(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **3 013 609**

(51) Int. CI.:

G06F 9/30 (2008.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **08.10.2019 E 20216494 (3)**

(97) Fecha y número de publicación de la concesión europea: **04.12.2024 EP 3822774**

(54) Título: **Sistemas y métodos para realizar instrucciones para convertir a formato de coma flotante de 16 bits**

(30) Prioridad:

09.11.2018 US 201816186384

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
14.04.2025

(73) Titular/es:

INTEL CORPORATION (100.00%)
2200 Mission College Blvd.
Santa Clara, CA 95054, US

(72) Inventor/es:

HEINECKE, ALEXANDER F.;
VALENTINE, ROBERT;
CHARNEY, MARK J.;
SADE, RAANAN;
ADELMAN, MENACHEM;
SPERBER, ZEEV;
GRADSTEIN, AMIT y
RUBANOVICH, SIMON

(74) Agente/Representante:

LEHMANN NOVO, María Isabel

DESCRIPCIÓN

Sistemas y métodos para realizar instrucciones para convertir a formato de coma flotante de 16 bits

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 para convertir a formato de coma flotante de 16 bits.

ANTECEDENTES

Un conjunto de instrucciones, o arquitectura de conjunto de instrucciones (ISA), es la parte de la arquitectura del ordenador relacionada con la programación, y puede incluir los tipos de datos nativos, las instrucciones, la arquitectura de registro, los modos de direccionamiento, la arquitectura de memoria, el manejo de interrupciones y excepciones, y la entrada y salida externa (E/S). Un conjunto de instrucciones incluye uno o más formatos de instrucciones. Un formato de instrucción dado define diversos campos (número de bits, ubicación de bits) para especificar, entre otras cosas, la operación a realizar y el operando u operandos en los que va a realizarse esa operación. Una instrucción dada se expresa usando un formato de instrucción dado y especifica la operación y los operandos. Un flujo de instrucciones es una secuencia específica de instrucciones, donde cada instrucción de la secuencia es una ocurrencia de una instrucción en un formato de instrucción.

Las aplicaciones científicas, financieras, auto-vectorizadas de propósito general, RMS (reconocimiento, minería y síntesis)/visuales y multimedia (por ejemplo, gráficos 2D/3D, procesamiento de imágenes, compresión/descompresión de vídeo, algoritmos de reconocimiento de voz y manipulación de audio) a menudo requieren que se realice la misma operación en una gran cantidad de elementos de datos (denominado como "paralelismo de datos"). Una instrucción única de datos múltiples (SIMD) se refiere a un tipo de instrucción que hace que un procesador realice la misma operación en múltiples elementos de datos. La tecnología de SIMD es especialmente adecuada para los procesadores que pueden dividir lógicamente los bits de un registro en una cantidad de elementos de datos de tamaño fijo, cada uno de los cuales representa un valor separado. Por ejemplo, los bits de un registro de 512 bits se pueden especificar como un operando de origen que se operará como dieciséis elementos de datos de coma flotante de precisión sencilla de 32 bits separados. Como otro ejemplo, los bits en un registro de 256 bits pueden especificarse como un operando de origen para ser operado como dieciséis elementos de datos empaquetados de coma flotante de 16 bits separados, ocho elementos de datos empaquetados de 32 bits separados (elementos de datos de tamaño de palabra doble), o treinta y dos elementos de datos de 8 bits separados (elementos de datos de tamaño byte (B)). Este tipo de datos se denomina tipo de datos empaquetados o tipo de datos vectoriales, y los operandos de este tipo de datos se denominan operandos de datos empaquetados u operandos vectoriales. En otras palabras, un elemento o vector de datos empaquetados se refiere a una secuencia de elementos de datos empaquetados; y un operando de datos empaquetados o un operando de vector es un operando de origen o de destino de una instrucción de SIMD (también conocida como instrucción de datos empaquetados o instrucción de vector).

A modo de ejemplo, un tipo de instrucción de SIMD especifica una única operación vectorial que va a realizarse en dos operandos de vector de origen en forma vertical para generar un operando de vector de destino del mismo tamaño, con el mismo número de elementos de datos, y en el mismo orden de elementos de datos. Los elementos de datos en los operandos de vector de origen se denominan elementos de datos de origen, mientras que los elementos de datos en el operando de vector de destino se denominan elementos de datos de destino o resultado. Estos operandos de vector de origen tienen el mismo tamaño y contienen elementos de datos de la misma anchura y, por lo tanto, contienen el mismo número de elementos de datos. Los elementos de datos de origen en las mismas posiciones de bits en los dos operandos de vector de origen forman pares de elementos de datos (también denominados elementos de datos correspondientes; es decir, el elemento de datos en la posición de elemento de datos 0 de cada operando fuente corresponde, el elemento de datos en la posición de elemento de datos 1 de cada operando de origen corresponde, y así sucesivamente). La operación especificada por esa instrucción de SIMD se realiza por separado en cada uno de estos pares de elementos de datos de origen para generar un número coincidente de elementos de datos de resultado y, por lo tanto, cada par de elementos de datos de origen tiene un elemento de datos de resultado correspondiente. Dado que la operación es vertical y que el operando de vector de resultado tiene el mismo tamaño, tiene el mismo número de elementos de datos y los elementos de datos de resultado se almacenan en el mismo orden de elementos de datos que los operandos de vector de origen, los elementos de datos de resultado están en las mismas posiciones de bit del operando de vector de resultado que su correspondiente par de elementos de datos de origen en los operandos de vector de origen. Además de este tipo ilustrativo de instrucción de SIMD, existe una diversidad de otros tipos de instrucciones de SIMD.

Algunas aplicaciones que procesan vectores que tienen precisión sencilla rinden casi igual de bien usando vectores formateados en coma flotante de 16 bits en su lugar.

En "Advanced Micro Devices AMD64 Technology AMD64 Architecture Programmer's Manual Volume 4: 128-Bit and 256-Bit Media Instructions", URL: https://developer.amd.com/word press/media/2012/10/26568_APM_v41.pdf, describe el conjunto de instrucciones de Extensiones de SIMD de Envío por Flujo Continuo (SSE) que se usará con

productos AMD.El conjunto de instrucciones SSE incluye instrucciones de medios de 128 bits y 256 bits. Las SSE incluyen tanto las formas tradicionales como las formas extendidas. Por ejemplo, el conjunto de instrucciones SSE incluye la instrucción VCVTPS2PH para convertir valores de coma flotante de precisión sencilla empaquetados en valores de coma flotante de 16 bits empaquetados y escribir los valores convertidos en el registro de destino o en memoria. Un operando inmediato de 8 bits proporciona un control dinámico de redondeo.

SUMARIO

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

BREVE DESCRIPCIÓN DE LOS DIBUJOS

La **Figura 1** es un diagrama de bloques que ilustra componentes de procesamiento para ejecutar una instrucción de conversión de formato (VCVTNEPS2BF16 o VCVTNE2PS2BF16), de acuerdo con una realización;

La **Figura 2A** es un diagrama de bloques que ilustra la ejecución de una instrucción de conversión de formato (VCVTNEPS2BF16), de acuerdo con una realización;

La **Figura 2B** es un diagrama de bloques que ilustra la ejecución de una instrucción de conversión de formato (VCVTNEPS2BF16), de acuerdo con una realización;

La **Figura 2C** es un diagrama de bloques que ilustra la ejecución de una instrucción de conversión de formato (VCVTNEPS2BF16), de acuerdo con una realización;

La **Figura 2D** es un diagrama de bloques que ilustra la ejecución de una instrucción de conversión de formato de 2 entradas (VCVTNE2PS2BF16), de acuerdo con una realización;

La **Figura 3A** es un pseudocódigo que ilustra la ejecución ilustrativa de una instrucción de conversión de formato (VCVTNEPS2BF16), de acuerdo con una realización;

La **Figura 3B** es un pseudocódigo que ilustra la ejecución ilustrativa de una instrucción de conversión de formato de 2 entradas (VCVTNE2PS2BF16), de acuerdo con una realización;

La **Figura 3C** es un pseudocódigo que ilustra una función auxiliar para su uso con el pseudocódigo de las **Figuras 3A** y **3B,** de acuerdo con una realización;

La **Figura 4A** es un diagrama de flujo de proceso que ilustra un procesador que responde a una instrucción de conversión de formato (VCVTNEPS2BF16), de acuerdo con una realización;

La **Figura 4B** es un diagrama de flujo de proceso que ilustra un procesador que responde a una instrucción de conversión de formato de 2 entradas (VCVTNE2PS2BF16), de acuerdo con una realización;

La **Figura 5A** es un diagrama de bloques que ilustra una instrucción de conversión de formato (VCVTNEPS2BF16), de acuerdo con una realización;

La **Figura 5B** es un diagrama de bloques que ilustra una instrucción de conversión de formato de 2 entradas (VCVTNE2PS2BF16), de acuerdo con una realización;

Las **Figuras 6A-6B** son diagramas de bloques que ilustran un formato de instrucción compatible con vectores genérico y las plantillas de instrucción del mismo de acuerdo con algunas realizaciones de la invención;

La **Figura 6A** es un diagrama de bloques que ilustra un formato de instrucción compatible con vectores genérico y las plantillas de instrucción de clase A del mismo de acuerdo con algunas realizaciones de la invención;

La **Figura 6B** es un diagrama de bloques que ilustra el formato de instrucción compatible con vectores genérico y las plantillas de instrucción de clase B del mismo de acuerdo con algunas realizaciones de la invención;

La **Figura 7A** es un diagrama de bloques que ilustra un formato de instrucción compatible con vectores específico ilustrativo de acuerdo con algunas realizaciones de la invención;

La **Figura 7B** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico que componen el campo de código de operación completo de acuerdo con una realización;

La **Figura 7C** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico que componen el campo de índice de registro de acuerdo con una realización;

La **Figura 7D** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico que componen el campo de operación de aumento de acuerdo con una realización;

La **Figura 8** es un diagrama de bloques de una arquitectura de registro de acuerdo con una realización;

La **Figura 9A** es un diagrama de bloques que ilustra tanto un canal ilustrativo en orden como un canal ilustrativo de emisión/ejecución desordenada y cambio de nombre de registro de acuerdo con algunas realizaciones;

La **Figura 9B** es un diagrama de bloques que ilustra tanto una realización ilustrativa de un núcleo de arquitectura en orden y un núcleo ilustrativo de arquitectura de emisión/ejecución desordenada y cambio de nombre de registro que se incluirá en un procesador de acuerdo con algunas realizaciones;

Las Figuras 10A-B ilustran un diagrama de bloques de una arquitectura de núcleo ordenada ilustrativa más específica, cuyo núcleo sería uno de varios bloques lógicos (que incluyen otros núcleos del mismo tipo y/o tipos diferentes) en un chip;

La **Figura 10A** es un diagrama de bloques de un único núcleo de procesador, junto con su conexión a la red de interconexión en chip y con su subconjunto local de la caché de Nivel 2 (L2), de acuerdo con algunas realizaciones;

La **Figura 10B** es una vista ampliada de parte del núcleo de procesador en la **Figura 10A** de acuerdo con algunas realizaciones;

La **Figura 11** 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 algunas realizaciones;

Las **Figuras 12-15** son diagramas de bloques de arquitecturas informáticas ilustrativas;

La **Figura 12** muestra un diagrama de bloques de un sistema de acuerdo con algunas realizaciones;

La **Figura 13** es un diagrama de bloques de un primer sistema ilustrativo más específico de acuerdo con alguna realización;

La **Figura 14** es un diagrama de bloques de un segundo sistema ilustrativo más específico de acuerdo con algunas realizaciones;

La **Figura 15** es un diagrama de bloques de un sistema en un chip (SoC) de acuerdo con algunas realizaciones; y

La **Figura 16** es un diagrama de bloques que contrasta el uso de un convertidor de instrucciones de software para convertir instrucciones binarias en un conjunto de instrucciones de origen a instrucciones binarias en un conjunto de instrucciones objetivo de acuerdo con algunas realizaciones.

DESCRIPCIÓN DETALLADA DE LAS REALIZACIONES

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

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

Como se ha mencionado anteriormente, algunas aplicaciones que procesan vectores que tienen precisión sencilla rinden casi igual de bien usando vectores formateados en coma flotante de 16 bits en su lugar. En el presente

documento se divulga e ilustra a través de las figuras una instrucción de conversión de formato de datos empaquetados en vectores (VCVTNEPS2BF16 y VCVTNE2PS2BF16) que implementa la conversión de formato de uno o dos vectores de origen. La nemotecnia VCVTNEPS2BF16 indica: "VCVT"=Vector ConVerT, "NE"=redondeo a **P**ar **M**ás_cercano, "PS"=Fuente de precisión sencilla empaquetada, "2"=para, y "BF16"=**BF**loat16. La versión de 2 entradas de la instrucción toma dos vectores de origen, cada uno con N elementos de precisión sencilla, y genera un vector de destino que tiene 2 veces N elementos formateados en coma flotante de 16 bits. La versión de 2 entradas permite una solución equilibrada donde los vectores de origen de N elementos se convierten en vectores de destino de N elementos. Con una solución tan equilibrada, todos los operandos, ya sean de origen o de destino, se pueden almacenar en el mismo tipo de registros vectoriales, ya sean registros vectoriales de 128, 256 o 512 bits. Un archivo de registro de procesador ilustrativo se ilustra y describe al menos con respecto a la **Figura 8.**

En comparación con los algoritmos que usan precisión sencilla tanto para los elementos de origen como de destino, se espera que la instrucción de conversión de formato divulgada (VCVTNEPS2BF16 o VCVTNE2PS2BF16) alcance una calidad comparable, pero con una utilización de memoria y requisitos de ancho de banda de memoria reducidos, lo que serviría para mejorar rendimiento y eficiencia energética, especialmente en un contexto de aprendizaje automático.

FORMATOS DE COMA FLOTANTE RELEVANTES

Los formatos de coma flotante de 16 bits usados en las realizaciones divulgadas incluyen bfloat16 (definido por Google, Inc., de Mountain View, California), al que en ocasiones se hace referencia en el presente documento como "bf16 o BF16", y binario16 (promulgado como IEEE754-2008 por el Instituto de Ingenieros Eléctricos y Electrónicos), al que en ocasiones se hace referencia en el presente documento como "media precisión" o "fp16". Los formatos de coma flotante de 32 bits usados en las realizaciones divulgadas incluyen el binario32 (también promulgado como parte del IEEE754-2008), al que en ocasiones se hace referencia en el presente documento como "precisión sencilla" o "fp32".

La **Tabla 1** enumera algunas características y distinciones relevantes entre los formatos de datos relevantes. Como se muestra, los tres formatos incluyen un bit de signo. El binario32, el binario16 y el bfloat16 tienen anchuras de exponente de 8 bits, 5 bits y 8 bits, respectivamente, y bits significativos (en ocasiones denominados en el presente documento "mantisa" o "fracción") de 24 bits, 11 bits y 8 bits, respectivamente. Una ventaja de bfloat16 sobre fp16 es que se pueden truncar los números fp32 y tener un número bfloat16 válido.

Tabla 1

Formato	Bits	Signo	Exponente	Significando
Binario32	32	1	8 bits	24 bits
Binario16	16	1	5 bits	11 bits
Bfloat16	16	1	8 bits	8 bits

Un procesador que implementa la instrucción de conversión de formato divulgada (VCVTNEPS2BF16 o VCVTNE2PS2BF16) incluiría circuitos de recuperación para recuperar una instrucción que tiene campos para especificar un código de operación y ubicaciones de la primera fuente, la segunda fuente (para la versión de 2 entradas) y los vectores de destino. El formato de la instrucción de conversión de formato (VCVTNEPS2BF16 o VCVTNE2PS2BF16) se ilustra y describe con más detalle al menos con respecto a las **Figuras 5A-B**, **6A-B**, y **7A-D.** Los vectores de origen y destino especificados pueden ubicarse en registros vectoriales o en la memoria. El código de operación para indicar la circuitería de ejecución consiste en convertir cada uno de los elementos del vector de origen especificado a coma flotante de 16 bits, la conversión para incluir truncamiento y redondeo, según sea necesario, y almacenar cada elemento convertido en una ubicación correspondiente del vector de destino especificado. Un procesador de este tipo incluiría además circuitería de decodificación para decodificar la instrucción recuperada y circuitería de ejecución para responder a la instrucción decodificada según se especifica en el código de operación. La circuitería de ejecución se describe e ilustra con más detalle a continuación, al menos con respecto a las **Figuras 1-2D**, 9A-B y **10A-B.**

La **Figura 1** es un diagrama de bloques que ilustra componentes de procesamiento para ejecutar una instrucción de conversión de formato (VCVTNEPS2BF16 o VCVTNE2PS2BF16), de acuerdo con algunas realizaciones. Como se muestra, el sistema informático 100 incluye almacenamiento 101 para almacenar instrucción o instrucciones de conversión de formato 103 que se van a ejecutar. En algunas realizaciones, el sistema informático 100 es un procesador de SIMD para procesar simultáneamente múltiples elementos de vectores de datos empaquetados.

En operación, la instrucción o instrucciones de conversión de formato 103 se recuperan del almacenamiento 101 mediante la circuitería de recuperación 105. La instrucción o instrucciones de conversión de formato 103 tienen campos, no mostrados en este punto, para especificar un código de operación y ubicaciones de un primer vector de origen que comprende N elementos de precisión sencilla, y un vector de destino que comprende al menos N elementos

de coma flotante de 16 bits, el código de operación para indicar que la circuitería de ejecución es para convertir cada uno de los elementos del vector de origen especificado a formato de coma flotante de 16 bits, la conversión para incluir truncamiento y redondeo, según sea necesario, y almacenar cada elemento convertido en una ubicación correspondiente del vector de destino especificado. El formato de instrucción de conversión de formato (VCVTNEPS2BF16 o VCVTNE2PS2BF16) se ilustra y describe además al menos con respecto a las Figuras 5A-B, 6A-B y 7A-D.

La instrucción de conversión de formato recuperada 107 se decodifica mediante la circuitería de decodificación 109, que decodifica la instrucción de conversión de formato recuperada (VCVTNEPS2BF16 o VCVTNE2PS2BF16) 107 en una o más operaciones. En algunas realizaciones, esta decodificación incluye generar una pluralidad de microoperaciones para que se realicen por la circuitería de ejecución (tal como la circuitería de ejecución 117). La circuitería de decodificación 109 también decodifica sufijos y prefijos de instrucciones (si se usan).

La circuitería de ejecución 117, que tiene acceso al archivo de registro y a la memoria 115, es para responder a la instrucción decodificada 111 como se especifica en el código de operación, y se describe e ilustra además a continuación, al menos con respecto a las Figuras 2A-D, 3A-C, 4A-B, 9A-B y 10A-B.

En algunas realizaciones, el circuito 113 de cambio de nombre de registro, asignación de registro y/o planificación proporciona funcionalidad para uno o más de: 1) cambiar el nombre de los valores de operandos lógicos a valores de operandos físicos (por ejemplo, una tabla de alias de registros en algunas realizaciones), 2) asignar bits de estado y banderas a la instrucción decodificada, y 3) planificar la instrucción de conversión de formato decodificada (VCVTNEPS2BF16 o VCVTNE2PS2BF16) 111 para ejecución en la circuitería de ejecución 117 fuera de una agrupación de instrucciones (por ejemplo, usando una estación de reserva en algunas realizaciones).

En algunas realizaciones, el circuito de reescritura 119 sirve para reescribir los resultados de la instrucción ejecutada. El circuito de reescritura 119 y el circuito de cambio de nombre/planificación de registros 113 son opcionales, según se indica por sus bordes de línea discontinua, en la medida en que puedan ocurrir en diferentes tiempos, o no ocurrir en absoluto.

La **Figura 2A** es un diagrama de bloques que ilustra la ejecución de una instrucción de conversión de formato (VCVTNEPS2BF16), de acuerdo con una realización. Como se muestra, el aparato informático 200 (por ejemplo, un procesador) ha de recibir, recuperar y decodificar (la circuitería de recuperación y decodificación no mostradas en este punto, pero se ilustran y describen al menos con respecto a la **Figura 1** y las Figuras 9A-B) la instrucción de conversión de formato 201. La instrucción de conversión de formato 201 incluye campos para especificar el código de operación 202 (VCVTNEPS2BF16) y ubicaciones del primer vector de origen 206 que comprende N elementos de precisión sencilla, y el vector de destino 204 que comprende al menos N elementos de coma flotante de 16 bits (por ejemplo, bfloat16 o binario16).

En este punto, N equivale a 4, y tanto el primer vector de origen 212 como el de destino 218 especificados tienen cuatro elementos. Pero los vectores de origen y destino no están equilibrados, en la medida en que tienen anchuras diferentes. El software podría emitir una instrucción de conversión de formato no equilibrada 201 asignando vectores de diferentes tamaños a los vectores de origen y de destino, por ejemplo, asignando un vector ymm de 256 bits como origen y un vector xmm de 128 bits como destino. Las Figuras 2B-D ilustran situaciones en las que se logra el equilibrio asignando los mismos tipos de vectores tanto al origen como al destino. Un archivo de registro de procesador ilustrativo se ilustra y describe además al menos con respecto a la **Figura 8.**

En algunas realizaciones, la instrucción de conversión de formato 201 también incluye una máscara {k} 208 y un control de puesta a cero {z} 210. El formato de la instrucción de conversión de formato 201, con código de operación de VCVTNEPS2BF16, se ilustra y describe además al menos con respecto a las **Figuras 5A**, 6A-B y 7A-D. También se muestra el primer vector de origen especificado 212, la circuitería de ejecución 214, que incluye la circuitería de conversión 216A-D, y el vector de destino especificado 218.

En operación, el aparato informático 200 (por ejemplo, un procesador) es para recuperar y decodificar, usando circuitería de recuperación y decodificación (no mostradas), teniendo la instrucción 201 campos para especificar el código de operación 202 y las ubicaciones de los primeros vectores de origen 206 y destino 204, el código de operación para indicar que el aparato informático (por ejemplo, procesador) es para convertir cada uno de los elementos del primer vector de origen especificado 212 a formato de coma flotante de 16 bits (por ejemplo, bfloat16), la circuitería de convertidor 216A-D incluye truncamiento y redondeo, según sea necesario, y para almacenar cada elemento convertido en una ubicación correspondiente del vector de destino especificado 218. Como se ilustra y describe además al menos con respecto a las **Figuras 5A**, 6A-B y 7A-D, la instrucción 201 en otras realizaciones puede especificar diferentes longitudes de vector, tales como 128 bits, 512 bits o 1024 bits. La circuitería de ejecución 214 en este punto es para responder a la instrucción decodificada como se especifica por el código de operación 202.

La **Figura 2B** es un diagrama de bloques que ilustra la ejecución de una instrucción de conversión de formato (VCVTNEPS2BF16), de acuerdo con una realización. Como se muestra, el aparato informático 220 (por ejemplo, un procesador) es para recibir, recuperar y decodificar (la circuitería de recuperación y decodificación no mostradas en

este punto, pero se ilustran y describen al menos con respecto a la **Figura 1** y las Figuras 9A-B), instrucción de conversión de formato 221, que incluye campos para especificar el código de operación 222 (VCVTNEPS2BF16) y ubicaciones del primer vector de origen 226 que comprende N elementos de precisión sencilla, y el vector de destino 224 que comprende al menos N elementos de coma flotante de 16 bits (por ejemplo, bfloat16 o binario16).

En este punto, el equilibrio se logra asignando el mismo tipo de registro que el primer origen 232 y el destino 238 especificados. Sin embargo, el vector de destino especificado 238, que tiene la mitad de la anchura del primer vector de origen especificado 232, tiene el doble de entradas. En operación, las entradas convertidas se escriben en las primeras cuatro entradas de destino y se escriben ceros en las cuatro entradas restantes.

En algunas realizaciones, la instrucción de conversión de formato 221 también incluye una máscara {k} 228 y un control de puesta a cero {z} 230. El formato de la instrucción de conversión de formato 221, con código de operación de VCVTNEPS2BF16, se ilustra y describe además al menos con respecto a las **Figuras 5A**, 6A-B y 7A-D. También se muestra el primer vector de origen especificado 232, la circuitería de ejecución 234, que incluye la circuitería de conversión 236A-D, y el vector de destino especificado 238.

En operación, el aparato informático 220 (por ejemplo, un procesador) es para recuperar y decodificar, usando circuitería de recuperación y decodificación (no mostradas), teniendo la instrucción 221 campos para especificar el código de operación 222 (es decir, VCVTNEPS2BF16) y las ubicaciones de los primeros vectores de origen 226 y destino 224, el código de operación para indicar que el aparato informático 220 (por ejemplo, procesador) es para convertir, usando los convertidores 236A-D en la circuitería de ejecución 234, cada uno de los elementos del primer vector de origen especificado 232 a formato de coma flotante de 16 bits (por ejemplo, bfloat16), la circuitería de convertidor 236A-D incluye truncamiento y redondeo, según sea necesario, y para almacenar cada elemento convertido en una ubicación correspondiente del vector de destino especificado 238. En este punto, las ubicaciones del vector de destino correspondientes comprenden los primeros cuatro elementos, y se escriben ceros en los cuatro elementos restantes. Como se ilustra y describe además al menos con respecto a las **Figuras 5A**, 6A-B y 7A-D, la instrucción 221 en otras realizaciones puede especificar diferentes longitudes de vector, tales como 128 bits, 512 bits o 1024 bits. La circuitería de ejecución 234 en este punto es para responder a la instrucción decodificada como se especifica por el código de operación 222.

