(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 993 213**

(51) Int. Cl.:

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 21166159 (0)**

(97) Fecha y número de publicación de la concesión europea: **28.08.2024 EP 3866001**

(54) Título: **Sistemas y métodos para realizar instrucciones de producto escalar de vectores en coma flotante de 16 bits**

(30) Prioridad:

09.11.2018 US 201816186378

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
26.12.2024

(73) Titular/es:

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

(72) Inventor/es:

HEINECKE, ALEXANDER F.;
VALENTINE, ROBERT R.;
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 de producto escalar de vectores en 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 de producto escalar de vectores en coma flotante de 16 bits.

ESTADO DE LA TÉCNICA ANTERIOR

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 registros, 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 instrucción. 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 se va a realizar 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 aparición de una instrucción en un formato de instrucción.

Las aplicaciones científicas, financieras, autovectorizadas 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 (lo que se denomina "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 simple de 32 bits separados. Como otro ejemplo, los bits de un registro de 256 bits se pueden especificar 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 vectorial es un operando de origen o de destino de una instrucción de SIMD (también conocida como instrucción de datos empaquetados o instrucción vectorial).

A modo de ejemplo, un tipo de instrucción de SIMD especifica una única operación vectorial que se va a realizar en dos operandos vectoriales de origen en forma vertical para generar un operando vectorial 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 vectoriales de origen se denominan elementos de datos de origen, mientras que los elementos de datos en el operando vectorial de destino se denominan elementos de datos de destino o resultado. Estos operandos vectoriales 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 bit en los dos operandos vectoriales 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 de origen se corresponde, el elemento de datos en la posición de elemento de datos 1 de cada operando de origen se 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 vectorial resultante tiene el mismo tamaño, tiene el mismo número de elementos de datos y los elementos de datos resultantes se almacenan en el mismo orden de elementos de datos que los operandos vectoriales de origen, los elementos de datos resultantes están en las mismas posiciones de bit del operando vectorial resultante que su correspondiente par de elementos de datos de origen en los operandos vectoriales de origen. Además de este tipo ilustrativo de instrucción de SIMD, existe una diversidad de otros tipos de instrucciones de SIMD.

La multiplicación por producto escalar de vectores que contienen elementos de coma flotante de 16 bits es útil en una serie de algoritmos que realizan la multiplicación en orígenes de 16 bits y acumulan los resultados de la multiplicación con elementos vectoriales de destino de 32 bits.

El documento EP 3 620 910 A1 (estado de la técnica de conformidad con el artículo 54(3) EPC) se refiere al formato FP16-S7E8 de precisión mixta para el aprendizaje profundo y otros algoritmos. Por ejemplo, un procesador incluye

circuitos de extracción para extraer una instrucción de compresión que tiene campos para especificar ubicaciones de un vector de origen que tiene N elementos formateados con precisión simple y un vector comprimido que tiene N elementos formateados con precisión neuronal media (NHP), circuitos de descodificación para descodificar la instrucción de compresión extraída, circuitos de ejecución para responder a la instrucción de compresión descodificada mediante un proceso de conversión de cada elemento del vector de origen al formato NHP y de escritura de cada elemento convertido en un elemento vectorial comprimido correspondiente, en donde el procesador, además, extrae, descodifica y ejecuta una instrucción MPVMAC para multiplicar los elementos correspondientes con formato NHP usando un multiplicador de 16 bits, y acumula cada uno de los productos con los contenidos anteriores de un destino correspondiente usando un acumulador de 32 bits.

El documento EP 3 396 524 A1 se refiere a un procesamiento de datos por medio de una unidad de procesamiento gráfico de propósito general. Por ejemplo, un acelerador de hardware de aprendizaje automático comprende una unidad de cálculo que tiene un sumador y un multiplicador que se comparten entre una ruta de datos de números enteros y una ruta de datos de coma flotante, y los bits superiores de los operandos de entrada al multiplicador se controlan durante la operación de coma flotante.

El documento "ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile", ARM Architecture Reference ManualARMv8, for ARMv8-A architecture profile, GB, páginas 1-6.354, describe la arquitectura ARM® v8 y el funcionamiento de un elemento de procesamiento ARMv8-A.

El documento de KESTER WALT (EDITOR) ED - KESTER WALT (EDITOR), "Mixed-signal and DSP design techniques", MIXED-SIGNAL AND DSP DESIGN TECHNIQUES; [ANALOG DEVICES SERIES], NEWNES, AMSTERDAM [U.A.], páginas 1-418, aborda el procesamiento de señales del mundo real usando técnicas analógicas y digitales.

El documento CN 107 038 016 A se refiere a un método de conversión de números en coma flotante y a un aparato basado en GPU (unidad de procesamiento gráfico). Por ejemplo, el método comprende las etapas de que un primer número en coma flotante se convierte en un número en coma flotante dentro de un intervalo preestablecido, y se obtiene un segundo número en coma flotante, en donde el primer número en coma flotante es un número en coma flotante de 32 bits a convertir, y el intervalo preestablecido es un intervalo continuo compuesto por números en coma flotante con bits de signo y bits de índice idénticos; el segundo número de coma flotante se convierte en un entero sin signo objetivo, y los valores en las posiciones, correspondientes a los bits de signo y los bits de índice del primer número en coma flotante, en el entero sin signo objetivo son todos 0; y se determina un número en coma flotante de 16 bits en base al entero sin signo objetivo para completar la conversión del primer número en coma flotante.

SUMARIO

La presente invención está definida por la reivindicación independiente 1. 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 la ejecución de una instrucción VDPBF16PS, de acuerdo con una realización;

La **Figura 2** es un diagrama de bloques que ilustra la ejecución de una instrucción VDPBF16PS, de acuerdo con una realización;

La **Figura 3A** es un pseudocódigo que ilustra la ejecución ilustrativa de una instrucción VDPBF16PS, de acuerdo con una realización;

La **Figura 3B** es un pseudocódigo que ilustra la ejecución ilustrativa de una instrucción VDPBF16PS, 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 4** es un diagrama de flujo de proceso que ilustra la respuesta de un procesador a una instrucción VDPBF16PS, de acuerdo con una realización;

La **Figura 5** es un diagrama de bloques que ilustra un formato de una instrucción VDPBF16PS, de acuerdo con una realización;

Las **Figuras 6A-6B** son diagramas de bloques que ilustran un formato de instrucción genérico compatible con vectores 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 genérico compatible con vectores 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 genérico compatible con vectores 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 específico compatible con vectores 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 específico compatible con vectores 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 específico compatible con vectores 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 específico compatible con vectores 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 registros de acuerdo con una realización;

La **Figura 9A** es un diagrama de bloques que ilustra tanto una canalización ilustrativa en orden como una canalización ilustrativa de emisión/ejecución desordenada y de 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 ejemplo más específico de un diagrama de bloques de una arquitectura de núcleo en orden, cuyo núcleo sería uno de varios bloques lógicos (que incluyen otros núcleos del mismo tipo y/o tipos diferentes) en un chip;

La **Figura 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 memoria caché de Nivel 2 (L2), de acuerdo con algunas realizaciones;

La **Figura 10B** es una vista ampliada de parte del núcleo de procesador de 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 de un conjunto de instrucciones de origen en instrucciones binarias de 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 se pueden poner en práctica sin estos detalles específicos. En otros casos, no se han mostrado en detalle circuitos, estructuras y técnicas bien conocidos para no complicar la comprensión de esta descripción.

Las referencias en la memoria descriptiva a "una realización", "una realización ilustrativa", etc., indican que la realización descrita puede incluir un rasgo, estructura o característica, pero cada realización no tiene que incluir necesariamente 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 en relación con otras realizaciones si se describe explícitamente.

Como se ha mencionado anteriormente, la multiplicación por producto escalar de vectores que contienen elementos de coma flotante de 16 bits, acumulando los resultados con elementos vectoriales de destino de 32 bits, es útil en una serie de algoritmos. En el presente documento se divulga e ilustra mediante las figuras una instrucción de datos empaquetados vectoriales (VDPBF16PS, una nomenclatura que denota "VDP"= VDPBF16PS **V**ector **D**ot **P**roduct, "BF16" = orígenes formateados como **BF**loat**16**, para acumular productos con una precisión "**P**acked **S**ingle" (precisión simple empaquetada) que implementa la multiplicación por producto escalar de pares de elementos de coma flotante de 16 bits en dos vectores de origen. La instrucción toma dos vectores de origen que tienen pares de valores bfloat16 por elemento (es decir, los pares de elementos de 16 bits forman elementos de origen de 32 bits) y genera un vector de destino que tiene elementos de precisión simple (es decir, también 32 bits, por lo que los registros de origen y de destino están equilibrados). La salida de la multiplicación es de 32 bits y se satura y acumula con los contenidos anteriores del registro de destino.

En comparación con los algoritmos que usan precisión simple tanto para los elementos de origen como los de destino, cabe esperar que la instrucción VDPBF16PS divulgada logre una calidad comparable, pero con una utilización de memoria y unos requisitos de ancho de banda de memoria reducidos, lo que serviría para mejorar el rendimiento y la eficacia energética, especialmente en un contexto de aprendizaje automático.

Como se ha mencionado anteriormente, la instrucción VDPBF16PS divulgada realiza operaciones de producto escalar en vectores multiplicando elementos de coma flotante de 16 bits (por ejemplo, bfloat16) de dos registros de origen y acumulando los resultados en un vector de destino de precisión simple. Sin la instrucción VDPBF16PS divulgada, realizar esta funcionalidad con las instrucciones existentes requeriría una secuencia de instrucciones para convertir los elementos de datos, realizar las multiplicaciones y acumular los resultados de la multiplicación con los datos de destino con saturación. En cambio, las realizaciones divulgadas implementan esta funcionalidad en una instrucción.

FORMATOS DE COMA FLOTANTE RELEVANTES

Los formatos de coma flotante de 16 bits usados por las realizaciones divulgadas incluyen bfloat16 (definido por Google, Inc., de Mountain View, California), en ocasiones denominado en el presente documento "bf16" o "BF16", y binary16 (promulgado como IEEE754-2008 por el Instituto de Ingenieros Eléctricos y Electrónicos), en ocasiones denominado en el presente documento "precisión media" o "fp16". Los formatos de coma flotante de 32 bits usados en las realizaciones divulgadas incluyen binary32 (también promulgado como parte del IEEE754-2008), en ocasiones denominado en el presente documento "precisión simple" 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. Los formatos binary32, binary16 y 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. Para que quepa el significando, cada uno de los formatos binary32, binary16 y bfloat16 usa un bit implícito para el significando, incluyéndose expresamente el resto de bits del significando. 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
Binary32	32	1	8 bits	24 bits
Binary16	16	1	5 bits	11 bits
Bfloat16	16	1	8 bits	8 bits

Un procesador que implemente la instrucción VDPBF16PS divulgada incluiría circuitos de extracción para extraer una instrucción que tenga campos para especificar un código de operación y las ubicaciones de los vectores de primer origen, de segundo origen y de destino. El formato de la instrucción VDPBF16PS se ilustra y describe con más detalle

al menos con respecto a las Figuras **5**, **6A-B** y **7A-D**. Los vectores de origen y destino especificados pueden estar ubicados en registros vectoriales o en memoria. El código de operación hace que el procesador multiplique N pares de elementos formateados en coma flotante de 16 bits (por ejemplo, bfloat16) de los primer y segundo orígenes especificados, siendo N en algunas realizaciones uno cualquiera de 4, 8 y 16 (aunque la invención no establece un límite superior de N, que podría ser 32, 64 o mayor), y estando asociado a vectores de origen que tengan 1024, 2048 o más bits. En algunas formas de realización, N es un número par mayor que 4 y acumula cada uno de los productos resultantes con contenidos anteriores de un elemento correspondiente de precisión simple (es decir, FP32 o binary32) del destino especificado. Un procesador de este tipo incluiría además circuitos de descodificación para descodificar la instrucción extraída y circuitos de ejecución para responder a la instrucción descodificada según especifica el código de operación. Los circuitos de ejecución se describen e ilustran con más detalle a continuación, al menos con respecto a las **Figuras 1-4**, **9A-B** y **10A-B**.

La **Figura 1** es un diagrama de bloques que ilustra componentes de procesamiento para la ejecución de una instrucción VDPBF16PS, de acuerdo con algunas realizaciones. Como se ilustra, el almacenamiento 101 almacena una o más instrucciones VDPBF16PS 103 a ejecutar. En algunas formas de realización, el sistema informático 100 es un procesador de SIMD para procesar simultáneamente múltiples elementos de vectores de datos empaquetados, tales como matrices.

En funcionamiento, las una o más instrucciones VDPBF16PS 103 se extraen del almacenamiento 101 mediante circuitos de extracción 105. Como se describe más adelante, al menos con respecto a las **Figuras 2 y 5**, la instrucción VDPBF16PS tiene campos para especificar un código de operación y las ubicaciones de los vectores de primer origen, de segundo origen y de destino, donde el código de operación indica que los circuitos de ejecución deben multiplicar N pares de elementos formateados en coma flotante de 16 bits de los primer y segundo orígenes especificados, y acumular los productos resultantes con los contenidos anteriores de un elemento correspondiente de precisión simple del destino especificado. Los circuitos de descodificación 109 descodifican la instrucción VDPBF16PS extraída 107. El formato de la instrucción VDPBF16PS, que se ilustra y describe con más detalle al menos con respecto a las **Figuras 5**, **6A-B** y **7A-D**, tiene campos (no mostrados aquí) que especifican las ubicaciones de los vectores de primer origen, de segundo origen y de destino. Los circuitos de descodificación 109 descodifican la instrucción VDPBF16PS extraída 107 en una o más operaciones. En algunas realizaciones, esta descodificación incluye la generación de una pluralidad de microoperaciones que realizarán los circuitos de ejecución (tales como los circuitos de ejecución 117). Los circuitos de descodificación 109 también descodifican sufijos y prefijos de instrucciones (si se usan). Los circuitos de ejecución 117, que tienen acceso al archivo de registros y a la memoria 115, deben responder a la instrucción descodificada 111 según lo especificado por el código de operación, y se describen e ilustran con más detalle a continuación, al menos con respecto a las **Figuras 3-4**, **9A-B** y **10A-B**.

