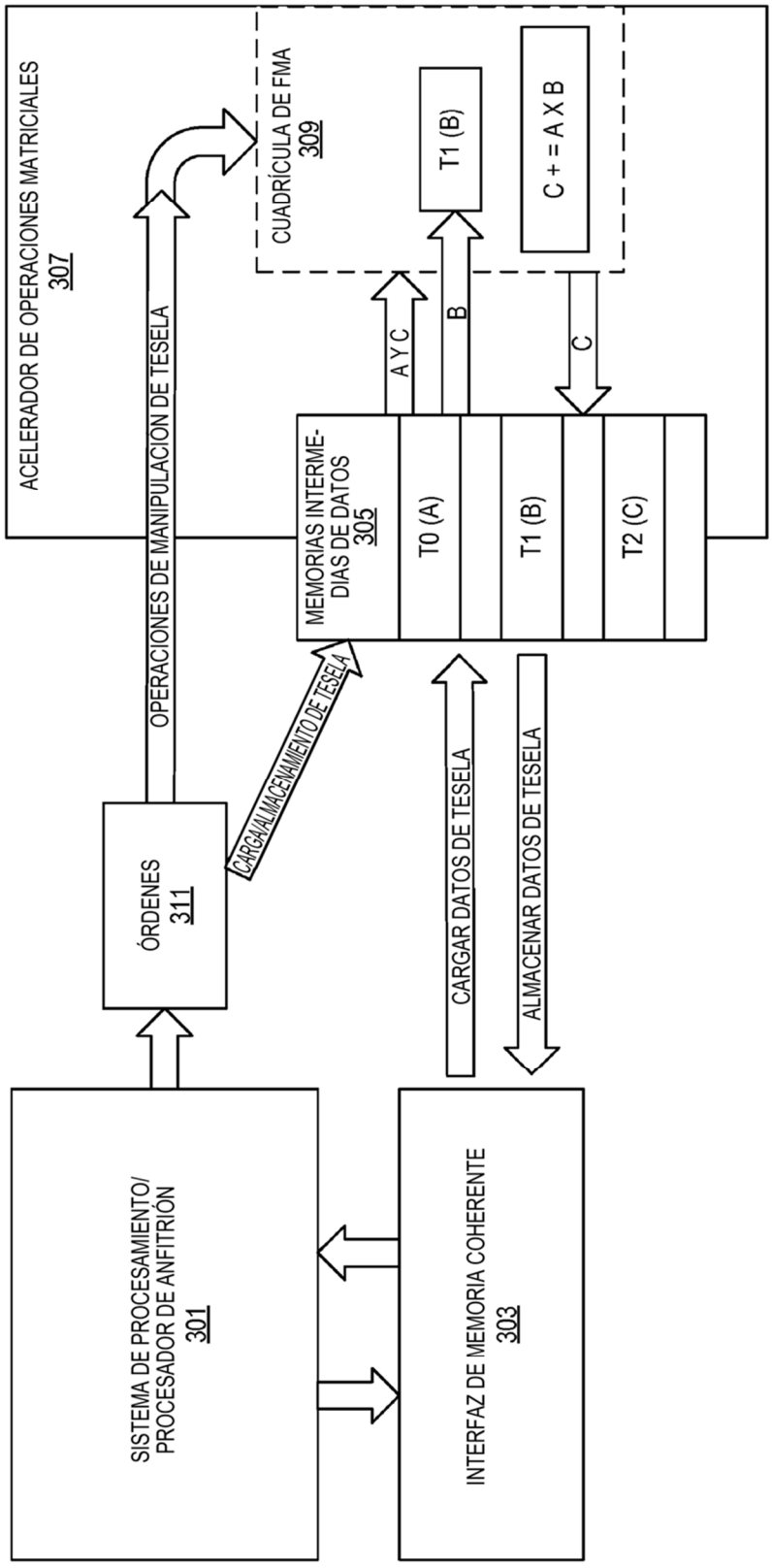

ACELERADOR DE OPERACIONES MATRICIALES
307
CUADRÍCULA DE FMA
309
T1 (B)
C + = A X B
A Y C
B
C
OPERACIONES DE MANIPULACIÓN DE TESELA
MEMORIAS INTERMEDIAS DE DATOS
305
T0 (A)
T1 (B)
T2 (C)
CARGA/ALMACENAMIENTO DE TESELA
ÓRDENES
311
CARGAR DATOS DE TESELA
ALMACENAR DATOS DE TESELA
SISTEMA DE PROCESAMIENTO/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

FIG. 3

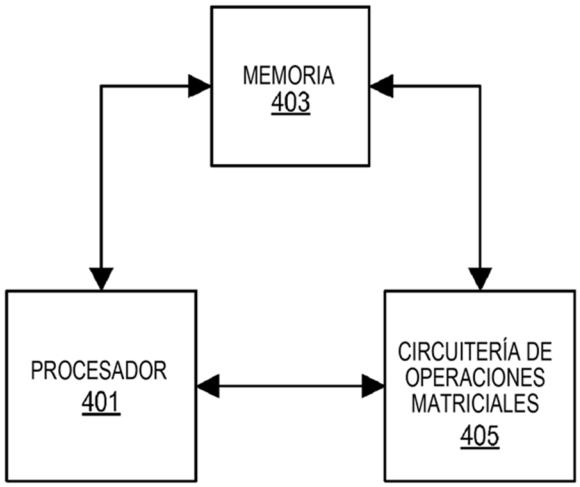

MEMORIA
403
PROCESADOR
401
CIRCUITERÍA DE
OPERACIONES
MATRICIALES
405

FIG. 4

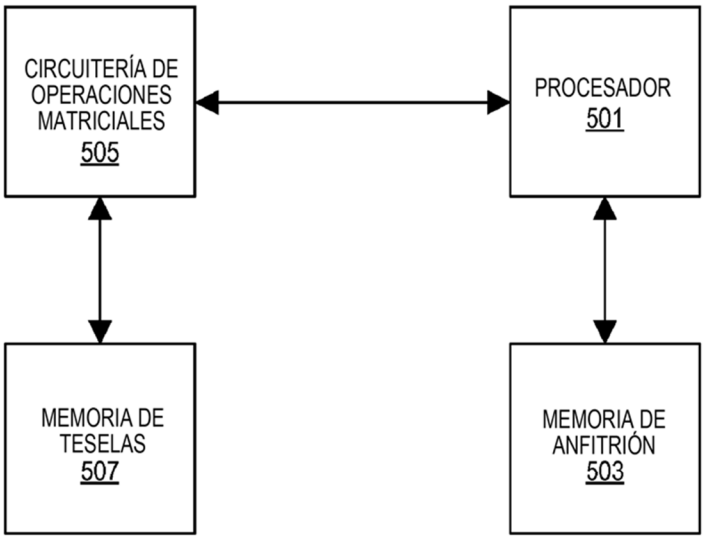

CIRCUITERÍA DE
OPERACIONES
MATRICIALES
505
PROCESADOR
501
MEMORIA DE
TESELAS
507
MEMORIA DE
ANFITRIÓN
503

FIG. 5

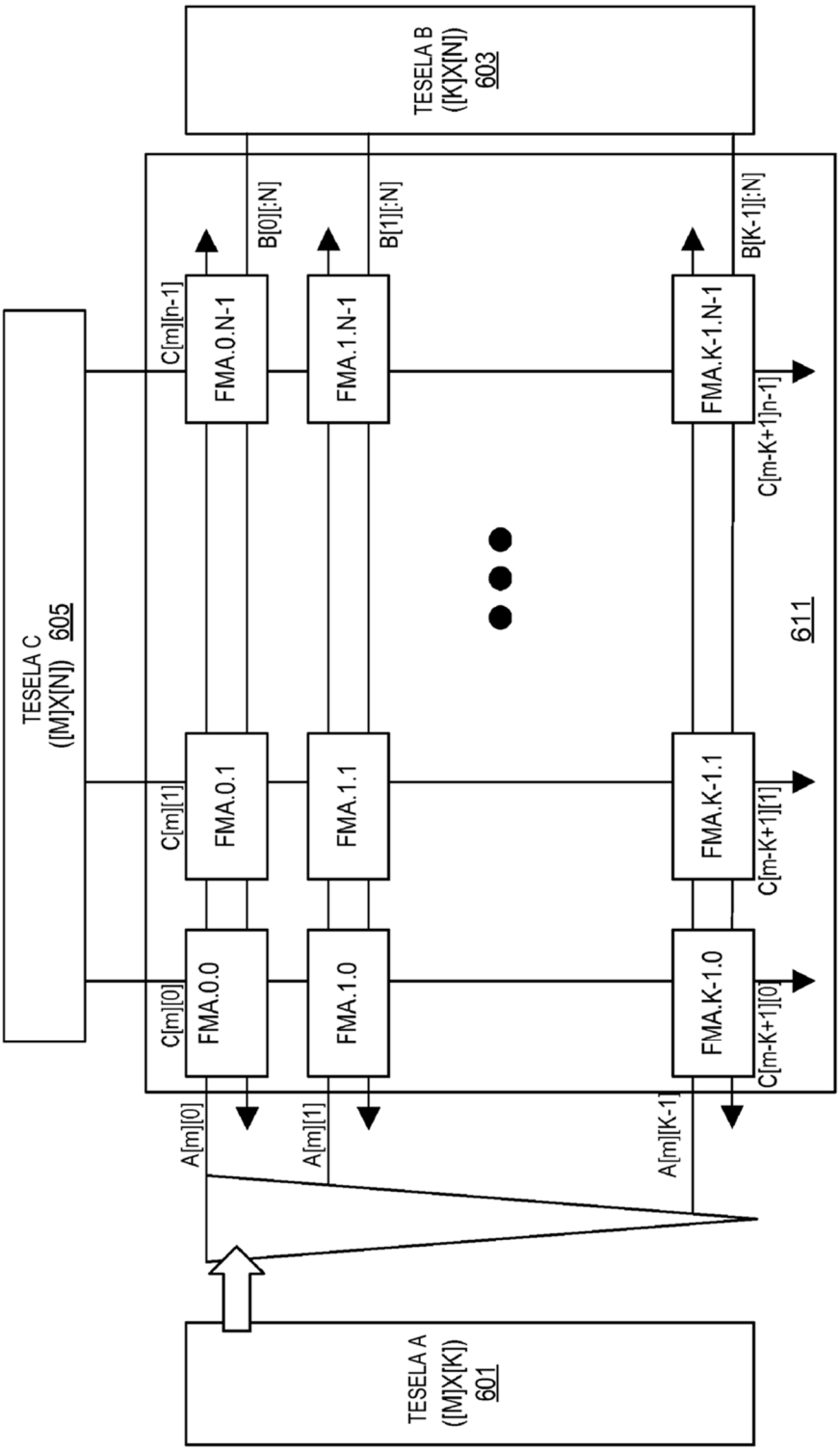

TESELA B
([K]X[N])
603
TESELA C
([M]X[N]) 605
TESELA A
([M]X[K])
601
611
B[0][:N]
B[1][:N]
B[K-1][:N]
C[m][n-1]
C[m][1]
C[m][0]
C[m-K+1][n-1]
C[m-K+1][1]
C[m-K+1][0]
A[m][0]
A[m][1]
A[m][K-1]
FMA.0.N-1
FMA.1.N-1
FMA.K-1.N-1
FMA.0.1
FMA.1.1
FMA.K-1.1
FMA.0.0
FMA.1.0
FMA.K-1.0

FIG. 6

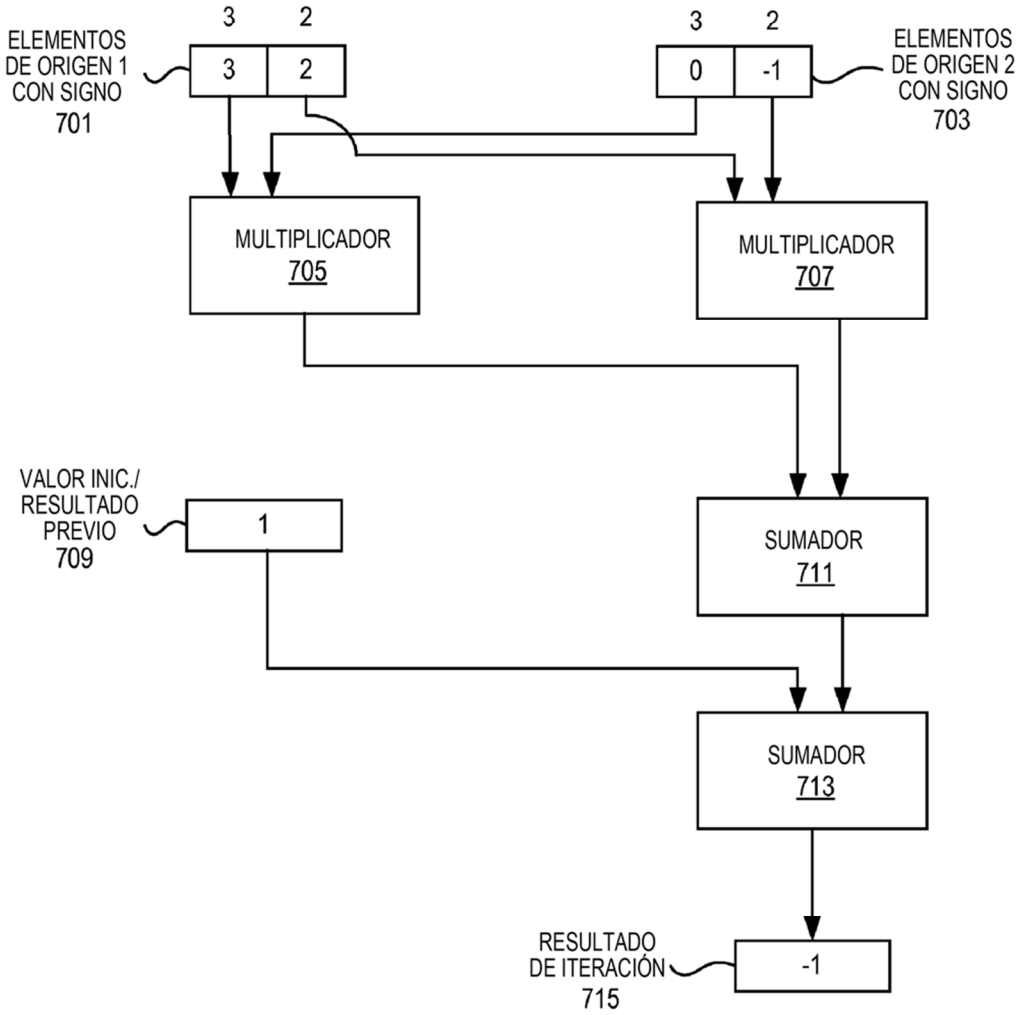

ELEMENTOS DE ORIGEN 1 CON SIGNO 701
3
2
3
2
ELEMENTOS DE ORIGEN 2 CON SIGNO 703
3
2
0
-1
MULTIPLICADOR 705
MULTIPLICADOR 707
VALOR INIC./ RESULTADO PREVIO 709
1
SUMADOR 711
SUMADOR 713
RESULTADO DE ITERACIÓN 715
-1