La **Figura 2C** es un diagrama de bloques que ilustra la ejecución de una instrucción de conversión de formato (VCVTNEPS2BF16), de acuerdo con una realización. Como se muestra, el aparato informático 240 (por ejemplo, un procesador) es para recibir, recuperar y decodificar (la circuitería de recuperación y decodificación no mostradas en este punto, pero se ilustran y describen al menos con respecto a la **Figura 1** y las Figuras 9A-B), instrucción de conversión de formato 241, que incluye campos para especificar el código de operación 242 (VCVTNEPS2BF16) y ubicaciones del primer vector de origen 246 que comprende N elementos de precisión sencilla, y el vector de destino 244 que comprende al menos N elementos de coma flotante de 16 bits (por ejemplo, bfloat16 o binario16).

En este punto, el equilibrio se logra asignando el mismo tipo de registro que los primeros vectores de origen 252 y de destino 258 especificados. Sin embargo, el vector de destino especificado 258, que tiene la mitad de la anchura del primer vector de origen especificado 252, tiene el doble de entradas. En operación, las entradas convertidas se escriben en las primeras cuatro entradas de destino y se escriben ceros en las cuatro entradas restantes. La puesta a cero no se muestra en la **Figura 2C**, pero ha de hacerse implícitamente en esta realización.

La puesta a cero implícita en algunas realizaciones es un tratamiento predeterminado de elementos enmascarados. En otras realizaciones, se debe programar mediante software un registro específico del modelo arquitectónico (MSR) para controlar si se aplica la puesta a cero o el enmascaramiento a elementos enmascarados. Aún en otras realizaciones, el comportamiento de puesta a cero se especifica mediante la instrucción de conversión de formato.

En algunas realizaciones, la instrucción de conversión de formato 241 también incluye una máscara {k} 248 y un control de puesta a cero {z} 250. El formato de la instrucción de conversión de formato 241, con código de operación de VCVTNEPS2BF16, se ilustra y describe además al menos con respecto a las **Figuras 5A**, 6A-B, y 7A-D.

También se muestran el primer vector de origen especificado 252, la circuitería de ejecución 254, que incluye la circuitería de conversión 256A-D, y el vector de destino especificado 258.

En operación, el aparato informático 240 (por ejemplo, un procesador) es para recuperar y decodificar, usando circuitería de recuperación y decodificación (no mostradas), teniendo la instrucción 241 campos para especificar el código de operación 242 (es decir, VCVTNEPS2BF16) y las ubicaciones de los primeros vectores de origen 246 y destino 244, el código de operación para indicar que el aparato informático 240 (por ejemplo, procesador) es para convertir, usando los convertidores 256A-D en la circuitería de ejecución 254, cada uno de los elementos del primer vector de origen especificado 252 a formato de coma flotante de 16 bits (por ejemplo, bfloat16), la circuitería de convertidor 256A-D incluye truncamiento y redondeo, según sea necesario, y para almacenar cada elemento convertido en una ubicación correspondiente del vector de destino especificado 258. En este punto, las ubicaciones del vector de destino correspondientes comprenden los primeros cuatro elementos, y se escriben ceros implícitamente en los cuatro elementos restantes. La puesta a cero implícita en algunas realizaciones es un tratamiento

predeterminado de elementos enmascarados. En otras realizaciones, se debe programar mediante software un registro específico del modelo arquitectónico (MSR) para controlar si se aplica la puesta a cero o el enmascaramiento a elementos enmascarados. Aún en otras realizaciones, el comportamiento de puesta a cero se especifica mediante la instrucción de conversión de formato.

Como se ilustra y describe además al menos con respecto a las **Figuras 5A**, 6A-B y 7A-D, la instrucción 241 en otras realizaciones puede especificar diferentes longitudes de vector, tales como 128 bits, 512 bits o 1024 bits. La circuitería de ejecución 254 en este punto es para responder a la instrucción decodificada como se especifica por el código de operación 242.

La **Figura 2D** es un diagrama de bloques que ilustra la ejecución de una instrucción de conversión de formato (VCVTNE2PS2BF16), de acuerdo con una realización. Como se muestra, el aparato informático 260 (por ejemplo, un procesador) es para recibir, recuperar y decodificar (la circuitería de recuperación y decodificación no mostradas en este punto, pero se ilustran y describen al menos con respecto a la **Figura 1** y las Figuras 9A-B), instrucción de conversión de formato 261, que incluye campos para especificar el código de operación 262 (VCVTNE2PS2BF16) y ubicaciones del primer y segundo vectores de origen 266 y 268 que comprende N elementos de precisión sencilla, y el vector de destino 264 que comprende al menos N elementos de coma flotante de 16 bits (por ejemplo, bfloat16 o binario16). En este punto, N es igual a 4 y el vector de destino especificado 264 incluye 8 elementos.

En este punto, el vector de destino tiene la mitad de la anchura que los vectores de origen, pero el equilibrio se logra asignando dos vectores de origen cuyos elementos han de convertirse y escribirse en el destino. En operación, las entradas convertidas desde el primer origen especificado 272A se escriben en las primeras cuatro entradas del destino especificado 278, y las entradas convertidas desde el segundo origen especificado 272B se escriben en las últimas cuatro entradas del destino especificado 278.

En algunas realizaciones, la instrucción de conversión de formato 261 también incluye una máscara {k} 268 y un control de puesta a cero {z} 270. El formato de la instrucción de conversión de formato 261, con código de operación de VCVTNEPS2BF16, se ilustra y describe además al menos con respecto a las **Figuras 5A**, 6A-B, y 7A-D.

También se muestran el primer y segundo vectores de origen especificados 272A-B, la circuitería de ejecución 274, que incluye la circuitería de conversión 276A-H, y el vector de destino especificado 278.

En operación, el aparato informático 260 (por ejemplo, un procesador) es para recuperar y decodificar, usando circuitería de recuperación y decodificación (no mostradas), teniendo la instrucción 261 campos para especificar el código de operación 262 (es decir, VCVTNE2PS2BF16) y las ubicaciones del primer y segundo vectores de origen 266 y 268 y destino 264, el código de operación para indicar que el aparato informático 260 (por ejemplo, procesador) es para convertir, usando los convertidores 276A-H en la circuitería de ejecución 274, cada uno de los elementos del primer y segundo vectores de origen especificados 272A-B a formato de coma flotante de 16 bits (por ejemplo, bfloat16), la circuitería de convertidor 276A-H incluye truncamiento y redondeo, según sea necesario, y para almacenar cada elemento convertido en una ubicación correspondiente del vector de destino especificado 278. En este punto, los primeros cuatro elementos del destino especificado 278 corresponden a la primera fuente especificada 272A, y los últimos cuatro elementos del destino especificado 278 corresponden a la segunda fuente especificada 272B. Como se ilustra y describe además al menos con respecto a las **Figuras 5A**, 6A-B y 7A-D, la instrucción 261 en otras realizaciones puede especificar diferentes longitudes de vector, tales como 128 bits, 512 bits o 1024 bits. La circuitería de ejecución 274 en este punto es para responder a la instrucción decodificada como se especifica por el código de operación 262.

La **Figura 3A** es un pseudocódigo que ilustra la ejecución ilustrativa de una instrucción de conversión de formato (VCVTNEPS2BF16), de acuerdo con una realización. Como se muestra, la instrucción de conversión de formato 301 tiene campos para especificar el código de operación 302 (VCVTNEPS2BF16) y ubicaciones de los primeros vectores de origen 306 (src) y destino 304 (dest), que, de acuerdo con VL constante, que se instancia en el código y se mantiene para "longitud del vector", puede ser cualquiera de 128 bits, 256 bits y 512 bits. En algunas realizaciones, la instrucción 301 tiene además campos para especificar una máscara 308 y un control de puesta a cero 310. El pseudocódigo 315 también muestra el uso de una máscara de escritura para controlar si se debe enmascarar cada uno de los elementos de destino, poniéndose a cero o fusionándose los elementos enmascarados (como se ilustra y describe más detalladamente al menos con respecto a **Figuras 5A**, 6A-B y 7A-D, la instrucción de conversión de formato en algunas realizaciones incluye campos para especificar la máscara y controlar si se pone a cero o se fusiona). La ejecución de la instrucción de conversión de formato 301 se ilustra y describe además al menos con respecto a las Figuras 2A-C, 4A y 9A-B.

La Figura 3B es un pseudocódigo que ilustra la ejecución ilustrativa de una instrucción de conversión de formato de 2 entradas (VCVTNE2PS2BF16), de acuerdo con una realización. Como se muestra, la instrucción de conversión de formato 321 tiene campos para especificar el código de operación 322 (VCVTNE2PS2BF16) y ubicaciones del primer vector de origen 326 (src1), segundo origen 328 (src2) y destino 324 (dest). El vector de destino, de acuerdo con VL constante, puede ser cualquiera de 128 bits, 256 bits y 512 bits. En este punto, las ubicaciones del vector de origen pueden estar en memoria o en registros. En algunas realizaciones, la instrucción de conversión de formato 321 tiene

campos para especificar una máscara de escritura {k} 330 y un control de puesta a cero {z} 331. El pseudocódigo 335 también muestra el uso de una máscara de escritura para controlar si se debe enmascarar cada uno de los elementos de destino, poniéndose a cero o fusionándose los elementos enmascarados (como se ilustra y describe más detalladamente al menos con respecto a **Figuras 5A**, 6A-B y 7A-D, la instrucción de conversión de formato en algunas realizaciones incluye campos para especificar la máscara y controlar si se pone a cero o se fusiona). La ejecución de la instrucción de conversión de formato 321 se ilustra y describe además al menos con respecto a las **Figuras 2D, 4B,** y 9A-B.

La **Figura 3C** es un pseudocódigo que ilustra una función auxiliar para usar con el pseudocódigo de las Figuras 3A-B, de acuerdo con una realización. En este punto, el pseudocódigo 354 define una función auxiliar, convert_fp32_to_bfloat16(), que convierte de un formato binario32 a un formato bfloat16.

El pseudocódigo 340 ilustra que las realizaciones divulgadas, en contraste con una conversión sencilla que simplemente truncaría los dieciséis bits inferiores del número binario32, realizan ventajosamente el redondeo de números normales y consideran un sesgo de redondeo. El código ilustra que la instrucción de conversión de formato tiene un comportamiento de redondeo mejorado en comparación con solamente truncamiento. El comportamiento de redondeo de las realizaciones divulgadas facilita un cálculo más preciso que la conversión por truncamiento. En algunas realizaciones, la circuitería de ejecución se adhiere al comportamiento de redondeo de acuerdo con las reglas de redondeo promulgadas como IEEE 754, por ejemplo, "NE" que indica redondeo al par más cercano. En algunas realizaciones, el comportamiento de redondeo se especifica por la instrucción, por ejemplo, incluyendo un sufijo, "NE", en el código de operación para indicar el redondeo al par más cercano. En otras realizaciones, el comportamiento de redondeo adopta un comportamiento predeterminado, como "NE". En otras realizaciones más, el comportamiento de redondeo se controla por un registro específico de modelo arquitectónico (MSR) que se configura mediante software.

El pseudocódigo 340 también ilustra que las realizaciones divulgadas realizan truncamiento cuando es necesario, por ejemplo, si la entrada a la función no es un número (nan).

La ejecución de la instrucción de conversión de formato se ilustra y describe además al menos con respecto a las Figuras 2A-D, 3A-B, 4A-B y 9A-B.

La **Figura 4A** es un diagrama de flujo de proceso que ilustra un procesador que responde a una instrucción de conversión de formato (VCVTNEPS2BF16), de acuerdo con una realización. La instrucción de conversión de formato 401 incluye campos para especificar el código de operación 402 (VCVTNEPS2BF16) y ubicaciones del primer vector de origen 406 que comprende N elementos de precisión sencilla, y el vector de destino 404 que comprende al menos N elementos de coma flotante de 16 bits (por ejemplo, bfloat16 o binario16).

Como se muestra, el procesador es para responder a una instrucción de conversión de formato decodificada realizando el flujo 400. En 421, el procesador ha de recuperar, usando circuitería de recuperación, una instrucción que tiene campos para especificar un código de operación (por ejemplo, VCVTNEPS2BF16) y ubicaciones de un primer vector de origen que comprende N elementos de precisión sencilla, y un vector de destino que comprende al menos N elementos de coma flotante de 16 bits, el código de operación para indicar que la circuitería de ejecución es para convertir cada uno de los elementos del vector de origen especificado a formato de coma flotante de 16 bits, la conversión para incluir truncamiento y redondeo, según sea necesario, y almacenar cada elemento convertido en una ubicación correspondiente del vector de destino especificado. En 423, el procesador ha de decodificar, usando circuitería de decodificación, la instrucción recuperada. En algunas realizaciones, el procesador en 425 ha de planificar la ejecución de la instrucción decodificada. En 427, el procesador ha de responder, usando circuitería de ejecución, a la instrucción decodificada como se especifica por el código de operación. En algunas realizaciones, el procesador en 429 ha de confirmar un resultado de la instrucción ejecutada. Las operaciones 425 y 429 son opcionales, según se indica por sus bordes discontinuos, en la medida en que pueden ocurrir en un tiempo diferente o no ocurrir en absoluto.

La **Figura 4B** es un diagrama de flujo de proceso que ilustra un procesador que responde a una instrucción de conversión de formato de 2 entradas (VCVTNE2PS2BF16), de acuerdo con una realización. La instrucción de conversión de formato 451 incluye campos para especificar el código de operación 452 (VCVTNE2PS2BF16) y ubicaciones del primer y segundo vectores de origen 456 y 462 que comprende N elementos de precisión sencilla, y el vector de destino 454 que comprende al menos N elementos de coma flotante de 16 bits (por ejemplo, bfloat16 o binario16).

Obsérvese que, la invención no pretende limitarse a ninguna nemotecnia particular para el código de operación. En este punto, se elige VCVTNEPS2BF16 como nemotecnia con letras que representan diversas características de instrucción. "VCVT", por ejemplo, se elige para indicar una conversión de vector. "NE", por otro lado, se elige para representar un modo redondo; en este punto, se selecciona el par más cercano, según lo promulgado por IEEE 754. "2PS" representa 2 paquetes individuales. "2" representa "para". Finalmente, "BF16" representa bfloat16.

Como se muestra, el procesador es para responder a una instrucción de conversión de formato decodificada realizando el flujo 450. En 471, el procesador ha de recuperar, usando circuitería de recuperación, una instrucción que tiene campos para especificar un código de operación (por ejemplo, VCVTNE2PS2BF16) y ubicaciones del primer y

segundo vectores de origen que comprenden N elementos de precisión sencilla, y un vector de destino que comprende al menos N elementos de coma flotante de 16 bits, el código de operación para indicar que la circuitería de ejecución es para convertir cada uno de los elementos del primer y segundo vectores de origen especificados a formato de coma flotante de 16 bits, la conversión para incluir truncamiento y redondeo, según sea necesario, y almacenar cada elemento convertido en una ubicación correspondiente del vector de destino especificado. En 473, el procesador ha de decodificar, usando circuitería de decodificación, la instrucción recuperada. En algunas realizaciones, el procesador en 475 ha de planificar la ejecución de la instrucción decodificada. En 477, el procesador ha de responder, usando circuitería de ejecución, a la instrucción decodificada como se especifica por el código de operación. En algunas realizaciones, el procesador en 479 ha de confirmar un resultado de la instrucción ejecutada. Las operaciones 475 y 479 son opcionales, según se indica por sus bordes discontinuos, en la medida en que pueden ocurrir en un tiempo diferente o no ocurrir en absoluto.

La **Figura 5A** es un diagrama de bloques que ilustra una instrucción de conversión de formato (VCVTNEPS2BF16), de acuerdo con una realización. Como se muestra, la instrucción de conversión de formato 500 incluye campos para especificar un código de operación 502 (VCVTNEPS2BF16) y ubicaciones de los vectores de destino 504 y del primer origen 506. Los vectores de origen y de destino pueden ubicarse cada uno en registros o en memoria.

El código de operación 502 se muestra que incluye un asterisco, lo que significa que se pueden añadir diversos campos opcionales como prefijos o sufijos al código de operación. Es decir, la instrucción de conversión de formato 500 incluye además parámetros opcionales para afectar el comportamiento de la instrucción, incluyendo la máscara {k} 508, el control de puesta a cero {z} 510, el formato de elemento 514, el tamaño del vector (N) 516 y el modo de redondeo 518. Uno o más de los modificadores de instrucción 508, 510, 514, 516 y 518 pueden especificarse usando prefijos o sufijos para el código de operación 502.

En algunas realizaciones, uno o más de los modificadores de instrucciones opcionales 508, 510, 514, 516 y 518 están codificados en un campo inmediato (no mostrado) incluido opcionalmente con la instrucción 500. En algunas realizaciones, uno o más de los modificadores de instrucciones opcionales 508, 510, 514, 516 y 518 se especifican a través de un registro de configuración, tal como registros específicos de modelo (MSR) incluidos en la arquitectura de conjunto de instrucciones.

El formato de instrucción de conversión de formato 500 se ilustra y describe además, al menos con respecto a **Figuras 5B**, 6A-B y 7A-D.

La **Figura 5B** es un diagrama de bloques que ilustra una instrucción de conversión de formato de 2 entradas (VCVTNE2PS2BF16), de acuerdo con una realización. Como se muestra, la instrucción de conversión de formato 550 incluye campos para especificar un código de operación 552 (VCVTNE2PS2BF16) y ubicaciones de los vectores de destino 554, primer origen 556 y segundo origen 562. Los vectores de origen y de destino pueden ubicarse cada uno en registros o en memoria.

El código de operación 552 se muestra que incluye un asterisco, lo que significa que se pueden añadir diversos campos opcionales como prefijos o sufijos al código de operación. Es decir, la instrucción de conversión de formato 550 incluye además parámetros opcionales para afectar el comportamiento de la instrucción, incluyendo la máscara {k} 558, el control de puesta a cero {z} 560, el formato de elemento 564, el tamaño del vector (N) 566 y el modo de redondeo 568. Uno o más de los modificadores de instrucción 558, 560, 564 y 566 pueden especificarse usando prefijos o sufijos para el código de operación 552.

En algunas realizaciones, uno o más de los modificadores de instrucciones opcionales 558, 560, 564, 566 y 568 están codificados en un campo inmediato (no mostrado) incluido opcionalmente con la instrucción 550. En algunas realizaciones, uno o más de los modificadores de instrucciones opcionales 558, 560, 564, 566 y 568 se especifican a través de un registro de configuración, tal como registros específicos de modelo (MSR) incluidos en la arquitectura de conjunto de instrucciones.

El formato de instrucción de conversión de formato 550 se ilustra y describe, además, al menos con respecto a **Figuras 5A**, 6A-B y 7A-D.

CONJUNTOS DE INSTRUCCIONES

Un conjunto de instrucciones puede incluir uno o más formatos de instrucciones. Un formato de instrucción dado puede definir diversos campos (por ejemplo, número de bits, ubicación de los bits) para especificar, entre otras cosas, la operación a realizar (por ejemplo, código de operación) y los operandos en los que se realizará esa operación y/u otro campo o campos de datos (por ejemplo, máscara). Algunos formatos de instrucción se desglosan aún más mediante la definición de plantillas de instrucción (o subformatos). Por ejemplo, las plantillas de instrucción de un formato de instrucción dado pueden definirse para tener diferentes subconjuntos de los campos del formato de instrucción (los campos incluidos típicamente están en el mismo orden, pero al menos algunos tienen posiciones de bits diferentes porque hay menos campos incluidos) y/o definirse para tener un campo dado interpretado de manera diferente. Por lo tanto, cada instrucción de una ISA se expresa utilizando un formato de instrucción dado (y, si se define, en una de las

plantillas de instrucción de ese formato de instrucción) e incluye campos para especificar la operación y los operandos. Por ejemplo, una instrucción ADD de ejemplo tiene un código de operación específico y un formato de instrucción que incluye un campo de código de operación para especificar ese código de operación y campos de operando para seleccionar operandos (fuente1/destino y fuente2); y una aparición de esta instrucción ADD en una secuencia de instrucciones tendrá contenidos específicos en los campos de operandos que seleccionan operandos específicos. Se ha lanzado y/o publicado un conjunto de extensiones de SIMD denominadas Extensiones Vectoriales Avanzadas (AVX) (AVX1 y AVX2) y que usan el esquema de codificación de Extensiones Vectoriales (VEX) (por ejemplo, véase Manual del desarrollador de Software de Arquitecturas Intel® 64 e IA-32, septiembre de 2014; y véase la referencia de programación de extensiones vectoriales avanzadas Intel®, octubre de 2014).

FORMATOS DE INSTRUCCIONES ILUSTRATIVOS

Las realizaciones de las instrucciones descritas en el presente documento pueden realizarse en diferentes formatos. Además, a continuación, se detallan sistemas, arquitecturas y canales ilustrativos. Se pueden ejecutar realizaciones de las instrucciones en dichos sistemas, arquitecturas y canales, pero no se limitan a las detalladas.

FORMATO DE INSTRUCCIONES COMPATIBLES CON VECTORES GENÉRICAS

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

Las **Figuras 6A-6B** son diagramas de bloques que ilustran un formato de instrucción compatible con vectores genérico y las plantillas de instrucción del mismo de acuerdo con algunas realizaciones de la invención. La **Figura 6A** es un diagrama de bloques que ilustra un formato de instrucción compatible con vectores genérico y las plantillas de instrucción de clase A del mismo de acuerdo con algunas realizaciones de la invención; mientras que la Figura 6B es un diagrama de bloques que ilustra el formato de instrucción compatible con vectores genérico y las plantillas de instrucción de clase B del mismo de acuerdo con algunas realizaciones de la invención. Específicamente, un formato de instrucción compatible con vectores genérico 600 para el que se definen plantillas de instrucción de clase A y clase B, ambas de las cuales incluyen las plantillas de instrucción sin acceso a memoria 605 y las plantillas de instrucción con acceso a memoria 620. El término genérico en el contexto del formato de instrucción compatible con vectores se refiere al formato de instrucción que no está vinculado a ningún conjunto de instrucciones específico.

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

Las plantillas de instrucción de clase A en la **Figura 6A** incluyen: 1) dentro de las plantillas de instrucción sin acceso a memoria 605 se muestra una plantilla de instrucción de operación de tipo de control de redondeo completo sin acceso a memoria 610 y una plantilla de instrucción de operación de tipo de transformación de datos sin acceso a memoria 615; y 2) dentro de las plantillas de instrucción con acceso a memoria 620 se muestra una plantilla de instrucción temporal con acceso a memoria 625 y una plantilla de instrucción no temporal con acceso a memoria 630. Las plantillas de instrucción de clase B en la **Figura 6B** incluyen: 1) dentro de las plantillas de instrucción sin acceso a memoria 605 se muestra una plantilla de instrucción de operación de tipo de control de redondeo parcial, control de máscara de escritura, sin acceso a memoria 612 y una plantilla de instrucción de operación de tipo vsize de control de máscara de escritura sin acceso a memoria 617; y 2) dentro de las plantillas de instrucción con acceso a memoria 620 se muestra una plantilla de instrucción de control de máscara de escritura con acceso a memoria 627.

El formato de instrucción compatible con vectores genérico 600 incluye los siguientes campos enumerados a continuación en el orden ilustrado en las **Figuras 6A-6B**.

Campo de formato 640 - un valor específico (un valor de identificador de formato de instrucción) en este campo identifica de forma única el formato de instrucción compatible con vectores y, por lo tanto, las ocurrencias de instrucciones en el formato de instrucción compatible con vectores en los flujos de instrucciones. Como tal, este campo es opcional en el sentido de que no es necesario para un conjunto de instrucciones que solo tiene el formato de instrucción compatible con vectores genérico.

Campo de operación de base 642 - su contenido distingue diferentes operaciones de base.

Campo de índice de registro 644 - su contenido, directamente o a través de la generación de direcciones, especifica las ubicaciones de los operandos de origen y destino, ya sea en los registros o en la memoria. Estos incluyen un número suficiente de bits para seleccionar N registros de un archivo de registros PxQ (por ejemplo, 32x512, 16x128, 32x1024, 64x1024). Mientras que en una realización N puede ser hasta tres registros de origen y uno de destino, realizaciones alternativas pueden soportar más o menos registros de origen y de destino (por ejemplo, pueden soportar hasta dos orígenes donde uno de estos orígenes también actúa como el destino, pueden soportar hasta tres orígenes donde uno de estos orígenes también actúa como el destino, puede soportar hasta dos orígenes y un destino).

Campo de modificador 646 - su contenido distingue las ocurrencias de instrucciones en el formato de instrucción de vector genérico que especifican el acceso a memoria de aquellas que no lo hacen; es decir, entre plantillas de instrucción sin acceso a memoria 605 y plantillas de instrucción con acceso a memoria 620. Las operaciones con acceso a memoria leen y/o escriben en la jerarquía de la memoria (en algunos casos, especificando las direcciones de origen y/o destino usando valores en los registros), mientras que las operaciones sin acceso a memoria no lo hacen (por ejemplo, el origen y los destinos son registros). Mientras que en una realización este campo también selecciona entre tres formas diferentes de realizar cálculos de direcciones de memoria, las realizaciones alternativas pueden soportar más, menos o diferentes formas de realizar cálculos de direcciones de memoria.

Campo de operación de aumento 650 - su contenido distingue cuál de una diversidad de operaciones diferentes se realizará además de la operación de base. Este campo es específico del contexto. En algunas realizaciones, este campo se divide en un campo de clase 668, un campo alfa 652 y un campo beta 654. El campo de operación de aumento 650 permite realizar grupos comunes de operaciones en una sola instrucción en lugar de 2, 3 o 4 instrucciones.

Campo de escala 660 - su contenido permite escalar el contenido del campo de índice para la generación de direcciones de memoria (por ejemplo, para la generación de direcciones que utiliza 2^{escala} * índice + base).

Campo de desplazamiento 662A - su contenido se usa como parte de la generación de direcciones de memoria (por ejemplo, para la generación de direcciones que usa 2^{escala} * índice + base + desplazamiento).