En algunas realizaciones, el circuito de cambio de nombre, asignación y/o planificación de registros 113 proporciona una o más de las siguientes funcionalidades: 1) renombrar los valores de operandos lógicos como valores de operandos físicos (por ejemplo, una tabla de solapamiento de registros en algunas realizaciones), 2) asignar bits de estado e indicadores a la instrucción descodificada, y 3) planificar la instrucción VDPBF16PS descodificada 111 para su ejecución en los circuitos de ejecución 117 a partir de un conjunto 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 pueden activarse en momentos diferentes, o no activarse en absoluto.

La **Figura 2** es un diagrama de bloques que ilustra la ejecución de una instrucción VDPBF16PS, de acuerdo con una realización. Tal y como se muestra, el aparato informático 200 (por ejemplo, un procesador) debe recibir, extraer y descodificar (los circuitos de extracción y descodificación no se muestran aquí, pero se ilustran y describen al menos con respecto a la **Figura 1** y las **Figuras 9A-B**) la instrucción VDPBF16PS 201, que incluye campos que especifican el código de operación 202 (VDPBF16PS) y las ubicaciones de los vectores de destino 204, de primer origen 206 y de segundo origen 208. El formato de la instrucción VDPBF16PS 201 se ilustra y describe con más detalle al menos con respecto a las **Figuras 5**, **6A-B** y **7A-D**. También se muestran los vectores de primer y segundo origen especificados 212A y 212B, los circuitos de ejecución 214, que incluyen multiplicadores 215A, 215B y un acumulador 216, y el vector de destino especificado 218.

En funcionamiento, el aparato informático 200 (por ejemplo, un procesador), debe extraer y descodificar, mediante circuitos de extracción y descodificación (no mostrados), una instrucción 201 que tiene campos que especifican el código de operación 202 y las ubicaciones de los vectores de primer origen 206, de segundo origen 208 y de destino 204, donde el código de operación indica que el aparato informático (por ejemplo, un procesador) debe multiplicar N pares de elementos de coma flotante de 16 bits (por ejemplo, bfloat16 o binary16) de los primer y segundo orígenes especificados, y acumular los productos resultantes con contenidos anteriores de un elemento correspondiente de precisión simple (por ejemplo, binary32) del destino especificado. En este caso, se muestra que N es igual a 8, por lo que los primer y segundo orígenes especificados son vectores de 256 bits, que pueden estar ubicados en registros vectoriales o en memoria, que contienen 8 pares de números formateados en coma flotante de 16 bits. Como se ilustra

y describe con más detalle al menos con respecto a las **Figuras 5**, **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. En este caso, los circuitos de ejecución 214 deben responder a la instrucción descodificada como se especifica mediante el código de operación 202. En algunas realizaciones, los circuitos de ejecución 214 saturan, según sea necesario, los resultados de los multiplicadores 215A y 215B y del acumulador 216. En algunas realizaciones, los circuitos de ejecución realizan las multiplicaciones con precisión infinita sin saturación y saturan los resultados de la acumulación a más o menos infinito en caso de un desbordamiento y a cero en caso de cualquier subdesbordamiento. En algunas realizaciones, los circuitos de ejecución 214 generan todos los N elementos de destino en paralelo.

La **Figura 3A** es un pseudocódigo que ilustra la ejecución ilustrativa de una instrucción VDPBF16PS, de acuerdo con una realización. Como se muestra en el pseudocódigo 300, la función VDPBF16PS tiene campos que especifican vectores de primer y segundo origen src1 y src2, que pueden tener una longitud cualquiera de 128 bits, 256 bits y 512 bits, y un destino, srcdest, que también sirve como origen para la acumulación. El pseudocódigo 300 también muestra el uso de una máscara de escritura para controlar si se debe enmascarar cada uno de los elementos de destino, donde los elementos enmascarados se ponen a cero o se fusionan. (Como se ilustra y describe con más detalle al menos con respecto a las **Figuras 5**, **6A-B** y **7A-D**, la instrucción VDPBF16PS en algunas realizaciones incluye campos que especifican la máscara y determinan si debe realizarse una puesta a cero o una fusión). La ejecución de la instrucción VDPBF16PS se ilustra y describe con más detalle al menos con respecto a las **Figuras 2**, **3B**, **4** y **9A-B**.

La **Figura 3B** es un pseudocódigo que ilustra la ejecución ilustrativa de una instrucción VDPBF16PS, de acuerdo con una realización. Como se muestra, el pseudocódigo 310 es similar al pseudocódigo 300 (**Figura 3A**), pero acumula el producto de los elementos de origen impares con el destino antes que el de los elementos pares.

La **Figura 3C** es un pseudocódigo que ilustra una función auxiliar para su uso con el pseudocódigo de las **Figura 3A**, de acuerdo con una realización. En este caso, el pseudocódigo 320 define una función auxiliar, make_fp32 (x), que convierte del formato bfloat16 al formato binary32. El código ilustra al menos una ventaja del formato bfloat16 sobre el formato binary16 o de media precisión; en concreto, la conversión puede simplemente empaquetar los 16 bits del número bfloat16 en los 16 bits superiores de una palabra doble, poniendo a cero los 16 bits inferiores. La conversión inversa se puede realizar simplemente truncando los 16 bits inferiores del número binary32.

La **Figura 4** es un diagrama de flujo de proceso que ilustra la respuesta de un procesador a una instrucción VDPBF16PS, de acuerdo con una realización. Como se muestra en el flujo 400, en 401, el procesador extrae, mediante circuitos de extracción, una instrucción que tiene campos que especifican un código de operación y las ubicaciones de los vectores de primer origen, de segundo origen y de destino, donde el código de operación indica que los circuitos de ejecución deben multiplicar N pares de elementos de precisión media de los primer y segundo orígenes especificados, y acumular los productos resultantes con los contenidos anteriores de un elemento correspondiente de precisión simple del destino especificado. En 403, el procesador ha de descodificar, mediante circuitos de descodificación, la instrucción extraída. En algunas realizaciones, en 405, el procesador planifica la ejecución de la instrucción descodificada como se especifica mediante el código de operación. En 407, el procesador ha de responder, mediante circuitos de ejecución, a la instrucción descodificada. En algunas realizaciones, los circuitos de ejecución saturan los resultados de las multiplicaciones y la acumulación, según sea necesario. En algunas realizaciones, los circuitos de ejecución realizan las multiplicaciones con precisión infinita sin saturación y saturan los resultados de la acumulación a más o menos infinito en caso de un desbordamiento y a cero en caso de cualquier subdesbordamiento. En algunas realizaciones, en 409, el procesador ha de confirmar un resultado de la instrucción ejecutada.

Las operaciones 405 y 409 son opcionales, según se indica por sus bordes de línea discontinua, en la medida en que pueden realizarse en un momento diferente, o no realizarse en absoluto.

La **Figura 5** es un diagrama de bloques que ilustra un formato de una instrucción VDPBF16PS, de acuerdo con una realización. Como se muestra, la instrucción VDPBF16PS 500 incluye campos que especifican un código de operación 502 y ubicaciones de los vectores de destino 504, primer origen 506 y segundo origen 508. Cada uno de los vectores de origen y de destino se pueden ubicar en registros o en memoria. Un ejemplo de código de operación 502 es VDPBF16PS*.

Como se muestra, el código de operación 502 incluye un asterisco, lo que significa que se pueden añadir diversos campos opcionales, como prefijos o sufijos, al código de operación. En concreto, la instrucción VDPBF16PS 500 incluye, además, parámetros opcionales que afectan al comportamiento de la instrucción, lo que incluye una máscara {k} 510, la puesta a cero {Z} 512, un formato de elemento 514 y un tamaño de vector (N) 516. Uno o más de los modificadores de instrucción 510, 512, 514 y 516 se pueden especificar usando prefijos o sufijos para el código de operación 502.

En algunas realizaciones, uno o más de los modificadores de instrucción opcionales 510, 512, 514 y 516 están codificados en un campo de valor inmediato (no mostrado) incluido de manera opcional en la instrucción 500. Por ejemplo, cada uno de los modificadores de instrucción opcionales 510, 512, 514 y 516 se puede indicar mediante una porción de 8 bits diferente de un valor inmediato de 32 bits. En algunas realizaciones, uno o más de los modificadores

de instrucción opcionales 510, 512, 514 y 516 se especifican por medio de un registro de configuración, tal como registros específicos de modelo (MSR) incluidos en la arquitectura de conjunto de instrucciones.

El formato de la instrucción VDPBF16PS 500 se ilustra y describe con más detalle al menos con respecto a las Figuras **6A-B** y **7A-D**.

CONJUNTOS DE INSTRUCCIONES

Un conjunto de instrucciones puede incluir uno o más formatos de instrucción. Un formato de instrucción dado puede definir diversos campos (por ejemplo, número de bits, ubicación de 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 se pueden definir 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 se incluyen menos campos) y/o definir para tener un campo dado interpretado de manera diferente. Por tanto, cada instrucción de una ISA se expresa mediante un formato de instrucción dado (y, si está definida, en una plantilla dada de las plantillas de instrucción de ese formato de instrucción) e incluye campos para especificar la operación y los operandos. Por ejemplo, una instrucción ADD ilustrativa tiene un código de operación específico y un formato de instrucción que incluye un campo de código de operación para especificar ese código de operación y campos de operando para seleccionar operandos (origen1/destino y origen2); y una aparición de esta instrucción ADD en un flujo de instrucciones tendrá contenido específico en los campos de operando 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 el 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 de Intel®, octubre de 2014).

FORMATOS DE INSTRUCCIÓN ILUSTRATIVOS

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

FORMATO DE INSTRUCCIÓN GENÉRICO COMPATIBLE CON VECTORES

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

Las **Figuras 6A-6B** son diagramas de bloques que ilustran un formato de instrucción genérico compatible con vectores 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 genérico compatible con vectores 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 genérico compatible con vectores y las plantillas de instrucción de clase B del mismo de acuerdo con algunas realizaciones de la invención. Específicamente, para un formato de instrucción genérico compatible con vectores 600 se definen plantillas de instrucción de clase A y clase B, las cuales incluyen plantillas de instrucción sin acceso a memoria 605 y plantillas de instrucción de 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 admite lo siguiente: una longitud (o tamaño) de operando vectorial de 64 bytes con anchuras (o tamaños) de elementos de datos de 32 bits (4 bytes) o 64 bits (8 bytes) (y, por lo tanto, un vector de 64 bytes consiste en 16 elementos de tamaño de palabra doble o, como alternativa, 8 elementos de tamaño de palabra cuádruple); una longitud (o tamaño) de operando vectorial de 64 bytes con anchuras (o tamaños) de elementos de datos de 16 bits (2 bytes) u 8 bits (1 byte); una longitud (o tamaño) de operando vectorial de 32 bytes con anchuras (o tamaños) de elementos de datos de 32 bits (4 bytes), 64 bits (8 bytes), 16 bits (2 bytes) u 8 bits (1 byte); y una longitud (o tamaño) de operando vectorial de 16 bytes con anchuras (o tamaños) de elementos de datos de 32 bits (4 bytes), 64 bits (8 bytes), 16 bits (2 bytes) u 8 bits (1 byte); realizaciones alternativas pueden admitir más, menos y/o diferentes tamaños de operandos vectoriales (por ejemplo, operandos vectoriales de 256 bytes) con más, menos o diferentes anchuras de elementos de datos (por ejemplo, anchuras de elementos de datos de 128 bits (16 bytes)).

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

El formato de instrucción genérico compatible con vectores 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 apariciones de instrucciones en el formato de instrucción compatible con vectores en los flujos de instrucciones. Por tanto, este campo es opcional en el sentido de que no es necesario para un conjunto de instrucciones que solo tiene el formato de instrucción genérico compatible con vectores.

Campo de operación base 642: su contenido distingue diferentes operaciones 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 admitir más o menos registros de origen y de destino (por ejemplo, pueden admitir hasta dos orígenes, donde uno de estos orígenes también actúa como el destino, pueden admitir hasta tres orígenes, donde uno de estos orígenes también actúa como el destino, pueden admitir hasta dos orígenes y un destino).

Campo de modificador 646: su contenido distingue apariciones de instrucciones en el formato de instrucción de vector genérico que especifican un acceso a memoria de aquellas que no lo hacen; es decir, entre las plantillas de instrucción sin acceso a memoria 605 y las plantillas de instrucción de acceso a memoria 620. Las operaciones de acceso a memoria leen y/o escriben en la jerarquía de memoria (especificando, en algunos casos, las direcciones de origen y/o de destino usando valores en registros), mientras que las operaciones sin acceso a memoria no lo hacen (por ejemplo, el origen y los destinos son registros). Si bien en una realización este campo también selecciona entre tres formas diferentes de realizar cálculos de direcciones de memoria, realizaciones alternativas pueden admitir 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 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 usa $2^{escala} * índice + base$).