FIG. 7

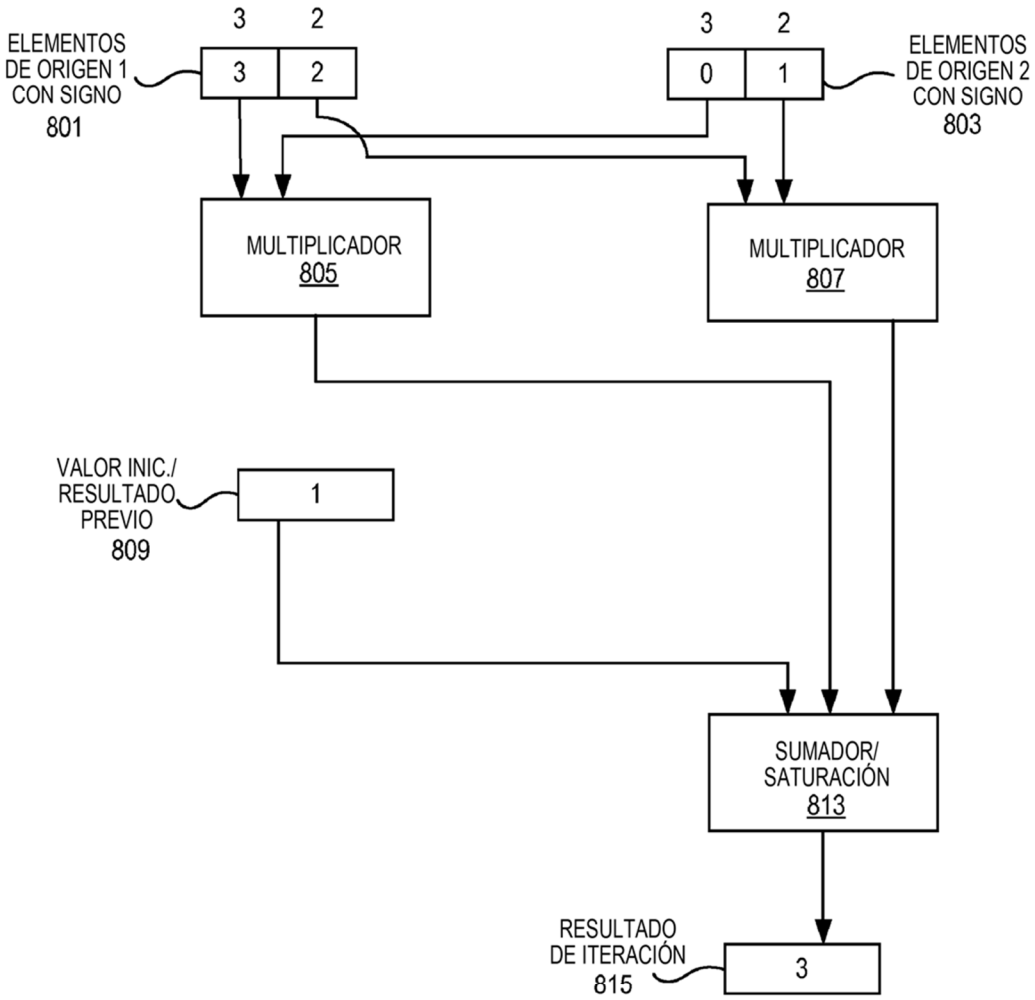

3
2
ELEMENTOS DE ORIGEN 1 CON SIGNO
801
3
2
3
2
0
1
ELEMENTOS DE ORIGEN 2 CON SIGNO
803
MULTIPLICADOR
805
MULTIPLICADOR
807
VALOR INIC./ RESULTADO PREVIO
809
1
SUMADOR/ SATURACIÓN
813
RESULTADO DE ITERACIÓN
815
3

FIG. 8

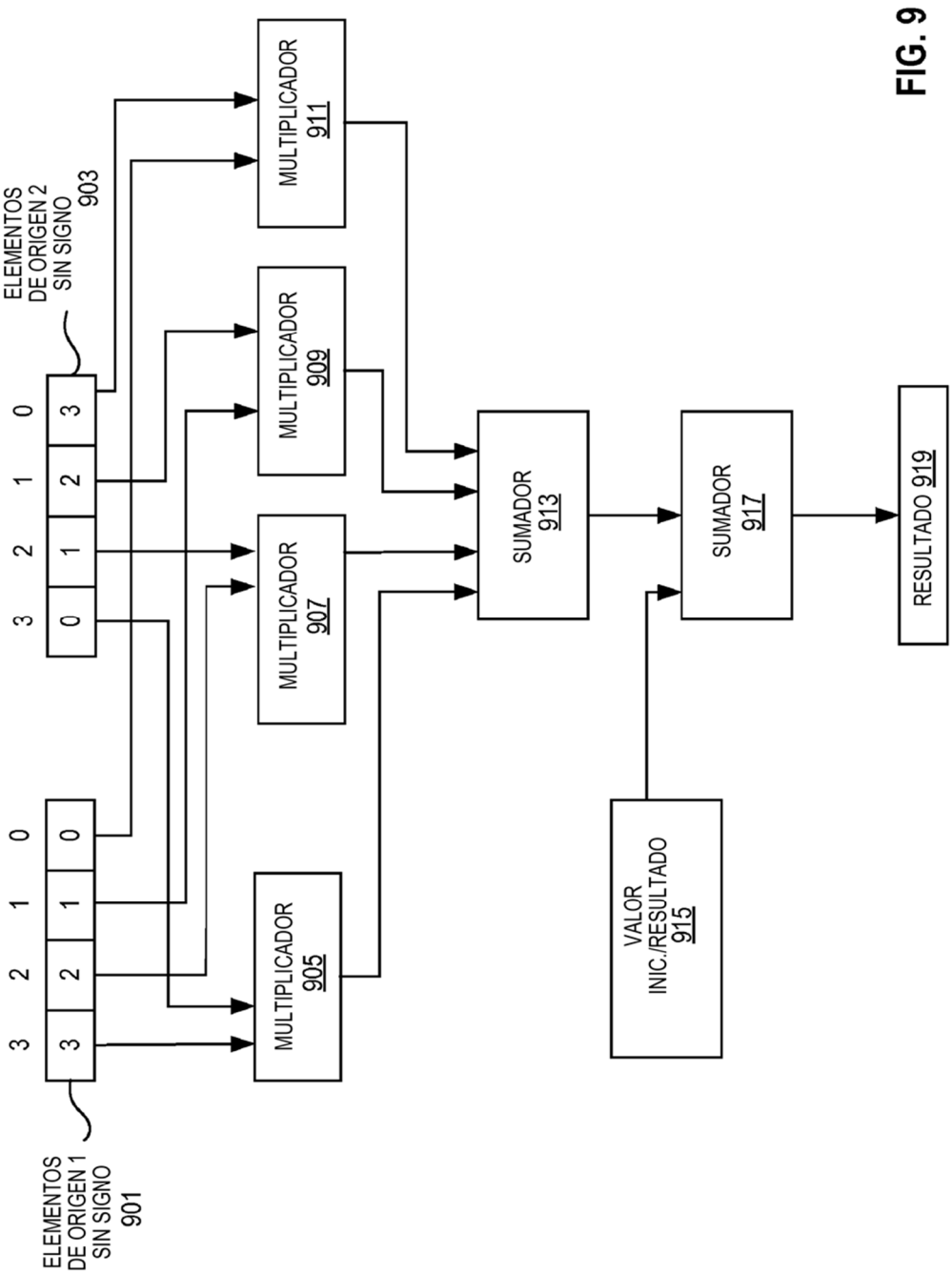

ELEMENTOS DE ORIGEN 1 SIN SIGNO 901
3 2 1 0
3 2 1 0
MULTIPLICADOR 905
MULTIPLICADOR 907
MULTIPLICADOR 909
MULTIPLICADOR 911
ELEMENTOS DE ORIGEN 2 SIN SIGNO 903
3 2 1 0
3 2 1 0
SUMADOR 913
VALOR INIC./RESULTADO 915
SUMADOR 917
RESULTADO 919

FIG. 9

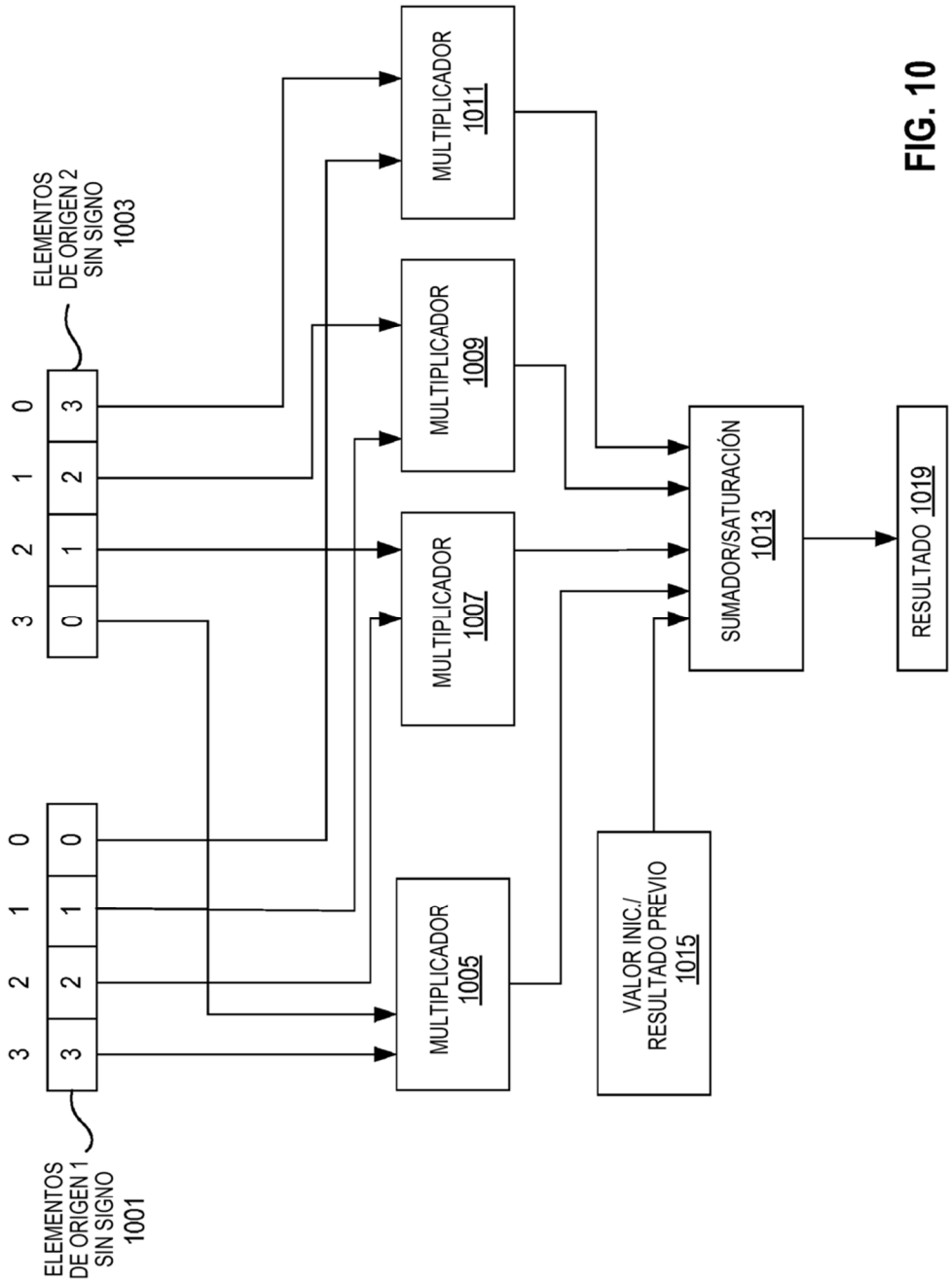

ELEMENTOS DE ORIGEN 1 SIN SIGNO 1001
3 2 1 0
ELEMENTOS DE ORIGEN 2 SIN SIGNO 1003
3 2 1 0
MULTIPLICADOR 1005
MULTIPLICADOR 1007
MULTIPLICADOR 1009
MULTIPLICADOR 1011
VALOR INIC./ RESULTADO PREVIO 1015
SUMADOR/SATURACIÓN 1013
RESULTADO 1019

FIG. 10

ACUMULADOR CON 2X LOS TAMAÑOS DE ENTRADA 1101

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

ACUMULADOR CON 4X LOS TAMAÑOS DE ENTRADA 1103

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

ACUMULADOR CON 8X LOS TAMAÑOS DE ENTRADA 1105

ORÍGENES	BITS	ACUMULADOR	BITS
BYTE	8	INT64/DPFP	64

FIG. 11

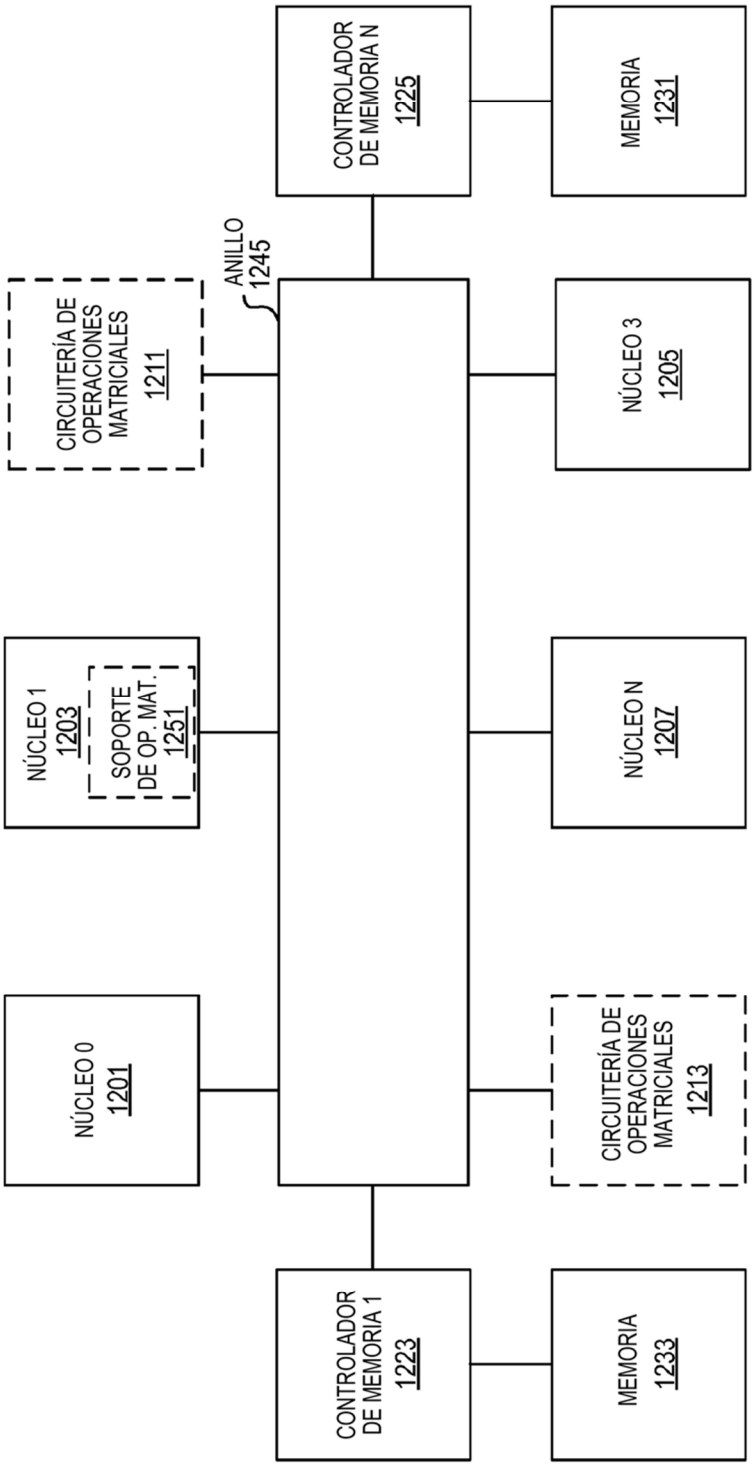

CONTROLADOR DE MEMORIA N 1225
MEMORIA 1231
ANILLO 1245
CIRCUITERÍA DE OPERACIONES MATRICIALES 1211
NÚCLEO 3 1205
NÚCLEO 1 1203
SOPORTE DE OP. MAT. 1251
NÚCLEO N 1207
NÚCLEO 0 1201
CIRCUITERÍA DE OPERACIONES MATRICIALES 1213
CONTROLADOR DE MEMORIA 1 1223
MEMORIA 1233