Campo de factor de desplazamiento 662B (obsérvese que la yuxtaposición del campo de desplazamiento 662A directamente sobre el campo de factor de desplazamiento 662B indica que se usa uno u otro) - su contenido se usa como parte de la generación de direcciones; especifica un factor de desplazamiento a escalar de acuerdo con el tamaño de un acceso a memoria (N), donde N es el número de bytes en el acceso a memoria (por ejemplo, para la generación de direcciones que utiliza 2^{escala} * índice + base + desplazamiento escalado). Los bits redundantes de bajo orden se ignoran y, por lo tanto, el contenido del campo del factor de desplazamiento se multiplica por el tamaño total de los operandos de memoria (N) para generar el desplazamiento final que se usará para calcular una dirección efectiva. El valor de N se determina por el hardware del procesador en tiempo de ejecución basándose en el campo de código de operación completo 674 (descrito más adelante en el presente documento) y el campo de manipulación de datos 654C. El campo de desplazamiento 662A y el campo de factor de desplazamiento 662B son opcionales en el sentido de que no se usan para las plantillas de instrucción sin acceso a memoria 605 y/o diferentes realizaciones pueden implementar únicamente uno o ninguno de los dos.

Campo de anchura de elemento de datos 664 - su contenido distingue cuál de un número de anchuras de elementos de datos va a usarse (en algunas realizaciones, para todas las instrucciones; en otras realizaciones, solo para algunas de las instrucciones). Este campo es opcional en el sentido de que no es necesario si únicamente se soporta una anchura de elemento de datos y/o se soportan anchuras de elementos de datos usando algún aspecto de los códigos de operación.

Campo de máscara de escritura 670 - su contenido controla, basándose en la posición del elemento de datos, si esa posición del elemento de datos en el operando de vector de destino refleja el resultado de la operación de base y la operación de aumento. Las plantillas de instrucción de clase A soportan la fusión de máscaras de escritura, mientras que las plantillas de instrucción de clase B soportan tanto la fusión como la puesta a cero de máscaras de escritura. Cuando se fusionan, las máscaras vectoriales permiten proteger de actualizaciones cualquier conjunto de elementos en el destino durante la ejecución de cualquier operación (especificada por la operación de base y la operación de aumento); en otra realización, conservando el valor antiguo de cada elemento del destino donde el bit de máscara correspondiente tiene un 0. Por el contrario, cuando las máscaras vectoriales de puesta a cero permiten poner a cero cualquier conjunto de elementos en el destino durante la ejecución de cualquier operación (especificada por la operación de base y la operación de aumento); en una realización, un elemento del destino se establece a 0 cuando el bit de máscara correspondiente tiene un valor 0. Un subconjunto de esta funcionalidad es la capacidad de controlar la longitud del vector de la operación que se realiza (es decir, el intervalo de elementos que se modifican, desde el primero hasta el último); sin embargo, no es necesario que los elementos que se modifican sean consecutivos. Por tanto, el campo de máscara de escritura 670 permite operaciones vectoriales parciales, incluyendo cargas, almacenamientos, aritméticas, lógicas, etc. Si bien se describen realizaciones de la invención en las que el contenido

del campo de máscara de escritura 670 selecciona uno de un número de registros de máscara de escritura que contiene la máscara de escritura que se va a utilizar (y por lo tanto el contenido del campo de máscara de escritura 670 identifica indirectamente ese enmascaramiento que se va a realizar), realizaciones alternativas en lugar o adicionalmente permiten que el contenido del campo de escritura de máscara 670 especifique directamente el enmascaramiento a realizar.

Campo inmediato 672 - su contenido permite la especificación de un inmediato. Este campo es opcional en el sentido de que no está presente en una implementación del formato compatible con vectores genéricos que no soporta inmediato y no está presente en instrucciones que no usan un inmediato.

Campo de clase 668 - su contenido distingue entre diferentes clases de instrucciones. Con referencia a las Figuras 6A-B, el contenido de este campo selecciona entre instrucciones de clase A y clase B. En las Figuras 6A-B, se utilizan cuadrados de esquinas redondeadas para indicar que un valor específico está presente en un campo (por ejemplo, clase A 668A y clase B 668B para el campo de clase 668 respectivamente en las Figuras 6A-B).

PLANTILLAS DE INSTRUCCIONES DE CLASE A

En el caso de las plantillas de instrucción sin acceso a memoria de clase A 605, el campo alfa 652 se interpreta como un campo de RS 652A, cuyo contenido distingue cuál de los diferentes tipos de operación de aumento va a realizarse (por ejemplo, la redondeo 652A.1 y la transformación de datos 652A.2 se especifican respectivamente para las plantillas de instrucción de operación de tipo de redondeo sin acceso a memoria 610 y operación de tipo de transformación de datos sin acceso a memoria 615), mientras que el campo beta 654 distingue cuál de las operaciones del tipo especificado va a realizarse. En las plantillas de instrucción sin acceso a memoria 605, el campo de escala 660, el campo de desplazamiento 662A y el campo de escala de desplazamiento 662B no están presentes.

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

En la plantilla de instrucción de operación de tipo de control de redondeo completo sin acceso a memoria 610, el campo beta 654 se interpreta como un campo de control de redondeo 654A, cuyo contenido o contenidos proporcionan redondeo estático. Mientras que, en las realizaciones descritas de la invención, el campo de control de redondeo 654A incluye un campo de supresión de todas las excepciones de coma flotante (SAE) 656 y un campo de control de operación de redondeo 658, las realizaciones alternativas pueden soportar la codificación de ambos conceptos en el mismo campo o solo tener uno o el otro de estos conceptos/campos (por ejemplo, pueden tener solo el campo de control de operación de redondeo 658).

Campo de SAE 656 - su contenido distingue si se debe o no deshabilitar el informe de eventos de excepción; cuando el contenido del campo de SAE 656 indica que la supresión está habilitada, una instrucción dada no informa ningún tipo de bandera de excepción de coma flotante y no genera ningún manejador de excepción de coma flotante.

Campo de control de operación de redondeo 658 - 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 658 permite el cambio del modo de redondeo por instrucción. En algunas realizaciones donde un procesador incluye un registro de control para especificar modos de redondeo, el contenido del campo de control de operación de redondeo 650 anula ese valor de registro.

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

En la plantilla de instrucción de operación de tipo de transformación de datos sin acceso a memoria 615, el campo beta 654 se interpreta como un campo de transformación de datos 654B, cuyo contenido distingue cuál de un número de transformaciones de datos se va a realizar (por ejemplo, sin transformación de datos, mezcla, difusión).

En el caso de una plantilla de instrucción de acceso a memoria 620 de clase A, el campo alfa 652 se interpreta como un campo de sugerencia de desalojo 652B, cuyo contenido distingue cuál de las sugerencias de desalojo se va a usar (en la **Figura 6A**, temporal 652B.1 y no temporal 652B.2 se especifican respectivamente para la plantilla de instrucción de acceso a memoria, temporal 625 y la plantilla de instrucción de acceso a memoria, no temporal 630), mientras que el campo beta 654 se interpreta como un campo de manipulación de datos 654C, cuyo contenido distingue cuál de un número de operaciones de manipulación de datos (también conocidas como primitivas) se va a realizar (por ejemplo, sin manipulación; difusión; conversión ascendente de un origen y conversión descendente de un destino). Las plantillas de instrucción con acceso a memoria 620 incluyen el campo de escala 660 y, opcionalmente, el campo de desplazamiento 662A o el campo de escala de desplazamiento 662B.

Las instrucciones de memoria vectoriales realizan cargas y almacenes de vectores en la memoria, con soporte de conversión. Al igual que con las instrucciones vectoriales normales, las instrucciones de memoria vectorial transfieren

datos desde/hacia la memoria en forma de elementos de datos, y los elementos que realmente se transfieren están dictados por el contenido de la máscara vectorial que se selecciona como máscara de escritura.

PLANTILLAS DE INSTRUCCIONES DE ACCESO A MEMORIA - TEMPORAL

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

PLANTILLAS DE INSTRUCCIONES DE ACCESO A MEMORIA - NO TEMPORAL

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

PLANTILLAS DE INSTRUCCIONES DE CLASE B

En el caso de las plantillas de instrucción de clase B, el campo alfa 652 se interpreta como un campo de control de máscara de escritura (Z) 652C, cuyo contenido distingue si el enmascaramiento de escritura controlado por el campo de máscara de escritura 670 debe ser una fusión o una puesta a cero.

En el caso de las plantillas de instrucción sin acceso a memoria 605 de clase B, parte del campo beta 654 se interpreta como un campo RL 657A, cuyo contenido distingue cuál de los diferentes tipos de operación de aumento se van a realizar (por ejemplo, redondeo 657A.1 y la longitud del vector (VSIZE) 657A.2 se especifican respectivamente para la plantilla de instrucción sin acceso a memoria, control de máscara de escritura, plantilla de instrucción de operación de tipo de control de redondeo parcial 612 y sin acceso a memoria, control de máscara de escritura, operación de tipo VSIZE 617), mientras que el resto del campo beta 654 distingue cuál de las operaciones del tipo especificado se va a realizar. En las plantillas de instrucción sin acceso a memoria 605, el campo de escala 660, el campo de desplazamiento 662A y el campo de escala de desplazamiento 662B no están presentes.

En la plantilla de instrucción de operación sin acceso a memoria, de control de máscara de escritura, tipo de control de redondeo parcial 610, el resto del campo beta 654 se interpreta como un campo de operación de redondeo 659A y el informe de eventos de excepción está deshabilitado (una instrucción dada no informa ninguna clase de bandera de excepción de coma flotante y no genera ningún manejador de excepción de coma flotante).

Campo de control de operación de redondeo 659A - al igual que el campo de control de operación de redondeo 658, 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 659A permite el cambio del modo de redondeo por instrucción. En algunas realizaciones donde un procesador incluye un registro de control para especificar modos de redondeo, el contenido del campo de control de operación de redondeo 650 anula ese valor de registro.

En la plantilla de instrucción de operación de tipo VSIZE 617, control de máscara de escritura, sin acceso a memoria, el resto del campo beta 654 se interpreta como un campo de longitud de vector 659B, cuyo contenido distingue cuál de varias longitudes de vector de datos se va a realizar en (por ejemplo, 128, 256 o 512 bytes).

En el caso de una plantilla de instrucción con acceso a memoria 620 de clase B, parte del campo beta 654 se interpreta como un campo de difusión 657B, cuyo contenido distingue si se va a realizar o no la operación de manipulación de datos de tipo difusión, mientras que el resto del campo beta 654 se interpreta como el campo de longitud vectorial 659B. Las plantillas de instrucción con acceso a memoria 620 incluyen el campo de escala 660 y, opcionalmente, el campo de desplazamiento 662A o el campo de escala de desplazamiento 662B.

Con respecto al formato de instrucción compatible con vectores genéricos 600, se muestra un campo de código de operación completo 674 que incluye el campo de formato 640, el campo de operación de base 642 y el campo de anchura de elemento de datos 664. Aunque se muestra una realización donde el campo de código de operación completo 674 incluye todos estos campos, el campo de código de operación completo 674 incluye menos de todos estos campos en realizaciones que no los soportan todos. El campo de código de operación completo 674 proporciona el código de operación (opcode).

El campo de operación de aumento 650, el campo de anchura de elemento de datos 664 y el campo de máscara de escritura 670 permiten que estas características se especifiquen en una base por instrucciones en el formato de instrucción compatible con vectores genérico.

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

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

FORMATO DE INSTRUCCIONES COMPATIBLE CON VECTORES ESPECÍFICO ILUSTRATIVO

La **Figura 7A** es un diagrama de bloques que ilustra un formato de instrucción compatible con vectores específico ilustrativo de acuerdo con algunas realizaciones de la invención. La **Figura 7A** muestra un formato de instrucción compatible con vectores específico 700 que es específico en el sentido de que especifica la ubicación, el tamaño, la interpretación y el orden de los campos, así como los valores para algunos de esos campos. El formato de instrucción compatible con vectores específicos 700 se puede usar para ampliar el conjunto de instrucciones x86 y, por lo tanto, algunos de los campos son similares o iguales a aquellos usados en el conjunto de instrucciones x86 existente y la extensión del mismo (por ejemplo, AVX). Este formato sigue siendo consistente con el campo de codificación de prefijo, el campo de bytes de código de operación real, el campo MOD R/M, el campo SIB, el campo de desplazamiento y los campos inmediatos del conjunto de instrucciones x86 existente con extensiones. Se ilustran los campos de la **Figura 6** a los que se mapean los campos de la **Figura 7A**.

Debe entenderse que, aunque las realizaciones de la invención se describen con referencia al formato de instrucción compatible con vectores específicos 700 en el contexto del formato de instrucción compatible con vectores genérico 600 con fines ilustrativos, la invención no se limita al formato de instrucción compatible con vectores específico. 700 excepto donde se reivindique. Por ejemplo, el formato de instrucción compatible con vectores genérico 600 contempla una diversidad de tamaños posibles para los diversos campos, mientras que el formato de instrucción compatible con vectores específicos 700 se muestra teniendo campos de tamaños específicos. A modo de ejemplo específico, mientras que el campo de anchura de elemento de datos 664 se ilustra como un campo de un bit en el formato de instrucción compatible con vectores específico 700, la invención no está así limitada (es decir, el formato de instrucción compatible con vectores genérico 600 contempla otros tamaños del campo de anchura de elemento de datos 664).

El formato de instrucción compatible con vectores genérico 600 incluye los siguientes campos enumerados a continuación en el orden ilustrado en la **Figura 7A.**

Prefijo EVEX (Bytes 0-3) 702: está codificado en formato de cuatro bytes.

Campo de formato 640 (EVEX Byte 0, bits [7:0]) - el primer byte (EVEX Byte 0) es el campo de formato 640 y contiene 0x62 (el valor único usado para distinguir el formato de instrucción compatible con vectores en algunas realizaciones).

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

El campo REX 705 (EVEX Byte 1, bits [7-5]) - consiste en un campo de bits EVEX.R (EVEX Byte 1, bit [7] - R), un campo de bits EVEX.X (EVEX byte 1, bit [6] - X), y 657BEX byte 1, bit[5] - B). Los campos de bits EVEX.R, EVEX.X y EVEX.B proporcionan la misma funcionalidad que los campos de bits VEX correspondientes y se codifican en forma de complemento a 1, es decir, ZMM0 se codifica como 1111B, ZMM15 se codifica como 0000B. Otros campos de las instrucciones codifican los tres bits inferiores de los índices de registro como es conocido en la técnica (rrr, xxx y bbb), de modo que Rrrr, Xxxx y Bbbb pueden formarse sumando EVEX.R, EVEX.X, y EVEX.B.

REX' 710A - esta es la primera parte del campo REX' 710 y es el campo de bits EVEX.R' (EVEX Byte 1, bit [4] - R') que se usa para codificar los 16 superiores o los 16 inferiores del conjunto de 32 registros extendido. En algunas

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

Campo de mapa de opcode 715 (EVEX byte 1, bits [3:0] - mmmm): su contenido codifica un byte de código de operación inicial implícito (0F, 0F 38 o 0F 3).

El campo de anchura de elemento de datos 664 (EVEX byte 2, bit [7] - W) - se representa mediante la notación EVEX.W. EVEX.W se usa para definir la granularidad (tamaño) del tipo de datos (ya sean elementos de datos de 32 bits o elementos de datos de 64 bits).

EVEX.vvvv 720 (EVEX Byte 2, bits [6:3]-vvvv)- la función de EVEX.vvvv puede incluir lo siguiente: 1) EVEX.vvvv codifica el primer operando de registro de origen, especificado en forma invertida (complemento a 1) y es válido para instrucciones con 2 o más operandos de origen; 2) EVEX.vvvv codifica el operando de registro de destino, especificado en forma de complemento a 1 para ciertos desplazamientos de vector; o 3) EVEX.vvvv no codifica ningún operando, el campo está reservado y debe contener 1111b. Por lo tanto, el campo 720 de EVEX.vvvv codifica los 4 bits de orden inferior del primer especificador de registro de origen almacenado en forma invertida (complemento a 1). Dependiendo de la instrucción, se usa un campo de bits EVEX adicional diferente para extender el tamaño del especificador a 32 registros.

EVEX.U 668 campo de Clase (EVEX byte 2, bit [2]-U) - Si EVEX.U = 0, indica clase A o EVEX.U0; si EVEX.U = 1 indica clase B o EVEX.U1.

Campo de codificación de prefijo 725 (EVEX byte 2, bits [1:0]-pp) - proporciona bits adicionales para el campo de operación de base. Además de proporcionar soporte para las instrucciones SSE heredadas en el formato de prefijo EVEX, esto también tiene la ventaja de compactar el prefijo de SIMD (en lugar de requerir un byte para expresar el prefijo de SIMD, el prefijo de EVEX requiere solo 2 bits). En una realización, para soportar instrucciones SSE heredadas que usan un prefijo de SIMD (66H, F2H, F3H) tanto en el formato heredado como en el formato de prefijo EVEX, estos prefijos de SIMD heredados se codifican en el campo de codificación de prefijo de SIMD; y en el tiempo de ejecución se expanden en el prefijo de SIMD heredado antes de proporcionarse al PLA del decodificador (para que el PLA pueda ejecutar tanto el formato heredado como el EVEX de estas instrucciones heredadas sin modificaciones). Aunque las instrucciones más nuevas podrían usar el contenido del campo de codificación del prefijo EVEX directamente como una extensión del código de operación, ciertas realizaciones se expanden de manera similar para mantener la consistencia, pero permiten que estos prefijos de SIMD heredados especifiquen diferentes significados. Una realización alternativa puede rediseñar el PLA para soportar las codificaciones de prefijo SIMD de 2 bits y, por lo tanto, no requerir la expansión.

Campo alfa 652 (EVEX byte 3, bit [7] - EH; también conocido como EVEX.EH, EVEX.rs, EVEX.RL, EVEX.control de máscara de escritura y EVEX.N; también ilustrado con α) - como se describió anteriormente, este campo es específico del contexto.

Campo beta 654 (EVEX byte 3, bits [6:4]-SSS, también conocido como EVEX.s_{2-0}, EVEX.r_{2-0}, EVEX.rr1, EVEX.LL0, EVEX.LLB; también ilustrado con βββ) - como se describió anteriormente, este campo es específico del contexto.

REX' 710B - este es el resto del campo REX' 710 y es el campo de bits EVEX.V' (EVEX Byte 3, bit [3] - V') que se puede usar para codificar los 16 superiores o los 16 inferiores del conjunto de 32 registros extendido. Este bit se almacena en formato de bits invertidos. Se usa un valor de 1 para codificar los 16 registros inferiores. En otras palabras, V'VVVV se forma combinando EVEX.V', EVEX.vvvv.

Campo de máscara de escritura 670 (EVEX byte 3, bits [2:0]-kkk) - su contenido especifica el índice de un registro en los registros de máscara de escritura como se describió anteriormente. En algunas realizaciones, el valor específico EVEX.kkk=000 tiene un comportamiento especial que implica que no se usa una máscara de escritura para la instrucción particular (esto se puede implementar de varias maneras, incluido el uso de una máscara de escritura conectada a todas las instrucciones o hardware que pasa por alto el hardware de enmascaramiento).

El campo de código de operación real 730 (byte 4) también se conoce como byte de código de operación. Parte del código de operación se especifica en este campo.

El campo MOD R/M 740 (byte 5) incluye el campo MOD 742, el campo Reg 744 y el campo R/M 746. Como se describió anteriormente, el contenido del campo MOD 742 distingue entre operaciones con acceso a memoria y operaciones sin acceso a memoria. La función del campo Reg 744 se puede resumir en dos situaciones: codificar el operando del registro de destino o un operando del registro de origen, o tratarse como una extensión de código de operación y no usarse para codificar un operando de instrucción. La función del campo R/M 746 puede incluir lo

siguiente: codificar el operando de instrucción que hace referencia a una dirección de memoria, o codificar el operando del registro de destino o un operando del registro de origen.

Byte de escala, índice, base (SIB) (Byte 6): como se describió anteriormente, el contenido el campo de escala 650 se utiliza para la generación de direcciones de memoria. SIB.xxx 754 y SIB.bbb 756 - el contenido de estos campos ha sido mencionado anteriormente con respecto a los índices de registro Xxxx y Bbbb.

Campo de desplazamiento 662A (Bytes 7-10) - cuando el campo MOD 742 contiene 10, los bytes 7-10 son el campo de desplazamiento 662A, y funciona igual que el desplazamiento de 32 bits heredado (disp32) y funciona con granularidad de byte.

Campo de factor de desplazamiento 662B (Byte 7) - cuando el campo MOD 742 contiene 01, el byte 7 es el campo de factor de desplazamiento 662B. La ubicación de este campo es la misma que la del desplazamiento de 8 bits (disp8) del conjunto de instrucciones x86 heredado, que funciona con granularidad de bytes. Dado que disp8 es un signo extendido, solo puede direccionar entre -128 y 127 compensaciones de bytes; en términos de líneas de caché de 64 bytes, disp8 usa 8 bits que se pueden establecer en solo cuatro valores realmente útiles: -128, -64, 0 y 64; dado que a menudo se necesita un rango mayor, se usa disp32; sin embargo, disp32 requiere 4 bytes. A diferencia de disp8 y disp32, el campo de factor de desplazamiento 662B es una reinterpretación de disp8; cuando se usa el campo de factor de desplazamiento 662B, el desplazamiento real se determina por el contenido del campo de factor de desplazamiento multiplicado por el tamaño del acceso de operando de memoria (N). Este tipo de desplazamiento se denomina disp8*N. Esto reduce la longitud de instrucción promedio (un solo byte de lo usado para el desplazamiento, pero con un rango mucho mayor). Tal desplazamiento comprimido se basa en la suposición de que el desplazamiento efectivo es un múltiplo de la granularidad del acceso a memoria y, por lo tanto, no es necesario codificar los bits de bajo orden redundantes de la compensación de dirección. En otras palabras, el campo de factor de desplazamiento 662B sustituye el desplazamiento de 8 bits del conjunto de instrucciones x86 heredado. Por lo tanto, el campo de factor de desplazamiento 662B se codifica de la misma manera que un desplazamiento de 8 bits del conjunto de instrucciones x86 (por lo que no hay cambios en las reglas de codificación ModRM/SIB) con la única excepción de que disp8 se sobrecarga a disp8*N. En otras palabras, no hay cambios en las reglas de codificación o longitudes de codificación, sino solo en la interpretación del valor de desplazamiento por el hardware (que necesita escalar el desplazamiento por el tamaño del operando de memoria para obtener una compensación de dirección por bytes). El campo inmediato 672 funciona como se ha descrito anteriormente.

CAMPO DE CÓDIGO DE OPCIÓN COMPLETO

La **Figura 7B** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico 700 que componen el campo de código de operación completo 674 de acuerdo con algunas realizaciones. Específicamente, el campo de código de operación completo 674 incluye el campo de formato 640, el campo de operación de base 642 y el campo de anchura de elemento de datos (W) 664. El campo de operación de base 642 incluye el campo de codificación de prefijo 725, el campo de mapa de código de operación 715 y el campo de código de operación real 730.

CAMPO DE ÍNDICE DE REGISTRO

La **Figura 7C** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico 700 que componen el campo de índice de registro 644 de acuerdo con algunas realizaciones. Específicamente, el campo de índice de registro 644 incluye el campo REX 705, el campo REX' 710, el campo MODR/M.reg 744, el campo MODR/M.r/m 746, el campo VVVV 720, el campo xxx 754 y el campo bbb 756.

CAMPO DE OPERACIÓN DE AUMENTO

La **Figura 7D** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico 700 que componen el campo de operación de aumento 650 de acuerdo con algunas realizaciones. Cuando el campo clase (U) 668 contiene 0, significa EVEX.U0 (clase A 668A); cuando contiene 1, significa EVEX.U1 (clase B 668B). Cuando U=0 y el campo MOD 742 contiene 11 (lo que significa una operación sin acceso a memoria), el campo alfa 652 (EVEX byte 3, bit [7] - EH) se interpreta como el campo de rs 652A. Cuando el campo de RS 652A contiene un 1 (redondeo 652A.1), el campo beta 654 (EVEX byte 3, bits [6:4]-SSS) se interpreta como el campo de control de redondeo 654A. El campo de control de redondeo 654A incluye un campo SAE 656 de un bit y un campo de operación de redondeo 658 de dos bits. Cuando el campo rs 652A contiene un 0 (transformación de datos 652A.2), el campo beta 654 (byte 3 EVEX, bits [6:4]- SSS) se interpreta como un campo de transformación de datos de tres bits 654B. Cuando U=0 y el campo MOD 742 contiene 00, 01 o 10 (lo que significa una operación de acceso a memoria), el campo alfa 652 (EVEX byte 3, bit [7] - EH) se interpreta como el campo de sugerencia de desalojo (EH) 652B y el campo beta 654 (EVEX byte 3, bits [6:4]-SSS) se interpreta como un campo de manipulación de datos de tres bits 654C.

Cuando U=1, el campo alfa 652 (EVEX byte 3, bit [7] - EH) se interpreta como el campo de control de máscara de escritura (Z) 652C. Cuando U=1 y el campo MOD 742 contiene 11 (lo que significa una operación sin acceso a

memoria), parte del campo beta 654 (EVEX byte 3, bit [4]- S_0) se interpreta como el campo de RL 657A; cuando contiene un 1 (redondeo 657A.1) el resto del campo beta 654 (EVEX byte 3, bit [6-5]- S_{2-1}) se interpreta como el campo de operación de redondeo 659A, mientras que cuando el campo de RL 657A contiene un 0 (VSIZE 657.A2) el resto del campo beta 654 (EVEX byte 3, bit [6-5]- S_{2-1}) se interpreta como el campo de longitud de vector 659B (EVEX byte 3, bit [6-5]- L_{1-0}). Cuando U=1 y el campo MOD 742 contiene 00, 01 o 10 (lo que significa una operación con acceso a memoria), el campo beta 654 (EVEX byte 3, bits [6:4]- SSS) se interpreta como el campo de longitud de vector 659B (EVEX byte 3, bit [6-5]- L_{1-0}) y el campo de difusión 657B (EVEX byte 3, bit [4]- B).