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 usa $2^{escala} * índice + base + desplazamiento\ escalado$). Los bits redundantes de bajo orden se ignoran y, por lo tanto, el contenido del campo de factor de desplazamiento se multiplica por el tamaño total de los operandos de memoria (N) para generar el desplazamiento final que se usará para calcular una dirección efectiva. El valor de N está determinado por el hardware del procesador en tiempo de ejecución en función del 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 se va a usar (en algunas realizaciones, para todas las instrucciones; en otras realizaciones, solo para algunas de las instrucciones). Este campo es opcional en el sentido de que no es necesario si únicamente se admite una

anchura de elemento de datos y/o se admiten anchuras de elementos de datos usando algún aspecto de los códigos de operación.

Campo de máscara de escritura 670: su contenido controla, para cada posición de elemento de datos, si esa posición de elemento de datos en el operando de vector de destino refleja el resultado de la operación base y la operación de aumento. Las plantillas de instrucción de clase A admiten el enmascaramiento de escritura de fusión, mientras que las plantillas de instrucción de clase B admiten el enmascaramiento de escritura tanto de fusión como de puesta a cero. Cuando se fusionan, las máscaras vectoriales permiten proteger de actualizaciones cualquier conjunto de elementos en el destino durante la ejecución de cualquier operación (especificada por la operación base y la operación de aumento); en otra realización, se conserva 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 base y la operación de aumento), en una realización, un elemento del destino se establece a 0 cuando el bit de máscara correspondiente tiene un valor 0. Un subconjunto de esta funcionalidad es la capacidad de controlar la longitud de vector de la operación que se realiza (es decir, el intervalo de elementos que se modifican, desde el primero hasta el último); sin embargo, no es necesario que los elementos que se modifican sean consecutivos. Por lo tanto, el campo de máscara de escritura 670 permite operaciones vectoriales parciales, incluidas cargas, almacenamientos, aritmética, lógica, etc. Si bien se describen realizaciones de la invención en las que el contenido del campo de máscara de escritura 670 selecciona uno de una pluralidad de registros de máscara de escritura que contiene la máscara de escritura que se va a usar (y, por lo tanto, el contenido del campo de máscara de escritura 670 identifica indirectamente que se va a realizar el enmascaramiento), realizaciones alternativas permiten, en cambio o de manera adicional, que el contenido del campo de escritura de máscara 670 especifique directamente el enmascaramiento que se va a realizar.

Campo de valor inmediato 672: su contenido permite especificar un valor inmediato. Este campo es opcional en el sentido de que no está presente en una implementación del formato compatible con vectores genérico que no admita valores inmediatos y no está presente en instrucciones que no usen un valor 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 usan 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 INSTRUCCIÓN DE CLASE A

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

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

En la plantilla de instrucción de operación de tipo control de redondeo completo sin acceso a memoria 610, el campo beta 654 se interpreta como un campo de control de redondeo 654A, cuyo contenido o contenidos proporcionan redondeo estático. Por otro lado, 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, donde realizaciones alternativas pueden admitir 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 inhabilitar o no la notificación de eventos de excepción; cuando el contenido del campo de SAE 656 indica que la supresión está habilitada, una instrucción dada no notifica ningún tipo de indicador de excepción de coma flotante y no genera ningún manejador de excepciones de coma flotante.

Campo de control de operación de redondeo 658: su contenido distingue cuál de un grupo de operaciones de redondeo se realizará (por ejemplo, redondeo hacia arriba, redondeo hacia abajo, redondeo hacia cero y redondeo hacia el valor más cercano). Por tanto, el campo de control de operación de redondeo 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 INSTRUCCIÓN SIN ACCESO A MEMORIA - OPERACIÓN DE TIPO TRANSFORMADA DE DATOS

En la plantilla de instrucción de operación de tipo transformada de datos sin acceso a memoria 615, el campo beta 654 se interpreta como un campo de transformada de datos 654B, cuyo contenido distingue cuál de un número de transformaciones de datos se va a realizar (por ejemplo, sin transformada 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 expulsión 652B, cuyo contenido distingue cuál de las sugerencias de expulsión se va a usar (en la **Figura 6A**, temporal 652B.1 y no temporal 652B.2 se especifican, respectivamente, para la plantilla de instrucción temporal de acceso a memoria 625 y la plantilla de instrucción no temporal de acceso a memoria 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 de 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 cargan vectores desde la memoria y almacenan vectores en la misma, con compatibilidad de conversión. Al igual que con las instrucciones vectoriales habituales, las instrucciones de memoria vectoriales 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 INSTRUCCIÓN DE ACCESO A MEMORIA - TEMPORAL

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

PLANTILLAS DE INSTRUCCIÓN DE ACCESO A MEMORIA - NO TEMPORAL

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

PLANTILLAS DE INSTRUCCIÓN 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 va a realizar (por ejemplo, el redondeo 657A.1 y la longitud de vector (VSIZE) 657A.2 se especifican, respectivamente, para la plantilla de instrucción de operación de tipo control de redondeo parcial, de control de máscara de escritura, sin acceso a memoria 612 y la plantilla de instrucción de operación de tipo VSIZE, de control de máscara de escritura, sin acceso a memoria 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, no están presentes el campo de escala 660, el campo de desplazamiento 662A ni el campo de escala de desplazamiento 662B.

En la plantilla de instrucción de operación de tipo control de redondeo parcial, de control de máscara de escritura, sin acceso a memoria 610, el resto del campo beta 654 se interpreta como un campo de operación de redondeo 659A y se inhabilita la notificación de eventos de excepción (una instrucción dada no notifica tipo alguno de indicador de excepción de coma flotante y no genera manejador de excepciones de coma flotante alguno).

Campo de control de operación de redondeo 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 valor más cercano). Por tanto, el campo de control de operación de redondeo 659A permite cambiar el modo de redondeo por cada 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, de control de máscara de escritura, sin acceso a memoria 617, el resto del campo beta 654 se interpreta como un campo de longitud de vector 659B, cuyo contenido distingue sobre cuál de varias longitudes de vector de datos se va a realizar una operación (por ejemplo, 128, 256 o 512 bytes).

En el caso de una plantilla de instrucción de 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 de vector 659B. Las plantillas de instrucción de 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 genérico compatible con vectores 600, se muestra un campo de código de operación completo 674 que incluye el campo de formato 640, el campo de operación 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 admiten 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 por cada instrucción en el formato de instrucción genérico compatible con vectores.

La combinación del campo de máscara de escritura y el campo de anchura de elemento de datos crea instrucciones de un tipo concreto que permiten que se aplique la máscara en función de 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 admitir 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 admitir solo la clase B, un núcleo destinado principalmente a gráficos y/o computación científica (de capacidad de procesamiento) puede admitir solo la clase A, y un núcleo destinado a ambos fines puede admitir ambas clases (por supuesto, un núcleo que tenga alguna combinación de plantillas e instrucciones de ambas clases, pero no todas las plantillas e instrucciones de ambas clases, está dentro del alcance de la invención). Además, un solo procesador puede incluir múltiples núcleos, todos los cuales admiten la misma clase o en los que diferentes núcleos admiten diferentes clases. Por ejemplo, en un procesador con núcleos para gráficos y de propósito general separados, uno de los núcleos de gráficos destinado principalmente a gráficos y/o computación científica puede admitir 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 admiten solo la clase B. Otro procesador que no tenga un núcleo de gráficos separado, puede incluir uno más núcleos dentro o fuera de orden de propósito general que admitan tanto la clase A como la clase B. Por supuesto, las características de una clase también se pueden implementar en la otra clase en diferentes realizaciones de la invención. Los programas escritos en un lenguaje de alto nivel (por ejemplo, compilados justo a tiempo o compilados estáticamente) se pondrían en una diversidad de formas ejecutables diferentes, que incluyen: 1) una forma que tiene únicamente instrucciones de la clase o clases admitidas 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 en función de las instrucciones admitidas por el procesador que actualmente está ejecutando el código.

FORMATO DE INSTRUCCIÓN ESPECÍFICO COMPATIBLE CON VECTORES ILUSTRATIVO

La **Figura 7A** es un diagrama de bloques que ilustra un formato de instrucción específico compatible con vectores ilustrativo de acuerdo con algunas realizaciones de la invención. La **Figura 7A** muestra un formato de instrucción específico compatible con vectores 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 específico compatible con vectores 700 se puede usar para ampliar el conjunto de instrucciones x86 y, por tanto, algunos de los campos son similares o iguales a los usados en el conjunto de instrucciones x86 existente y la extensión del mismo (por ejemplo, AVX). Este formato sigue siendo consecuente 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** con los que se correlacionan los campos de la **Figura 7A**.

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

El formato de instrucción genérico compatible con vectores 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 (byte de EVEX 0, bits [7:0]): el primer byte (byte de EVEX 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).

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

Campo REX 705 (byte de EVEX 1, bits [7-5]): consiste en un campo de bits EVEX.R (byte de EVEX 1, bit [7] - R), campo de bits EVEX.X (byte de EVEX 1, bit [6] - X) y byte de 657BEX 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 se conoce en la técnica (rrr, xxx y 3), de modo que Rrrr, Xxxx y 3b se pueden formar 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' (byte de EVEX 1, bit [4] - R') que se usa para codificar los 16 superiores o los 16 inferiores del conjunto extendido de 32 registros. En algunas realizaciones, este bit, junto con otros como se indica a continuación, se almacena en formato invertido de bits para distinguirlo (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 ni 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 correlación de código de operación 715 (byte de EVEX 1, bits [3:0] - mmmm): su contenido codifica un byte de código de operación inicial implícito (0F, 0F 38 o 0F 3).

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

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

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

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

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

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

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

Campo de máscara de escritura 670 (byte de EVEX 3, bits [2:0]-kkk): su contenido especifica el índice de un registro en los registros de máscara de escritura como se describió anteriormente. En 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 cableada a todo unos 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 de acceso a memoria y operaciones sin acceso a memoria. La función del campo Reg 744 se puede resumir en dos situaciones: codificar el operando de registro de destino o un operando de registro de origen, o tratarse como una extensión de código de operación y no usarse para codificar un operando de instrucción. La función del campo R/M 746 puede incluir lo siguiente: codificar el operando de instrucción que hace referencia a una dirección de memoria, o codificar el operando de registro de destino o un operando de registro de origen.

Byte de escala, índice, base (SIB) (byte 6): como se describió anteriormente, el contenido del campo de escala 650 se usa para la generación de direcciones de memoria. SIB.xxx 754 y SIB.bbb 756: se ha hecho referencia previamente a los contenidos de estos campos con respecto a los índices de registro Xxxx y 3b.

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 abordar desplazamientos entre -128 y 127 bytes; en términos de líneas de memoria caché de 64 bytes, disp8 usa 8 bits que se pueden establecer en solo cuatro valores realmente útiles: -128, -64, 0 y 64; dado que a menudo se necesita un intervalo mayor, se usa disp32; sin embargo, disp32 requiere 4 bytes. A diferencia de disp8 y disp32, el campo de factor de desplazamiento 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 intervalo mucho mayor). Tal desplazamiento comprimido se basa en la suposición de que el desplazamiento efectivo es un múltiplo de la granularidad del acceso a memoria y, por lo tanto, no es necesario codificar los bits de bajo orden redundantes del desplazamiento de dirección. En otras palabras, el campo de factor de desplazamiento 662B sustituye a desplazamiento de 8 bits del conjunto de instrucciones x86 heredado. Por 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 sobrecargue 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 un desplazamiento de dirección byte a byte). 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 específico compatible con vectores 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 base 642 y el campo de anchura de elemento de datos (W) 664. El campo de operación base 642 incluye el campo de codificación de prefijo 725, el campo de correlación 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 específico compatible con vectores 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 específico compatible con vectores 700 que componen el campo de operación de aumento 650 de acuerdo con algunas realizaciones. Cuando el campo de 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 (byte de EVEX 3, bit [7] - EH) se interpreta como el campo rs 652A. Cuando el campo rs 652A contiene un 1 (redondeo 652A.1), el campo beta 654 (byte de EVEX 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 de SAE de un bit 656 y un campo de operación de redondeo de dos bits 658. Cuando el campo rs 652A contiene un 0 (transformada de datos 652A.2), el campo beta 654 (byte de EVEX 3, bits [6:4]-SSS) se interpreta como un campo de transformada 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 (byte de EVEX 3, bit [7] - EH) se interpreta como el campo de sugerencia de expulsión (EH) 652B y el campo beta 654 (byte de EVEX 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 (byte de EVEX 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 (byte de EVEX 3, bit [4]- S_0) se interpreta como el campo RL 657A; cuando contiene un 1 (redondeo 657A.1) el resto del campo beta 654 (byte de EVEX 3, bit [6-5]- S_{2-1}) se interpreta como el campo de operación de redondeo 659A, mientras que cuando el campo RL 657A contiene un 0 (VSIZE 657.A2) el resto del campo beta 654 (byte de EVEX 3, bit [6-5]- S_{2-1}) se interpreta como el campo de longitud de vector 659B (byte de EVEX 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 (byte de EVEX 3, bits [6:4]- SSS) se interpreta como el campo de longitud de vector 659B (byte de EVEX 3, bit [6-5]- L_{1-0}) y el campo de difusión 657B (byte de EVEX 3, bit [4]- B).

ARQUITECTURA DE REGISTROS ILUSTRATIVA

La **Figura 8** es un diagrama de bloques de una arquitectura de registros 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 con 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 con los registros xmm0-15. El formato de instrucción específico compatible con vectores 700 opera en este archivo de registros superpuesto, como se ilustra en las siguientes tablas.