FIG. 12

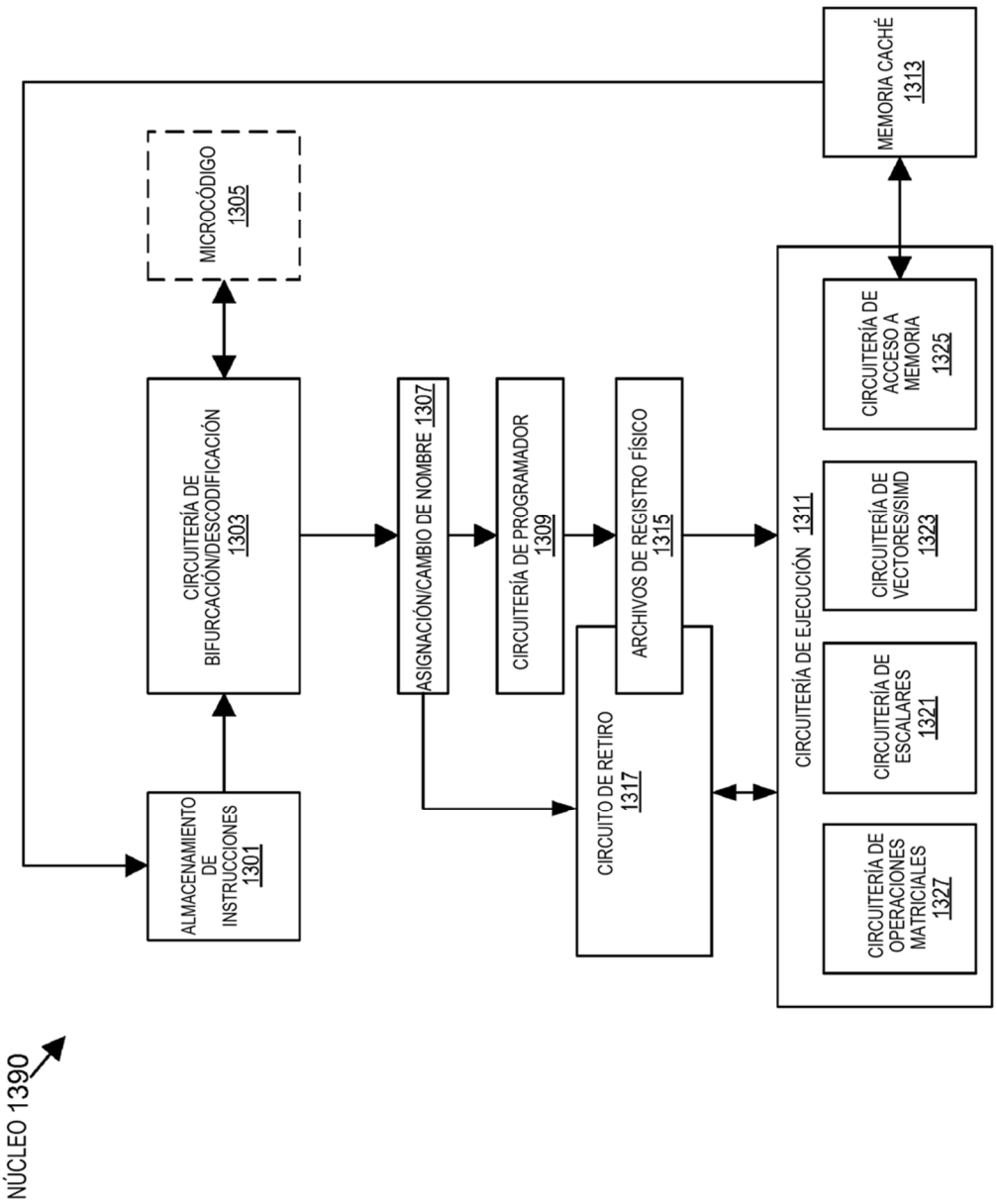

NÚCLEO 1390
ALMACENAMIENTO DE INSTRUCCIONES 1301
CIRCUITERÍA DE BIFURCACIÓN/DESCODIFICACIÓN 1303
MICROCÓDIGO 1305
ASIGNACIÓN/CAMBIO DE NOMBRE 1307
CIRCUITERÍA DE PROGRAMADOR 1309
ARCHIVOS DE REGISTRO FÍSICO 1315
CIRCUITO DE RETIRO 1317
CIRCUITERÍA DE EJECUCIÓN 1311
CIRCUITERÍA DE OPERACIONES MATRICIALES 1327
CIRCUITERÍA DE ESCALARES 1321
CIRCUITERÍA DE VECTORES/SIMD 1323
CIRCUITERÍA DE ACCESO A MEMORIA 1325
MEMORIA CACHÉ 1313

FIG. 13

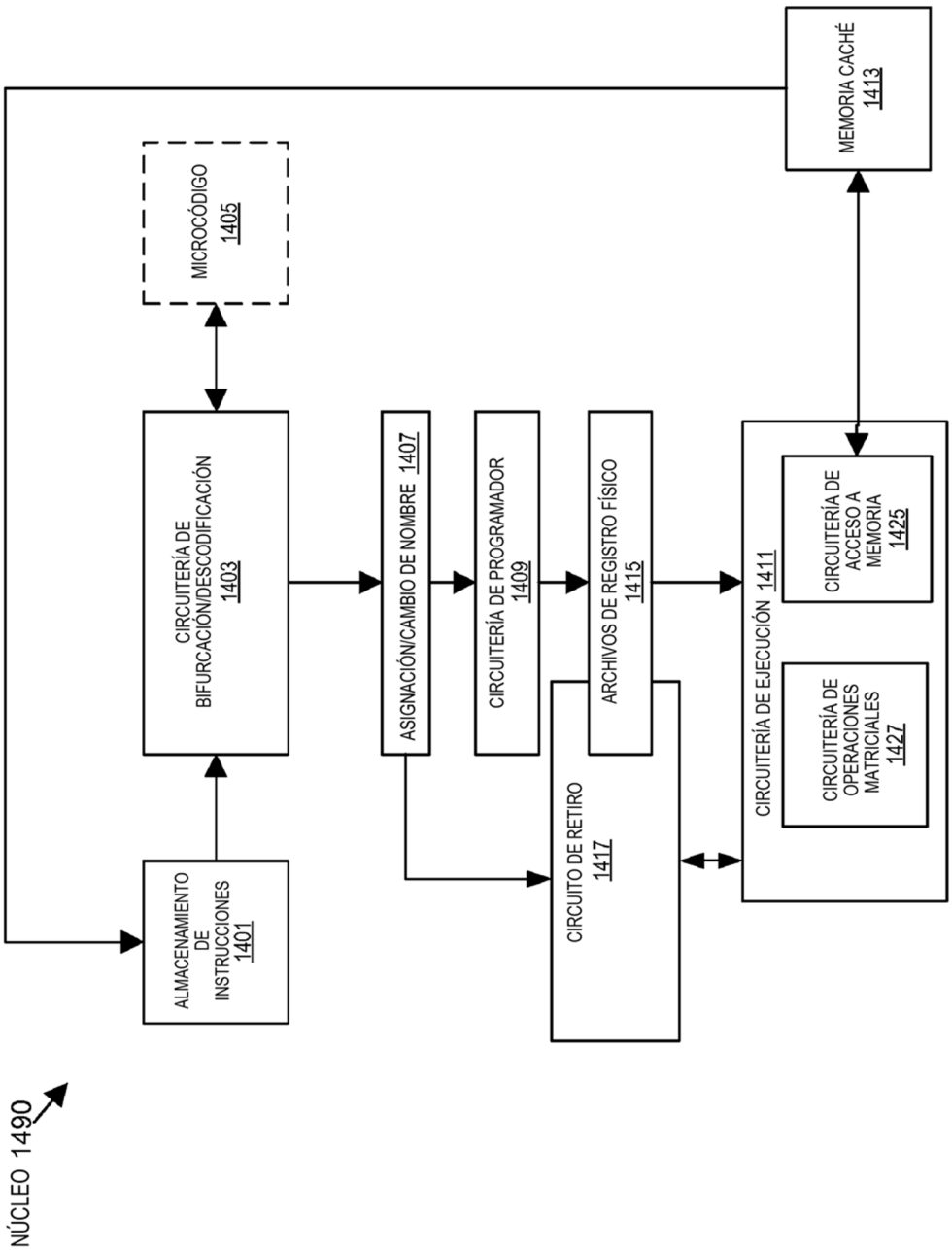

NÚCLEO 1490
ALMACENAMIENTO DE INSTRUCCIONES 1401
CIRCUITERÍA DE BIFURCACIÓN/DESCODIFICACIÓN 1403
MICROCÓDIGO 1405
ASIGNACIÓN/CAMBIO DE NOMBRE 1407
CIRCUITERÍA DE PROGRAMADOR 1409
ARCHIVOS DE REGISTRO FÍSICO 1415
CIRCUITO DE RETIRO 1417
CIRCUITERÍA DE EJECUCIÓN 1411
CIRCUITERÍA DE ACCESO A MEMORIA 1425
CIRCUITERÍA DE OPERACIONES MATRICIALES 1427
MEMORIA CACHÉ 1413

FIG. 14

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

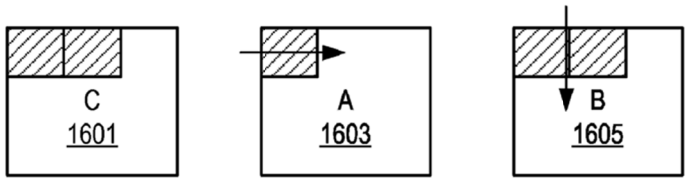
DIREC.	VALOR
0	A_{11}
1	A_{12}
2	A_{13}
3	A_{21}
4	A_{22}
5	A_{23}

ORDENADO POR FILAS

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

ORDENADO POR COLUMNAS

FIG. 15




```
TILECONFIG [RAX]
// SUPONER QUE ALGUNOS BUCLES EXTERNOS ACCIONAN LA DIVISIÓN EN TESELAS DE MEMORIA CACHÉ (NO MOSTRADO)
{
TILELOAD TMM0, RSI+RDI // SCRDST, RSI APUNTA A C, RDI TIENE
TILELOAD TMM1, RSI+RDI+N // LA SEGUNDA TESELA DE C, EXPANDIR EN DIMENSIÓN DE SIMD N
MOV KK, 0
LOOP:
TILELOAD TMM2, R8+R9 // SCR2 ES UNA CARGA CON ZANCADA DE A, REUTILIZADO PARA 2 INSTR. DE TMMA
TILELOAD TMM3, R10+R11 // SCR1 ES UNA CARGA CON ZANCADA DE B
TMMAPS TMM0, TMM2, TMM3 // ACTUALIZAR TESELA IZQUIERDA DE C
TILELOAD TMM3, R10+R11+N // SCR1 CARGADO CON B A PARTIR DE TESELA MÁS A LA DERECHA SIGUIENTE
TMMAPS TMM1, TMM2, TMM3 // ACTUALIZAR TESELA DERECHA DE C
ADD R8, K // ACTUALIZAR PUNTEROS MEDIANTE 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 INIC.
```

FIG. 16

ES 2 905 697 T3

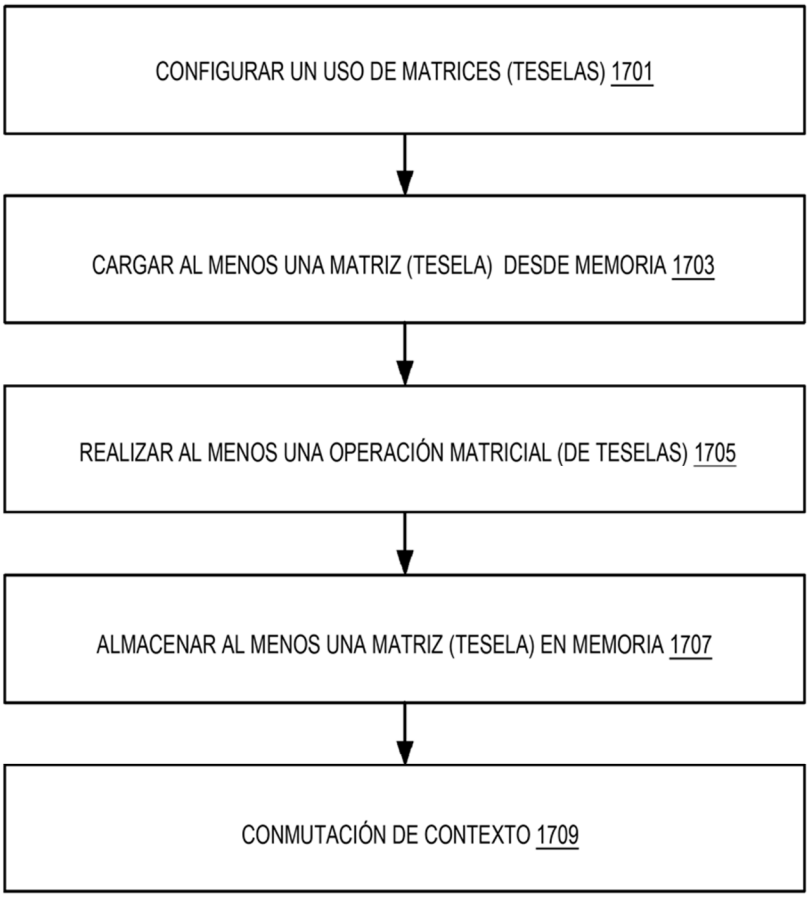

CONFIGURAR UN USO DE MATRICES (TESELAS) 1701
CARGAR AL MENOS UNA MATRIZ (TESELA) DESDE MEMORIA 1703
REALIZAR AL MENOS UNA OPERACIÓN MATRICIAL (DE TESELAS) 1705
ALMACENAR AL MENOS UNA MATRIZ (TESELA) EN MEMORIA 1707
CONMUTACIÓN DE CONTEXTO 1709

FIG. 17

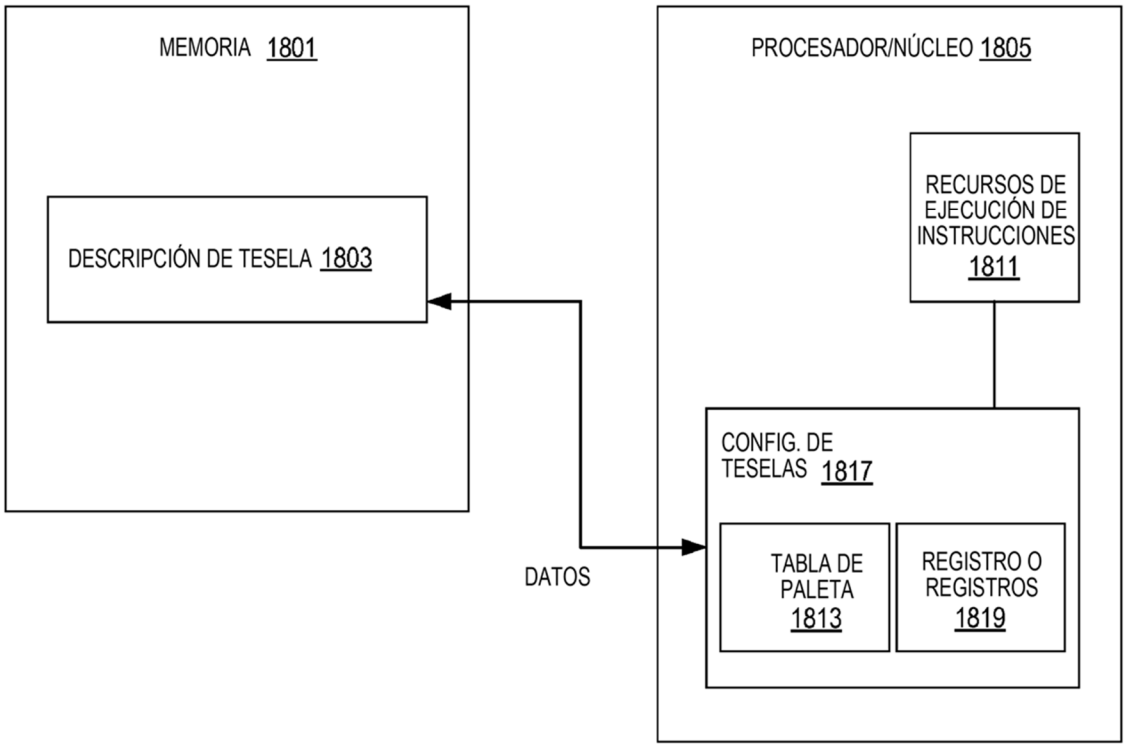

MEMORIA 1801
DESCRIPCIÓN DE TESELA 1803
DATOS
PROCESADOR/NÚCLEO 1805
RECURSOS DE EJECUCIÓN DE INSTRUCCIONES 1811
CONFIG. DE TESELAS 1817
TABLA DE PALETA 1813
REGISTRO O REGISTROS 1819

FIG. 18

ID DE PALETA 1901	MINICIO 1903
PINICIO 1905	INDICADORES DE PAR 1907
0	0
0	0

. . .