ARQUITECTURA DE REGISTRO ILUSTRATIVA

La **Figura 8** es un diagrama de bloques de una arquitectura de registro 800 de acuerdo con algunas realizaciones. En la realización ilustrada, hay 32 registros vectoriales 810 que tienen 512 bits de anchura; estos registros se referencian como zmm0 a zmm31. Los 256 bits de orden inferior de los 16 registros zmm inferiores se superponen en los registros ymm0-16. Los 128 bits de orden inferior de los 16 registros zmm inferiores (los 128 bits de orden inferior de los registros ymm) se superponen en los registros xmm0-15. El formato de instrucción compatible con vectores específico 700 opera en este archivo de registro superpuesto, 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 659B	A (Figura 6A; U=0)	610, 615, 625, 630	registros zmm (la longitud del vector es de 64 bytes)
	B (Figura 6B; U=1)	612	registros zmm (la longitud del vector es de 64 bytes)
Plantillas de instrucción que incluyen el campo de longitud de vector 659B	B (Figura 6B; U=1)	617, 627	registros zmm, ymm o xmm (la longitud del vector es de 64 bytes, 32 bytes o 16 bytes) dependiendo del campo de longitud de vector 659B

En otras palabras, el campo de longitud de vector 659B selecciona entre una longitud máxima y una o más longitudes más cortas, donde cada una de dichas longitudes más cortas es la mitad de la longitud de la longitud precedente; y las plantillas de instrucción sin el campo de longitud de vector 659B operan en la longitud de vector máxima. Además, en una realización, las plantillas de instrucción de clase B del formato de instrucción compatible con vectores específico 700 operan con datos de coma flotante de precisión sencilla/doble empaquetados o escalares y con datos enteros empaquetados o escalares. Las operaciones escalares son operaciones realizadas en la posición del elemento de datos de orden más bajo en un registro zmm/ymm/xmm; las posiciones de los elementos de datos de orden superior se dejan igual que antes de la instrucción o se ponen a cero dependiendo de la realización.

Registros de máscara de escritura 815 - en la realización ilustrada, hay 8 registros de máscara de escritura (k0 a k7), cada uno de 64 bits de tamaño. En una realización alternativa, los registros de máscara de escritura 815 tienen un tamaño de 16 bits. Como se ha descrito anteriormente, en algunas realizaciones, el registro de máscara vectorial k0 no se puede usar como una máscara de escritura; cuando se usa la codificación que normalmente indicaría k0 para una máscara de escritura, selecciona una máscara de escritura predeterminada de 0xffff, deshabilitando de manera efectiva el enmascaramiento de escritura para esa instrucción.

Registros de propósito general 825 - en la realización ilustrada, hay dieciséis registros de propósito general de 64 bits que se usan junto con los modos de direccionamiento x86 existentes para direccionar operandos de memoria. A estos registros se hace referencia con los nombres RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP y R8 a R15.

Archivo de registro de pila de coma flotante escalar (pila x87) 845, en el que tiene un alias el archivo de registro plano entero empaquetado MMX 850; en la realización ilustrada, la pila x87 es una pila de ocho elementos utilizada para realizar operaciones escalares de coma flotante en datos de coma flotante de 32/64/80 bits utilizando la extensión del conjunto de instrucciones x87; mientras que los registros MMX se utilizan para realizar operaciones con datos enteros empaquetados de 64 bits, así como para contener operandos para algunas operaciones realizadas entre los registros MMX y XMM.

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

ARQUITECTURAS DE NÚCLEO, PROCESADORES Y ARQUITECTURAS INFORMÁTICAS ILUSTRATIVAS

Los núcleos de procesador se pueden implementar de diferentes maneras, para diferentes propósitos y en diferentes procesadores. Por ejemplo, las implementaciones de tales núcleos pueden incluir: 1) un núcleo ordenado de propósito general destinado a la computación de propósito general; 2) un núcleo desordenado de propósito general de alto rendimiento destinado a computación de propósito general; 3) un núcleo de propósito especial destinado

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

ARQUITECTURAS DE NÚCLEO ILUSTRATIVAS

DIAGRAMA DE BLOQUES DEL NÚCLEO ORDENADO Y DESORDENADO

La **Figura 9A** es un diagrama de bloques que ilustra tanto una canalización ordenada ilustrativa como una canalización de emisión/ejecución en desorden de cambio de nombre de registro ilustrativo de acuerdo con algunas realizaciones de la invención. La **Figura 9B** es un diagrama de bloques que ilustra tanto una realización ilustrativa de un núcleo de arquitectura ordenado y un núcleo ilustrativo de arquitectura de emisión/ejecución desordenada y cambio de nombre de registro que se incluirá en un procesador de acuerdo con algunas realizaciones de la invención. Los cuadros con líneas continuas en las Figuras 9A-B ilustran la canalización ordenada y el núcleo ordenado, mientras que la adición opcional de los cuadros con líneas discontinuas ilustra el cambio de nombre del registro, la canalización y el núcleo de emisión/ejecución desordenada. Dado que el aspecto ordenado es un subconjunto del aspecto desordenado, se describirá el aspecto desordenado.

En la **Figura 9A,** una canalización de procesador 900 incluye una etapa de recuperación 902, una etapa de decodificación de longitud 904, una etapa de decodificación 906, una etapa de asignación 908, una etapa de cambio de nombre 910, una etapa de planificación (también conocida como despacho o emisión) 912, una etapa de lectura de registro/lectura de memoria 914, una etapa de ejecución 916, una etapa de reescritura/escritura de memoria 918, una etapa de manejo de excepciones 922 y una etapa de confirmación 924.

La **Figura 9B** muestra el núcleo de procesador 990 que incluye una unidad de extremo frontal 930 acoplada a una unidad de motor de ejecución 950, y ambas están acopladas a una unidad de memoria 970. El núcleo 990 puede ser un núcleo de computación de conjunto de instrucciones reducido (RISC), un conjunto de instrucciones complejo de computación (CISC), un núcleo de palabra de instrucción muy larga (VLIW) o un tipo de núcleo híbrido o alternativo. Como otra opción más, el núcleo 990 puede ser un núcleo de propósito especial, tal como, por ejemplo, un núcleo de red o comunicación, motor de compresión, núcleo de coprocesador, núcleo de unidad de procesamiento de gráficos informáticos de propósito general (GPGPU), núcleo de gráficos o s.

La unidad frontal 930 incluye una unidad de predicción de rama 932 acoplada a una unidad de caché de instrucciones 934, que está acoplada a una memoria intermedia de recuperación de traducción de instrucciones (TLB) 936, que está acoplada a una unidad de recuperación de instrucción 938, que está acoplada a una unidad de decodificación 940. La unidad de decodificación 940 (o decodificador) puede decodificar instrucciones y generar como salida una o más microoperaciones, puntos de entrada de microcódigo, microinstrucciones, otras instrucciones u otras señales de control, que se decodifican a partir de, o que de otro modo reflejan o se derivan de las instrucciones originales. La unidad de decodificación 940 puede implementarse usando varios mecanismos diferentes. Ejemplos de mecanismos adecuados incluyen, entre otros, tablas de consulta, implementaciones de hardware, matrices lógicas programables (PLA), memorias de solo lectura (ROM) de microcódigo, etc. En una realización, el núcleo 990 incluye una ROM de microcódigo u otros medio que almacena microcódigo para ciertas macroinstrucciones (por ejemplo, en la unidad de decodificación 940 o de otro modo dentro de la unidad frontal 930). La unidad de decodificación 940 está acoplada a una unidad de cambio de nombre/asignación 952 en la unidad de motor de ejecución 950.

La unidad de motor de ejecución 950 incluye la unidad de cambio de nombre/asignación 952 acoplada a una unidad de retiro 954 y un conjunto de una o más unidades de planificación 956. La unidad de planificación 956 representa cualquier número de planificadores diferentes, incluyendo estaciones de reserva, ventana de instrucción central, etc. La o las unidades de planificación 956 están acopladas a la o las unidades de archivos de registro físico 958. Cada una de las unidades de archivo o archivos de registro físico 958 representa uno o más archivos de registro físico, diferentes de los que almacenan uno o más tipos de datos diferentes, tales como entero escalar, coma flotante escalar, entero empaquetado, coma flotante empaquetado, entero vectorial, coma flotante vectorial, estado (por ejemplo, un puntero de instrucción que es la dirección de la siguiente instrucción a ejecutar), etc. En una realización, la unidad de archivo o archivos de registros físicos 958 comprende una unidad de registros vectoriales, una unidad de registros de máscara y una unidad de registros escalares. Estas unidades de registro pueden proporcionar registros vectoriales arquitectónicos, registros de máscara vectorial y registros de propósito general. La unidad o unidades de archivo o

archivos de registro físico 958 se superponen con la unidad de retiro 954 para ilustrar diversas formas en las que se puede implementar el cambio de nombre de registro y la ejecución desordenada (por ejemplo, usando una memoria o memorias intermedias de reorden y un archivo o archivos de registro de retiro; usando un archivos o archivos futuros, una memoria o memorias intermedias de historial y un archivo o archivos de registro de retiro; usando mapas de registros y una agrupación de registros; etc.). La unidad de retiro 954 y la o las unidades de archivos de registro físico 958 están acopladas al o a los grupos de ejecución 960. El o los grupos de ejecución 960 incluye un conjunto de una o más unidades de ejecución 962 y un conjunto de una o más unidades con acceso a memoria 964. Las unidades de ejecución 962 pueden realizar diversas operaciones (por ejemplo, desplazamientos, suma, resta, multiplicación) y en diversos tipos de datos (por ejemplo, coma flotante escalar, entero empaquetado, coma flotante empaquetado, entero vectorial, coma flotante vectorial). Si bien algunas realizaciones pueden incluir varias unidades de ejecución dedicadas a funciones o conjuntos de funciones específicas, otras realizaciones pueden incluir solo una unidad de ejecución o múltiples unidades de ejecución que realizan todas las funciones. La unidad o unidades de planificación 956, la unidad o unidades de archivo de registro físico 958 y la agrupación o agrupaciones de ejecución 960 se muestran como posiblemente plurales porque ciertas realizaciones crean canalizaciones separadas para ciertos tipos de datos/operaciones (por ejemplo, una canalización de enteros escalares, una canalización de coma flotante escalar/entero empaquetado/coma flotante empaquetado/entero vectorial/coma flotante vectorial, y/o una canalización de acceso a memoria que cada una tiene su propia unidad de planificación, unidad de archivo o archivos de registro físico, y/o agrupación de ejecución - y en el caso de una canalización de acceso a memoria separada, se implementan ciertas realizaciones en las que únicamente el grupo de ejecución de esta canalización tiene la unidad o unidades de acceso a memoria 964). También debe entenderse que cuando se utilizan canales separados, una o más de estas canales pueden estar desordenados en servicio/ejecución y el resto ordenados.

El conjunto de unidades con acceso a memoria 964 está acoplado a la unidad de memoria 970, que incluye una unidad TLB de datos 972 acoplada a una unidad de caché de datos 974 acoplada a una unidad de caché de nivel 2 (L2) 976. En una realización ilustrativa, las unidades con acceso a memoria 964 puede incluir una unidad de carga, una unidad de dirección de almacenamiento y una unidad de datos de almacenamiento, cada una de las cuales está acoplada a la unidad TLB de datos 972 en la unidad de memoria 970. La unidad de caché de instrucciones 934 está además acoplada a una unidad de caché de nivel 2 (L2) 976 en la unidad de memoria 970. La unidad de caché L2 976 está acoplada a uno o más niveles de caché y eventualmente a una memoria principal.

A modo de ejemplo, la arquitectura central ilustrativa de cambio de nombre de registro, emisión/ejecución desordenada puede implementar el canal 900 de la siguiente manera: 1) la recuperación de instrucciones 938 realiza las etapas de recuperación y decodificación de longitud 902 y 904; 2) la unidad de decodificación 940 realiza la etapa de decodificación 906; 3) la unidad de cambio de nombre/asignación 952 realiza la etapa de asignación 908 y la etapa de cambio de nombre 910; 4) la o las unidades de planificación 956 realizan la etapa de planificación 912; 5) la o las unidades de archivos de registro físico 958 y la unidad de memoria 970 realizan la etapa 914 de lectura de registro/lectura de memoria; el grupo de ejecución 960 realiza la etapa de ejecución 916; 6) la unidad de memoria 970 y la o las unidades de archivos de registro físico 958 realizan la etapa 918 de reescritura/escritura en memoria; 7) varias unidades pueden estar involucradas en la etapa de manejo de excepciones 922; y 8) la unidad de retiro 954 y la o las unidades de archivos de registro físico 958 realizan la etapa de confirmación 924.

El núcleo 990 puede soportar uno o más conjuntos de instrucciones (por ejemplo, el conjunto de instrucciones x86 (con algunas extensiones que se han agregado con versiones más recientes); el conjunto de instrucciones MIPS de MIPS Technologies de Sunnyvale, CA; el conjunto de instrucciones ARM (con extensiones opcionales adicionales tal como NEON) de ARM Holdings de Sunnyvale, CA), incluidas las instrucciones descritas en el presente documento. En una realización, el núcleo 990 incluye lógica para soportar una extensión del conjunto de instrucciones de datos empaquetados (por ejemplo, AVX1, AVX2), permitiendo de esta manera que las operaciones usadas por muchas aplicaciones multimedia se realicen usando datos empaquetados.

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

Si bien el cambio de nombre de registros se describe en el contexto de la ejecución desordenada, debe entenderse que el cambio de nombre de registros puede usarse en una arquitectura ordenada. Si bien la realización ilustrada del procesador también incluye unidades de caché de datos e instrucciones separadas 934/974 y una unidad de caché L2 compartida 976, realizaciones alternativas pueden tener una única caché interna tanto para instrucciones como para datos, tal como, por ejemplo, una caché interna de nivel 1 (L1), o múltiples niveles de caché interna. En algunas realizaciones, el sistema puede incluir una combinación de una memoria caché interna y una memoria caché externa que es externa al núcleo y/o al procesador. Alternativamente, toda la caché puede ser externa al núcleo y/o al procesador.

ARQUITECTURA DE NÚCLEO ORDENADO ILUSTRATIVA ESPECÍFICA

Las Figuras 10A-B ilustran un diagrama de bloques de una arquitectura de núcleo ordenado ilustrativa más específica, cuyo núcleo sería uno de varios bloques lógicos (que incluyen otros núcleos del mismo tipo y/o tipos diferentes) en un chip. Los bloques lógicos se comunican a través de una red de interconexión de gran ancho de banda (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, de acuerdo con la aplicación.

La **Figura 10A** es un diagrama de bloques de un único núcleo de procesador, junto con su conexión a la red de interconexión en chip 1002 y con su subconjunto local de la caché de nivel 2 (L2) 1004, de acuerdo con las realizaciones de la invención. En una realización, un decodificador de instrucciones 1000 soporta el conjunto de instrucciones x86 con una extensión del conjunto de instrucciones de datos empaquetados. Una caché L1 1006 permite accesos de baja latencia a la memoria caché en las unidades escalares y vectoriales. Mientras que en una realización (para simplificar el diseño), una unidad escalar 1008 y una unidad vectorial 1010 usan conjuntos de registros separados (respectivamente, registros escalares 1012 y registros vectoriales 1014) y los datos transferidos entre ellos se escriben en la memoria y luego se vuelven a leer desde una memoria caché de nivel 1 (L1) 1006, realizaciones alternativas de la invención pueden usar un enfoque diferente (por ejemplo, usar un único conjunto de registros o incluir una ruta de comunicaciones que permita que los datos se transfieran entre los dos archivos de registro sin tener que escribirse ni volverse a leer).

El subconjunto local de la caché L2 1004 es parte de una caché L2 global que está dividida en subconjuntos locales separados, uno por núcleo de procesador. Cada núcleo de procesador tiene una ruta de acceso directo a su propio subconjunto local de la caché L2 1004. Los datos leídos por un núcleo de procesador se almacenan en su subconjunto de caché L2 1004 y se puede acceder a ellos rápidamente, en paralelo con otros núcleos de procesador que acceden a sus propios subconjuntos de caché L2 locales. Los datos escritos por un núcleo de procesador se almacenan en su propio subconjunto de caché L2 1004 y se eliminan de otros subconjuntos, si es necesario. La red en anillo garantiza la coherencia de los datos compartidos. La red en anillo es bidireccional para permitir que agentes tales como núcleos de procesador, caché L2 y otros bloques lógicos se comuniquen entre sí dentro del chip. Cada ruta de datos del anillo tiene 1012 bits de anchura por dirección.

La **Figura 10B** es una vista ampliada de parte del núcleo del procesador en la Figura 10A de acuerdo con algunas realizaciones de la invención. La **Figura 10B** incluye una caché de datos L1 1006A parte de la caché L1 1004, así como más detalles con respecto a la unidad de vector 1010 y los registros vectoriales 1014. Específicamente, la unidad vectorial 1010 es una unidad de procesamiento vectorial (VPU) de 16 de ancho (véase la ALU 1028 de 16 de ancho), que ejecuta una o más instrucciones flotantes de precisión de enteros, sencilla y doble. La VPU soporta el mezclado de las entradas de registro con la unidad de mezcla 1020, la conversión numérica con las unidades de conversión numérica 1022A-B y la replicación con la unidad de replicación 1024 en la entrada de memoria. Los registros de máscara de escritura 1026 permiten predecir escrituras vectoriales resultantes.

La **Figura 11** es un diagrama de bloques de un procesador 1100 que puede tener más de un núcleo, puede tener un controlador de memoria integrado y puede tener gráficos integrados de acuerdo con algunas realizaciones de la invención. Los recuadros con líneas continuas en la **Figura 11** ilustran un procesador 1100 con un único núcleo 1102A, un agente de sistema 1110, un conjunto de una o más unidades de controlador de bus 1116, mientras que, la adición opcional de los recuadros con líneas discontinuas ilustra un procesador alternativo 1100 con múltiples núcleos 1102A-N, un conjunto de una o más unidad o unidades de controlador de memoria integrado 1114 en la unidad de agente de sistema 1110 y lógica de propósito especial 1108.

Por lo tanto, diferentes implementaciones del procesador 1100 pueden incluir: 1) una CPU con la lógica de propósito especial 1108 que es gráficos integrados y/o lógica científica (de rendimiento) (que puede incluir uno o más núcleos), y siendo los núcleos 1102A-N uno o más núcleos de propósito general (por ejemplo, núcleos ordenados de propósito general, núcleos desordenados de propósito general, una combinación de los dos); 2) un coprocesador con núcleos 1102A-N que son un gran número de núcleos de propósito especial destinados principalmente a gráficos y/o científicos (rendimiento); y 3) un coprocesador con los núcleos 1102A-N que son un gran número de núcleos ordenados de propósito general. Por lo tanto, el procesador 1100 puede ser un procesador de propósito general, coprocesador o procesador de propósito especial, tal como, por ejemplo, un procesador de red o comunicación, motor de compresión, procesador de gráficos, GPGPU (unidad de procesamiento de gráficos de propósito general), un coprocesador de alto rendimiento de muchos núcleos integrados (MIC) (que incluye 30 o más núcleos), procesador integrado o similar. El procesador puede implementarse en uno o más chips. El procesador 1100 puede ser parte y/o puede implementarse en uno o más sustratos usando cualquiera de un número de tecnologías de proceso, tales como, por ejemplo, BiCMOS, CMOS o NMOS.

La jerarquía de memoria incluye uno o más niveles de caché dentro de los núcleos, un conjunto o una o más unidades de caché compartidas 1106 y memoria externa (no mostrada) acoplada al conjunto de unidades de controlador de memoria integrado 1114. El conjunto de unidades de caché compartida 1106 puede incluir una o más caché de nivel medio, tales como de nivel 2 (L2), nivel 3 (L3), nivel 4 (L4) u otros niveles de caché, una caché de último nivel (LLC) y/o combinaciones de las mismas. Mientras que, en una realización, una unidad de interconexión basada en anillo 1112 interconecta la lógica de gráficos integrada 1108 (la lógica de gráficos integrada 1108 es un ejemplo y también

se denomina en el presente documento lógica de propósito especial), el conjunto de unidades de caché compartida 1106 y la unidad de agente del sistema 1110/unidad o unidades de controlador de memoria integrada 1114, 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 caché 1106 y núcleos 1102-A-N.

En algunas realizaciones, uno o más de los núcleos 1102A-N son capaces de realizar subprocesos múltiples. El agente de sistema 1110 incluye aquellos componentes que coordinan y operan los núcleos 1102A-N. La unidad de agente de sistema 1110 puede incluir, por ejemplo, una unidad de control de energía (PCU) y una unidad de visualización. La PCU puede ser o incluir la lógica y los componentes necesarios para regular el estado de energía de los núcleos 1102A-N y la lógica de gráficos integrados 1108. La unidad de visualización es para controlar una o más pantallas conectadas externamente.

Los núcleos 1102A-N pueden ser homogéneos o heterogéneos en términos de conjunto de instrucciones de arquitectura; es decir, dos o más de los núcleos 1102A-N pueden ser capaces de ejecutar el mismo conjunto de instrucciones, mientras que otros pueden ser capaces de ejecutar solo un subconjunto de ese conjunto de instrucciones o un conjunto de instrucciones diferente.

ARQUITECTURAS INFORMÁTICAS ILUSTRATIVAS

Las **Figuras 12-15** son diagramas de bloques de arquitecturas informáticas ilustrativas. Otros diseños y configuraciones de sistema conocidos en la técnica para portátiles, equipos de sobremesa, PC portátiles, asistentes digitales personales, estaciones de trabajo de ingeniería, servidores, dispositivos de red, concentradores de red, conmutadores, procesadores integrados, procesadores de señales digitales (DSP), dispositivos de gráficos, dispositivos de videojuegos, decodificadores de salón, microcontroladores, teléfonos móviles, reproductores multimedia portátiles, dispositivos de mano y diversos otros dispositivos electrónicos, también son adecuados. En general, son generalmente adecuados una gran diversidad de sistemas o dispositivos electrónicos que pueden incorporar un procesador y/u otra lógica de ejecución como se divulga en el presente documento.

Con referencia ahora a la **Figura 12**, se muestra un diagrama de bloques de un sistema 1200 de acuerdo con una realización de la presente invención. El sistema 1200 puede incluir uno o más procesadores 1210, 1215, que están acoplados a un concentrador de controlador 1220. En una realización, el concentrador DE controlador 1220 incluye un concentrador DE controlador de memoria gráfica (GMCH) 1290 y un concentrador de entrada/salida (IOH) 1250 (que puede estar en chips separados); el GMCH 1290 incluye controladores de memoria y gráficos a los que están acoplados la memoria 1240 y un coprocesador 1245; el IOH 1250 acopla los dispositivos de entrada/salida (E/S) 1260 al GMCH 1290. Alternativamente, uno o ambos controladores de memoria y gráficos están integrados dentro del procesador (como se describe en el presente documento), la memoria 1240 y el coprocesador 1245 están acoplado directamente al procesador 1210, y al concentrador de controlador 1220 en un único chip con el IOH 1250.

La naturaleza opcional de los procesadores adicionales 1215 se indica en la **Figura 12** con líneas discontinuas. Cada procesador 1210, 1215 puede incluir uno o más de los núcleos de procesamiento descritos en el presente documento y puede ser alguna versión del procesador 1100.

La memoria 1240 puede ser, por ejemplo, una memoria dinámica de acceso aleatorio (DRAM), una memoria de cambio de fase (PCM) o una combinación de ambas. Para al menos una realización, el concentrador de controlador 1220 se comunica con el o los procesadores 1210, 1215 a través de un bus multipunto, tal como un bus frontal (FSB), una interfaz punto a punto tal como QuickPath Interconnect (QPI), o conexión similar 1295.

En una realización, el coprocesador 1245 es un procesador de propósito especial, tal como, por ejemplo, un procesador MIC de alto rendimiento, un procesador de red o comunicación, motor de compresión, procesador de gráficos, GPGPU, procesador integrado o similar. En una realización, el concentrador de controlador 1220 puede incluir un acelerador de gráficos integrado.

Puede haber una diversidad de diferencias entre los recursos físicos 1210, 1215 en términos de un espectro de métricas de mérito que incluyen características arquitectónicas, microarquitectónicas, térmicas, de consumo de energía y similares.

En una realización, el procesador 1210 ejecuta instrucciones que controlan operaciones de procesamiento de datos de un tipo general. Incrustadas dentro de las instrucciones puede haber instrucciones de coprocesador. El procesador 1210 reconoce estas instrucciones de coprocesador como de un tipo que debería ser ejecutado por el coprocesador adjunto 1245. En consecuencia, el procesador 1210 emite estas instrucciones de coprocesador (o señales de control que representan instrucciones de coprocesador) en un bus del coprocesador u otra interconexión, al coprocesador 1245. Los coprocesadores 1245 aceptan y ejecutan las instrucciones de coprocesador recibidas.

Con referencia ahora a la **Figura 13**, se muestra un diagrama de bloques de un primer sistema 1300 ilustrativo más específico de acuerdo con una realización de la presente invención. Como se muestra en la **Figura 13**, el sistema multiprocesador 1300 es un sistema de interconexión punto a punto e incluye un primer procesador 1370 y un segundo

procesador 1380 acoplados a través de una interconexión punto a punto 1350. Cada uno de los procesadores 1370 y 1380 puede ser alguna versión del procesador 1100. En algunas realizaciones, los procesadores 1370 y 1380 son respectivamente procesadores 1210 y 1215, mientras que el coprocesador 1338 es el coprocesador 1245. En otra realización, los procesadores 1370 y 1380 son respectivamente el procesador 1210 y el coprocesador 1245.