Longitud de vector ajustable	Clase	Operaciones	Registros
Plantillas de instrucción que no incluyen el campo de longitud de vector 659B	A (Figura 6A; U=0)	610, 615, 625, 630	registros zmm (la longitud de vector es de 64 bytes)
	B (Figura 6B; U=1)	612	registros zmm (la longitud de 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 de 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 tales 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 específico compatible con vectores 700 operan con datos de coma flotante de precisión simple/doble empaquetados o escalares y con datos de números enteros empaquetados o escalares. Las operaciones escalares son operaciones realizadas sobre la posición de elemento de datos del orden más bajo en un registro zmm/ymm/xmm; las posiciones de elemento de datos de orden superior o bien se dejan igual que como estaban antes de la instrucción o bien se ponen a cero dependiendo de la realización.

Registros de máscara de escritura 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, inhabilitando 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 registros de pila de coma flotante escalar (pila x87) 845, en el que se establece un solapamiento para el archivo de registros plano de números enteros empaquetados MMX 850: en la realización ilustrada, la pila x87 es una pila de ocho elementos que se usa para realizar operaciones escalares de coma flotante en datos de coma flotante de 32/64/80 bits usando la extensión del conjunto de instrucciones x87, mientras que los registros MMX se usan para realizar operaciones con datos de números enteros empaquetados de 64 bits, así como para contener operandos para algunas operaciones realizadas entre los registros MMX y XMM.

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

ARQUITECTURAS DE NÚCLEO, PROCESADORES Y ARQUITECTURAS INFORMÁTICAS ILUSTRATIVOS

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

ARQUITECTURAS DE NÚCLEO ILUSTRATIVAS

DIAGRAMA DE BLOQUES DE UN NÚCLEO EN ORDEN Y FUERA DE ORDEN

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

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

La **Figura 9B** muestra el núcleo de procesador 990 que incluye una unidad de sección de entrada 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 núcleo de computación de conjunto de instrucciones complejo (CISC), un núcleo de palabras de instrucción muy largas (VLIW) o un tipo de núcleo híbrido o alternativo. Como otra opción más, el núcleo 990 puede ser un núcleo de propósito especial, tal como, por ejemplo, un núcleo de red o comunicación, un motor de compresión, un núcleo de coprocesador, un núcleo de unidad de procesamiento de gráficos informáticos de propósito general (GPGPU), un núcleo de gráficos o similar.

La unidad de sección de entrada 930 incluye una unidad de predicción de ramificación 932 acoplada a una unidad de memoria caché de instrucciones 934, que está acoplada a una memoria intermedia de traducción anticipada (TLB) de instrucciones 936, que está acoplada a una unidad de extracción de instrucciones 938, que está acoplada a una unidad de descodificación 940. La unidad de descodificación 940 (o descodificador) puede descodificar instrucciones y generar como salida una o más microoperaciones, puntos de entrada de microcódigo, microinstrucciones, otras instrucciones u otras señales de control, que se descodifican a partir de, o que de otro modo reflejan o se derivan de, las instrucciones originales. La unidad de descodificación 940 se puede implementar usando varios mecanismos diferentes. Ejemplos de mecanismos adecuados incluyen, pero no se limitan, a tablas de consulta, implementaciones de hardware, matrices lógicas programables (PLA), memorias de solo lectura (ROM) de microcódigo, etc. En una realización, el núcleo 990 incluye una ROM de microcódigo u otro medio que almacena microcódigo para determinadas macroinstrucciones (por ejemplo, en la unidad de descodificación 940 o, de otro modo, dentro de la unidad de sección de entrada 930). La unidad de descodificación 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. Las(s) unidad(es) de planificación 956 representa(n) cualquier número de planificadores diferentes, incluyendo estaciones de reserva, ventana de instrucciones central, etc. Las(s) unidad(es) de planificación 956 está(n) acoplada(s) a las(s) unidad(es) de archivo(s) de registros físicos 958. Cada una de las unidades de archivo(s) de registros físicos 958 representa uno o más archivos de registros físico, diferentes de los que almacenan uno o más tipos de datos diferentes, tales como entero escalar, coma flotante escalar, entero empaquetado, coma flotante empaquetada, entero vectorial, coma flotante vectorial, estado (por ejemplo, un puntero de instrucción que es la dirección de la siguiente instrucción a ejecutar), etc. En una realización, la unidad de archivo(s) de registros físicos 958 comprende una unidad de registros vectoriales, una unidad de registros de máscara de escritura y una unidad de registros escalares. Estas unidades de registros pueden proporcionar registros vectoriales arquitectónicos, registros de máscara vectorial y registros de propósito general. La(s) unidad(es) de archivo(s) de registros físicos 958 se superponen con la unidad de retiro 954 para ilustrar diversas maneras en las que se puede implementar el cambio de nombre de registro y la ejecución fuera de orden (por ejemplo, usando una o más memorias intermedias de reordenación y uno o más archivos de registros de retiro; usando uno o más archivos futuros, una o más memorias intermedias de historial y uno o más archivos de registros de retiro; usando correlaciones de registros y un conjunto de registros; etc.). La unidad de retiro 954 y la(s) unidad(es) de archivo(s) de registros físicos 958 están acopladas al o a los grupos de ejecución 960. El o los grupos de ejecución 960 incluye(n) 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 empaquetada, entero vectorial, coma flotante vectorial). Si bien algunas realizaciones pueden incluir varias unidades de ejecución dedicadas a funciones específicas o conjuntos de funciones, otras realizaciones pueden incluir solo una unidad de ejecución o múltiples unidades de ejecución que realizan todas las funciones. La(s) unidad(es) de planificación 956, las(s) unidad(es) de archivo(s) de registros físicos 958 y el o los grupos de ejecución 960 se muestran posiblemente como una pluralidad porque ciertas realizaciones crean canalizaciones separadas para ciertos tipos de datos/operaciones (por ejemplo, una canalización de números enteros escalares, una canalización de coma flotante escalar/número entero empaquetado/coma flotante empaquetada/número 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(s) de registros físicos y/o grupo 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 las(s) unidad(es) de acceso a memoria 964). También se debería entender que, cuando se usan canalizaciones separadas, una o más de estas canalizaciones pueden ser de emisión/ejecución fuera de orden y, el resto, en orden.

El conjunto de unidades 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 memoria caché de datos 974 acoplada a una unidad de memoria caché de nivel 2 (L2) 976. En una realización ilustrativa, las unidades con acceso a memoria 964 pueden 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 de TLB de datos 972 de la unidad de memoria 970. La unidad de memoria caché de instrucciones 934 está acoplada además a una unidad de memoria caché de nivel 2 (L2) 976 de la unidad de memoria 970. La unidad de memoria caché de L2 976 está acoplada a otro u otros niveles más de memoria caché y, finalmente, a una memoria principal.

A modo de ejemplo, la arquitectura de núcleo ilustrativa de emisión/ejecución fuera de orden y de cambio de nombre de registro puede implementar la canalización 900 como sigue: 1) la extracción de instrucciones 938 realiza las fases de extracción y descodificación de longitud 902 y 904; 2) la unidad de descodificación 940 realiza la fase de descodificación 906; 3) la unidad de cambio de nombre/asignación 952 realiza la fase de asignación 908 y la fase de cambio de nombre 910; 4) las(s) unidad(es) de planificación 956 realiza(n) la fase de planificación 912; 5) las(s) unidad(es) de archivo(s) de registros físicos 958 y la unidad de memoria 970 realizan la fase de lectura de registro/lectura de memoria 914; el grupo de ejecución 960 realiza la fase de ejecución 916; 6) la unidad de memoria 970 y las(s) unidad(es) de archivo(s) de registros físicos 958 realizan la fase de reescritura/escritura en memoria 918; 7) varias unidades pueden estar involucradas en la fase de manejo de excepciones 922; y 8) la unidad de retiro 954 y las(s) unidad(es) de archivo(s) de registros físicos 958 realizan la fase de confirmación 924.

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

Se debería entender que el núcleo puede admitir el procesamiento de múltiples hilos (ejecutar dos o más conjuntos paralelos de operaciones o hilos), y puede hacer esto de diversas maneras, incluyendo procesamiento de múltiples hilos segmentado en el tiempo, procesamiento de múltiples hilos simultáneo (en donde un único núcleo físico proporciona un núcleo lógico para cada uno de los hilos para los que ese núcleo físico está realizando simultáneamente un procesamiento de múltiples hilos), o una combinación de los mismos (por ejemplo, extracción y descodificación segmentadas en el tiempo y, a continuación, procesamiento de múltiples hilos simultáneo, tal como en la tecnología Hyperthreading de Intel®).

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

ARQUITECTURA DE NÚCLEO EN ORDEN ILUSTRATIVA ESPECÍFICA

Las **Figuras 10A-B** ilustran un diagrama de bloques de una arquitectura de núcleo en orden, ilustrativa y 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 memoria caché de nivel 2 (L2) 1004, de acuerdo con las realizaciones de la invención. En una realización, un descodificador de instrucciones 1000 admite el conjunto de instrucciones x86 con una extensión del conjunto de instrucciones de datos empaquetados. Una memoria caché de L1 1006 permite accesos de baja latencia a la memoria caché en las unidades escalares y vectoriales. Aunque 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 los mismos se escriben en memoria y, a continuación, 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 comunicación que permita que se transfieran datos entre los dos archivos de registros sin que se escriban y se vuelvan a leer).

El subconjunto local de la memoria caché de L2 1004 es parte de una memoria caché de L2 global que se divide en subconjuntos locales separados, uno por núcleo de procesador. Cada núcleo de procesador tiene una ruta de acceso directo a su propio subconjunto local de la memoria caché de L2 1004. Los datos leídos por un núcleo de procesador se almacenan en su subconjunto de memoria caché de L2 1004 y se puede acceder a ellos rápidamente, en paralelo con otros núcleos de procesador que acceden a sus propios subconjuntos de memoria caché de L2 locales. Los datos escritos por un núcleo de procesador se almacenan en su propio subconjunto de memoria caché de L2 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, memorias caché de 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 de procesador de la **Figura 10A** de acuerdo con algunas realizaciones de la invención. La **Figura 10B** incluye una parte de memoria caché de datos de L1 1006A de la memoria caché de L1 1004, así como más detalles con respecto a la unidad vectorial 1010 y los registros vectoriales 1014. Específicamente, la unidad vectorial 1010 es una unidad de procesamiento vectorial (VPU) de anchura 16 (véase la ALU 1028 de anchura 16), que ejecuta una o más de instrucciones de números enteros, flotantes de precisión simple y flotantes de precisión doble. La VPU admite el mezclado de las entradas de registro con una unidad de mezcla 1020, la conversión numérica con unidades de conversión numérica 1022A-B y la replicación con una 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 unidades de controlador de memoria integrado 1114 en la unidad de agente de sistema 1110 y lógica de propósito especial 1108.

Por tanto, diferentes implementaciones del procesador 1100 pueden incluir: 1) una CPU con la lógica de propósito especial 1108, que es una lógica de gráficos y/o científica (de capacidad de procesamiento) integrada (que puede incluir uno o más núcleos), y los núcleos 1102A-N, que son uno o más núcleos de propósito general (por ejemplo, núcleos en orden de propósito general, núcleos fuera de orden de propósito general, una combinación de los dos tipos); 2) un coprocesador con los núcleos 1102A-N, que son un gran número de núcleos de propósito especial destinados principalmente a cálculos gráficos y/o científicos (de capacidad de procesamiento); y 3) un coprocesador con los núcleos 1102A-N, que son un gran número de núcleos en orden de propósito general. Por 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 núcleos o más), procesador integrado o similar. El procesador se puede implementar en uno o más chips. El procesador 1100 puede ser parte y/o se puede implementar en uno o más sustratos usando cualquiera de varias tecnologías de proceso, tales como, por ejemplo, BiCMOS, CMOS o NMOS.

La jerarquía de memoria incluye uno o más niveles de memoria caché dentro de los núcleos, un conjunto o una o más unidades de memoria caché compartidas 1106, y memoria externa (no mostrada) acoplada al conjunto de unidades de controlador de memoria integrado 1114. El conjunto de unidades de memoria caché compartidas 1106 puede incluir una o más memorias caché de nivel medio, tales como de nivel 2 (L2), de nivel 3 (L3), de nivel 4 (L4) o de otros niveles de memoria caché, una memoria caché de último nivel (LLC) y/o combinaciones de las mismas. Aunque, en una realización, una unidad de interconexión basada en anillo 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 memoria caché compartidas 1106 y la unidad de agente de sistema 1110/unidad o unidades de controlador de memoria integrado 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 memoria caché 1106 y núcleos 1102-A-N.

En algunas realizaciones, uno o más de los núcleos 1102A-N admiten el procesamiento de múltiples hilos. 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 integrada 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, descodificadores de salón, microcontroladores, teléfonos móviles, reproductores multimedia portátiles, dispositivos de mano y otros diversos dispositivos electrónicos, también son adecuados. En general, son 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 controladores 1220. En una realización, el concentrador de controladores 1220 incluye un concentrador de controladores 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. De forma alternativa, los controladores de memoria y/o los controladores de gráficos están integrados dentro del procesador (como se describe en el presente documento), la memoria 1240 y el coprocesador 1245 están

acoplados directamente al procesador 1210 y al concentrador de controladores 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 las dos. Para al menos una realización, el concentrador de controladores 1220 se comunica con el o los procesadores 1210, 1215 por medio de un bus multipunto, tal como un bus frontal (FSB), una interfaz punto a punto tal como QuickPath Interconnect (QPI) o una 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, un motor de compresión, un procesador de gráficos, una GPGPU, un procesador integrado o similar. En una realización, el concentrador de controladores 1220 puede incluir un acelerador de gráficos integrado.