0	0
FILAS DE TMM0 1913	COLUMNAS DE TMM0 1915
FILAS DE TMM1	COLUMNAS DE TMM1
▪ ▪ ▪	
FILAS DE TMM5	COLUMNAS DE TMM15
0	

FIG. 19

FIG. 20(A)

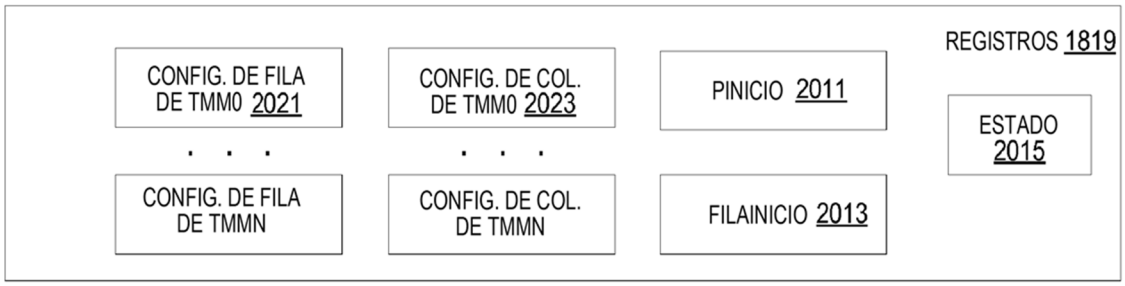


FIG. 20(B)

FIG. 20(C)

FIG. 20(D)

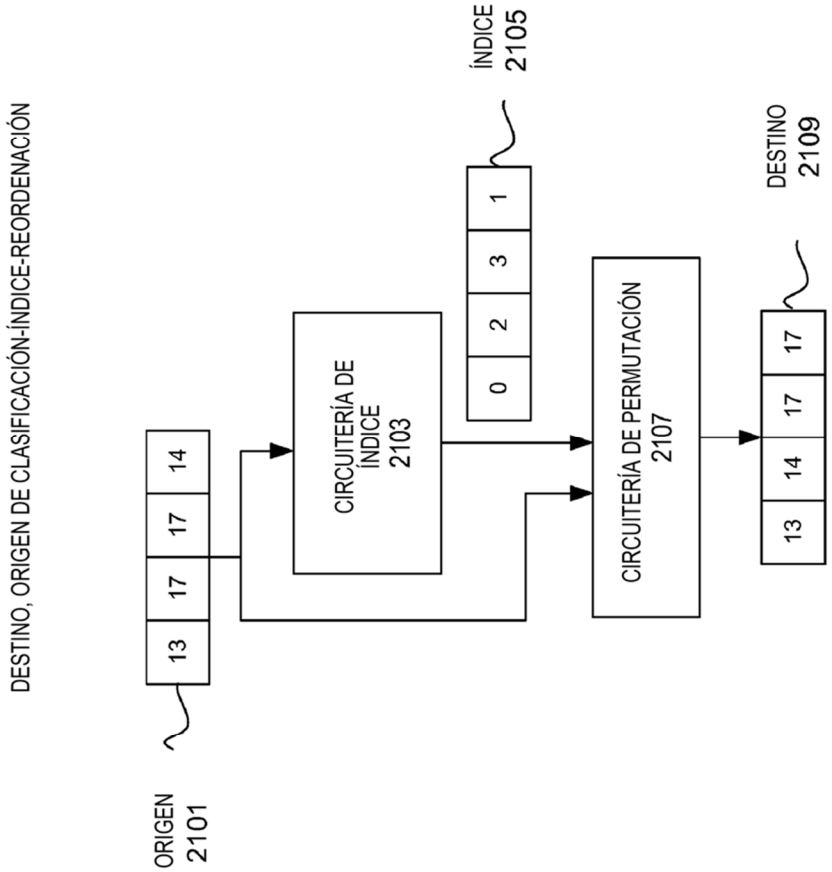

DESTINO, ORIGEN DE CLASIFICACIÓN-ÍNDICE-REORDENACIÓN
ORIGEN
2101
13
17
17
14
CIRCUITERÍA DE ÍNDICE
2103
ÍNDICE
2105
0
2
3
1
CIRCUITERÍA DE PERMUTACIÓN
2107
DESTINO
2109
13
14
17
17

FIG. 21

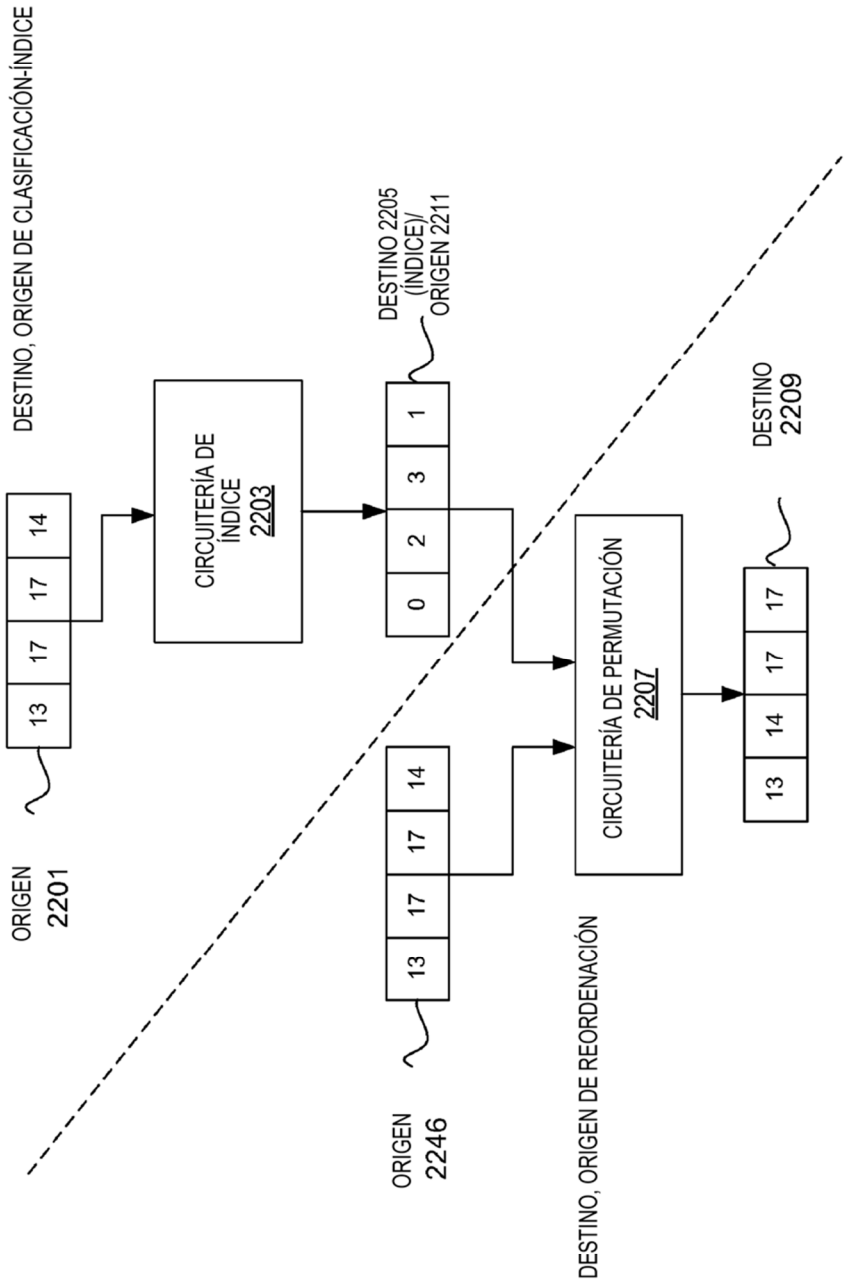

DESTINO, ORIGEN DE CLASIFICACIÓN-ÍNDICE
ORIGEN 2201
13
17
17
14
CIRCUITERÍA DE ÍNDICE 2203
DESTINO 2205 (ÍNDICE)/ ORIGEN 2211
0
2
3
1
ORIGEN 2246
13
17
17
14
DESTINO, ORIGEN DE REORDENACIÓN
CIRCUITERÍA DE PERMUTACIÓN 2207
DESTINO 2209
13
14
17
17

FIG. 22

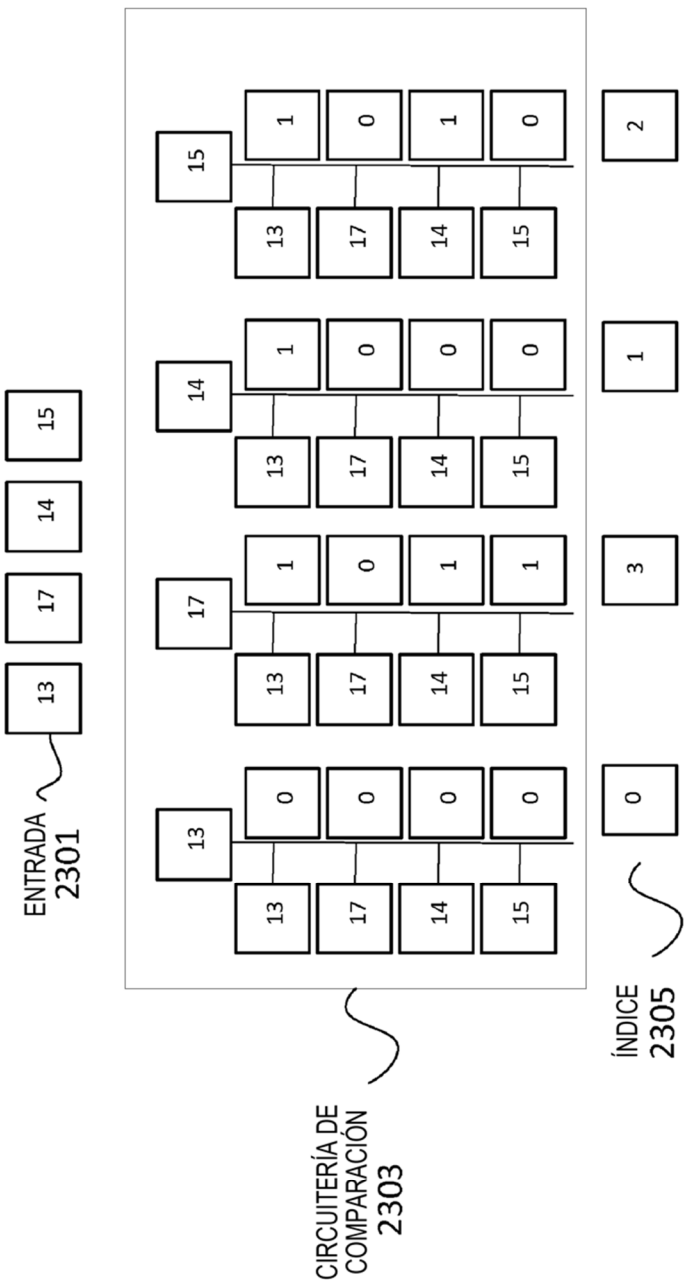

ENTRADA
2301
13
17
14
15
CIRCUITERÍA DE
COMPARACIÓN
2303
ÍNDICE
2305
0
1
2
3
13
17
14
15
0
0
0
0
1
1
1
0
1
0
0
0
1
1
0
1
1
0
1
0

FIG. 23

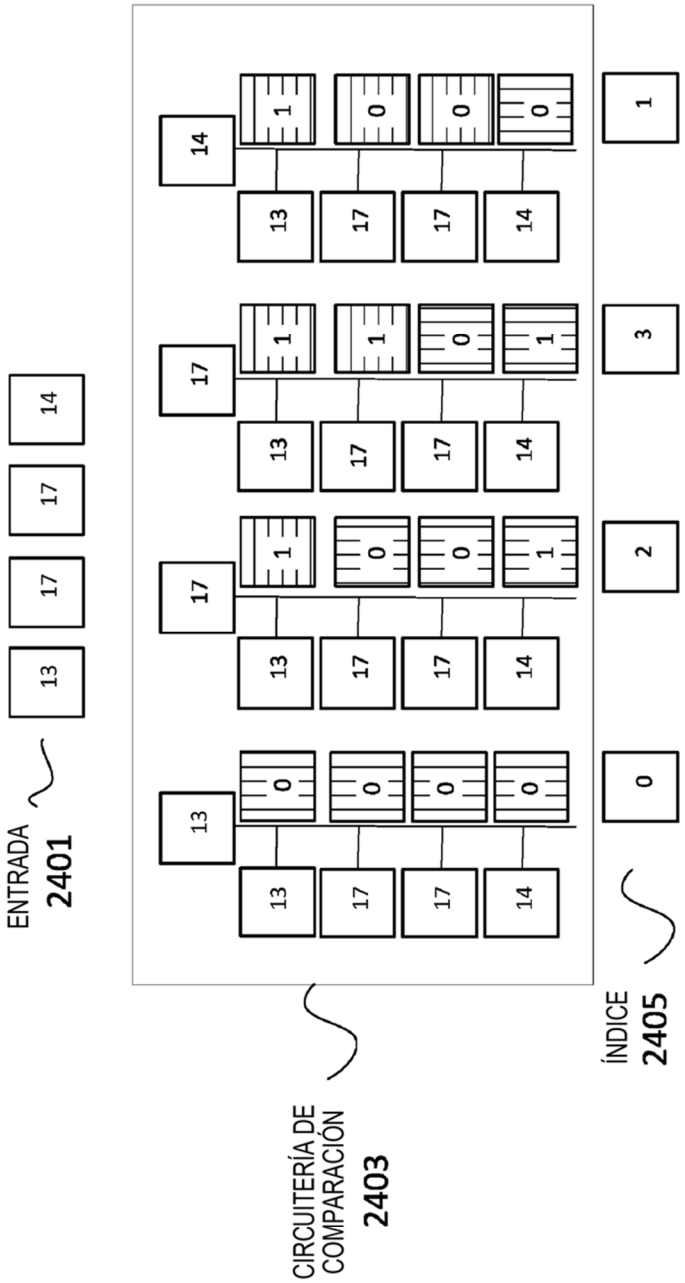
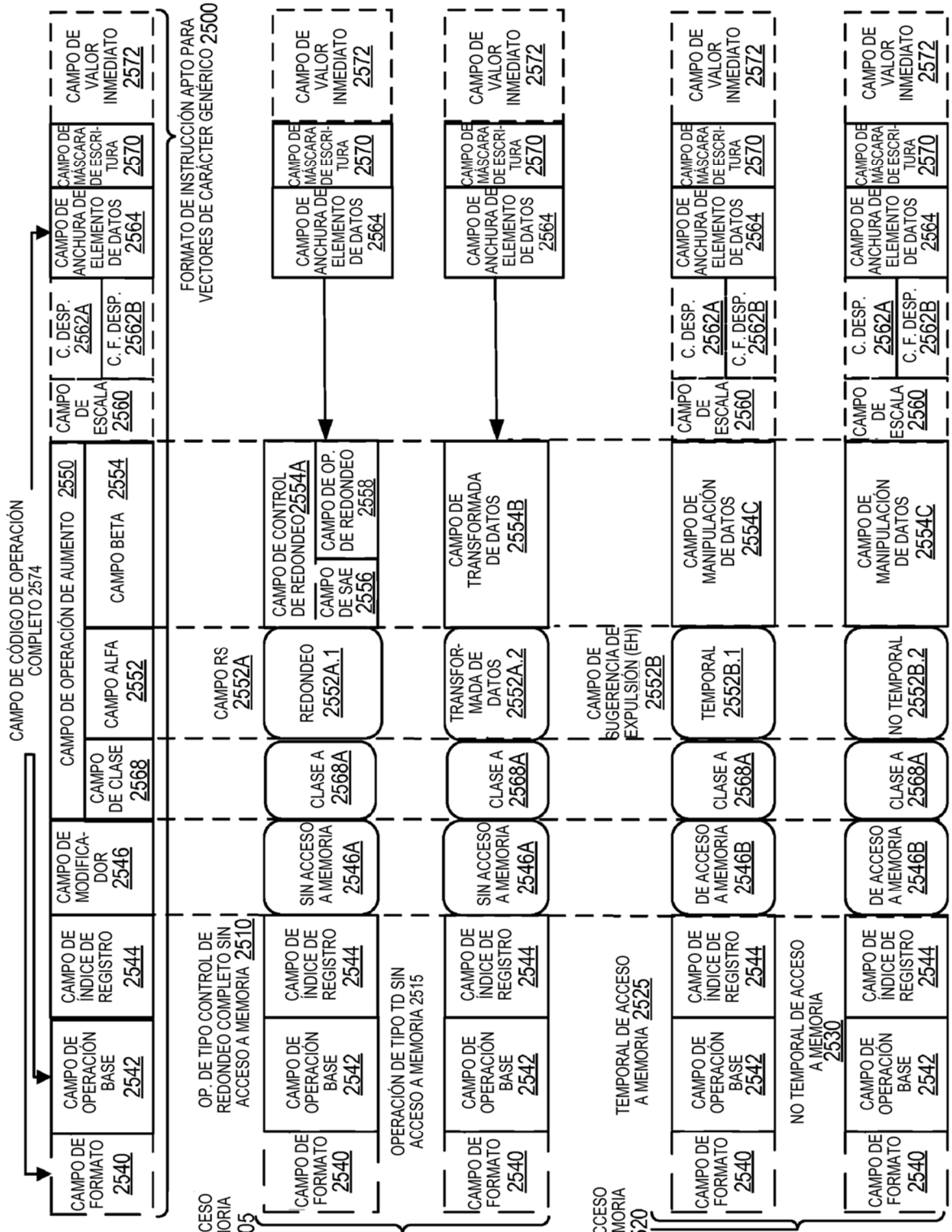