Los procesadores 1370 y 1380 se muestran incluyendo las unidades de controlador de memoria integrada (IMC) 1372 y 1382, respectivamente. El procesador 1370 también incluye como parte de sus unidades de controlador de bus, interfaces punto a punto (P-P) 1376 y 1378; de manera similar, el segundo procesador 1380 incluye interfaces P-P 1386 y 1388. Los procesadores 1370, 1380 pueden intercambiar información a través de una interfaz punto a punto (P-P) 1350 usando circuitos de interfaz P-P 1378, 1388. Como se muestra en la **Figura 13**, los IMC 1372 y 1382 acoplan los procesadores a las respectivas memorias, en concreto, una memoria 1332 y una memoria 1334, que pueden ser porciones de la memoria principal conectadas localmente a los respectivos procesadores.

Cada uno de los procesadores 1370, 1380 puede intercambiar información con un conjunto de chips 1390 a través de interfaces P-P 1352, 1354 individuales usando circuitos de interfaz punto a punto 1376, 1394, 1386, 1398. El conjunto de chips 1390 puede opcionalmente intercambiar información con el coprocesador 1338 a través de una interfaz de alto rendimiento 1392. En una realización, el coprocesador 1338 es un procesador de propósito especial, tal como, por ejemplo, un procesador MIC de alto rendimiento, un procesador de red o comunicación, motor de compresión, procesador de gráficos, GPGPU, procesador integrado o similar.

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

El conjunto de chips 1390 se puede acoplar a un primer bus 1316 a través de una interfaz 1396. En una realización, el primer bus 1316 puede ser un bus de interconexión de componentes periféricos (PCI), o un bus tal como un bus PCI Express u otro bus de interconexión de E/S de tercera generación, aunque el alcance de la presente invención no está así limitado.

Como se muestra en la **Figura 13**, diversos dispositivos de E/S 1314 pueden acoplarse al primer bus 1316, junto con un puente de bus 1318 que acopla el primer bus 1316 a un segundo bus 1320. En una realización, uno o más procesadores adicionales 1315, tales como coprocesadores, procesadores MIC de alto rendimiento, GPGPU, aceleradores (tales como, por ejemplo, aceleradores de gráficos o unidades de procesamiento de señales digitales (DSP)), conjuntos de puertas programables en campo o cualquier otro procesador, están acoplados al primer bus 1316. En una realización, el segundo bus 1320 puede ser un bus de recuento bajo de pines (LPC). Se pueden acoplar diversos dispositivos a un segundo bus 1320 que incluye, por ejemplo, un teclado y/o ratón 1322, dispositivos de comunicaciones 1327 y una unidad de almacenamiento 1328 tal como una unidad de disco u otro dispositivo de almacenamiento masivo que puede incluir instrucciones/códigos y datos 1330, en una realización. Además, se puede acoplar una E/S de audio 1324 al segundo bus 1320. Obsérvese que son posibles otras arquitecturas. Por ejemplo, en lugar de la arquitectura de punto a punto de la **Figura 13**, un sistema puede implementar un bus multipunto u otra arquitectura de este tipo.

Con referencia ahora a la **Figura 14**, se muestra un diagrama de bloques de un segundo sistema 1400 ilustrativo más específico de acuerdo con una realización de la presente invención. Elementos similares en las **Figuras 13 y 14** llevan números de referencia similares, y ciertos aspectos de la **Figura 13** se han omitido de la **Figura 14** para evitar complicar otros aspectos de la **Figura 14**.

La **Figura 14** ilustra que los procesadores 1370, 1380 pueden incluir memoria integrada y lógica de control de E/S ("CL") 1472 y 1482, respectivamente. Por lo tanto, la CL 1472, 1482 incluyen unidades controladoras de memoria integradas e incluyen lógica de control de E/S. La **Figura 14** ilustra que no solo las memorias 1332, 1334 están acopladas a la CL 1472, 1482, sino que también los dispositivos de E/S 1414 también están acoplados a la lógica de control 1472, 1482. Los dispositivos de E/S heredados 1415 están acoplados al conjunto de chips 1390.

Con referencia ahora a la **Figura 15**, se muestra un diagrama de bloques de un SoC 1500 de acuerdo con una realización de la presente invención. Elementos similares en la **Figura 11** llevan los mismos números de referencia. Además, los recuadros con línea discontinua son características opcionales en los SoC más avanzados. En la **Figura 15**, una unidad o unidades de interconexión 1502 están acopladas a: un procesador de aplicaciones 1510 que incluye un conjunto de uno o más núcleos 1102A-N, que incluyen una o más unidades de caché 1104A-N y una unidad o unidades de caché compartida 1106; una unidad de agente de sistema 1110; una unidad o unidades de controlador de bus 1116; una unidad o unidades de controlador de memoria integrado 1114; un conjunto o uno o más coprocesadores 1520 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) 1530; una unidad de acceso directo a memoria (DMA) 1532; y una unidad de visualización 1540 para acoplarse a una o más pantallas externas. En una realización, los coprocesadores 1520 incluyen un procesador de propósito especial, tal como, por

ejemplo, un procesador de red o comunicación, motor de compresión, GPGPU, un procesador MIC de alto rendimiento, procesador integrado o similar.

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

El código de programa, tal como el código 1330 ilustrado en la **Figura 13**, se puede aplicar a las instrucciones de entrada para realizar las funciones descritas en el presente documento y generar información de salida. La información de salida puede aplicarse a uno o más dispositivos de salida, de forma conocida. Para los propósitos de esta solicitud, un sistema de procesamiento incluye cualquier sistema que tenga un procesador, tal como, por ejemplo; un procesador de señales digitales (DSP), un microcontrolador, un circuito integrado de específico de la aplicación (ASIC) o un microprocesador.

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

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

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

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

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

En algunos casos, se puede usar un convertidor de instrucciones para convertir una instrucción de un conjunto de instrucciones de origen a un conjunto de instrucciones 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 para que las procese el núcleo. El convertidor de instrucciones puede implementarse en software, hardware, firmware o una combinación de los mismos. El convertidor de instrucciones puede estar en el procesador, fuera del procesador o en parte dentro y fuera del procesador.

La **Figura 16** es un diagrama de bloques que contrasta el uso de un convertidor de instrucciones de software para convertir instrucciones binarias en un conjunto de instrucciones de origen en instrucciones binarias en un conjunto de instrucciones objetivo de acuerdo con algunas realizaciones de la invención. En la realización ilustrada, el convertidor de instrucciones es un convertidor de instrucciones de software, aunque, como alternativa, el convertidor de instrucciones puede implementarse en software, firmware, hardware o diversas combinaciones de los mismos. La **Figura 16** muestra un programa en un lenguaje de alto nivel 1602 que puede compilarse usando un compilador x86 1604 para generar código binario x86 1606 que puede ejecutarse de forma nativa mediante un procesador con al menos un núcleo de conjunto de instrucciones x86 1616. El procesador con al menos un núcleo de conjunto de instrucciones x86 1616 representa cualquier procesador que puede realizar sustancialmente las mismas funciones que un procesador Intel con al menos un núcleo del conjunto de instrucciones x86 ejecutando o procesando de otro modo (1) de manera compatible una parte sustancial del conjunto de instrucciones del núcleo de conjunto de instrucciones Intel x86 o (2) versiones de código objeto de aplicaciones u otro software destinado a ejecutarse en un

procesador Intel con al menos un núcleo de conjunto de instrucciones x86, para lograr sustancialmente el mismo resultado que un procesador Intel con al menos un núcleo de conjunto de instrucciones x86. El compilador x86 1604 representa un compilador que puede funcionar para generar código binario x86 1606 (por ejemplo, código objeto) que puede, con o sin procesamiento de vinculación adicional, ejecutarse en el procesador con al menos un núcleo de conjunto de instrucciones x86 1616. De manera similar, la **Figura 16** muestra que el programa en el lenguaje de alto nivel 1602 puede compilarse usando un compilador de conjunto de instrucciones alternativo 1608 para generar un código binario de conjunto de instrucciones alternativo 1610 que puede ejecutarse de forma nativa por un procesador sin al menos un núcleo de conjunto de instrucciones x86 1614 (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 1612 se usa para convertir el código binario x86 1606 en un código que el procesador puede ejecutar de forma nativa sin un núcleo de conjunto de instrucciones x86 1614. No es probable que este código convertido sea el mismo que el código binario del conjunto de instrucciones alternativo 1610 porque es difícil hacer un convertidor de instrucciones apto para esto; sin embargo, el código convertido conseguirá la operación general y estará compuesto por instrucciones del conjunto de instrucciones alternativo. Por lo tanto, el convertidor de instrucciones 1612 representa software, firmware, hardware o una combinación de los mismos que, mediante emulación, simulación o cualquier otro proceso, permite que un procesador u otro dispositivo electrónico que no tiene un procesador o núcleo de conjunto de instrucciones x86 ejecute el código binario x86 1606.

REIVINDICACIONES

1. Un procesador (1100) que tiene una pluralidad de núcleos (1102), que incluye un núcleo configurado, en respuesta a una instrucción de conversión de formato que tiene campos para especificar un primer registro de origen que almacena un primer elemento de datos de coma flotante de precisión sencilla de 32 bits, y un segundo registro de origen que almacena un segundo elemento de datos de coma flotante de precisión sencilla de 32 bits, configurado para:

convertir el primer elemento de datos de coma flotante de precisión sencilla de 32 bits en un primer elemento de datos de coma flotante de 16 bits, donde, cuando el primer elemento de datos de coma flotante de precisión sencilla de 32 bits es un elemento de datos normal, la conversión se debe realizar de acuerdo con un modo de redondeo especificado mediante un código de operación de la instrucción de conversión de formato, y el primer elemento de datos de coma flotante de 16 bits debe tener un bit de signo, un exponente de 8 bits, siete bits de mantisa explícitos y un bit de mantisa implícito, y donde, cuando el primer elemento de datos de coma flotante de precisión sencilla de 32 bits es un elemento de datos que no es un número, NaN, el núcleo se configura para definir un bit más significativo de una mantisa del primer elemento de datos de coma flotante de 16 bits como uno, y donde, cuando el primer elemento de datos de coma flotante de precisión sencilla de 32 bits es un elemento de datos denormal o cero, el núcleo se configura para preservar un bit de signo del primer elemento de datos de coma flotante de precisión sencilla de 32 bits definiendo el bit de signo del primer elemento de datos de coma flotante de 16 bits como el bit de signo del primer elemento de datos de coma flotante de precisión sencilla de 32 bits, y definir los bits restantes del primer elemento de datos de coma flotante de 16 bits como cero;

convertir el segundo elemento de datos de coma flotante de precisión sencilla de 32 bits en un segundo elemento de datos de coma flotante de 16 bits, donde, cuando el segundo elemento de datos de coma flotante de precisión sencilla de 32 bits es un elemento de datos normal, la conversión se debe realizar de acuerdo con el modo de redondeo especificado por el código de operación de la instrucción de conversión de formato, y el segundo elemento de datos de coma flotante de 16 bits debe tener un bit de signo, un exponente de 8 bits, siete bits de mantisa explícitos y un bit de mantisa implícito, y donde, cuando el segundo elemento de datos de coma flotante de precisión sencilla de 32 bits es un elemento de datos NaN, el núcleo se configura para definir un bit más significativo de una mantisa del segundo elemento de datos de coma flotante de 16 bits como uno, y donde, cuando el segundo elemento de datos de coma flotante de precisión sencilla de 32 bits es un elemento de datos denormal o cero, el núcleo se configura para preservar un bit de signo del segundo elemento de datos de coma flotante de precisión sencilla de 32 bits definiendo el bit de signo del segundo elemento de datos de coma flotante de 16 bits como el bit de signo del segundo elemento de datos de coma flotante de precisión sencilla de 32 bits, y definir los bits restantes del segundo elemento de datos de coma flotante de 16 bits como cero; y

almacenar el primer elemento de datos de coma flotante de 16 bits en una mitad de orden inferior de un registro de destino y el segundo elemento de datos de coma flotante de 16 bits en una mitad de orden superior del registro de destino,

donde el procesador (1100) comprende, además:

el primer registro de origen configurado para almacenar el primer elemento de datos de coma flotante de precisión sencilla de 32 bits; y

el segundo registro de origen configurado para almacenar el segundo elemento de datos de coma flotante de precisión sencilla de 32 bits.

2. El procesador (1100) de la reivindicación 1, donde el primer y segundo elemento de datos de coma flotante de 16 bits son el primer y segundo elemento de datos de formato BF16.

3. El procesador (1100) de una cualquiera de las reivindicaciones 1 o 2, que comprende además un convertidor de instrucciones configurado para convertir la instrucción de conversión de formato en una o más instrucciones de un conjunto de instrucciones diferente ejecutable por el núcleo.

4. El procesador (1100) de una cualquiera de las reivindicaciones 1 a 3, donde la pluralidad de núcleos (1102) comprende núcleos gráficos.

5. El procesador (1100) de una cualquiera de las reivindicaciones 1 a 4, donde la pluralidad de núcleos (1102) son heterogéneos.

6. El procesador (1100) de una cualquiera de las reivindicaciones 1 a 5, que comprende, además:

una memoria caché (1104); y

un archivo de registro.

7. Un chip (1500) que comprende:

una pluralidad de controladores de memoria (1114);

una memoria caché de nivel dos, L2, acoplada a la pluralidad de controladores de memoria (1114);

un procesador (1100) de acuerdo con una cualquiera de las reivindicaciones 1 a 6, donde el procesador (1100) está acoplado a la pluralidad de controladores de memoria (1114) y acoplado a la memoria caché L2;

una interconexión (1502) acoplada al procesador (1100); y

un controlador de bus (1116) acoplado al procesador (1100).

8. Un sistema que comprende:

el chip (1500) de la reivindicación 7; y

una memoria de sistema acoplada con el chip (1500).

9. Un método, que comprende:

procesar datos con una pluralidad de núcleos de un procesador, incluyendo la pluralidad de núcleos un núcleo;

realizar una instrucción de conversión de formato con el núcleo, teniendo la instrucción de conversión de formato campos para especificar un primer registro de origen que almacena un primer elemento de datos de coma flotante de precisión sencilla de 32 bits y un segundo registro de origen que almacena un segundo elemento de datos de coma flotante de precisión sencilla de 32 bits, donde realizar la instrucción de conversión de formato incluye:

convertir el primer elemento de datos de coma flotante de precisión sencilla de 32 bits en un primer elemento de datos de coma flotante de 16 bits, donde, cuando el primer elemento de datos de coma flotante de precisión sencilla de 32 bits es un elemento de datos normal, la conversión se debe realizar de acuerdo con un modo de redondeo especificado por un código de operación de la instrucción de conversión de formato, y el primer elemento de datos de coma flotante de 16 bits debe tener un bit de signo, un exponente de 8 bits, siete bits de mantisa explícitos y un bit de mantisa implícito, y donde, cuando el primer elemento de datos de coma flotante de precisión sencilla de 32 bits es un elemento de datos que no es un número, NaN, la conversión incluye definir un bit más significativo de una mantisa del primer elemento de datos de coma flotante de 16 bits como uno, y donde, cuando el primer elemento de datos de coma flotante de precisión sencilla de 32 bits es un elemento de datos denormal o cero, la conversión incluye preservar un bit de signo del primer elemento de datos de coma flotante de precisión sencilla de 32 bits definiendo el bit de signo del primer elemento de datos de coma flotante de 16 bits como el bit de signo del primer elemento de datos de coma flotante de precisión sencilla de 32 bits, y definir los bits restantes del primer elemento de datos de coma flotante de 16 bits como cero;

convertir el segundo elemento de datos de coma flotante de precisión sencilla de 32 bits en un segundo elemento de datos de coma flotante de 16 bits, donde, cuando el segundo elemento de datos de coma flotante de precisión sencilla de 32 bits es un elemento de datos normal, la conversión se debe realizar de acuerdo con el modo de redondeo especificado por el código de operación de la instrucción de conversión de formato, y el segundo elemento de datos de coma flotante de 16 bits debe tener un bit de signo, un exponente de 8 bits, siete bits de mantisa explícitos y un bit de mantisa implícito, y donde, cuando el segundo elemento de datos de coma flotante de precisión sencilla de 32 bits es un elemento de datos NaN, la conversión incluye definir un bit más significativo de una mantisa del segundo elemento de datos de coma flotante de 16 bits como uno, y donde, cuando el segundo elemento de datos de coma flotante de precisión sencilla de 32 bits es un elemento de datos denormal o cero, la conversión incluye preservar un bit de signo del segundo elemento de datos de coma flotante de precisión sencilla de 32 bits definiendo el bit de signo del segundo elemento de datos de coma flotante de 16 bits como el bit de signo del segundo elemento de datos de coma flotante de precisión sencilla de 32 bits, y definir los bits restante del segundo elemento de datos de coma flotante de 16 bits como cero; y

almacenar el primer elemento de datos de coma flotante de 16 bits en una mitad de orden inferior de un registro de destino y almacenar el segundo elemento de datos de coma flotante de 16 bits en una mitad de orden superior del registro de destino,

donde el método comprende, además:

recibir el primer elemento de datos de coma flotante de precisión sencilla de 32 bits del primer registro de origen; y

recibir el segundo elemento de datos de coma flotante de precisión sencilla de 32 bits del segundo registro de origen.

10. El método de la reivindicación 9, donde el primer y segundo elemento de datos de coma flotante de 16 bits son el primer y segundo elemento de datos de formato BF16.

11. El método de una cualquiera de las reivindicaciones 9 a 10, que comprende además convertir la instrucción de conversión de formato en una o más instrucciones de un conjunto de instrucciones diferente ejecutable por el núcleo.

12. El método de una cualquiera de las reivindicaciones 9 a 11, donde dicho procesamiento de los datos con la pluralidad de núcleos comprende procesar los datos con núcleos gráficos.

13. El método de una cualquiera de las reivindicaciones 9 a 12, donde dicho procesamiento de los datos con la pluralidad de núcleos comprende procesar los datos con núcleos heterogéneos.

14. El método de una cualquiera de las reivindicaciones 9 a 13, que comprende, además:

acceder a una memoria con una pluralidad de controladores de memoria de un chip;

almacenar datos en una memoria caché de nivel dos, L2, del chip;

transmitir datos del procesador a una interconexión del chip; y

acceder a un bus con un controlador de bus del chip,

donde el chip comprende el procesador.

15. Un medio legible por máquina que almacena un código, el cual, cuando se ejecuta, hace que el procesador de la reivindicación 1 realice cualquier método de las reivindicaciones 9 a 14.

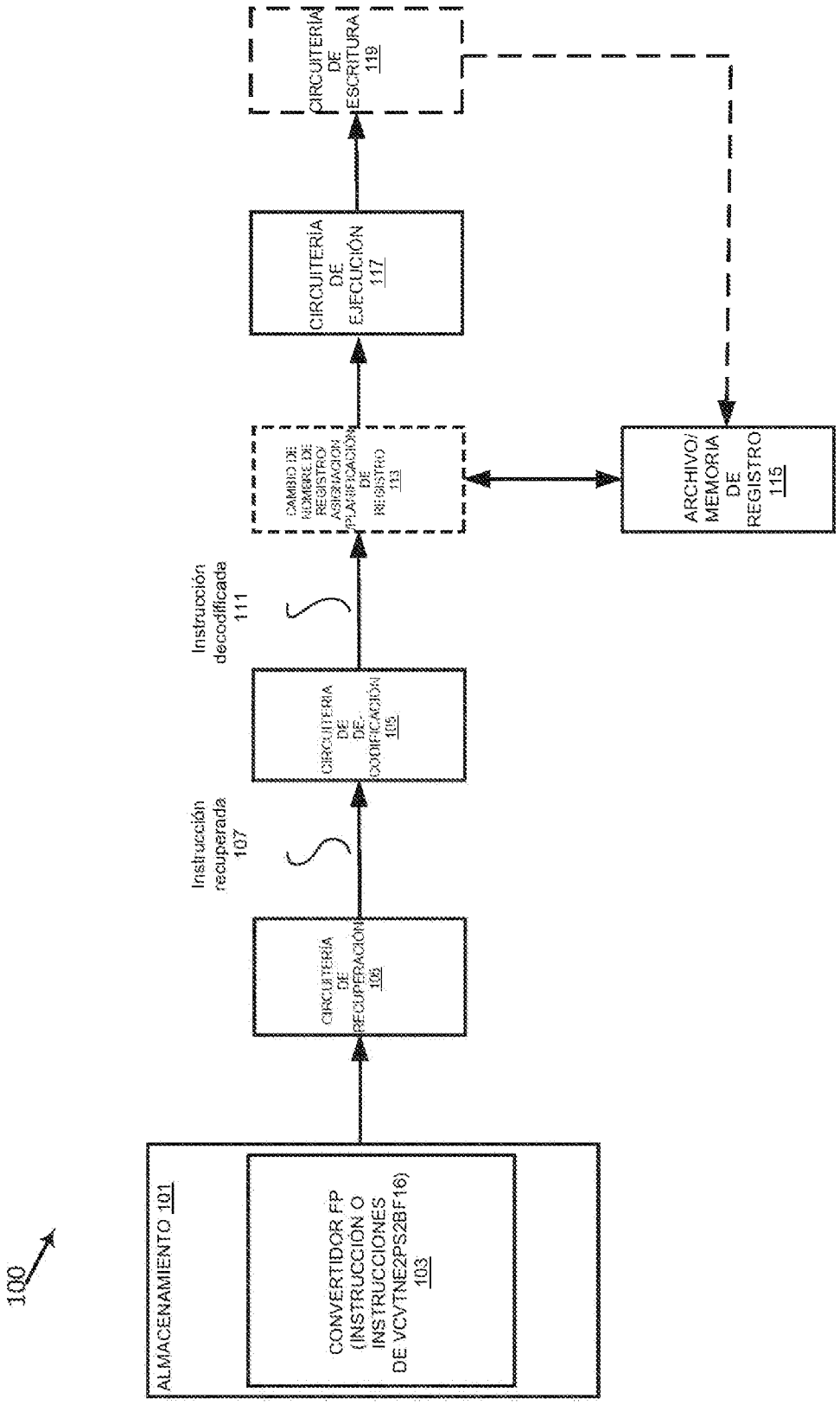

100
ALMACENAMIENTO 101
CONVERTIDOR FP (INSTRUCCIÓN O INSTRUCCIONES DE VCVTNE2PS2BF16) 103
CIRCUITERÍA DE RECUPERACIÓN 105
Instrucción recuperada 107
CIRCUITERÍA DE DECODIFICACIÓN 109
Instrucción decodificada 111
CAMBIO DE NOMBRE DE REGISTRO/ ASIGNACIÓN/AMPLIACIÓN DE REGISTRO 113
CIRCUITERÍA DE EJECUCIÓN 117
CIRCUITERÍA DE ESCRITURA 119
ARCHIVO/ MEMORIA DE REGISTRO 115

FIG. 1

Instrucción 201

Código de operación VCVTNEPS2BF16* 202	Ubicación de destino (registro/memoria) 204	Primera ubicación de origen (registro/pieza/ memoria) 206	{k} 208	{Z} 210

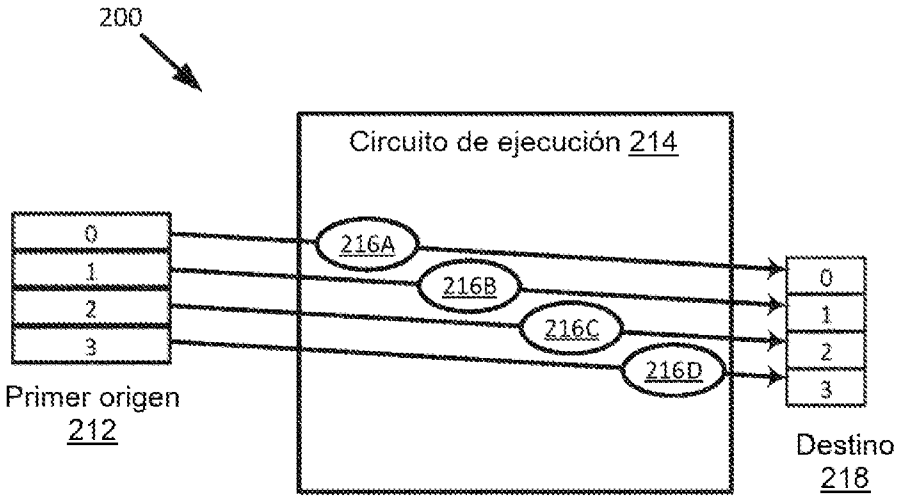


FIG. 2A

Instrucción 221

Código de operación VCVTNEPS2BF16* 222	Ubicación de destino (registro/memoria) 224	Primera ubicación de origen (registro/pieza/ memoria) 226	{k} 228	{Z} 230

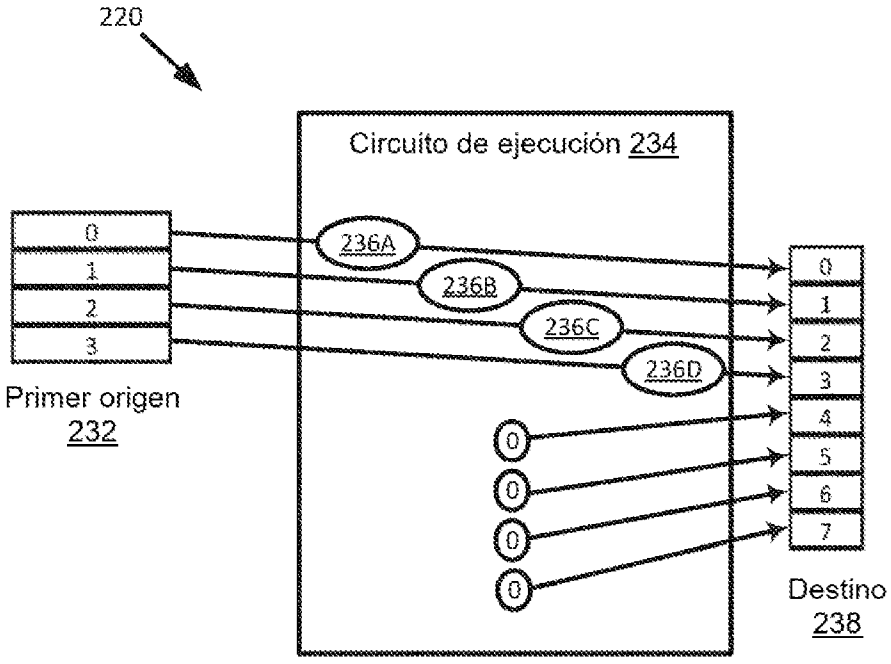


FIG. 2B

Instrucción 241

Código de operación VCVTNEPS2BF16* 242	Ubicación de destino (registro/memoria) 244	Primera ubicación de origen (registro/pieza/ memoria) 246	{k} 248	{Z} 250