Puede haber varias diferencias entre los recursos físicos 1210, 1215 en lo que respecta a un espectro de métricas de mérito, que incluyen características arquitectónicas, microarquitectónicas, térmicas, de consumo de energía y similares.

En una realización, el procesador 1210 ejecuta instrucciones que controlan operaciones de procesamiento de datos de un tipo general. Las instrucciones pueden incluir 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 al coprocesador 1245 estas instrucciones de coprocesador (o señales de control que representan instrucciones de coprocesador) en un bus de coprocesador u otra interconexión. El/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 por medio 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.

Como se muestra, los procesadores 1370 y 1380 incluyen unidades de controlador de memoria integrado (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 por medio 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 individuales P-P 1352, 1354 usando circuitos de interfaz de 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, un motor de compresión, un procesador de gráficos, una GPGPU, un procesador integrado o similar.

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

El conjunto de chips 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 se limita a ello.

Como se muestra en la **Figura 13**, diversos dispositivos de E/S 1314 se pueden acoplar 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)), matrices de puertas programables *in situ* o cualquier otro procesador, están acoplados al primer bus 1316. En

una realización, el segundo bus 1320 puede ser un bus de bajo número de pines (LPC). Se pueden acoplar diversos dispositivos a un segundo bus 1320 que incluyan, por ejemplo, un teclado y/o un ratón 1322, dispositivos de comunicación 1327 y una unidad de almacenamiento 1328 tal como una unidad de disco u otro dispositivo de almacenamiento masivo que pueda incluir instrucciones/código 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 no complicar otros aspectos de la **Figura 14**.

La **Figura 14** ilustra que los procesadores 1370, 1380 pueden incluir lógica de control ("CL") integrada de memoria y de E/S 1472 y 1482, respectivamente. Por tanto, la CL 1472, 1482 incluye unidades de controlador de memoria integrado e incluye 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 números de referencia parecidos. Además, los recuadros con línea discontinua son características opcionales en los SoC más avanzados. En la **Figura 15**, una o más 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 unidades de memoria caché 1104A-N y una o más unidades de caché compartida 1106; una unidad de agente de sistema 1110; una o más unidades de controlador de bus 1116; una o más 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, el o los coprocesadores 1520 incluyen un procesador de propósito especial, tal como, por ejemplo, un procesador de red o comunicación, un motor de compresión, una GPGPU, un procesador MIC de alto rendimiento, un procesador integrado o similares.

Las realizaciones de los mecanismos divulgados en el presente documento se pueden implementar en hardware, software, firmware o una combinación de tales enfoques de implementación. Las realizaciones de la invención se pueden implementar 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 se puede aplicar a uno o más dispositivos de salida, de forma conocida. Para los propósitos de esta solicitud, un sistema de procesamiento incluye cualquier sistema que tenga un procesador, tal como, por ejemplo, un procesador de señales digitales (DSP), un microcontrolador, un circuito integrado específico de la aplicación (ASIC) o un microprocesador.

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

Uno o más aspectos de al menos una realización se pueden implementar 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 genere lógica para realizar las técnicas descritas en el presente documento. Tales representaciones, conocidas como "núcleos de IP", se pueden almacenar en un medio tangible legible por máquina y suministrar 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, incluidos disquetes, discos ópticos, memorias de solo lectura en disco compacto (CD-ROM), discos compactos reescribibles (CD-RW) y discos magnetoópticos, dispositivos 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 el almacenamiento de instrucciones electrónicas.

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

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

En algunos casos, se puede usar un convertidor de instrucciones para convertir una instrucción de un conjunto de instrucciones de origen a un conjunto de instrucciones objetivo. Por ejemplo, el convertidor de instrucciones puede traducir (por ejemplo, usando traducción binaria estática, traducción binaria dinámica que incluye compilación dinámica), transformar, emular o convertir de otro modo una instrucción en una o más instrucciones a procesar por el núcleo. El convertidor de instrucciones se puede implementar en software, hardware, firmware o una combinación de los mismos. El convertidor de instrucciones puede estar en el procesador, fuera del procesador o en parte dentro y en parte 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 de un conjunto de instrucciones de origen en instrucciones binarias de 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 se puede implementar en software, firmware, hardware o diversas combinaciones de los mismos. La **Figura 16** muestra que un programa en un lenguaje de alto nivel 1602 se puede compilar usando un compilador x86 1604 para generar un código binario x86 1606 que un procesador puede ejecutar de forma nativa 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 pueda realizar sustancialmente las mismas funciones que un procesador de Intel con al menos un núcleo de conjunto de instrucciones x86 ejecutando o procesando de otro modo, de forma compatible, (1) una porción sustancial del conjunto de instrucciones del núcleo de conjunto de instrucciones x86 de Intel 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, con el fin de lograr sustancialmente el mismo resultado que un procesador de 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 se puede compilar usando un compilador de conjunto de instrucciones alternativo 1608 para generar un código binario de conjunto de instrucciones alternativo 1610 que se puede ejecutar 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 de conjunto de instrucciones alternativo 1610 porque es difícil fabricar un convertidor de instrucciones apto para esto; sin embargo, el código convertido conseguirá la operación general y estará compuesto por instrucciones del conjunto de instrucciones alternativo. Por tanto, el convertidor de instrucciones 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 tenga un procesador o núcleo de conjunto de instrucciones x86 ejecute el código binario x86 1606.

REIVINDICACIONES

1. Un procesador (100), que comprende:

circuitos de extracción (105) configurados para extraer una instrucción de producto escalar en coma flotante (201);
una unidad de descodificación (109) configurada para descodificar la instrucción de producto escalar en coma flotante (201), teniendo la instrucción de producto escalar en coma flotante (201) un primer campo (206) para especificar un primer registro vectorial de origen, un segundo campo (208) para especificar un segundo registro vectorial de origen, y un tercer campo (204) para especificar un registro vectorial de origen/destino, el primer registro vectorial de origen configurado para almacenar un primer vector de origen (212A) que tiene una primera pluralidad de pares de elementos de datos de coma flotante de 16 bits, el segundo registro vectorial de origen configurado para almacenar un segundo vector de origen (212B) que tiene una segunda pluralidad de pares de elementos de datos de coma flotante de 16 bits, correspondiendo cada par del primer vector de origen (212A) a un par del segundo vector de origen (212B) en las mismas posiciones de bit, correspondiendo cada elemento de datos de coma flotante de 16 bits del primer vector de origen (212A) a un elemento de datos de coma flotante de 16 bits del segundo vector de origen (212B) en las mismas posiciones de bits, teniendo los elementos de datos de coma flotante de 16 bits de los primer y segundo vectores de origen (212A, 212B) un formato, teniendo el formato un bit de signo, un exponente de 8 bits, siete bits de mantisa explícitos y un bit de mantisa implícito, el registro vectorial de origen/destino configurado para almacenar un tercer vector de origen (218) que tiene una pluralidad de elementos de datos de coma flotante de precisión simple de 32 bits, correspondiendo cada elemento de datos de coma flotante de precisión simple de 32 bits a un par del primer vector de origen (212A) y a un par del segundo vector de origen (212B), en las mismas posiciones de bit;
y
circuitos de ejecución (117, 214) acoplados a la unidad de descodificación (109), los circuitos de ejecución (117, 214) configurados para ejecutar la instrucción de producto escalar en coma flotante descodificada (201) para

multiplicar los elementos de datos de coma flotante de 16 bits de los pares del primer vector de origen (212A) por los correspondientes elementos de datos de coma flotante de 16 bits de los correspondientes pares del segundo vector de origen (212B) para generar una pluralidad de pares de productos;
generar una pluralidad de elementos de datos de coma flotante de precisión simple de 32 bits resultantes mediante la suma de los respectivos pares de productos con un elemento de datos de coma flotante de precisión simple de 32 bits del tercer vector de origen (218) correspondiente a un par del primer vector de origen (212A) usado para generar el respectivo par de productos, y la aplicación de un modo de redondeo de coma flotante para la instrucción de producto escalar en coma flotante (201); y
almacenar la pluralidad de elementos de datos de coma flotante de precisión simple de 32 bits resultantes en el registro vectorial de origen/destino,
caracterizado por que cada par de elementos de datos de coma flotante de 16 bits del primer vector de origen (212A) forma un elemento de 32 bits del primer vector de origen (212A) y cada par de elementos de datos de coma flotante de 16 bits del segundo vector de origen (212B) forma un elemento de 32 bits del segundo vector de origen (212B) de modo que los elementos de 32 bits del primer vector de origen (212A), del segundo vector de origen (212B) y del tercer vector de origen (218) están equilibrados en tamaño,

en donde los circuitos de ejecución comprenden:

un primer multiplicador (215A) configurado para multiplicar un primer elemento de datos de coma flotante de 16 bits de un par del primer vector de origen (212A) por un primer elemento de datos de coma flotante de 16 bits de un par del segundo vector de origen (212B); y
un segundo multiplicador (215B) configurado para multiplicar un segundo elemento de datos de coma flotante de 16 bits del par del primer vector de origen (212A) por un segundo elemento de datos de coma flotante de 16 bits del par del segundo vector de origen (212B).

2. El procesador (100) de la reivindicación 1, que comprende además un registro de control para especificar un modo de redondeo de coma flotante, en donde el modo de redondeo de coma flotante se aplica independientemente del modo de redondeo de coma flotante especificado por el registro de control.

3. El procesador (100) de la reivindicación 2, en donde los circuitos de ejecución (117, 214), independientemente del registro de control, establecen en cero los valores desnormalizados de los primer y segundo vectores de origen (212A, 212B).

4. El procesador (100) de la reivindicación 2 o 3, en donde los circuitos de ejecución (117, 214), independientemente del registro de control, ponen a cero un elemento de datos de coma flotante de precisión simple de 32 bits resultante.

5. El procesador (100) de cualquiera de las reivindicaciones 2 a 4, en donde el procesador (100) no actualiza el registro de control durante la ejecución de la instrucción descodificada de producto escalar en coma flotante (201).

6. El procesador (100) de cualquiera de las reivindicaciones 2 a 5, en donde el procesador (100) no consulta el registro de control durante la ejecución de la instrucción descodificada de producto escalar en coma flotante (201).

7. El procesador (100) de cualquiera de las reivindicaciones 1 a 6, en donde los circuitos de ejecución (117, 214) convierten además al menos uno de los elementos de datos de coma flotante de 16 bits de los primer y segundo vectores de origen (212A, 212B) en un elemento de datos de coma flotante de precisión simple de 32 bits, lo que incluye: almacenar ceros en los bits [15:0] del elemento de datos de coma flotante de precisión simple de 32 bits, y almacenar el elemento de datos de coma flotante de 16 bits en los bits [31:16] del elemento de datos de coma flotante de precisión simple de 32 bits.

8. El procesador (100) de cualquiera de las reivindicaciones 1 a 7, en donde el primer registro vectorial de origen es un registro vectorial de 512 bits.

9. El procesador (100) de cualquiera de las reivindicaciones 1 a 8, que comprende además una pluralidad de registros de máscara, en donde la pluralidad de registros de máscara incluye una pluralidad de registros que se usarán para enmascarar operaciones en base a la posición de cada elemento de datos, y un registro que no se puede usar para enmascarar operaciones en base a la posición de cada elemento de datos.

10. El procesador (100) de la reivindicación 9, en donde la pluralidad de registros que se van a usar para el enmascaramiento en base a la posición de cada elemento de datos se pueden usar en el enmascaramiento fusionado, en el que los elementos enmascarados conservan los valores iniciales que tenían antes del enmascaramiento fusionado.

11. El procesador (100) de la reivindicación 10, en donde la pluralidad de registros que se van a usar para el enmascaramiento en base a la posición de cada elemento de datos se pueden usar en el enmascaramiento de puesta a cero, en el que los elementos enmascarados se van a poner a cero.

12. El procesador (100) de cualquiera de las reivindicaciones 1 a 11, en donde el procesador (100) es un procesador de conjunto de instrucciones reducido, RISC.

13. El procesador (100) de cualquiera de las reivindicaciones 1 a 12, en donde el procesador (100) es un núcleo de CPU de propósito general.

14. El procesador (100) de cualquiera de las reivindicaciones 1 a 13, en donde el formato es un formato bfloat16.

15. El procesador (100) de cualquiera de las reivindicaciones 1 a 14, que comprende además múltiples niveles de memoria caché, incluida una memoria caché de nivel 2, L2.