ENTRADA
2401
13
17
17
14
CIRCUITERÍA DE COMPARACIÓN
2403
ÍNDICE
2405
0
2
17
1
3
14
1

FIG. 24

FIG. 25A


CAMPO DE CÓDIGO DE OPERACIÓN COMPLETO 2574
FORMATO DE INSTRUCCIÓN APTO PARA VECTORES DE CARÁCTER GENÉRICO 2500
SIN ACCESO A MEMORIA 2505
DE ACCESO A MEMORIA 2520
OP. DE TIPO CONTROL DE REDONDEO COMPLETO SIN ACCESO A MEMORIA 2510
OPERACIÓN DE TIPO TD SIN ACCESO A MEMORIA 2515
TEMPORAL DE ACCESO A MEMORIA 2525
NO TEMPORAL DE ACCESO A MEMORIA 2530
CAMPO DE FORMATO 2540
CAMPO DE OPERACIÓN BASE 2542
CAMPO DE ÍNDICE DE REGISTRO 2544
CAMPO DE MODIFICADOR 2546
SIN ACCESO A MEMORIA 2546A
DE ACCESO A MEMORIA 2546B
CAMPO DE OPERACIÓN DE AUMENTO 2550
CAMPO DE CLASE 2568
CAMPO ALFA 2552
CAMPO BETA 2554
CAMPO RS 2552A
REDONDEO 2552A.1
TRANSFORMADA DE DATOS 2552A.2
CLASE A 2568A
CAMPO DE CONTROL DE REDONDEO 2554A
CAMPO DE OP. DE REDONDEO 2558
CAMPO DE SAE 2556
CAMPO DE TRANSFORMADA DE DATOS 2554B
CAMPO DE SUGERENCIA DE EXPULSIÓN (EH) 2552B
TEMPORAL 2552B.1
NO TEMPORAL 2552B.2
CAMPO DE MANIPULACIÓN DE DATOS 2554C
CAMPO DE ESCALA 2560
C. DESP. 2562A
C. F. DESP. 2562B
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 2564
CAMPO DE MÁSCARA DE ESCRITURA 2570
CAMPO DE VALOR INMEDIATO 2572

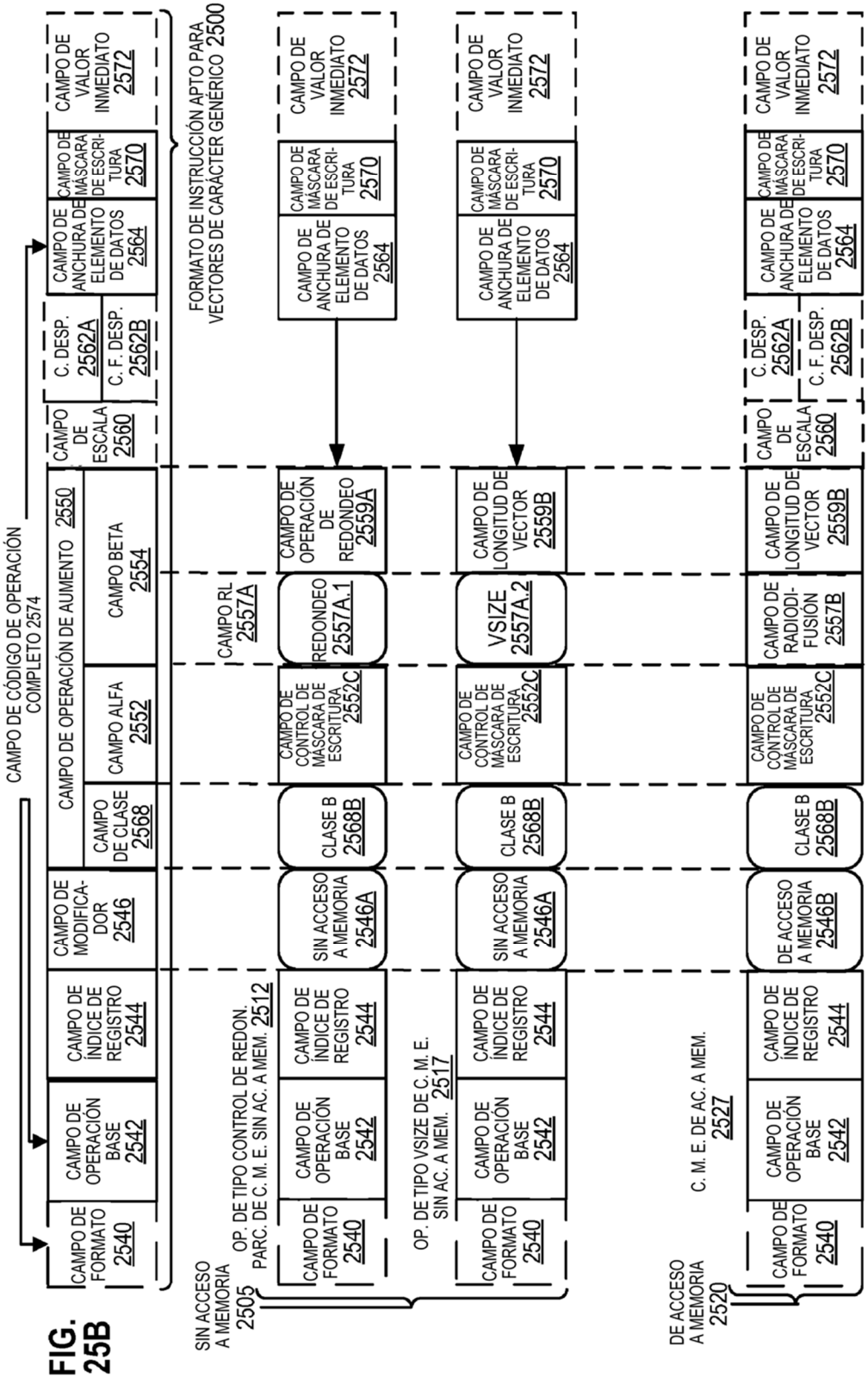


FIG. 25B
FORMATO DE INSTRUCCIÓN APTO PARA VECTORES DE CARÁCTER GENÉRICO 2500
CAMPO DE CÓDIGO DE OPERACIÓN COMPLETO 2574
SIN ACCESO A MEMORIA 2505
DE ACCESO A MEMORIA 2520
OP. DE TIPO CONTROL DE REDON. PARC. DE C. M. E. SIN AC. A MEM. 2512
OP. DE TIPO VSIZE DE C. M. E. SIN AC. A MEM. 2517
C. M. E. DE AC. A MEM. 2527
CAMPO DE FORMATO 2540
CAMPO DE OPERACIÓN BASE 2542
CAMPO DE ÍNDICE DE REGISTRO 2544
CAMPO DE MODIFICADOR 2546
SIN ACCESO A MEMORIA 2546A
DE ACCESO A MEMORIA 2546B
CAMPO DE OPERACIÓN DE AUMENTO 2550
CAMPO DE CLASE 2568
CLASE B 2568B
CAMPO ALFA 2552
CAMPO BETA 2554
CAMPO DE CONTROL DE MÁSCARA DE ESCRITURA 2552C
CAMPO RL 2557A
REDONDEO 2557A.1
VSIZE 2557A.2
CAMPO DE OPERACIÓN DE REDONDEO 2559A
CAMPO DE LONGITUD DE VECTOR 2559B
CAMPO DE RADIODIFUSIÓN 2557B
CAMPO DE ESCALA 2560
C. DESP. 2562A
C. F. DESP. 2562B
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 2564
CAMPO DE MÁSCARA DE ESCRITURA 2570
CAMPO DE VALOR INMEDIATO 2572

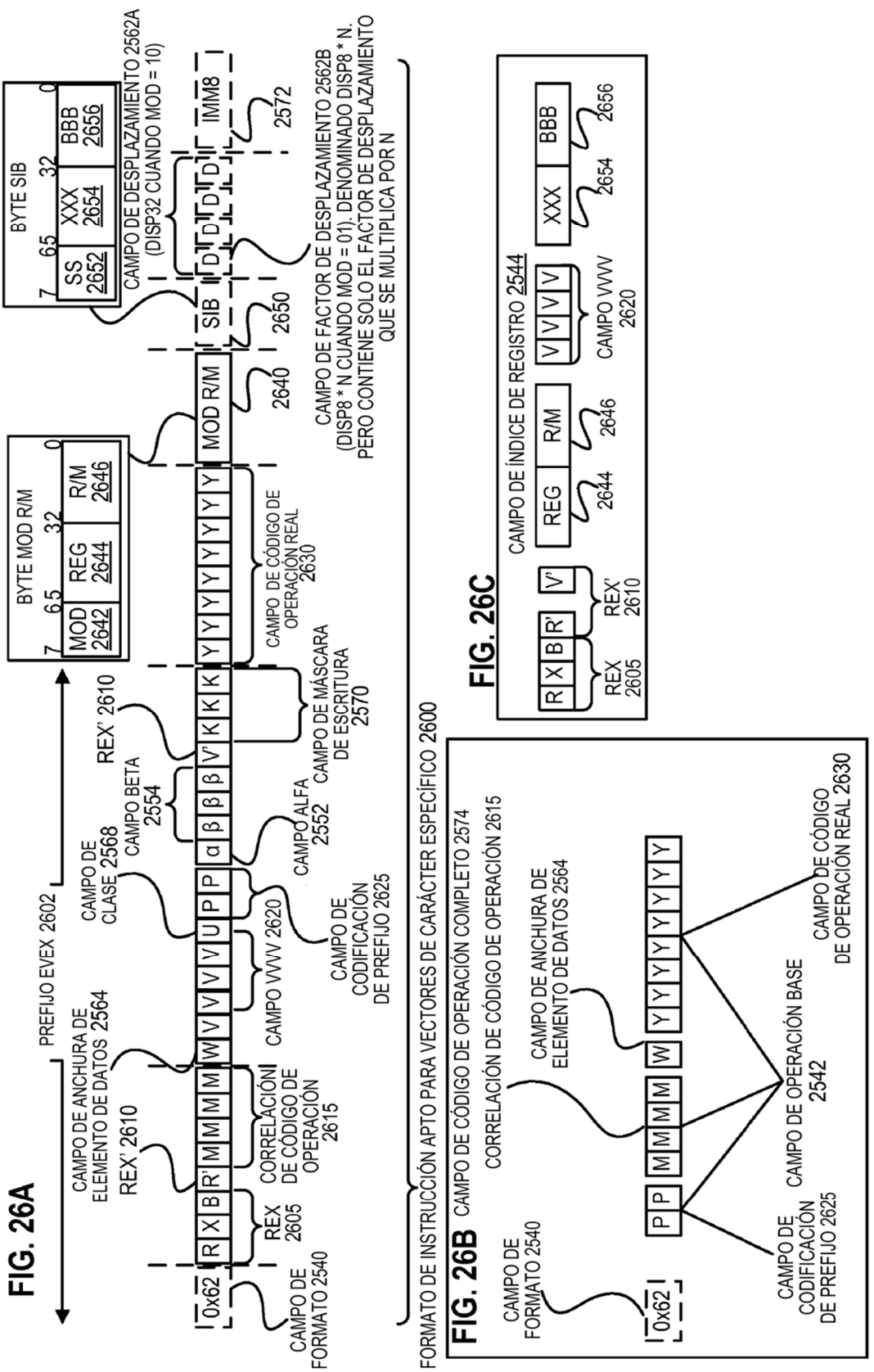
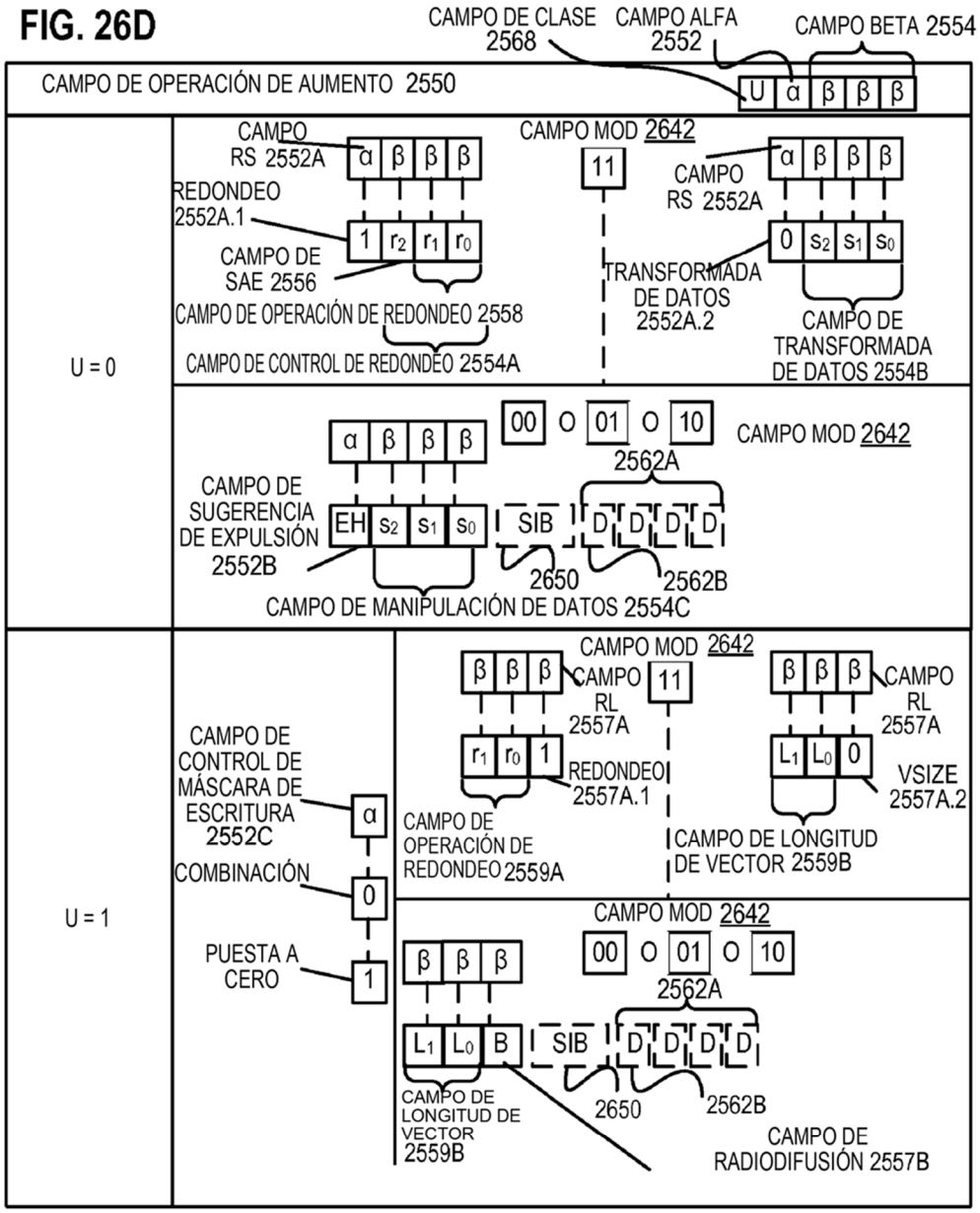