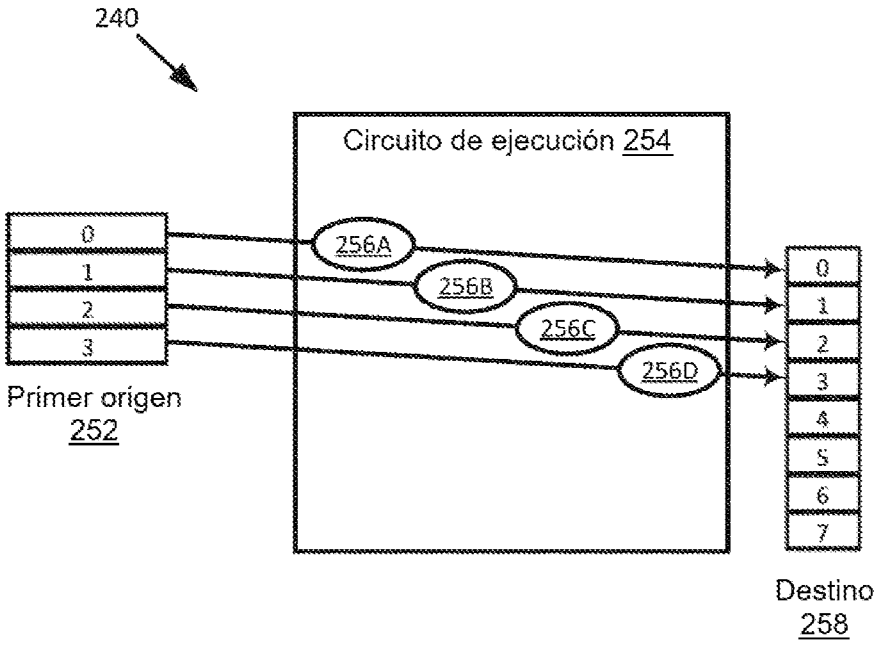


FIG. 2C

Instrucción 261

Código de operación VCVTNEPS2BF16* 262	Ubicación de destino (registro/memoria) 264	Primera ubicación de origen (registro/pieza/ memoria) 266	Segunda ubicación de origen (registro/pieza/ memoria) 268	{k} 270	{Z} 271

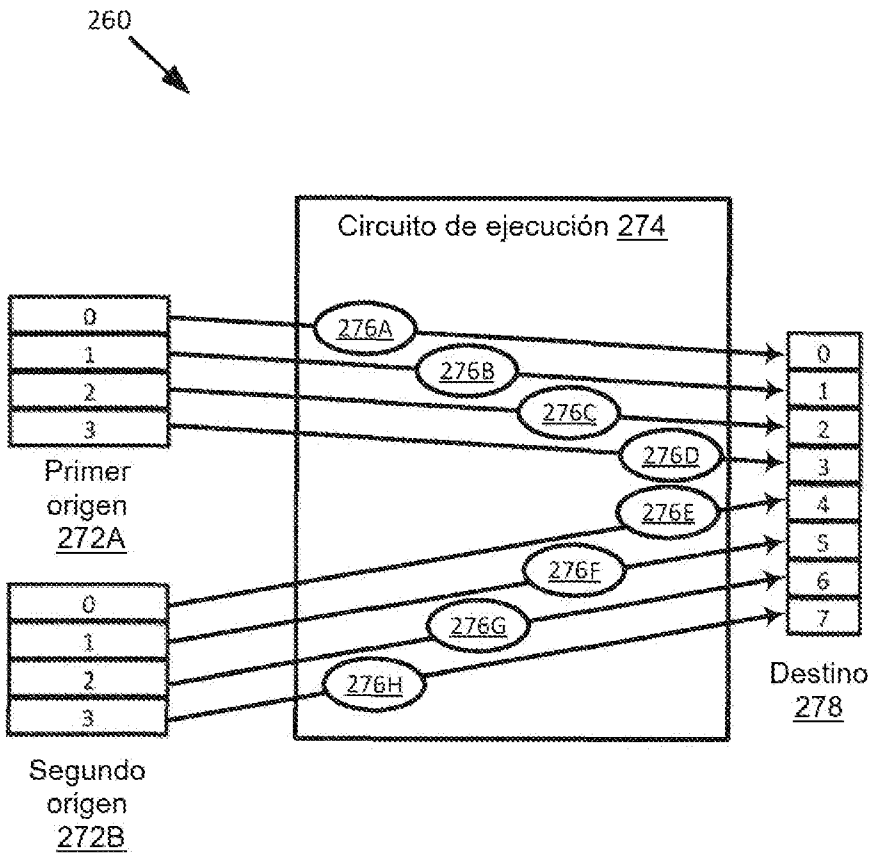


FIG. 2D

Instrucción 301

Código de operación VCVTNEPS2BF16* 302	Ubicación de destino (registro/memoria) 304	Primera ubicación de origen (registro/pieza/ memoria) 306	{k} 308	{Z} 310

Pseudocódigo
315

```
VCVTNEPS2BF16 dest, src
        VL = (128,256,512)
        KL = VL/16
        origdest := dest

        for i := 0 to kl/2-1:
                if k1[i] or *no writemask*:
                        if src is memory and evex.b == 1:
                                t := src.fp32[0]
                        else:
                                t := src.fp32[i]

                        dest.word[i] := convert_fp32_to_bfloat16(t)

                else if *zeroing*:
                        dest.word[i] := 0
                else: // fusionar enmascaramiento, elemento de destino sin cambios
                        dest.word[i] := origdest.word[i]

        dest[max_vl-1:vl/2] := 0
```

FIG. 3A

Instrucción 321

Código de operación VCVTNEPS2BF16* 322	Ubicación de destino (registro/memoria) 324	Primera ubicación de origen (registro/pieza/ memoria) 326	Segunda ubicación de origen (registro/pieza/ memoria) 328	{k} 330	{Z} 331

Pseudocódigo
335

```
VCVTNE2PS2BF16 dest, src1, src2
        VL = (128,256,512)
        KL = VL/16
        origdest := dest
        for i := 0 to kl-1:
                if k1[i] or *no writemask*:
                        if src is memory and evex.b == 1:
                                t := src2.fp32[0]
                        else if i < kl/2:
                                t := src2.fp32[i]
                        else:
                                t := src1.fp32[i-kl/2]

                        // véase VCVTNEPS2BF16 para definición de función auxiliar de convertidor
                        dest.word[i] := convert_fp32_to_bfloat16(t)

                else if *zeroing*:
                        dest.word[i] := 0
                else: //  fusionar enmascaramiento, elemento de destino sin cambios
                        dest.word[i] := origdest.word[i]

        dest[max_vl-1:vl] := 0
```

FIG. 3B

Pseudocódigo
340

Función auxiliar

```
define convert_fp32_to_bfloat16(x):
si x es cero o anormal:
        dest[15] := x[31] // conservación de signo cero (denormal pasar a cero)
        dest[14:0] := 0
else if x es infinito:
        dest[15:0] := x[31:16]
else if x es nan:
        dest[15:0] := x[31:16] //truncar y establecer msb de la fuerza de la mantisa qnan
        dest[7] := 1
else // número normal
        lsb := x[16]
        rounding_bias := 0x00007FFF + lsb
        temp[31:0] := x[31:0] + rounding_bias // número entero añadir
        dest[15:0] := temp[31:16]
return dest
```

FIG. 3C

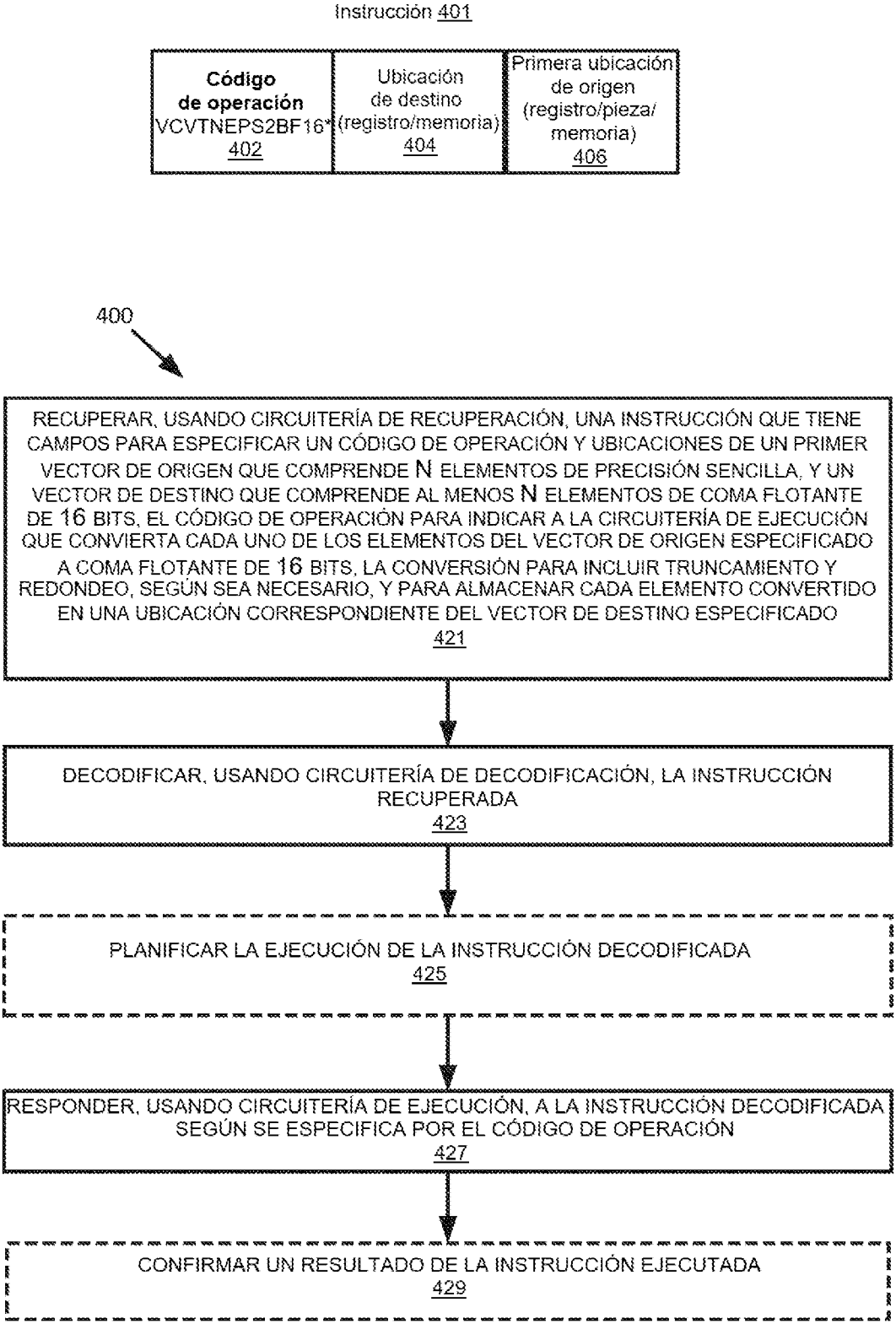

Instrucción 401
Código de operación VCVTNEPS2BF16* 402
Ubicación de destino (registro/memoria) 404
Primera ubicación de origen (registro/pieza/ memoria) 406
400
RECUPERAR, USANDO CIRCUITERÍA DE RECUPERACIÓN, UNA INSTRUCCIÓN QUE TIENE CAMPOS PARA ESPECIFICAR UN CÓDIGO DE OPERACIÓN Y UBICACIONES DE UN PRIMER VECTOR DE ORIGEN QUE COMPRENDE N ELEMENTOS DE PRECISIÓN SENCILLA, Y UN VECTOR DE DESTINO QUE COMPRENDE AL MENOS N ELEMENTOS DE COMA FLOTANTE DE 16 BITS, EL CÓDIGO DE OPERACIÓN PARA INDICAR A LA CIRCUITERÍA DE EJECUCIÓN QUE CONVIERTA CADA UNO DE LOS ELEMENTOS DEL VECTOR DE ORIGEN ESPECIFICADO A COMA FLOTANTE DE 16 BITS, LA CONVERSIÓN PARA INCLUIR TRUNCAMIENTO Y REDONDEO, SEGÚN SEA NECESARIO, Y PARA ALMACENAR CADA ELEMENTO CONVERTIDO EN UNA UBICACIÓN CORRESPONDIENTE DEL VECTOR DE DESTINO ESPECIFICADO 421
DECODIFICAR, USANDO CIRCUITERÍA DE DECODIFICACIÓN, LA INSTRUCCIÓN RECUPERADA 423
PLANIFICAR LA EJECUCIÓN DE LA INSTRUCCIÓN DECODIFICADA 425
RESPONDER, USANDO CIRCUITERÍA DE EJECUCIÓN, A LA INSTRUCCIÓN DECODIFICADA SEGÚN SE ESPECIFICA POR EL CÓDIGO DE OPERACIÓN 427
CONFIRMAR UN RESULTADO DE LA INSTRUCCIÓN EJECUTADA 429

FIG. 4A

Instrucción 451

Código de operación VCVTNEPS2BF16* 452	Ubicación de destino (registro/memoria) 454	Primera ubicación de origen (registro/pieza/ memoria) 456	Segunda ubicación de origen (registro/pieza/ memoria) 462

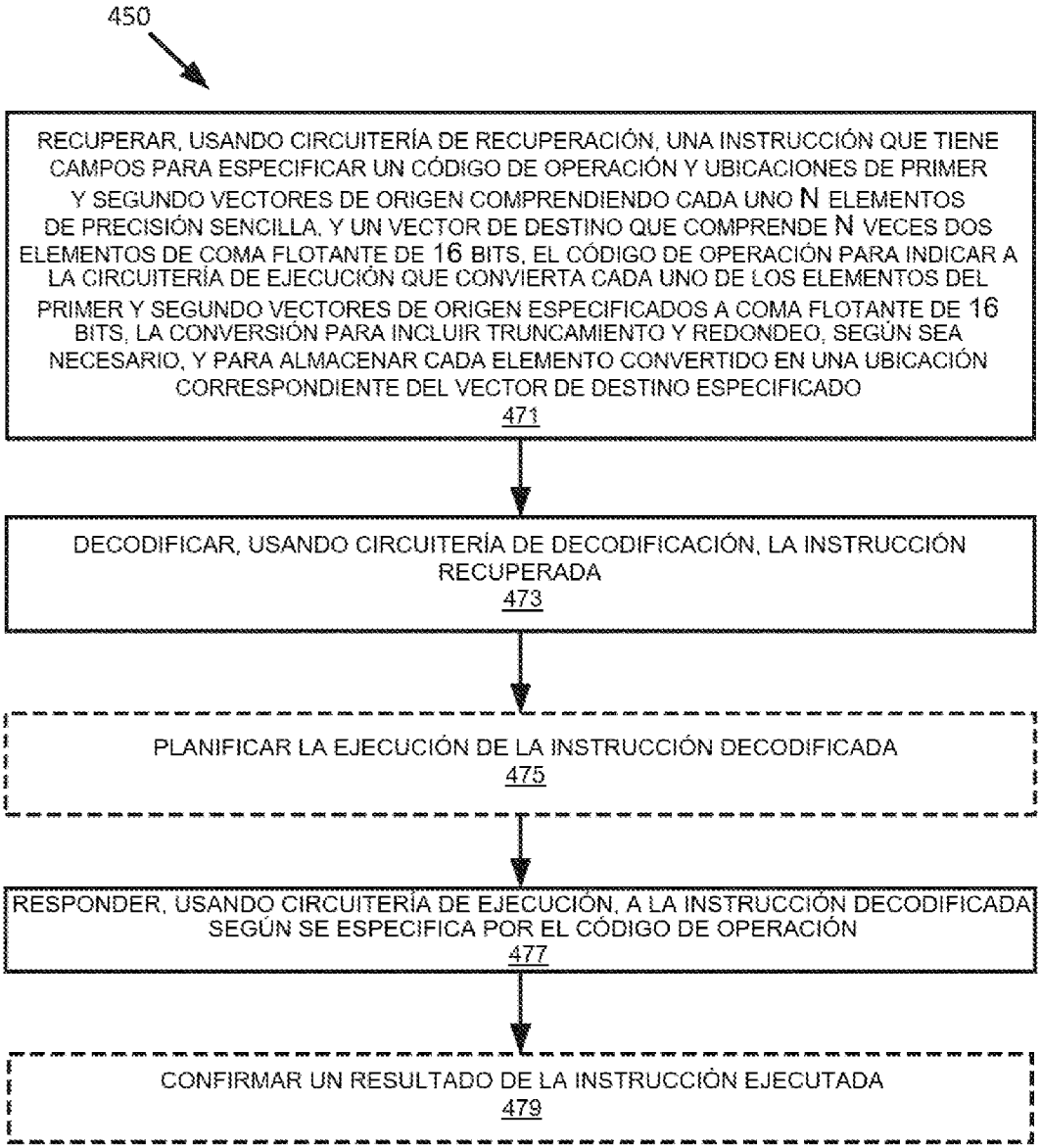


FIG. 4B

Instrucción VCVTNEPS2BF16 500

Código de operación VCVTNEPS2BF16* 502	Ubicación de destino (registro/ memoria) 504	Primera ubicación de origen (registro/ memoria) 506	Máscara {k} 508	Puesta a cero {Z} 510	Formato de elemento 514	Tamaño de vector (N) 516	Modo de redondeo 518

FIG. 5A

Instrucción VCVTNE2PS2BF16 550

Código de operación VCVTNEPS2BF16* 552	Ubicación de destino (registro/ memoria) 554	Primera ubicación de origen (registro/ memoria) 556	Segunda ubicación de origen (registro/ memoria) 562	Máscara {k} 558	Puesta a cero {Z} 560	Formato de elemento 564	Tamaño de vector (N) 566	Modo de redondeo 568

FIG. 5B

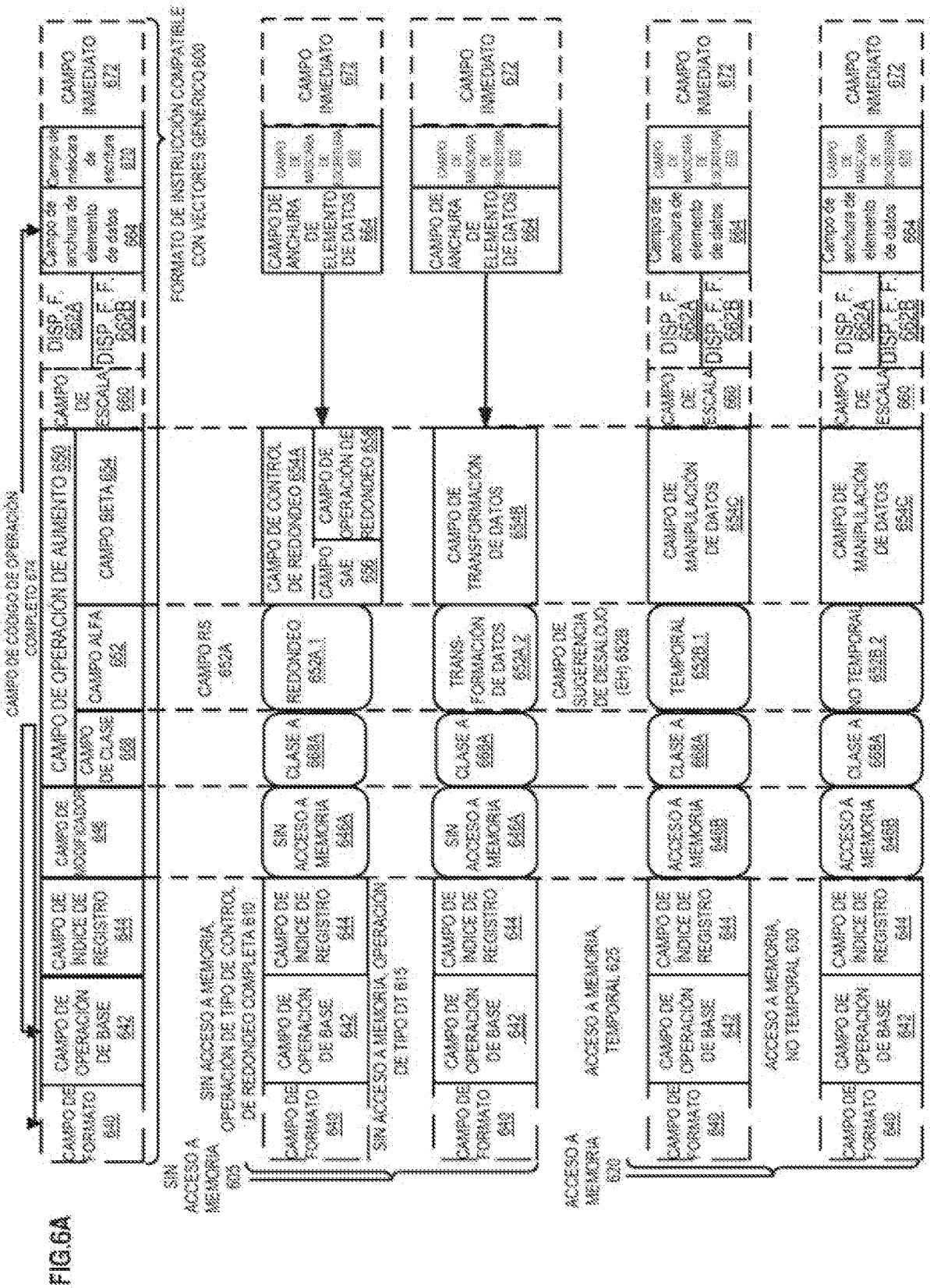
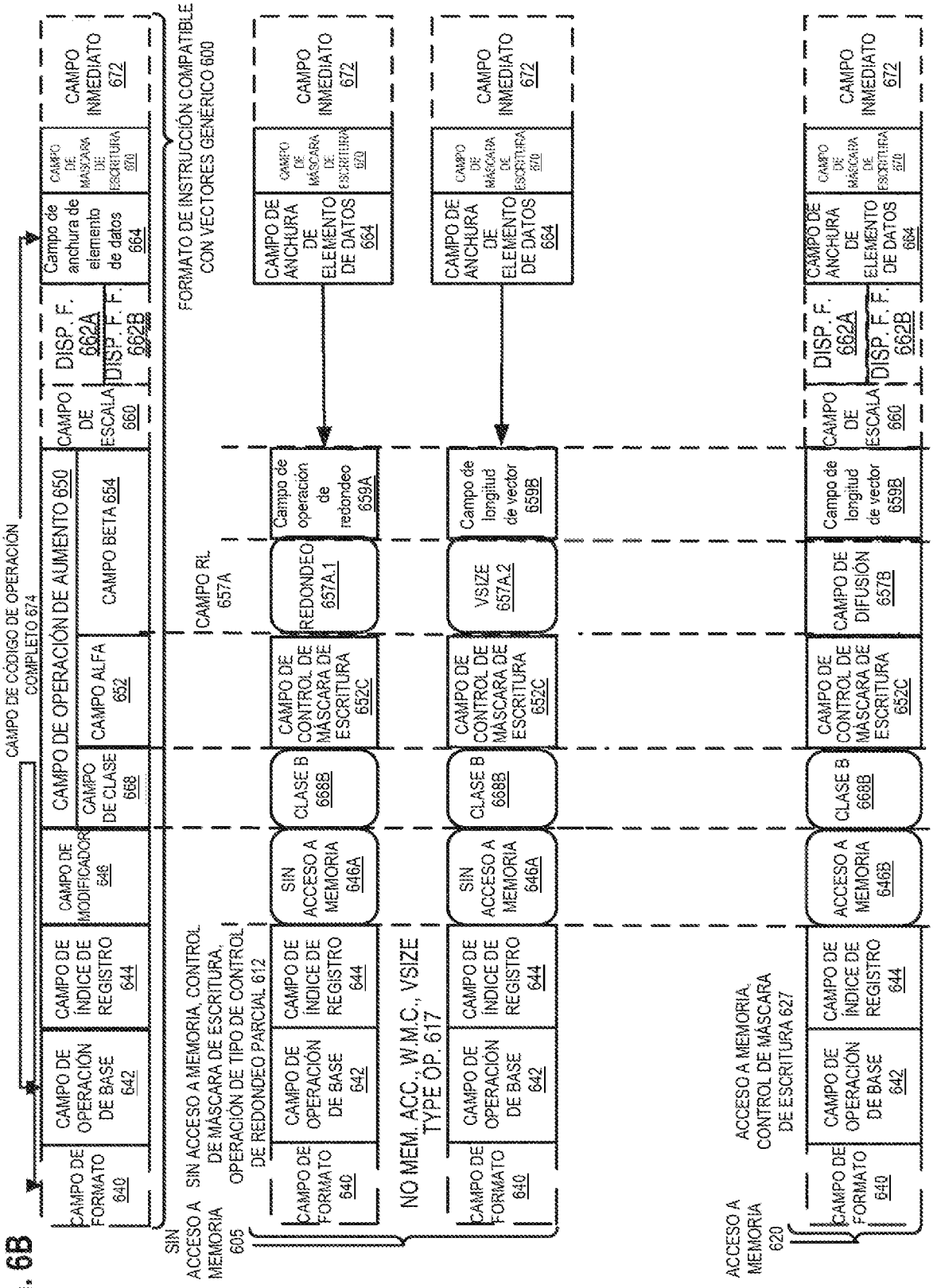