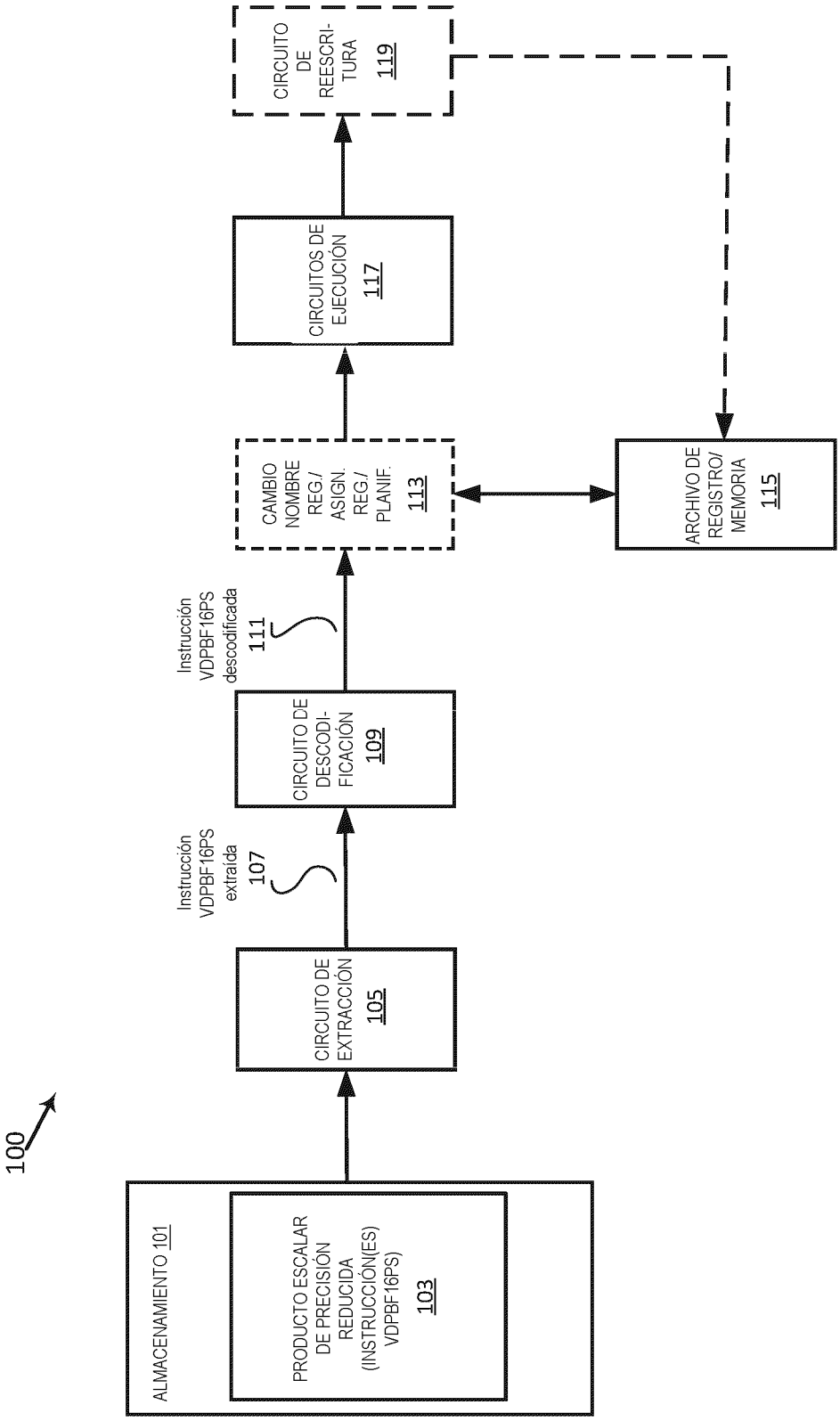

100
ALMACENAMIENTO 101
PRODUCTO ESCALAR DE PRECISIÓN REDUCIDA (INSTRUCCIÓN(ES) VDPBF16PS) 103
CIRCUITO DE EXTRACCIÓN 105
Instrucción VDPBF16PS extraída 107
CIRCUITO DE DESCODIFICACIÓN 109
Instrucción VDPBF16PS descodificada 111
CAMBIO NOMBRE REG./ ASIGN. REG./ PLANIF. 113
ARCHIVO DE REGISTRO/ MEMORIA 115
CIRCUITOS DE EJECUCIÓN 117
CIRCUITO DE REESCRITURA 119

FIG. 1

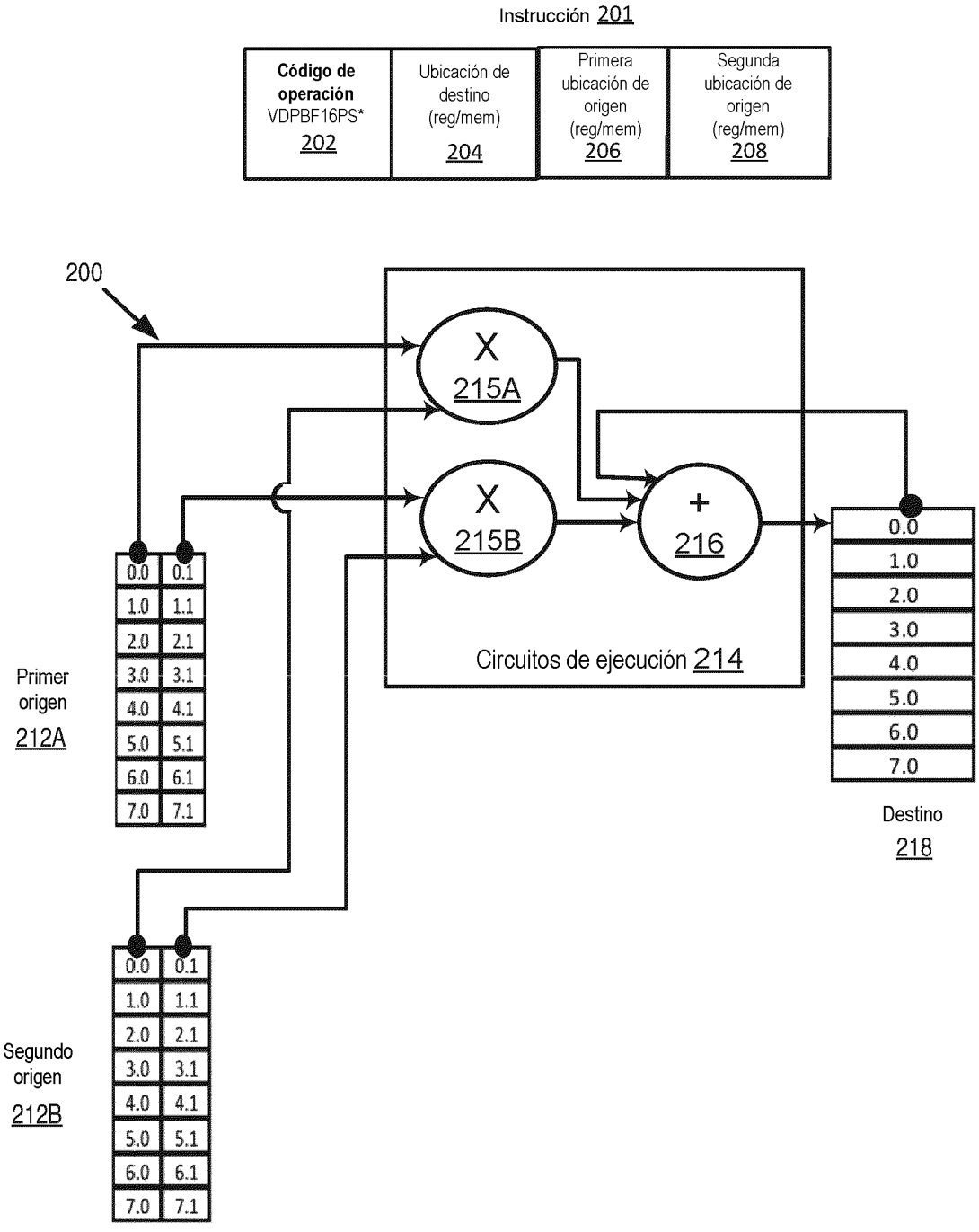

Instrucción 201
Código de operación VDPBF16PS* 202
Ubicación de destino (reg/mem) 204
Primera ubicación de origen (reg/mem) 206
Segunda ubicación de origen (reg/mem) 208
200
X 215A
X 215B
+ 216
Circuitos de ejecución 214
Primer origen 212A
Segundo origen 212B
Destino 218
0.0 0.1
1.0 1.1
2.0 2.1
3.0 3.1
4.0 4.1
5.0 5.1
6.0 6.1
7.0 7.1
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0

FIG. 2

Pseudocódigo
300

```
VDPBF16PS srcdest, src1, src2
        VL = (128,256,512)
        kl = vl/32
        origdest := srcdest
        for i := 0 to kl-1:
                if k1[i] or *no writemask*:
                        if src2 is memory and evex.b == 1:
                                t := src2.dword[0]
                        else:
                                t := src2.dword[i]

                        // FP32 FMA con daz in, ftz out y redondeo RNE
                        // MXCSR no consultado ni actualizado

                        srcdest.fp32[i] += make_fp32(src1.bfloat16[2*i+0]) *
                                           make_fp32(t.bfloat[0])
                        srcdest.fp32[i] += make_fp32(src1.bfloat16[2*i+1]) *
                                           make_fp32(t.bfloat[1])
                else if *zeroing*:
                        srcdest.dword[i] := 0

                else: // Fusionar enmascaramiento, elemento dest sin modificar
                        srcdest.dword[i] := origdest.dword[i]

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

FIG. 3A

Pseudocódigo
310

```
VDPBF16PS srcdest, src1, src2
        VL = (128,256,512)
        kl = vl/32
        origdest := srcdest
        for i := 0 to kl-1:
                if k1[i] or *no writemask*:
                        if src2 is memory and evex.b == 1:
                                t := src2.dword[0]
                        else:
                                t := src2.dword[i]

                        // FP32 FMA con daz in, ftz out y redondeo RNE
                        // MXCSR no consultado ni actualizado

                        srcdest.fp32[i] += make_fp32(src1.bfloat16[2*i+1]) *
                                           make_fp32(t.bfloat[1])
                        srcdest.fp32[i] += make_fp32(src1.bfloat16[2*i+0]) *
                                           make_fp32(t.bfloat[0])
                else if *zeroing*:
                        srcdest.dword[i] := 0

                else: // Fusionar enmascaramiento, elemento dest sin modificar
                        srcdest.dword[i] := origdest.dword[i]

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

FIG. 3B

Pseudocódigo
320

Función auxiliar

```
define make_fp32(x):
// El parámetro x es bfloat16. Empaquetarlo en 16 bits superiores de una palabra doble
// El patrón de bits es un valor fp32 válido. Devolver ese patrón de bits
dword: = 0
dword[31:16] := x
return dword
```

FIG. 3C

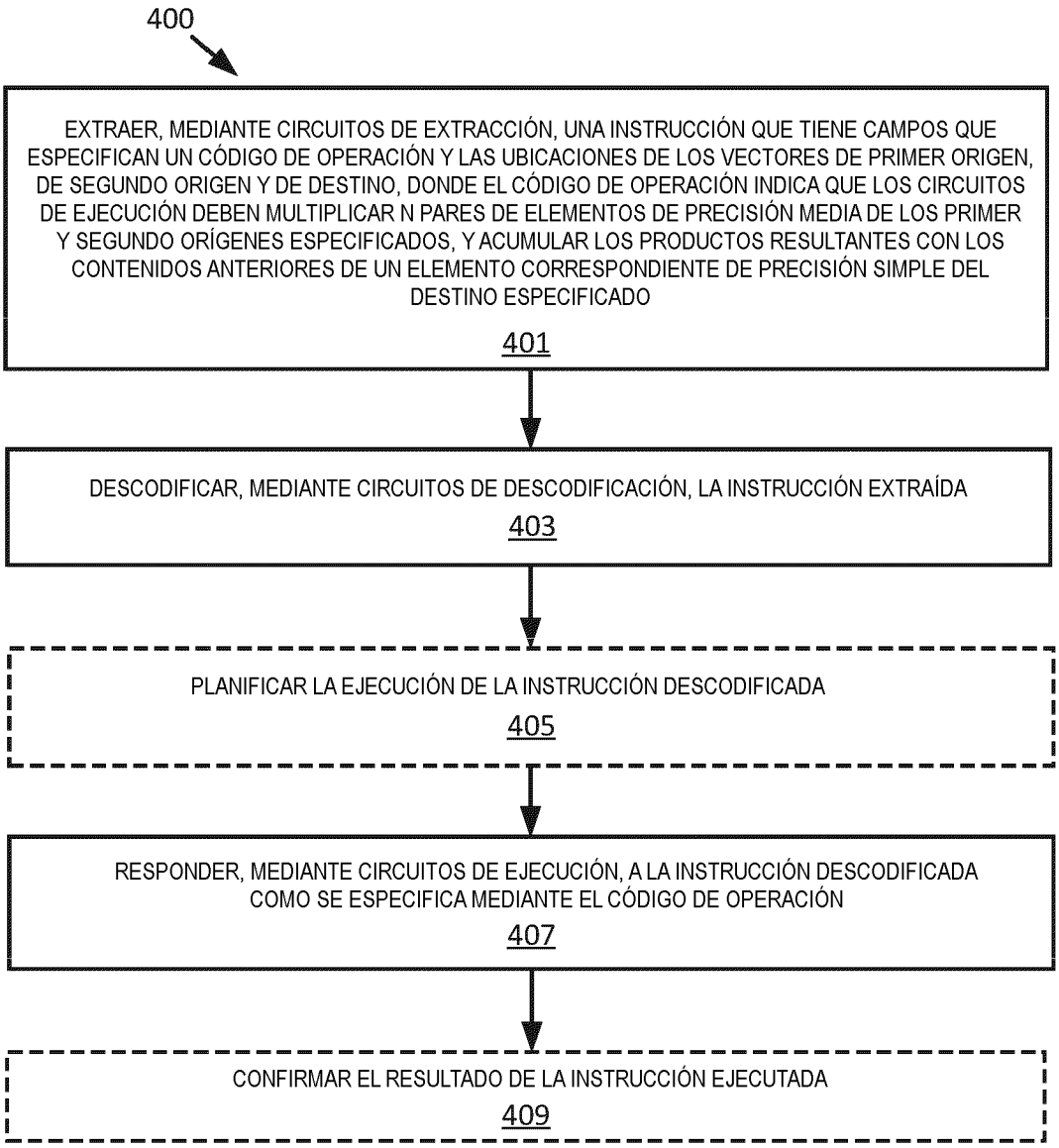

400
EXTRAER, MEDIANTE CIRCUITOS DE EXTRACCIÓN, UNA INSTRUCCIÓN QUE TIENE CAMPOS QUE ESPECIFICAN UN CÓDIGO DE OPERACIÓN Y LAS UBICACIONES DE LOS VECTORES DE PRIMER ORIGEN, DE SEGUNDO ORIGEN Y DE DESTINO, DONDE EL CÓDIGO DE OPERACIÓN INDICA QUE LOS CIRCUITOS DE EJECUCIÓN DEBEN MULTIPLICAR N PARES DE ELEMENTOS DE PRECISIÓN MEDIA DE LOS PRIMER Y SEGUNDO ORÍGENES ESPECIFICADOS, Y ACUMULAR LOS PRODUCTOS RESULTANTES CON LOS CONTENIDOS ANTERIORES DE UN ELEMENTO CORRESPONDIENTE DE PRECISIÓN SIMPLE DEL DESTINO ESPECIFICADO
401
DESCODIFICAR, MEDIANTE CIRCUITOS DE DESCODIFICACIÓN, LA INSTRUCCIÓN EXTRAÍDA
403
PLANIFICAR LA EJECUCIÓN DE LA INSTRUCCIÓN DESCODIFICADA
405
RESPONDER, MEDIANTE CIRCUITOS DE EJECUCIÓN, A LA INSTRUCCIÓN DESCODIFICADA COMO SE ESPECIFICA MEDIANTE EL CÓDIGO DE OPERACIÓN
407
CONFIRMAR EL RESULTADO DE LA INSTRUCCIÓN EJECUTADA
409