FIG. 26A
PREFIJO EVEX 2602
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 2564
REX' 2610
CAMPO DE CLASE 2568
CAMPO BETA 2554
REX' 2610
CAMPO ALFA 2552
CAMPO DE MÁSCARA DE ESCRITURA 2570
CAMPO DE CÓDIGO DE OPERACIÓN REAL 2630
CAMPO VVVV 2620
CAMPO DE CODIFICACIÓN DE PREFIJO 2625
CORRELACIÓN DE CÓDIGO DE OPERACIÓN 2615
REX 2605
CAMPO DE FORMATO 2540
0x62
R X B R' M M M W V V V V U P P α β β β V' K K K Y Y Y Y Y Y Y Y
BYTE MOD R/M
MOD 2642
REG 2644
R/M 2646
MOD R/M 2640
BYTE SIB
SS 2652
XXX 2654
BBB 2656
SIB 2650
CAMPO DE DESPLAZAMIENTO 2562A (DISP32 CUANDO MOD = 10)
D D D D
IMM8 2572
CAMPO DE FACTOR DE DESPLAZAMIENTO 2562B (DISP8 * N CUANDO MOD = 01). DENOMINADO DISP8 * N. PERO CONTIENE SOLO EL FACTOR DE DESPLAZAMIENTO QUE SE MULTIPLICA POR N
FORMATO DE INSTRUCCIÓN APTO PARA VECTORES DE CARÁCTER ESPECÍFICO 2600
FIG. 26B
CAMPO DE CÓDIGO DE OPERACIÓN COMPLETO 2574
CORRELACIÓN DE CÓDIGO DE OPERACIÓN 2615
CAMPO DE FORMATO 2540
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 2564
0x62
P P M M M W Y Y Y Y Y Y Y Y
CAMPO DE CODIFICACIÓN DE PREFIJO 2625
CAMPO DE OPERACIÓN BASE 2542
CAMPO DE CÓDIGO DE OPERACIÓN REAL 2630
FIG. 26C
CAMPO DE ÍNDICE DE REGISTRO 2544
R X B R'
REX 2605
V'
REX' 2610
REG
2644
R/M
2646
V V V V
CAMPO VVVV 2620
XXX
2654
BBB
2656

FIG. 26D


CAMPO DE CLASE 2568
CAMPO ALFA 2552
CAMPO BETA 2554
CAMPO DE OPERACIÓN DE AUMENTO 2550
U α β β β
U = 0
CAMPO 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 MOD 2642
11
CAMPO RS 2552A
α β β β
0 s2 s1 s0
TRANSFORMADA DE DATOS 2552A.2
CAMPO DE TRANSFORMADA DE DATOS 2554B
α β β β
CAMPO DE SUGERENCIA DE EXPULSIÓN 2552B
EH s2 s1 s0
00 O 01 O 10
2562A
SIB D D D D
2650
2562B
CAMPO MOD 2642
CAMPO DE MANIPULACIÓN DE DATOS 2554C
U = 1
CAMPO DE CONTROL DE MÁSCARA DE ESCRITURA 2552C
α
COMBINACIÓN
0
PUESTA A CERO
1
CAMPO MOD 2642
β β β CAMPO RL 2557A
11
r1 r0 1
REDONDEO 2557A.1
CAMPO DE OPERACIÓN DE REDONDEO 2559A
β β β CAMPO RL 2557A
L1 L0 0
VSIZE 2557A.2
CAMPO DE LONGITUD DE VECTOR 2559B
CAMPO MOD 2642
β β β
00 O 01 O 10
2562A
L1 L0 B
SIB D D D D
CAMPO DE LONGITUD DE VECTOR 2559B
2650
2562B
CAMPO DE RADIODIFUSIÓN 2557B

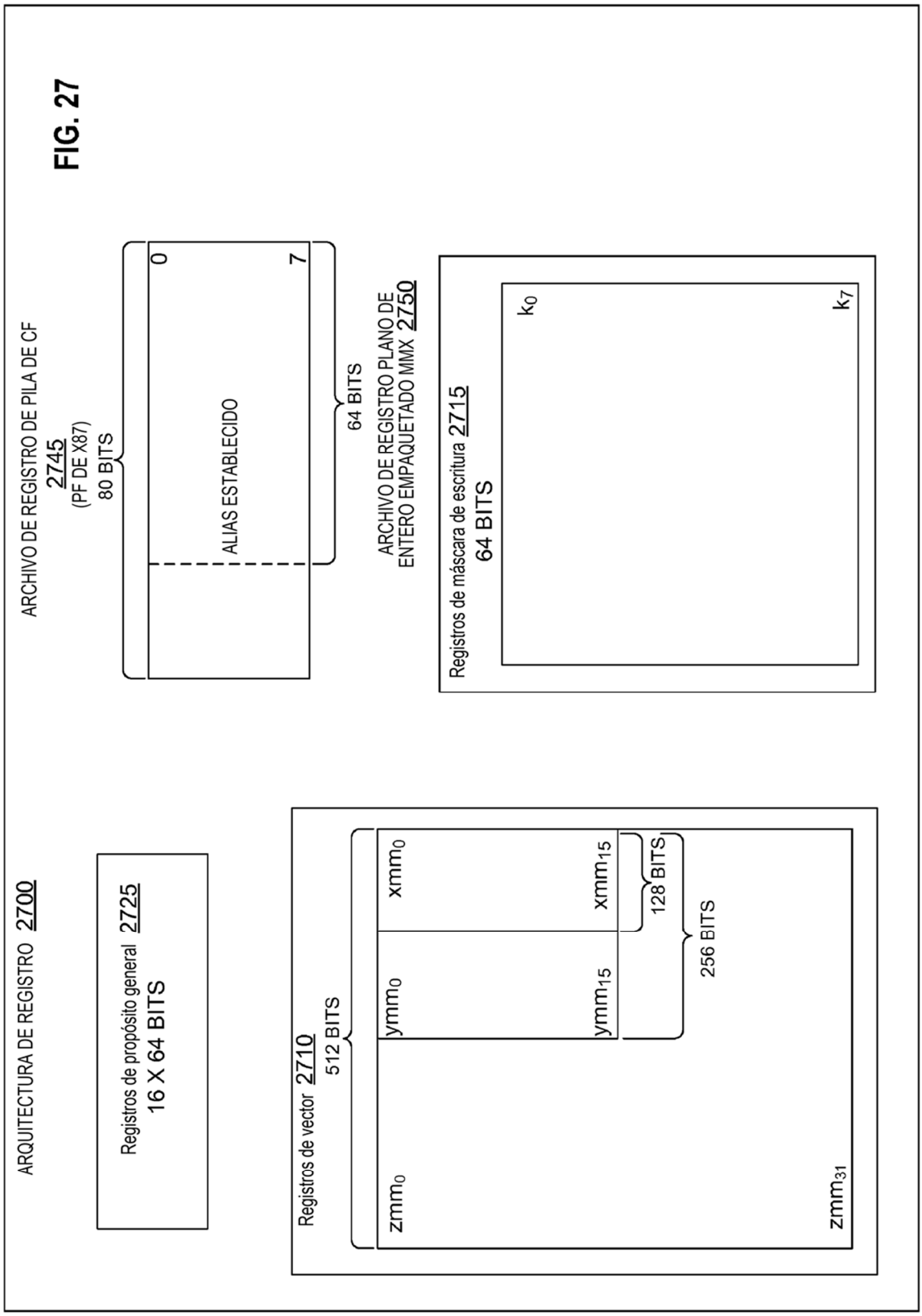

ARQUITECTURA DE REGISTRO 2700
Registros de propósito general 2725
16 X 64 BITS
Registros de vector 2710
512 BITS
zmm0
ymm0
xmm0
ymm15
xmm15
128 BITS
256 BITS
zmm31
ARCHIVO DE REGISTRO DE PILA DE CF
2745
(PF DE X87)
80 BITS
0
ALIAS ESTABLECIDO
7
64 BITS
ARCHIVO DE REGISTRO PLANO DE ENTERO EMPAQUETADO MMX 2750
Registros de máscara de escritura 2715
64 BITS
k0
k7

FIG. 27

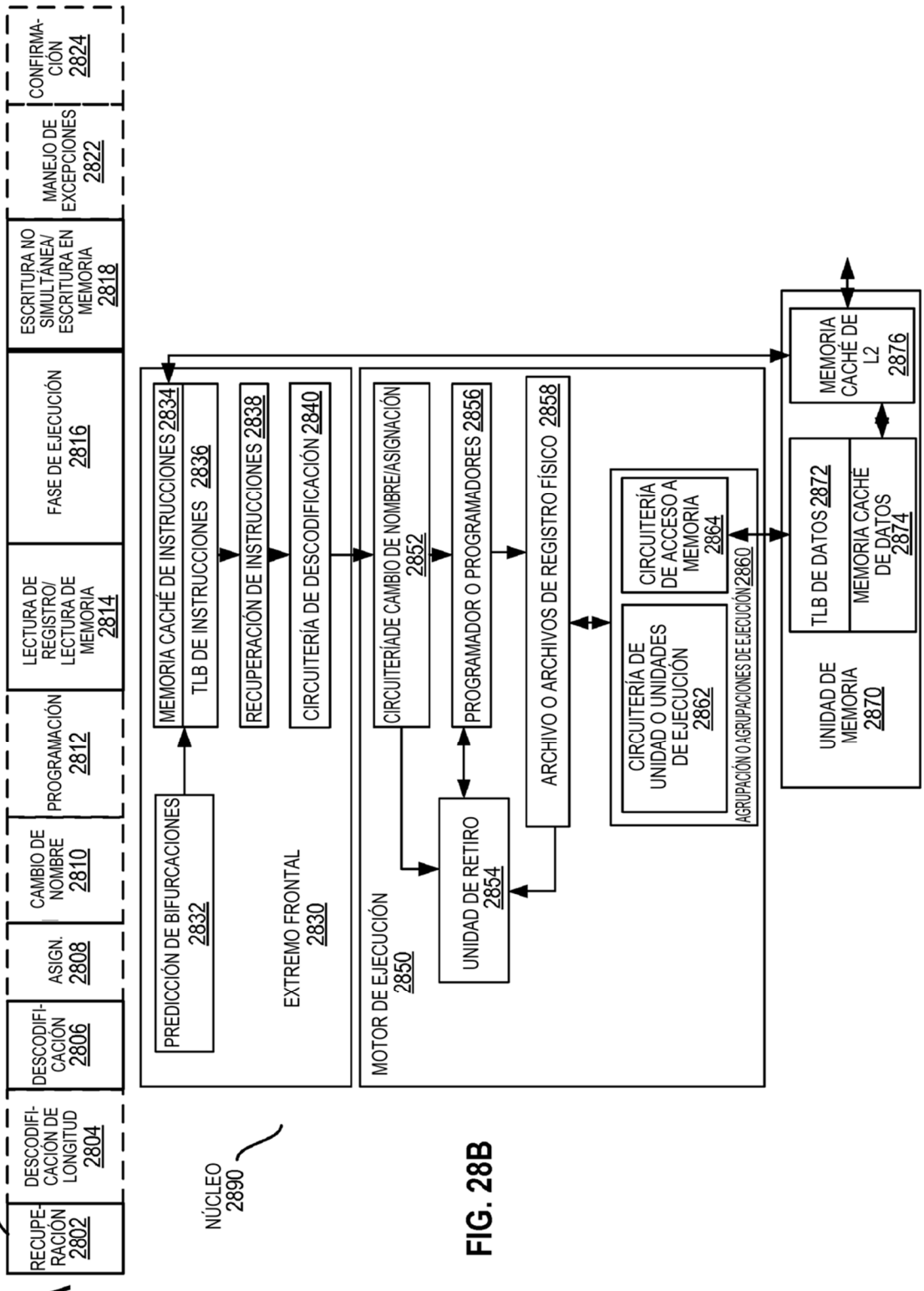


FIG. 28A

FIG. 28B

FIG. 29B

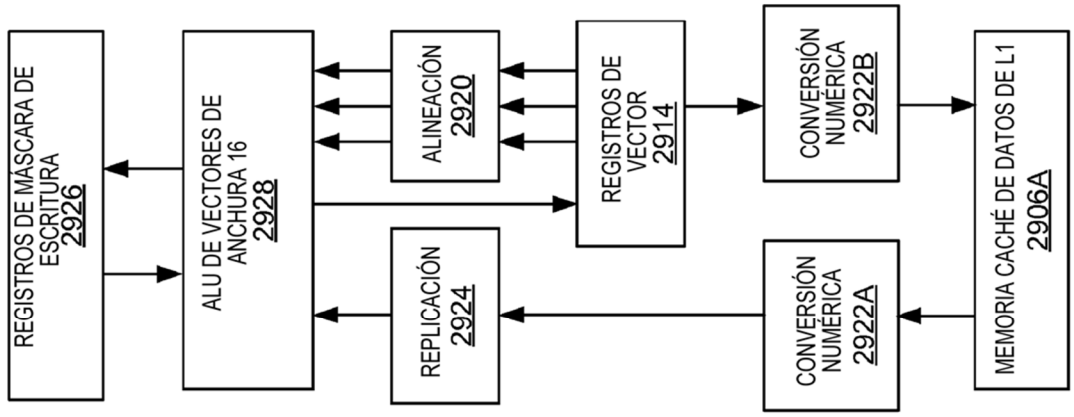
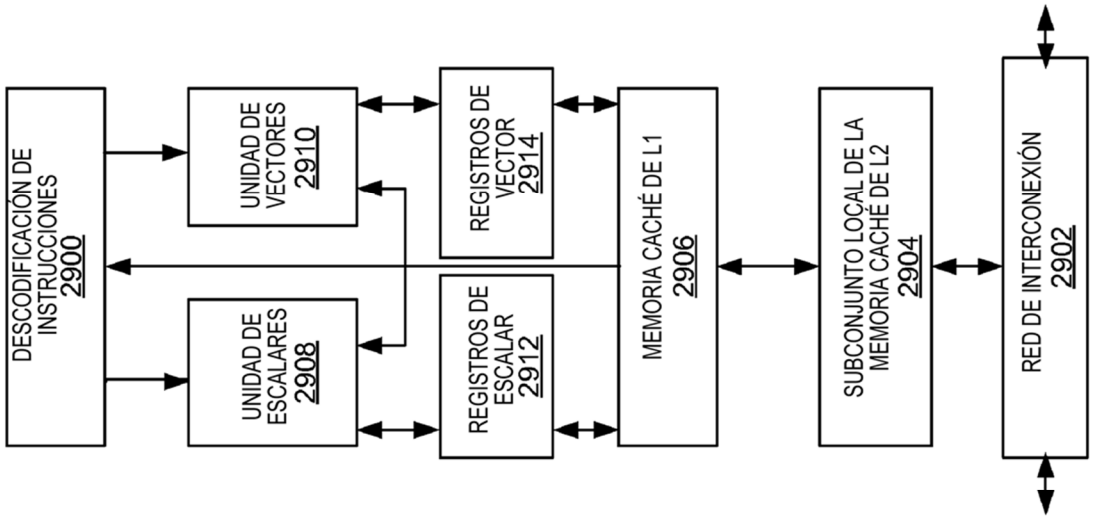