FORMATO DE INSTRUCCIÓN COMPATIBLE CON VECTORES GENÉRICO 600
CAMPO DE CÓDIGO DE OPERACIÓN COMPLETO 674
SIN ACCESO A MEMORIA 605
SIN ACCESO A MEMORIA, OPERACIÓN DE TIPO DE CONTROL DE REDONDEO COMPLETA 610
SIN ACCESO A MEMORIA, OPERACIÓN DE TIPO DT 615
ACCESO A MEMORIA 620
ACCESO A MEMORIA, NO TEMPORAL 630
ACCESO A MEMORIA, TEMPORAL 625
CAMPO DE FORMATO 640
CAMPO DE OPERACIÓN DE BASE 642
CAMPO DE ÍNDICE DE REGISTRO 644
CAMPO DE MODIFICACIÓN 646
CAMPO DE OPERACIÓN DE AUMENTO 650
CAMPO DE CLASE 668
CAMPO ALFA 652
CAMPO BETA 654
CAMPO RS 652A
CAMPO DE ESCALA 660
DISP. F. 662A
DISP. F. F. 662B
Campo de anchura de elemento de datos 664
Campo de máscara de escritura 670
CAMPO INMEDIATO 672
SIN ACCESO A MEMORIA 646A
CLASE A 668A
REDONDEO 652A.1
CAMPO DE CONTROL DE REDONDEO 654A
CAMPO SAE 656
CAMPO DE OPERACIÓN DE REDONDEO 658
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 664
CAMPO DE MÁSCARA DE ESCRITURA 670
ACCESO A MEMORIA 646B
CLASE B 668B
TEMPORAL 652B.1
NO TEMPORAL 652B.2
CAMPO DE TRANSFORMACIÓN DE DATOS 654B
TRANSFORMACIÓN DE DATOS 652A.2
CAMPO DE SUGERENCIA DE DESALOJO (EN) 652B
CAMPO DE MANIPULACIÓN DE DATOS 654C

FIG.6A

FIG. 6B


FORMATO DE INSTRUCCIÓN COMPATIBLE CON VECTORES GENÉRICO 600
CAMPO DE CÓDIGO DE OPERACIÓN COMPLETO 674
CAMPO DE FORMATO 640
CAMPO DE OPERACIÓN DE BASE 642
CAMPO DE ÍNDICE DE REGISTRO 644
CAMPO DE MODIFICADOR 646
CAMPO DE OPERACIÓN DE AUMENTO 650
CAMPO DE CLASE 668
CAMPO ALFA 652
CAMPO BETA 654
CAMPO DE ESCALA 660
DISP. F. 662A
DISP. F. F. 662B
Campo de anchura de elemento de datos 664
CAMPO DE MÁSCARA DE ESCRITURA 670
CAMPO INMEDIATO 672
SIN ACCESO A MEMORIA 605
SIN ACCESO A MEMORIA, CONTROL DE MÁSCARA DE ESCRITURA, OPERACIÓN DE TIPO DE CONTROL DE REDONDEO PARCIAL 612
SIN ACCESO A MEMORIA 646A
CLASE B 668B
CAMPO DE CONTROL DE MÁSCARA DE ESCRITURA 652C
CAMPO RL 657A
REDONDEO 657A.1
Campo de operación de redondeo 659A
NO MEM. ACC., W.M.C., VSIZE TYPE OP. 617
VSIZE 657A.2
Campo de longitud de vector 659B
ACCESO A MEMORIA 620
ACCESO A MEMORIA, CONTROL DE MÁSCARA DE ESCRITURA 627
ACCESO A MEMORIA 646B
CAMPO DE DIFUSIÓN 657B
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 664

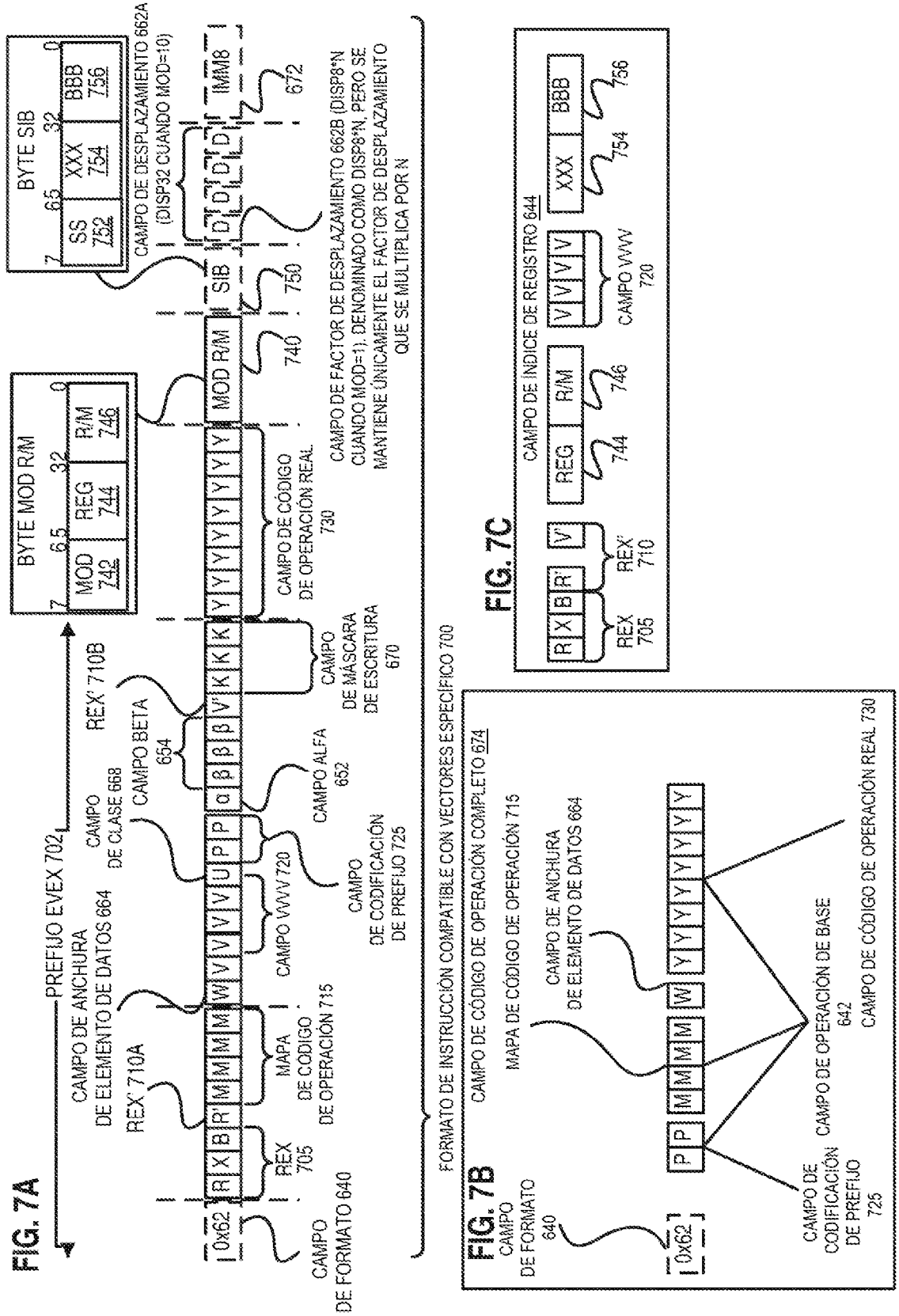


FIG. 7A
PREFIJO EVEX 702
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 664
CAMPO DE CLASE 668
REX' 710B
REX' 710A
CAMPO BETA 654
CAMPO ALFA 652
CAMPO DE MÁSCARA DE ESCRITURA 670
CAMPO DE CÓDIGO DE OPERACIÓN REAL 730
CAMPO DE FACTOR DE DESPLAZAMIENTO 662B (DISP8*N CUANDO MOD=1); DENOMINADO COMO DISP8*N, PERO SE MANTIENE ÚNICAMENTE EL FACTOR DE DESPLAZAMIENTO QUE SE MULTIPLICA POR N
BYTE MOD R/M
MOD 742
REG 744
R/M 746
BYTE SIB
SS 752
XXX 754
BBB 756
CAMPO DE DESPLAZAMIENTO 662A
(DISP32 CUANDO MOD=10)
SIB 750
DISP8 672
IMM8
REX 705
MAPA DE CÓDIGO DE OPERACIÓN 715
CAMPO VVV 720
CAMPO DE FORMATO 640
FORMATO DE INSTRUCCIÓN COMPATIBLE CON VECTORES ESPECÍFICO 700
FIG. 7B
CAMPO DE CÓDIGO DE OPERACIÓN COMPLETO 674
MAPA DE CÓDIGO DE OPERACIÓN 715
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 664
CAMPO DE CODIFICACIÓN DE PREFIJO 725
CAMPO DE OPERACIÓN DE BASE 642
CAMPO DE CÓDIGO DE OPERACIÓN REAL 730
FIG. 7C
CAMPO DE ÍNDICE DE REGISTRO 644
REX 705
REX' 710
CAMPO VVVV 720

FIG. 7D

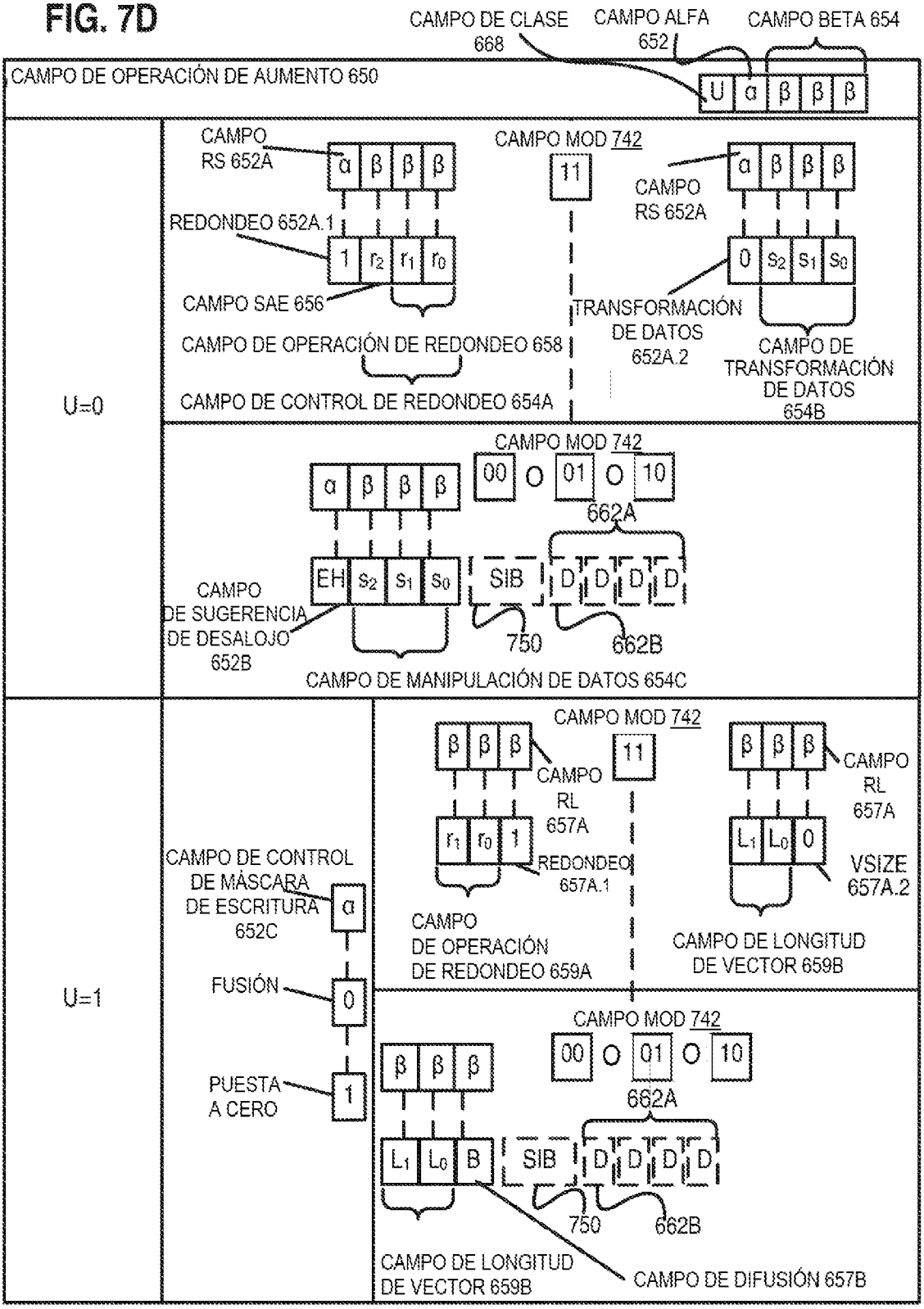
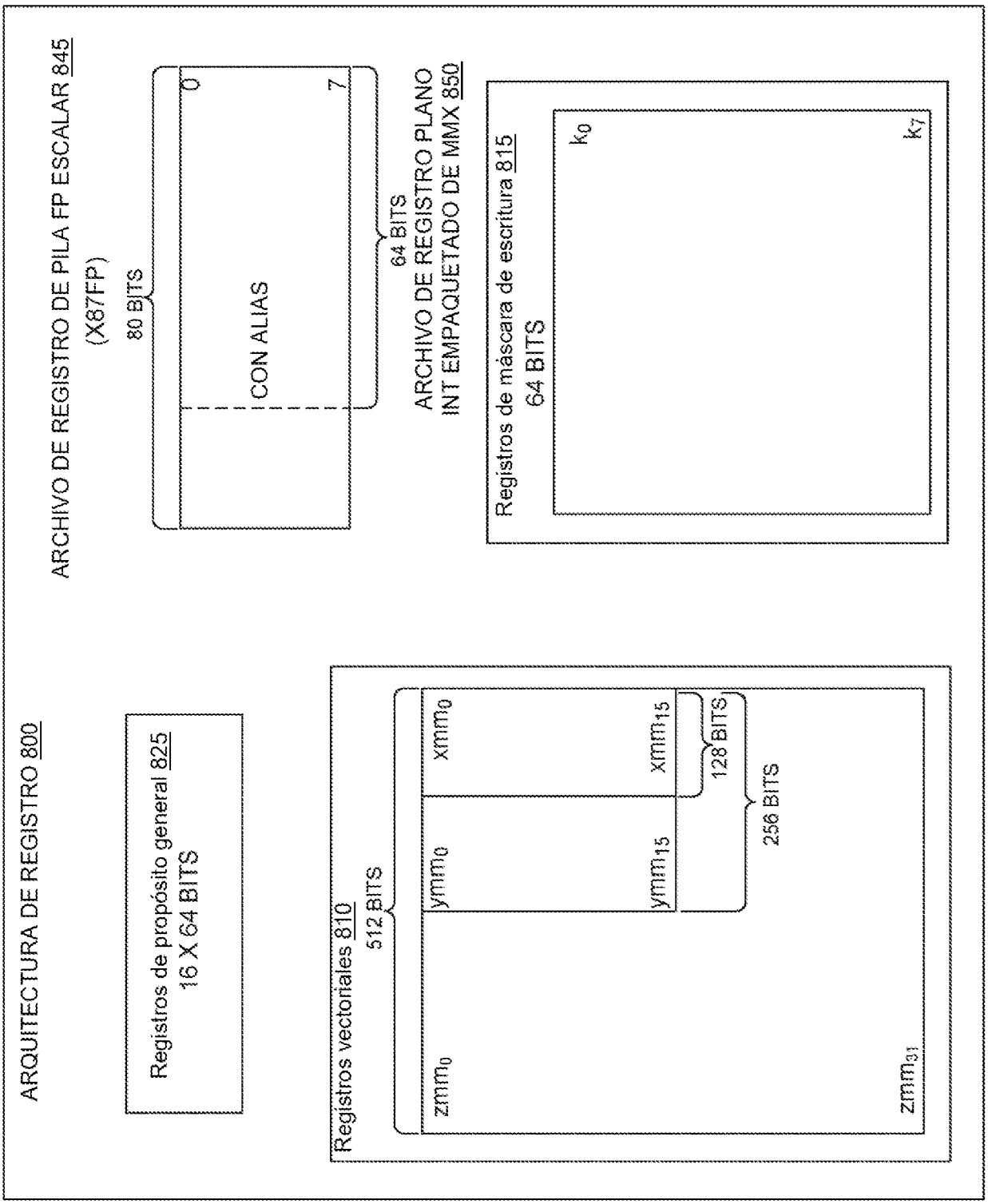

CAMPO DE CLASE 668
CAMPO ALFA 652
CAMPO BETA 654
CAMPO DE OPERACIÓN DE AUMENTO 650
U α β β β
U=0
CAMPO RS 652A
α β β β
CAMPO MOD 742
11
REDONDEO 652A.1
1 r2 r1 r0
CAMPO SAE 656
CAMPO DE OPERACIÓN DE REDONDEO 658
CAMPO DE CONTROL DE REDONDEO 654A
CAMPO RS 652A
α β β β
0 s2 s1 s0
TRANSFORMACIÓN DE DATOS 652A.2
CAMPO DE TRANSFORMACIÓN DE DATOS 654B
CAMPO MOD 742
00 O 01 O 10
662A
α β β β
EH s2 s1 s0
SIB D D D D
CAMPO DE SUGERENCIA DE DESALOJO 652B
750
662B
CAMPO DE MANIPULACIÓN DE DATOS 654C
U=1
CAMPO DE CONTROL DE MÁSCARA DE ESCRITURA 652C
α
FUSIÓN
0
PUESTA A CERO
1
CAMPO MOD 742
β β β
11
CAMPO RL 657A
r1 r0 1
REDONDEO 657A.1
CAMPO DE OPERACIÓN DE REDONDEO 659A
β β β
CAMPO RL 657A
L1 L0 0
VSIZE 657A.2
CAMPO DE LONGITUD DE VECTOR 659B
CAMPO MOD 742
00 O 01 O 10
662A
β β β
L1 L0 B
SIB D D D D
750
662B
CAMPO DE LONGITUD DE VECTOR 659B
CAMPO DE DIFUSIÓN 657B

FIG. 8


ARQUITECTURA DE REGISTRO 800
Registros de propósito general 825
16 X 64 BITS
ARCHIVO DE REGISTRO DE PILA FP ESCALAR 845
(X87FP)
80 BITS
0
CON ALIAS
7
64 BITS
ARCHIVO DE REGISTRO PLANO
INT EMPAQUETADO DE MMX 850
Registros de máscara de escritura 815
64 BITS
k0
k7
Registros vectoriales 810
512 BITS
zmm0
ymm0
xmm0
ymm15
xmm15
128 BITS
256 BITS
zmm31

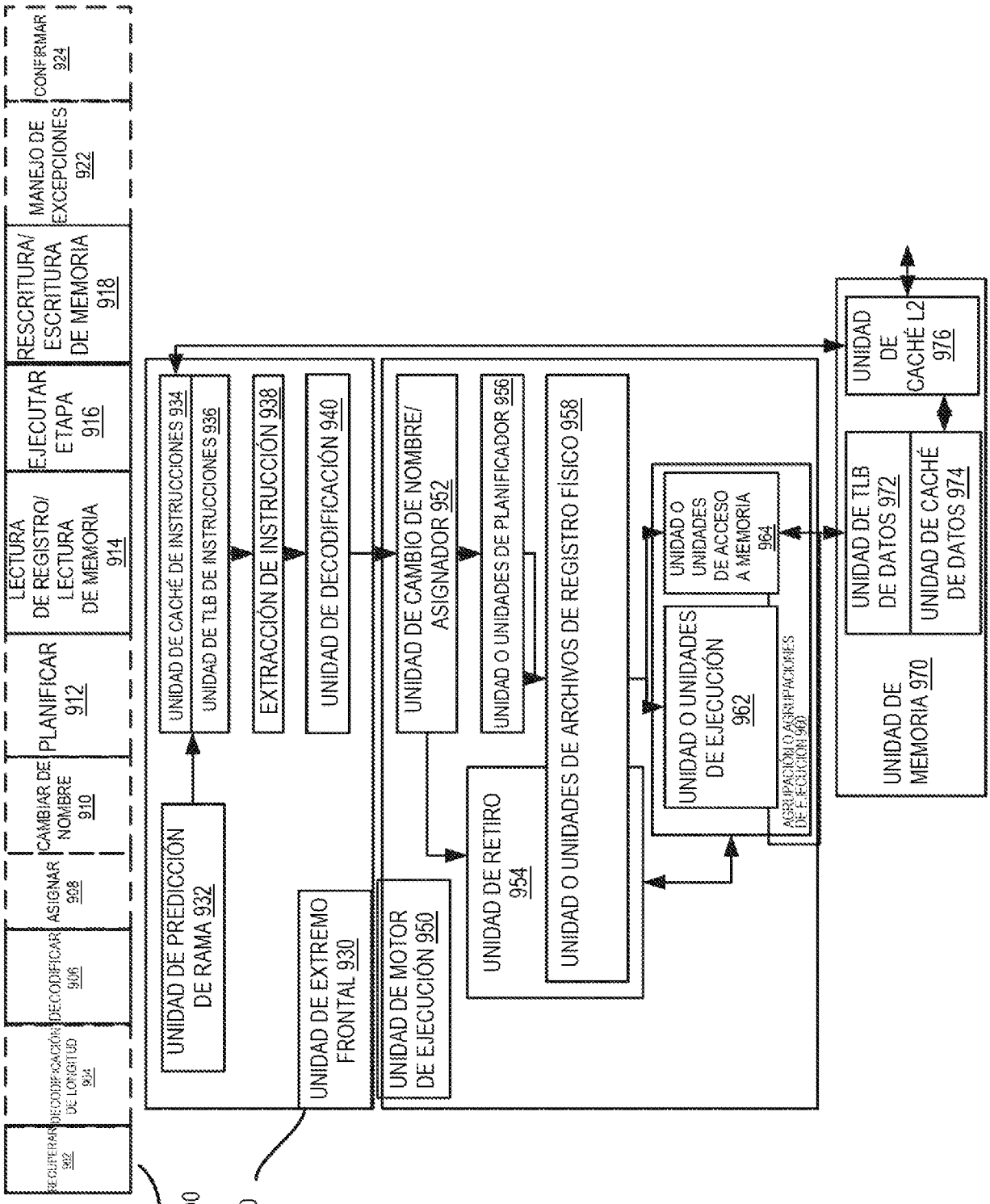

RECUPERAR 902
DECODIFICACIÓN DE LONGITUD 904
DECODIFICAR 906
ASIGNAR 908
CAMBIAR DE NOMBRE 910
PLANIFICAR 912
LECTURA DE REGISTRO/ LECTURA DE MEMORIA 914
EJECUTAR ETAPA 916
RESCRITURA/ ESCRITURA DE MEMORIA 918
MANEJO DE EXCEPCIONES 922
CONFIRMAR 924
CANALIZACIÓN 900
NÚCLEO 990
UNIDAD DE PREDICCIÓN DE RAMA 932
UNIDAD DE CACHÉ DE INSTRUCCIONES 934
UNIDAD DE TLB DE INSTRUCCIONES 936
EXTRACCIÓN DE INSTRUCCIÓN 938
UNIDAD DE DECODIFICACIÓN 940
UNIDAD DE EXTREMO FRONTAL 930
UNIDAD DE MOTOR DE EJECUCIÓN 950
UNIDAD DE CAMBIO DE NOMBRE/ ASIGNADOR 952
UNIDAD DE RETIRO 954
UNIDAD O UNIDADES DE PLANIFICADOR 956
UNIDAD O UNIDADES DE ARCHIVOS DE REGISTRO FÍSICO 958
AGRUPACIÓN O AGRUPACIONES DE EJECUCIÓN 960
UNIDAD O UNIDADES DE EJECUCIÓN 962
UNIDAD O UNIDADES DE ACCESO A MEMORIA 964
UNIDAD DE MEMORIA 970
UNIDAD DE TLB DE DATOS 972
UNIDAD DE CACHÉ DE DATOS 974
UNIDAD DE CACHÉ L2 976