FIG. 4

Instrucción VDPBF16PS 500

Código de operación VDPBF16PS* 502	Ubicación de destino (reg/mem) 504	Ubicación de primer origen (reg/mem) 506	Ubicación de segundo origen (reg/mem) 508	Máscara {k} 510	Puesta a cero {Z} 512	Formato de elemento 514	Tamaño de vector (N) 516

FIG. 5

FIG.6A

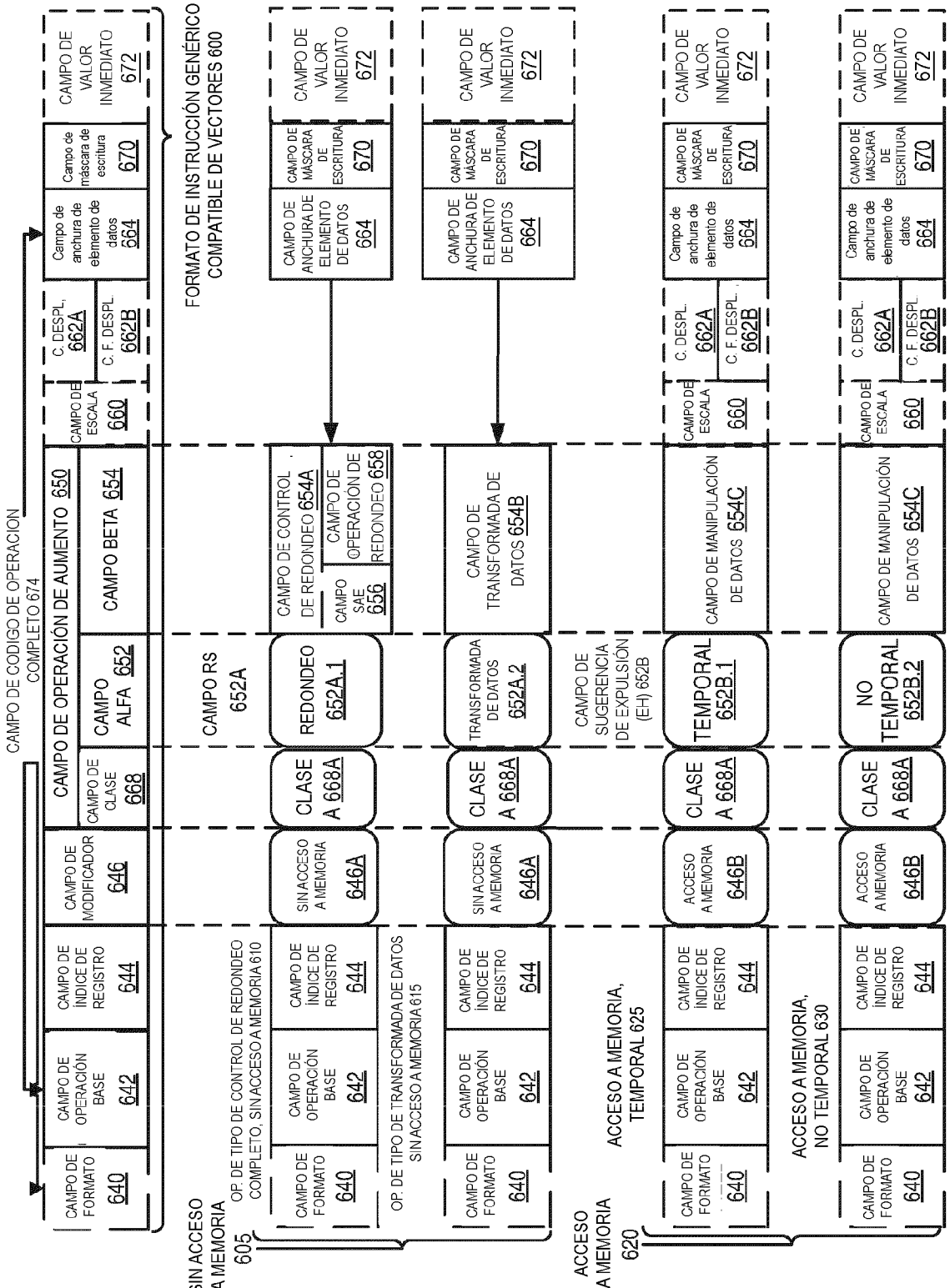
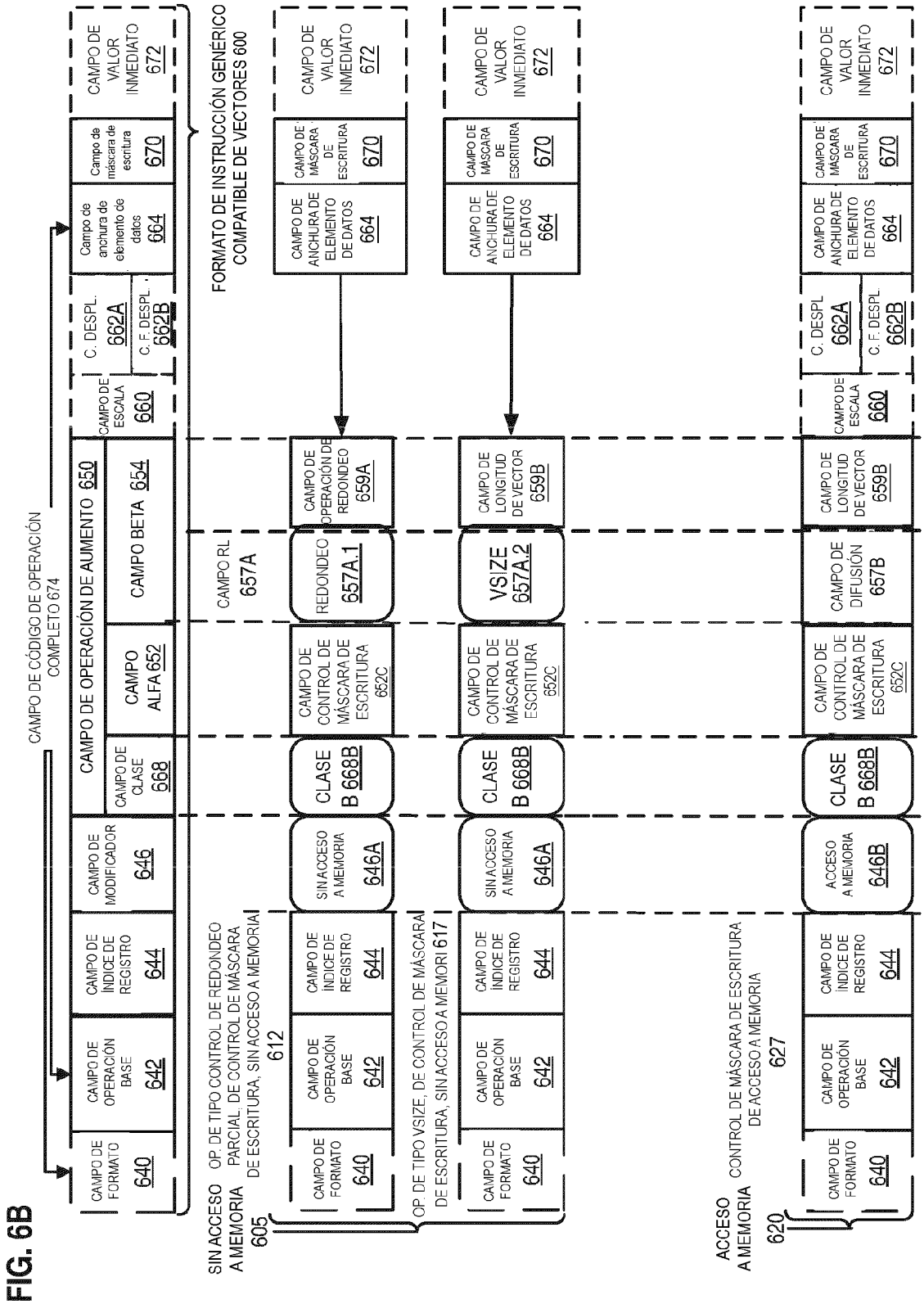

CAMPO DE CODIGO DE OPERACION COMPLETO 674
FORMATO DE INSTRUCCIÓN GENÉRICO COMPATIBLE DE VECTORES 600
CAMPO DE FORMATO 640
CAMPO DE OPERACIÓN BASE 642
CAMPO DE ÍNDICE DE REGISTRO 644
CAMPO DE MODIFICADOR 646
CAMPO DE CLASE 668
CAMPO DE OPERACIÓN DE AUMENTO 650
CAMPO ALFA 652
CAMPO BETA 654
CAMPO DE ESCALA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

FIG. 6B


FORMATO DE INSTRUCCIÓN GENÉRICO COMPATIBLE DE VECTORES 600
CAMPO DE CÓDIGO DE OPERACIÓN COMPLETO 674
CAMPO DE FORMATO 640
CAMPO DE OPERACIÓN BASE 642
CAMPO DE ÍNDICE DE REGISTRO 644
CAMPO DE MODIFICADOR 646
CAMPO DE CLASE 668
CAMPO DE OPERACIÓN DE AUMENTO 650
CAMPO ALFA 652
CAMPO BETA 654
CAMPO DE ESCALA 660
C. DESPL. 662A
C. F. DESPL. 662B
Campo de anchura de elemento de datos 664
Campo de máscara de escritura 670
CAMPO DE VALOR INMEDIATO 672
SIN ACCESO A MEMORIA 605
OP. DE TIPO CONTROL DE REDONDEO PARCIAL, DE CONTROL DE MÁSCARA DE ESCRITURA, SIN ACCESO A MEMORIA 612
CAMPO RL 657A
SIN ACCESO A MEMORIA 646A
CLASE B 668B
CAMPO DE CONTROL DE MÁSCARA DE ESCRITURA 652C
REDONDEO 657A.1
CAMPO DE OPERACIÓN DE REDONDEO 659A
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 664
CAMPO DE MÁSCARA DE ESCRITURA 670
OP. DE TIPO VSIZE, DE CONTROL DE MÁSCARA DE ESCRITURA, SIN ACCESO A MEMORIA 617
VSIZE 657A.2
CAMPO DE LONGITUD DE VECTOR 659B
ACCESO A MEMORIA 620
CONTROL DE MÁSCARA DE ESCRITURA DE ACCESO A MEMORIA 627
ACCESO A MEMORIA 646B
CAMPO DE DIFUSIÓN 657B