FIG. 29A



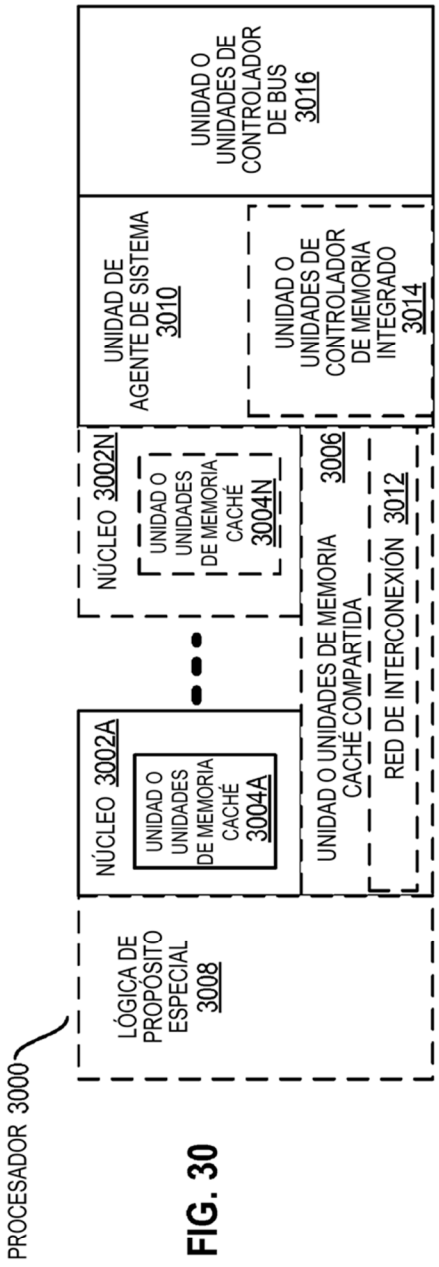

PROCESADOR 3000
LÓGICA DE PROPÓSITO ESPECIAL 3008
NÚCLEO 3002A
UNIDAD O UNIDADES DE MEMORIA CACHÉ 3004A
NÚCLEO 3002N
UNIDAD O UNIDADES DE MEMORIA CACHÉ 3004N
UNIDAD O UNIDADES DE MEMORIA CACHÉ COMPARTIDA 3006
RED DE INTERCONEXIÓN 3012
UNIDAD DE AGENTE DE SISTEMA 3010
UNIDAD O UNIDADES DE CONTROLADOR DE MEMORIA INTEGRADO 3014
UNIDAD O UNIDADES DE CONTROLADOR DE BUS 3016

FIG. 30

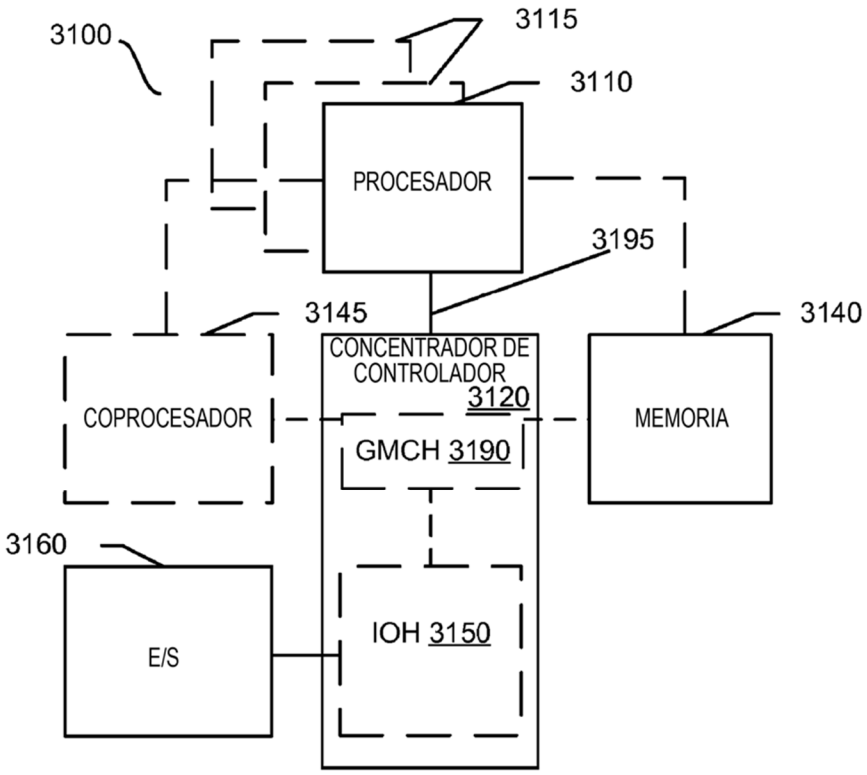

3100
3115
3110
PROCESADOR
3195
3145
3140
CONCENTRADOR DE CONTROLADOR
3120
COPROCESADOR
GMCH 3190
MEMORIA
3160
E/S
IOH 3150

FIG. 31

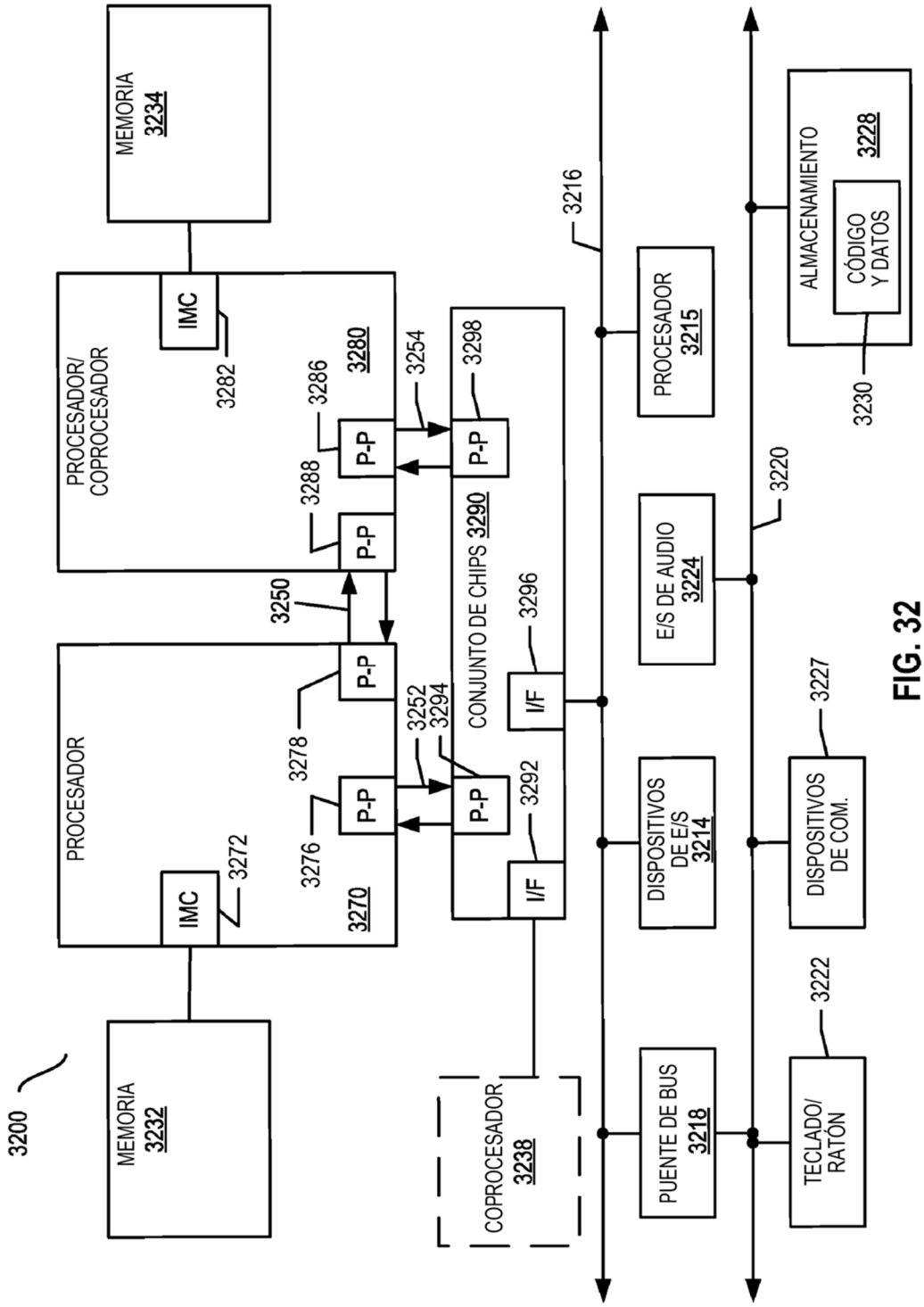

3200
MEMORIA 3232
MEMORIA 3234
PROCESADOR
IMC
3272
3270
PROCESADOR/ COPROCESADOR
IMC
3282
3280
P-P
3278
3276
3252
3294
3288
3286
3254
3298
3250
CONJUNTO DE CHIPS 3290
I/F
3296
3292
COPROCESADOR 3238
3216
PROCESADOR 3215
E/S DE AUDIO 3224
DISPOSITIVOS DE E/S 3214
PUENTE DE BUS 3218
3220
ALMACENAMIENTO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

FIG. 32

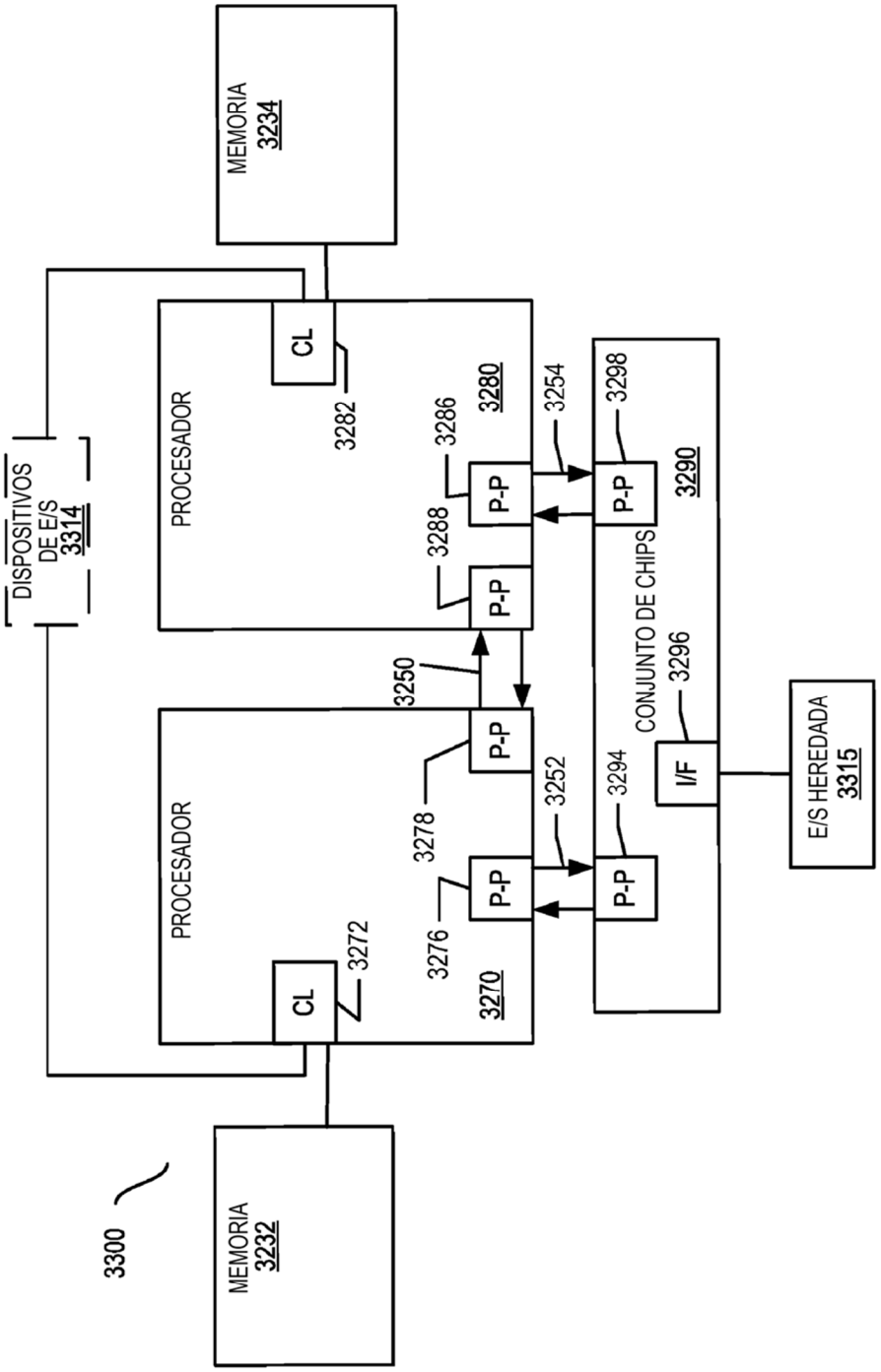

3300
MEMORIA
3232
MEMORIA
3234
DISPOSITIVOS DE E/S
3314
PROCESADOR
CL
3272
3270
PROCESADOR
CL
3282
3280
P-P
3276
P-P
3278
P-P
3288
P-P
3286
3250
3252
3254
CONJUNTO DE CHIPS
3290
P-P
3294
P-P
3298
I/F
3296
E/S HEREDADA
3315

FIG. 33

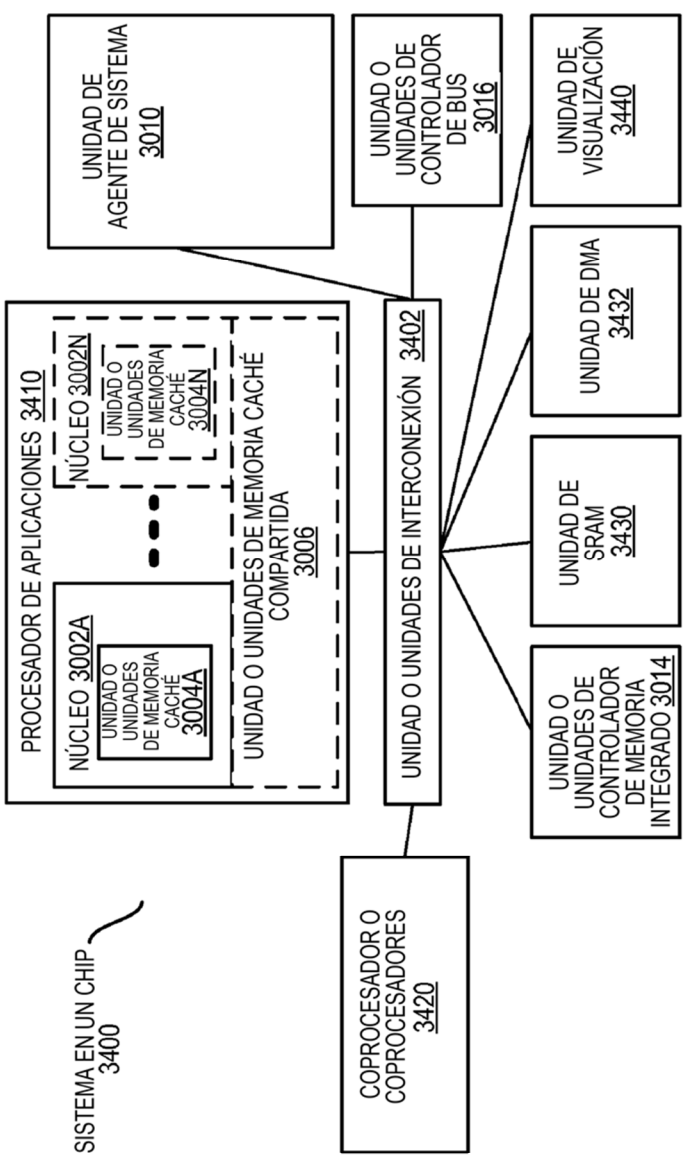

SISTEMA EN UN CHIP
3400
PROCESADOR DE APLICACIONES 3410
NÚCLEO 3002A
UNIDAD O UNIDADES DE MEMORIA CACHÉ 3004A
NÚCLEO 3002N
UNIDAD O UNIDADES DE MEMORIA CACHÉ 3004N
UNIDAD O UNIDADES DE MEMORIA CACHÉ COMPARTIDA 3006
UNIDAD DE AGENTE DE SISTEMA 3010
UNIDAD O UNIDADES DE CONTROLADOR DE BUS 3016
UNIDAD DE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