FIG. 9A

FIG. 9B

FIG. 10A

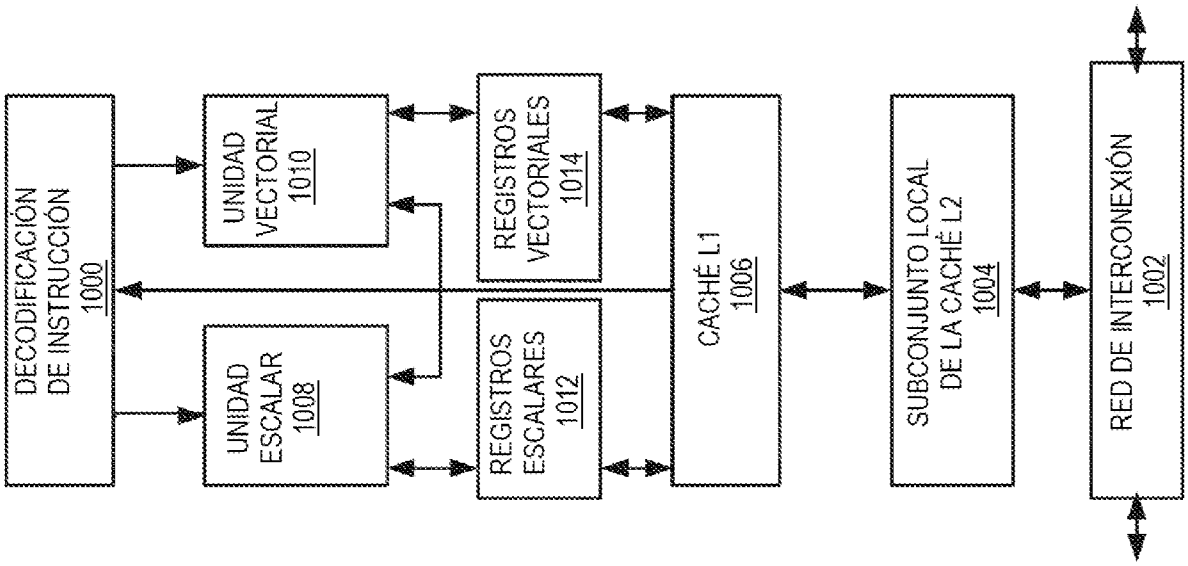
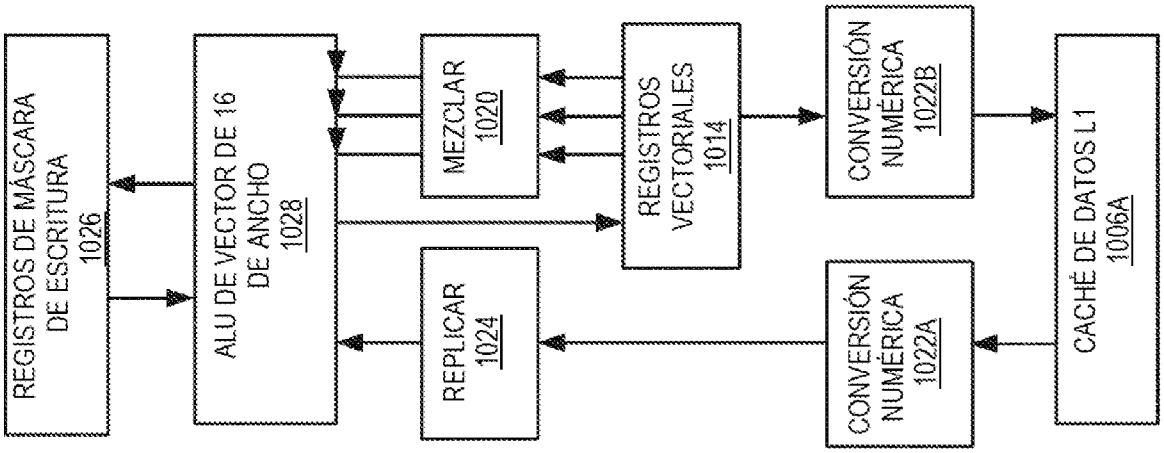


FIG. 10B



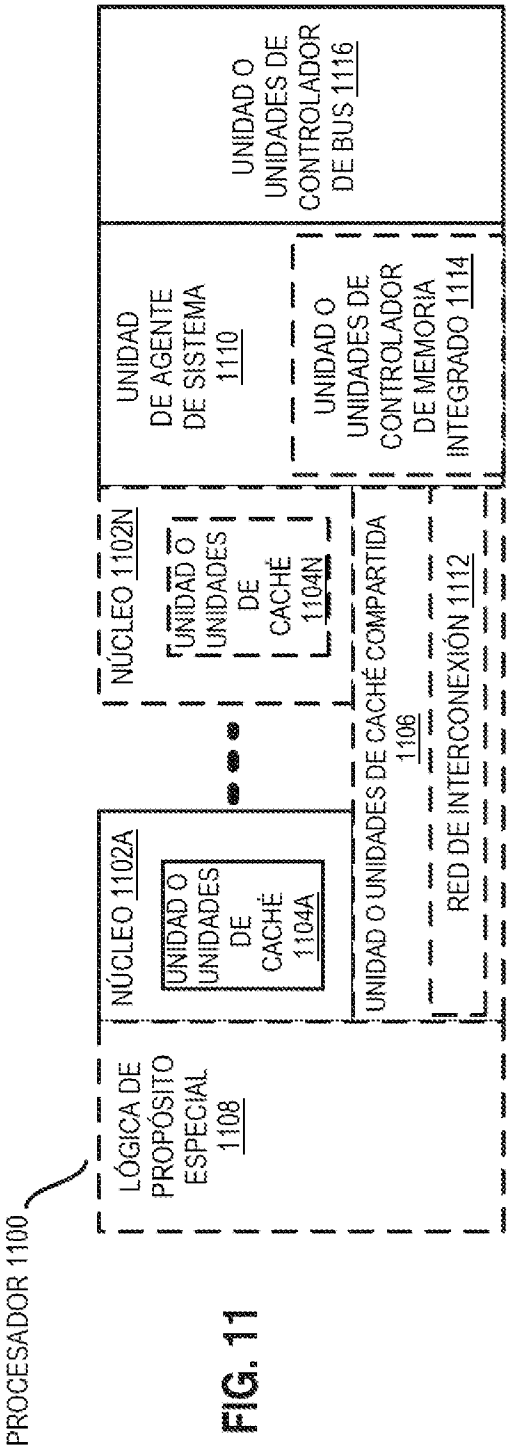

PROCESADOR 1100
LÓGICA DE PROPÓSITO ESPECIAL 1108
NÚCLEO 1102A
UNIDAD O UNIDADES DE CACHÉ 1104A
NÚCLEO 1102N
UNIDAD O UNIDADES DE CACHÉ 1104N
UNIDAD O UNIDADES DE CACHÉ COMPARTIDA 1106
RED DE INTERCONEXIÓN 1112
UNIDAD DE AGENTE DE SISTEMA 1110
UNIDAD O UNIDADES DE CONTROLADOR DE MEMORIA INTEGRADO 1114
UNIDAD O UNIDADES DE CONTROLADOR DE BUS 1116

FIG. 11

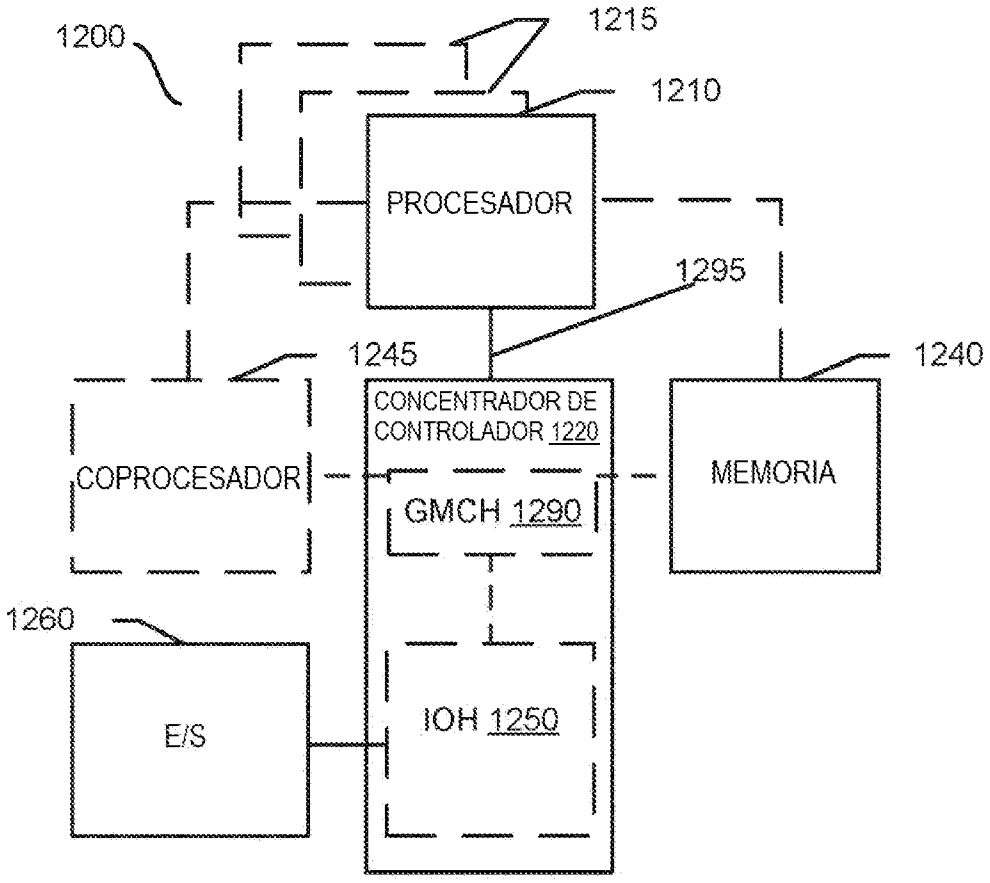

1200
1215
1210
PROCESADOR
1295
1245
1240
CONCENTRADOR DE CONTROLADOR 1220
COPROCESADOR
GMCH 1290
MEMORIA
1260
E/S
IOH 1250

FIG. 12

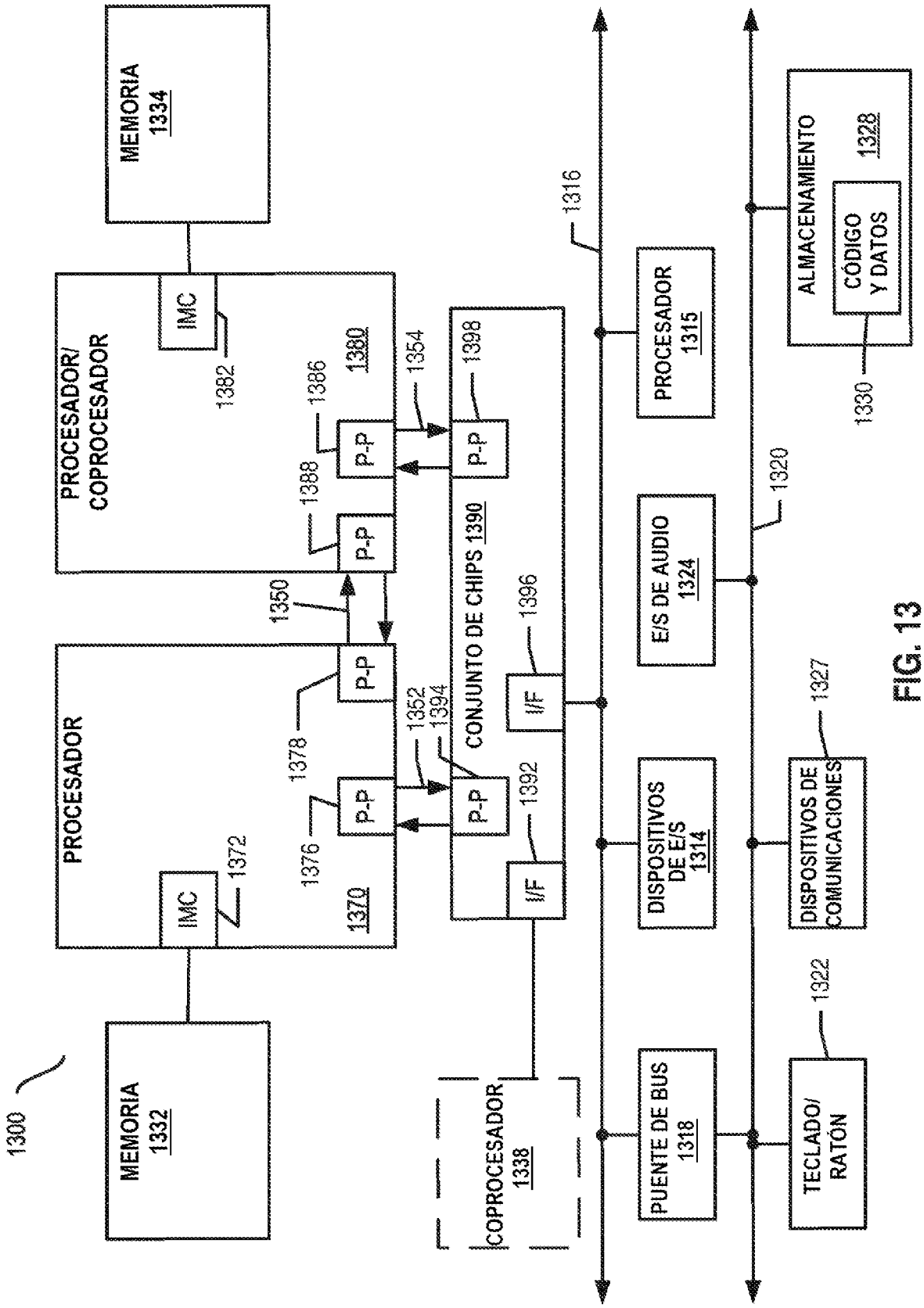

1300
MEMORIA 1332
MEMORIA 1334
PROCESADOR
IMC
1372
1370
1376
1378
P-P
1352
1394
PROCESADOR/ COPROCESADOR
IMC
1382
1380
1386
1388
1350
1354
CONJUNTO DE CHIPS 1390
1398
1396
I/F
1392
COPROCESADOR 1338
1316
PROCESADOR 1315
E/S DE AUDIO 1324
DISPOSITIVOS DE E/S 1314
PUENTE DE BUS 1318
1320
ALMACENAMIENTO
CÓDIGO Y DATOS
1328
1330
1327
DISPOSITIVOS DE COMUNICACIONES
1322
TECLADO/ RATÓN

FIG. 13

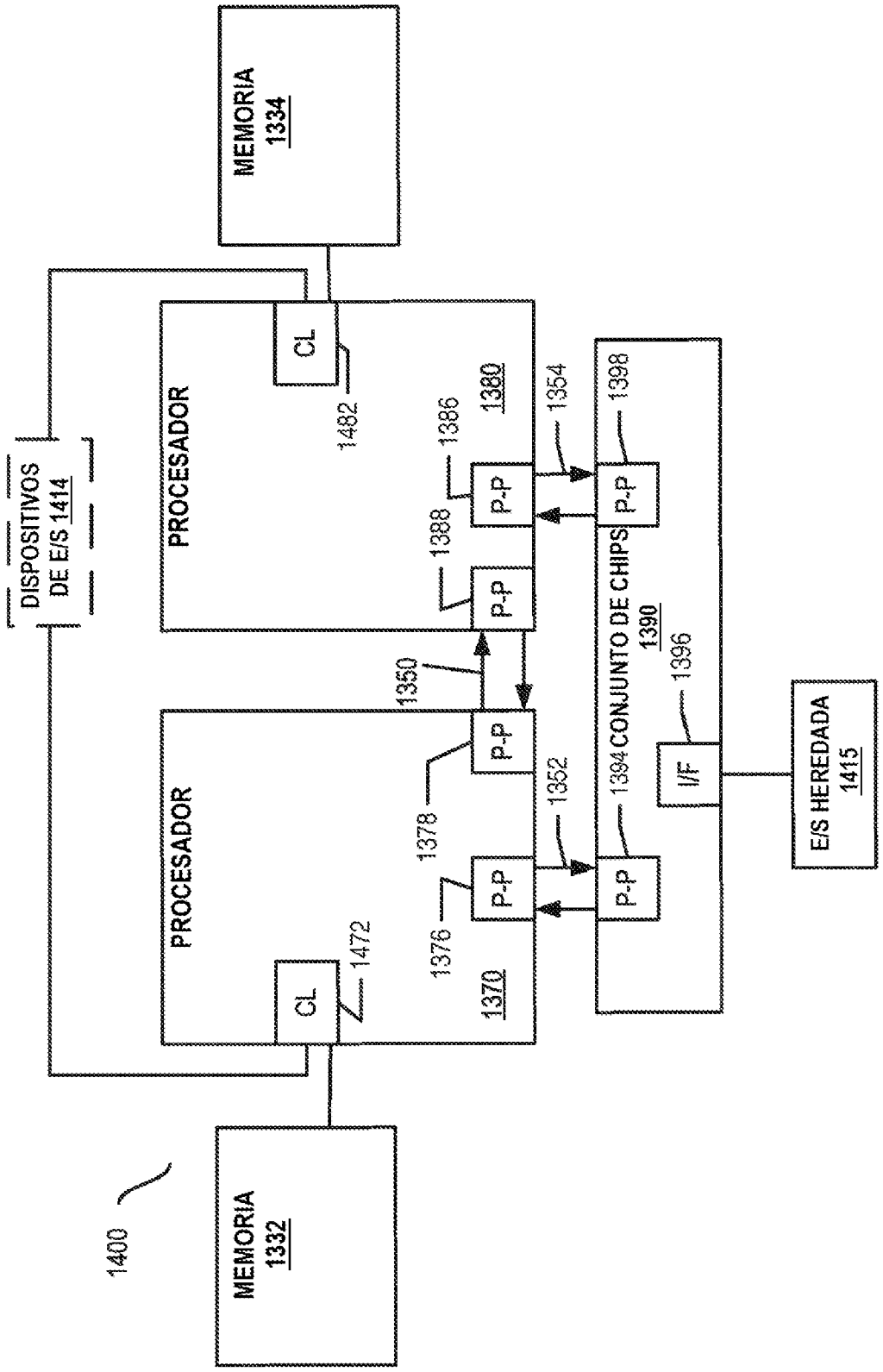

1400
MEMORIA 1332
MEMORIA 1334
DISPOSITIVOS DE E/S 1414
PROCESADOR
CL
1472
1482
1370
1380
P-P
1376
1378
1386
1388
1350
1352
1354
1394 CONJUNTO DE CHIPS 1390
1396
1398
I/F
E/S HEREDADA 1415

FIG. 14

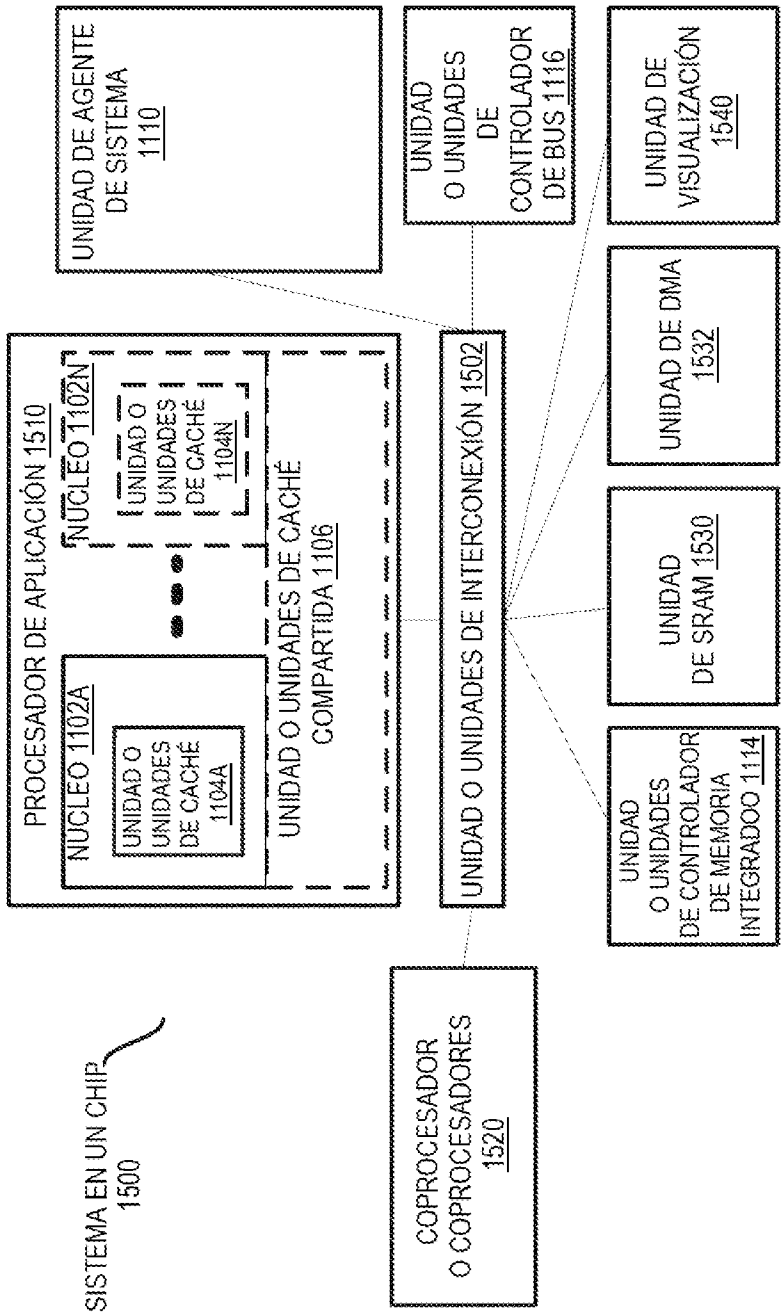

SISTEMA EN UN CHIP
1500
PROCESADOR DE APLICACIÓN 1510
NUCLEO 1102A
UNIDAD O UNIDADES DE CACHÉ 1104A
NÚCLEO 1102N
UNIDAD O UNIDADES DE CACHÉ 1104N
UNIDAD O UNIDADES DE CACHÉ COMPARTIDA 1106
UNIDAD DE AGENTE DE SISTEMA 1110
UNIDAD O UNIDADES DE CONTROLADOR DE BUS 1116
UNIDAD DE VISUALIZACIÓN 1540
UNIDAD DE DMA 1532
UNIDAD DE SRAM 1530
UNIDAD O UNIDADES DE CONTROLADOR DE MEMORIA INTEGRADOO 1114
UNIDAD O UNIDADES DE INTERCONEXIÓN 1502
COPROCESADOR O COPROCESADORES 1520

FIG. 15

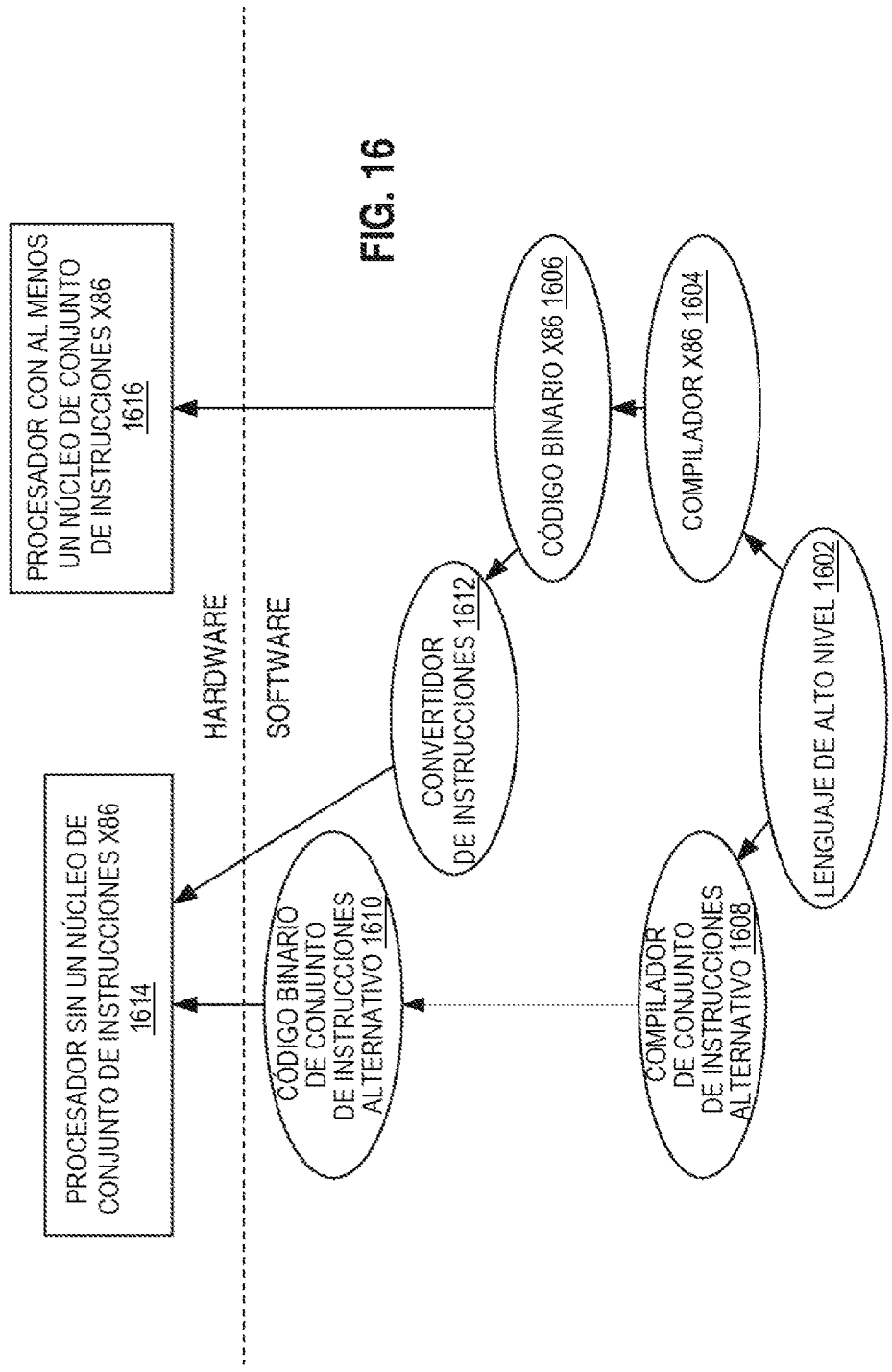

PROCESADOR CON AL MENOS UN NÚCLEO DE CONJUNTO DE INSTRUCCIONES X86 1616
HARDWARE
SOFTWARE
PROCESADOR SIN UN NÚCLEO DE CONJUNTO DE INSTRUCCIONES X86 1614
CÓDIGO BINARIO DE CONJUNTO DE INSTRUCCIONES ALTERNATIVO 1610
CONVERTIDOR DE INSTRUCCIONES 1612
CÓDIGO BINARIO X86 1606
COMPILADOR DE CONJUNTO DE INSTRUCCIONES ALTERNATIVO 1608
COMPILADOR X86 1604
LENGUAJE DE ALTO NIVEL 1602

FIG. 16