FIG. 34

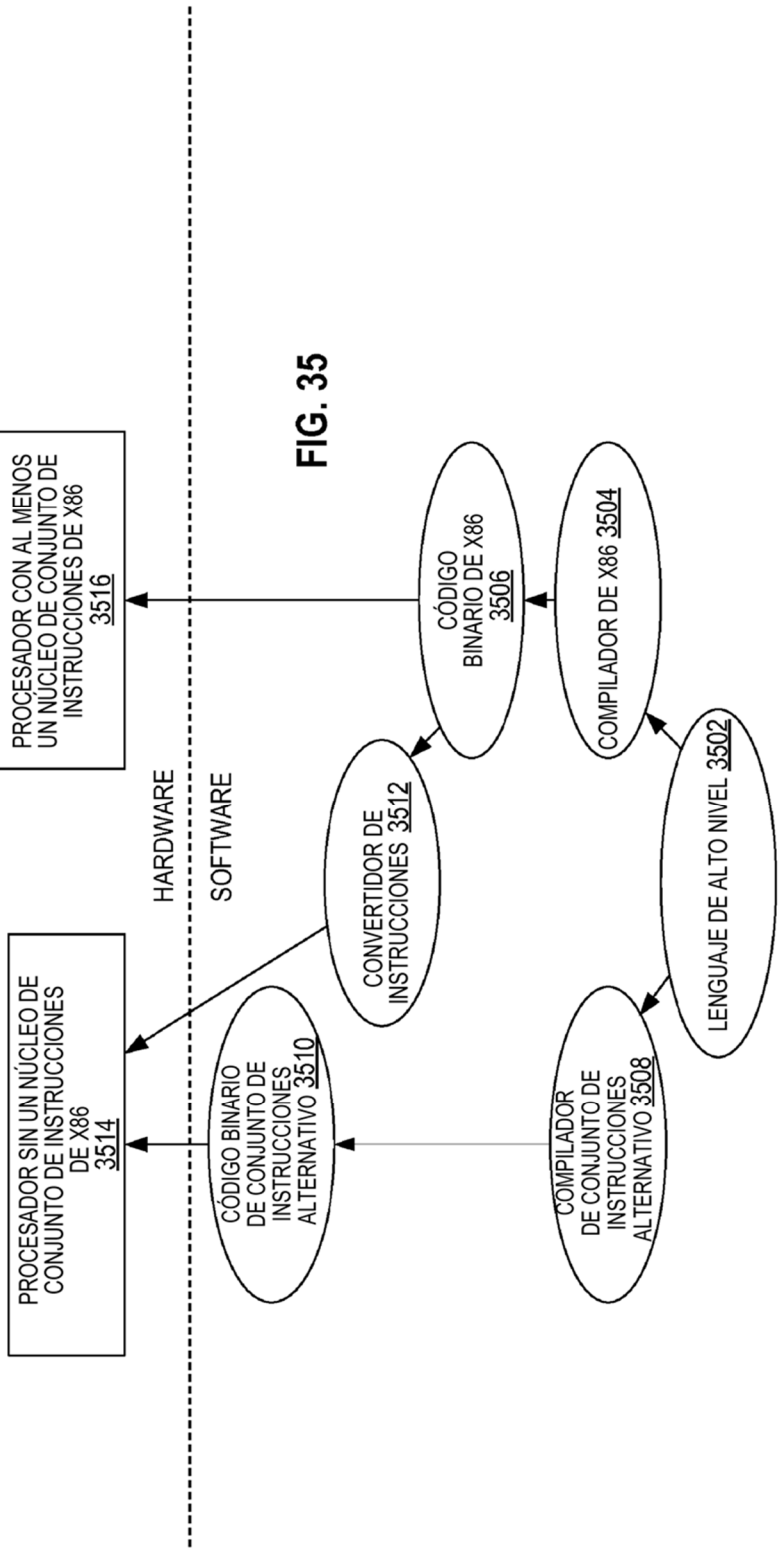

PROCESADOR CON AL MENOS UN NÚCLEO DE CONJUNTO DE INSTRUCCIONES DE X86 3516
HARDWARE
SOFTWARE
PROCESADOR SIN UN NÚCLEO DE CONJUNTO DE INSTRUCCIONES DE X86 3514
CÓDIGO BINARIO DE CONJUNTO DE INSTRUCCIONES ALTERNATIVO 3510
CONVERTIDOR DE INSTRUCCIONES 3512
CÓDIGO BINARIO DE X86 3506
COMPILADOR DE X86 3504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

FIG. 35


RECUPERAR, USANDO UNA CIRCUITERÍA DE RECUPERACIÓN. UNA INSTRUCCIÓN. INCLUYENDO LA INSTRUCCIÓN UN PRIMER CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN VECTOR DE ORIGEN, UN SEGUNDO CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN VECTOR DE DESTINO Y UN CÓDIGO DE OPERACIÓN PARA INDICAR A UNA CIRCUITERÍA DE EJECUCIÓN QUE EJECUTE LA INSTRUCCIÓN DESCODIFICADA PARA CLASIFICAR VALORES DEL VECTOR DE ORIGEN Y ALMACENAR UN RESULTADO DE LA CLASIFICACIÓN EN EL VECTOR DE DESTINO GENERANDO, POR CADA ELEMENTO DEL VECTOR DE ORIGEN, UN VALOR DE ÍNDICE USANDO UNA O MÁS COMPARACIONES DEL ELEMENTO CONSIGO MISMO Y CON OTROS ELEMENTOS DEL VECTOR DE ORIGEN, Y PERMUTANDO VALORES DE LOS ELEMENTOS DEL VECTOR DE ORIGEN BASÁNDOSE EN EL ÍNDICE Y ALMACENANDO LOS VALORES PERMUTADOS EN EL VECTOR DE DESTINO
3601
DESCODIFICAR, USANDO UNA CIRCUITERÍA DE DESCODIFICACIÓN, LA INSTRUCCIÓN RECUPERADA 3603
PROGRAMAR LA EJECUCIÓN DE LA INSTRUCCIÓN DESCODIFICADA
3605
EJECUTAR LA INSTRUCCIÓN DESCODIFICADA (Y PROGRAMADA) COMO ES INDICADO POR EL CÓDIGO DE OPERACIÓN
3607
CONFIRMAR UN RESULTADO DE LA INSTRUCCIÓN EJECUTADA
3609

FIG. 36


RECUPERAR, USANDO UNA CIRCUITERÍA DE RECUPERACIÓN, UNA INSTRUCCIÓN. INCLUYENDO LA INSTRUCCIÓN UN PRIMER CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN VECTOR DE ORIGEN, UN SEGUNDO CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN VECTOR DE DESTINO Y UN CÓDIGO DE OPERACIÓN PARA INDICAR A UNA CIRCUITERÍA DE EJECUCIÓN QUE EJECUTE LA INSTRUCCIÓN DESCODIFICADA PARA CLASIFICAR VALORES DEL VECTOR DE ORIGEN Y ALMACENAR UN RESULTADO DE LA CLASIFICACIÓN EN EL VECTOR DE DESTINO GENERANDO, POR CADA ELEMENTO DEL VECTOR DE ORIGEN, UN VALOR DE ÍNDICE, Y PERMUTANDO LOS VALORES DE LOS ELEMENTOS DEL VECTOR DE ORIGEN EN EL VECTOR DE DESTINO BASÁNDOSE EN LOS VALORES DE ÍNDICE PARA LOS ELEMENTOS
3701
DESCODIFICAR, USANDO UNA CIRCUITERÍA DE DESCODIFICACIÓN, LA INSTRUCCIÓN RECUPERADA 3703
PROGRAMAR LA EJECUCIÓN DE LA INSTRUCCIÓN DESCODIFICADA
3705
EJECUTAR LA INSTRUCCIÓN DESCODIFICADA (Y PROGRAMADA) COMO ES INDICADO POR EL CÓDIGO DE OPERACIÓN
3707
CONFIRMAR UN RESULTADO DE LA INSTRUCCIÓN EJECUTADA
3709

FIG. 37


RECUPERAR, USANDO UNA CIRCUITERÍA DE RECUPERACIÓN, UNA INSTRUCCIÓN. INCLUYENDO LA INSTRUCCIÓN UN PRIMER CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN VECTOR DE ORIGEN, UN SEGUNDO CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN VECTOR DE DESTINO Y UN CÓDIGO DE OPERACIÓN PARA INDICAR A UNA CIRCUITERÍA DE EJECUCIÓN QUE EJECUTE LA INSTRUCCIÓN DESCODIFICADA PARA INDEXAR VALORES DEL VECTOR DE ORIGEN Y ALMACENAR UN RESULTADO DE LA INDEXACIÓN EN EL VECTOR DE DESTINO GENERANDO, POR CADA ELEMENTO DEL VECTOR DE ORIGEN, UN VALOR DE ÍNDICE USANDO UNA O MÁS COMPARACIONES
3801
DESCODIFICAR, USANDO UNA CIRCUITERÍA DE DESCODIFICACIÓN, LA INSTRUCCIÓN RECUPERADA 3803
PROGRAMAR LA EJECUCIÓN DE LA INSTRUCCIÓN DESCODIFICADA
3805
EJECUTAR LA INSTRUCCIÓN DESCODIFICADA (Y PROGRAMADA) COMO ES INDICADO POR EL CÓDIGO DE OPERACIÓN
3807
CONFIRMAR UN RESULTADO DE LA INSTRUCCIÓN EJECUTADA
3809

FIG. 38


RECUPERAR, USANDO UNA CIRCUITERÍA DE RECUPERACIÓN, UNA INSTRUCCIÓN. INCLUYENDO LA INSTRUCCIÓN UN PRIMER CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN VECTOR DE ORIGEN, UN SEGUNDO CAMPO DE OPERANDO PARA IDENTIFICAR UNA UBICACIÓN DE UN VECTOR DE DESTINO Y UN CÓDIGO DE OPERACIÓN PARA INDICAR A UNA CIRCUITERÍA DE EJECUCIÓN QUE EJECUTE LA INSTRUCCIÓN DESCODIFICADA PARA INDEXAR VALORES DEL VECTOR DE ORIGEN Y ALMACENAR UN RESULTADO DE LA INDEXACIÓN EN EL VECTOR DE DESTINO GENERANDO, POR CADA ELEMENTO DEL VECTOR DE ORIGEN, UN VALOR DE ÍNDICE
3901
DESCODIFICAR, USANDO UNA CIRCUITERÍA DE DESCODIFICACIÓN, LA INSTRUCCIÓN RECUPERADA 3903
PROGRAMAR LA EJECUCIÓN DE LA INSTRUCCIÓN DESCODIFICADA
3905
EJECUTAR LA INSTRUCCIÓN DESCODIFICADA (Y PROGRAMADA) COMO ES INDICADO POR EL CÓDIGO DE OPERACIÓN
3907
CONFIRMAR UN RESULTADO DE LA INSTRUCCIÓN EJECUTADA
3909

FIG. 39


RECUPERAR, USANDO UNA CIRCUITERÍA DE RECUPERACIÓN, UNA INSTRUCCIÓN. INCLUYENDO LA INSTRUCCIÓN UN PRIMER CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN PRIMER VECTOR DE ORIGEN, UN SEGUNDO CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN VECTOR DE DESTINO, UN TERCER CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN SEGUNDO VECTOR DE ORIGEN Y UN CÓDIGO DE OPERACIÓN PARA INDICAR A UNA CIRCUITERÍA DE EJECUCIÓN QUE EJECUTE LA INSTRUCCIÓN DESCODIFICADA PARA PERMUTAR LOS VALORES DE ELEMENTOS DEL PRIMER VECTOR DE ORIGEN BASÁNDOSE EN UN ÍNDICE ALMACENADO EN EL SEGUNDO VECTOR DE ORIGEN Y ALMACENAR EL RESULTADO DE LA PERMUTACIÓN EN EL VECTOR DE DESTINO
4001
DESCODIFICAR, USANDO UNA CIRCUITERÍA DE DESCODIFICACIÓN, LA INSTRUCCIÓN RECUPERADA 4003
PROGRAMAR LA EJECUCIÓN DE LA INSTRUCCIÓN DESCODIFICADA
4005
EJECUTAR LA INSTRUCCIÓN DESCODIFICADA (Y PROGRAMADA) COMO ES INDICADO POR EL CÓDIGO DE OPERACIÓN
4007
CONFIRMAR UN RESULTADO DE LA INSTRUCCIÓN EJECUTADA
4009

FIG. 40


RECUPERAR, USANDO UNA CIRCUITERÍA DE RECUPERACIÓN, UNA INSTRUCCIÓN. INCLUYENDO LA INSTRUCCIÓN UN PRIMER CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN PRIMER VECTOR DE ORIGEN, UN SEGUNDO CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN PRIMER VECTOR DE ORIGEN, UN TERCER CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN VECTOR DE DESTINO Y UN CÓDIGO DE OPERACIÓN PARA INDICAR A UNA CIRCUITERÍA DE EJECUCIÓN QUE EJECUTE LA INSTRUCCIÓN DESCODIFICADA PARA CLASIFICAR VALORES DEL VECTOR DE ORIGEN Y ALMACENAR UN RESULTADO DE LA CLASIFICACIÓN EN EL VECTOR DE DESTINO GENERANDO, POR CADA POSICIÓN DE ELEMENTO DEL PRIMER Y EL SEGUNDO VECTORES DE ORIGEN, UN VALOR DE ÍNDICE USANDO UNA O MÁS COMPARACIONES DE LOS ELEMENTOS, Y PERMUTANDO Y ALMACENANDO LOS VALORES DE LOS ELEMENTOS BASÁNDOSE EN LOS VALORES DE ÍNDICE PARA LOS ELEMENTOS EN EL VECTOR DE DESTINO
4101
DESCODIFICAR, USANDO UNA CIRCUITERÍA DE DESCODIFICACIÓN, LA INSTRUCCIÓN RECUPERADA 4103
PROGRAMAR LA EJECUCIÓN DE LA INSTRUCCIÓN DESCODIFICADA
4105
EJECUTAR LA INSTRUCCIÓN DESCODIFICADA (Y PROGRAMADA) COMO ES INDICADO POR EL CÓDIGO DE OPERACIÓN
4107
CONFIRMAR UN RESULTADO DE LA INSTRUCCIÓN EJECUTADA
4109

FIG. 41


RECUPERAR, USANDO UNA CIRCUITERÍA DE RECUPERACIÓN, UNA INSTRUCCIÓN. INCLUYENDO LA INSTRUCCIÓN UN PRIMER CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN PRIMER VECTOR DE ORIGEN, UN SEGUNDO CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN PRIMER VECTOR DE ORIGEN, UN TERCER CAMPO PARA IDENTIFICAR UNA UBICACIÓN DE UN VECTOR DE DESTINO Y UN CÓDIGO DE OPERACIÓN PARA INDICAR A UNA CIRCUITERÍA DE EJECUCIÓN QUE EJECUTE LA INSTRUCCIÓN DESCODIFICADA PARA, PARA CADA POSICIÓN DE ELEMENTO DEL PRIMER Y EL SEGUNDO VECTORES DE ORIGEN, GENERAR UN VALOR DE ÍNDICE USANDO UNA O MÁS COMPARACIONES DE LOS ELEMENTOS Y ALMACENAR EL VALOR DE ÍNDICE GENERADO EN UNA POSICIÓN DE ELEMENTO CORRESPONDIENTE DEL VECTOR DE DESTINO
4201
DESCODIFICAR, USANDO UNA CIRCUITERÍA DE DESCODIFICACIÓN, LA INSTRUCCIÓN RECUPERADA 4203
PROGRAMAR LA EJECUCIÓN DE LA INSTRUCCIÓN DESCODIFICADA
4205
EJECUTAR LA INSTRUCCIÓN DESCODIFICADA (Y PROGRAMADA) COMO ES INDICADO POR EL CÓDIGO DE OPERACIÓN
4207
CONFIRMAR UN RESULTADO DE LA INSTRUCCIÓN EJECUTADA
4209

FIG. 42