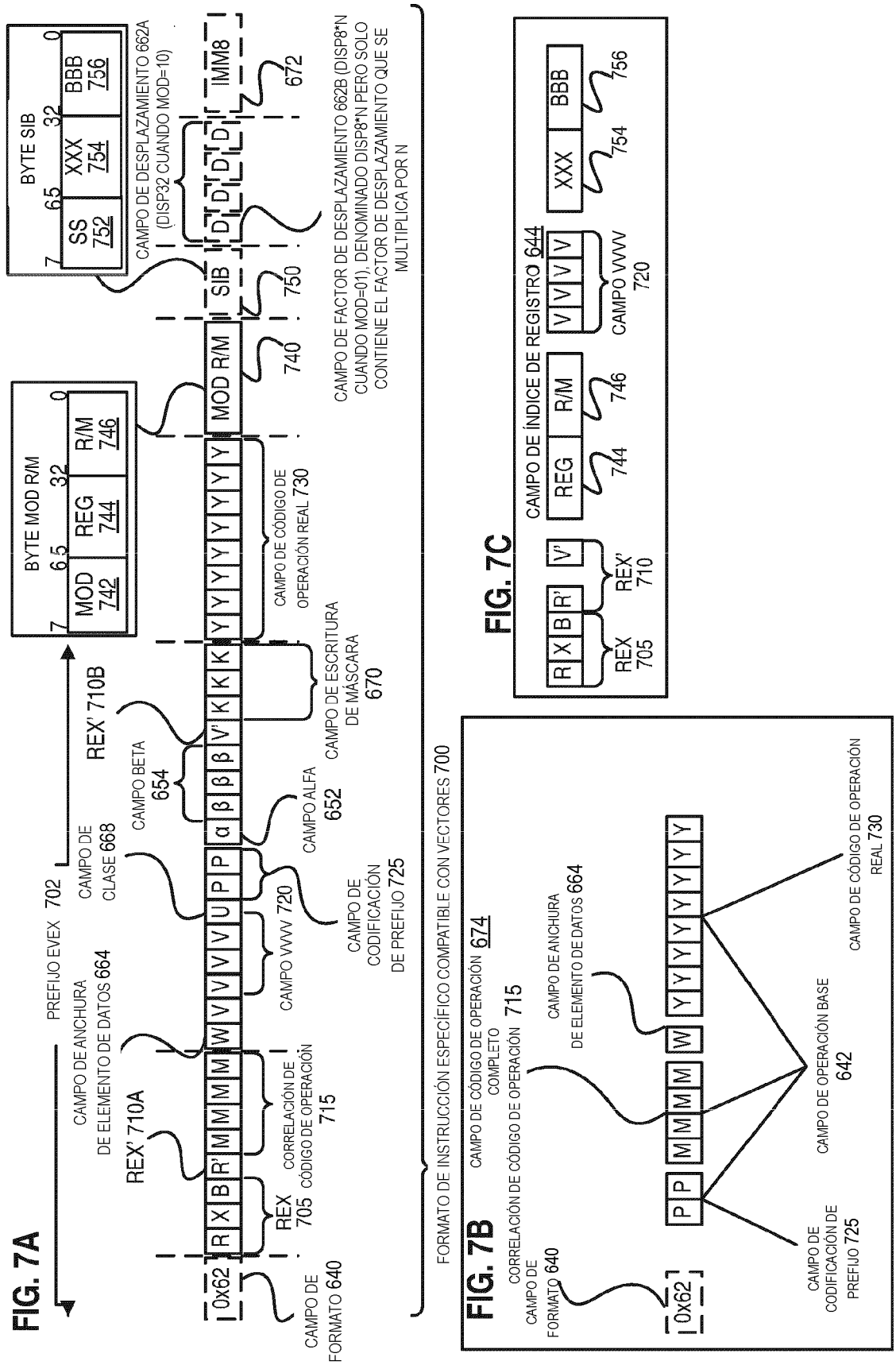


FIG. 7A
PREFIJO EVEX 702
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 664
CAMPO DE CLASE 668
REX' 710B
CAMPO BETA 654
REX' 710A
CAMPO ALFA 652
CAMPO DE ESCRITURA DE MÁSCARA 670
CAMPO VVVV 720
CORRELACIÓN DE CÓDIGO DE OPERACIÓN 715
CAMPO DE CÓDIGO DE OPERACIÓN REAL 730
CAMPO DE CODIFICACIÓN DE PREFIJO 725
REX 705
CAMPO DE FORMATO 640
0x62
BYTE MOD R/M
MOD 742
REG 744
R/M 746
MOD R/M
SIB
BYTE SIB
SS 752
XXX 754
BBB 756
CAMPO DE DESPLAZAMIENTO 662A (DISP32 CUANDO MOD=10)
CAMPO DE FACTOR DE DESPLAZAMIENTO 662B (DISP8*N CUANDO MOD=01), DENOMINADO DISP8*N PERO SOLO CONTIENE EL FACTOR DE DESPLAZAMIENTO QUE SE MULTIPLICA POR N
740
750
672
IMM8
FORMATO DE INSTRUCCIÓN ESPECÍFICO COMPATIBLE CON VECTORES 700
FIG. 7B
CAMPO DE CÓDIGO DE OPERACIÓN COMPLETO 674
CORRELACIÓN DE CÓDIGO DE OPERACIÓN 715
CAMPO DE ANCHURA DE ELEMENTO DE DATOS 664
CAMPO DE FORMATO 640
CAMPO DE OPERACIÓN BASE 642
CAMPO DE CÓDIGO DE OPERACIÓN REAL 730
CAMPO DE CODIFICACIÓN DE PREFIJO 725
FIG. 7C
CAMPO DE ÍNDICE DE REGISTRO 644
REX 705
REX' 710
REG 744
R/M 746
CAMPO VVVV 720
XXX 754
BBB 756

FIG. 7D

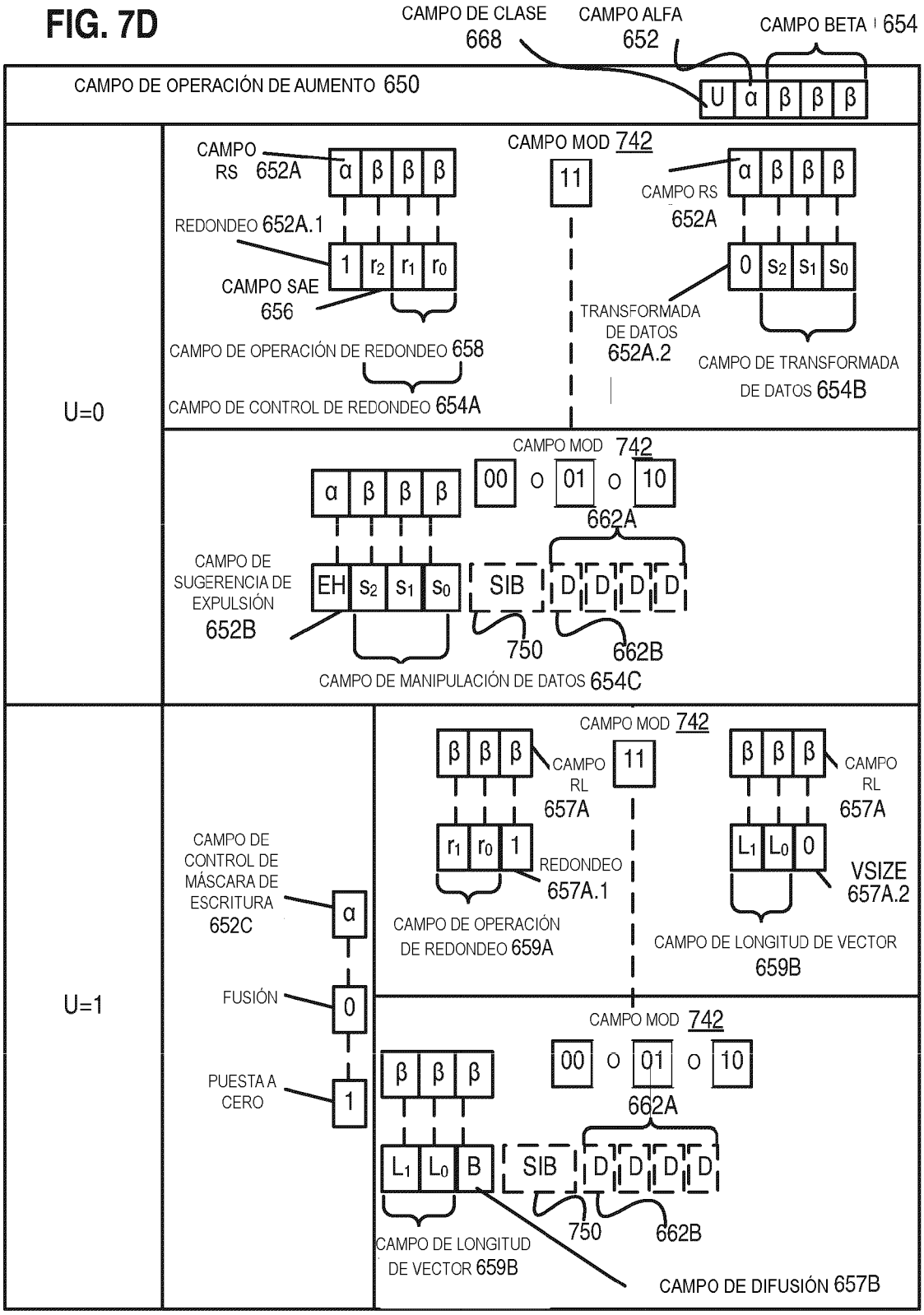
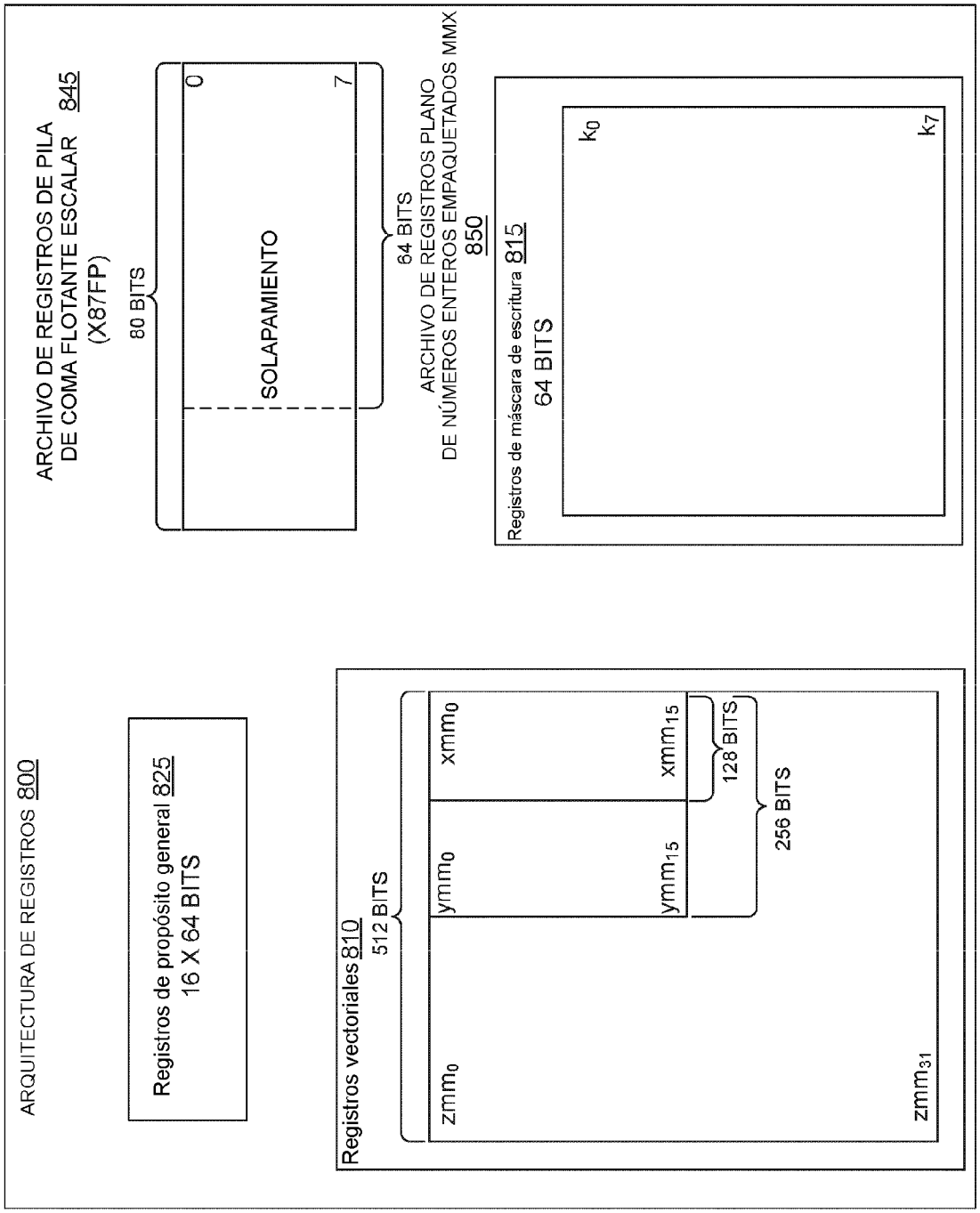

CAMPO DE CLASE 668
CAMPO ALFA 652
CAMPO BETA 654
CAMPO DE OPERACIÓN DE AUMENTO 650
U α β β β
U=0
CAMPO RS 652A
α β β β
REDONDEO 652A.1
1 r2 r1 r0
CAMPO SAE 656
CAMPO DE OPERACIÓN DE REDONDEO 658
CAMPO DE CONTROL DE REDONDEO 654A
CAMPO MOD 742
11
CAMPO RS 652A
α β β β
0 s2 s1 s0
TRANSFORMADA DE DATOS 652A.2
CAMPO DE TRANSFORMADA DE DATOS 654B
CAMPO MOD 742
00 o 01 o 10
662A
α β β β
CAMPO DE SUGERENCIA DE EXPULSIÓN 652B
EH s2 s1 s0
SIB
D D D D
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
CAMPO RL 657A
11
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 REGISTROS 800
Registros de propósito general 825
16 X 64 BITS
Registros vectoriales 810
512 BITS
zmm0
ymm0
xmm0
ymm15
xmm15
128 BITS
256 BITS
zmm31
ARCHIVO DE REGISTROS DE PILA
DE COMA FLOTANTE ESCALAR 845
(X87FP)
80 BITS
0
7
SOLAPAMIENTO
64 BITS
ARCHIVO DE REGISTROS PLANO
DE NÚMEROS ENTEROS EMPAQUETADOS MMX
850
Registros de máscara de escritura 815
64 BITS
k0
k7

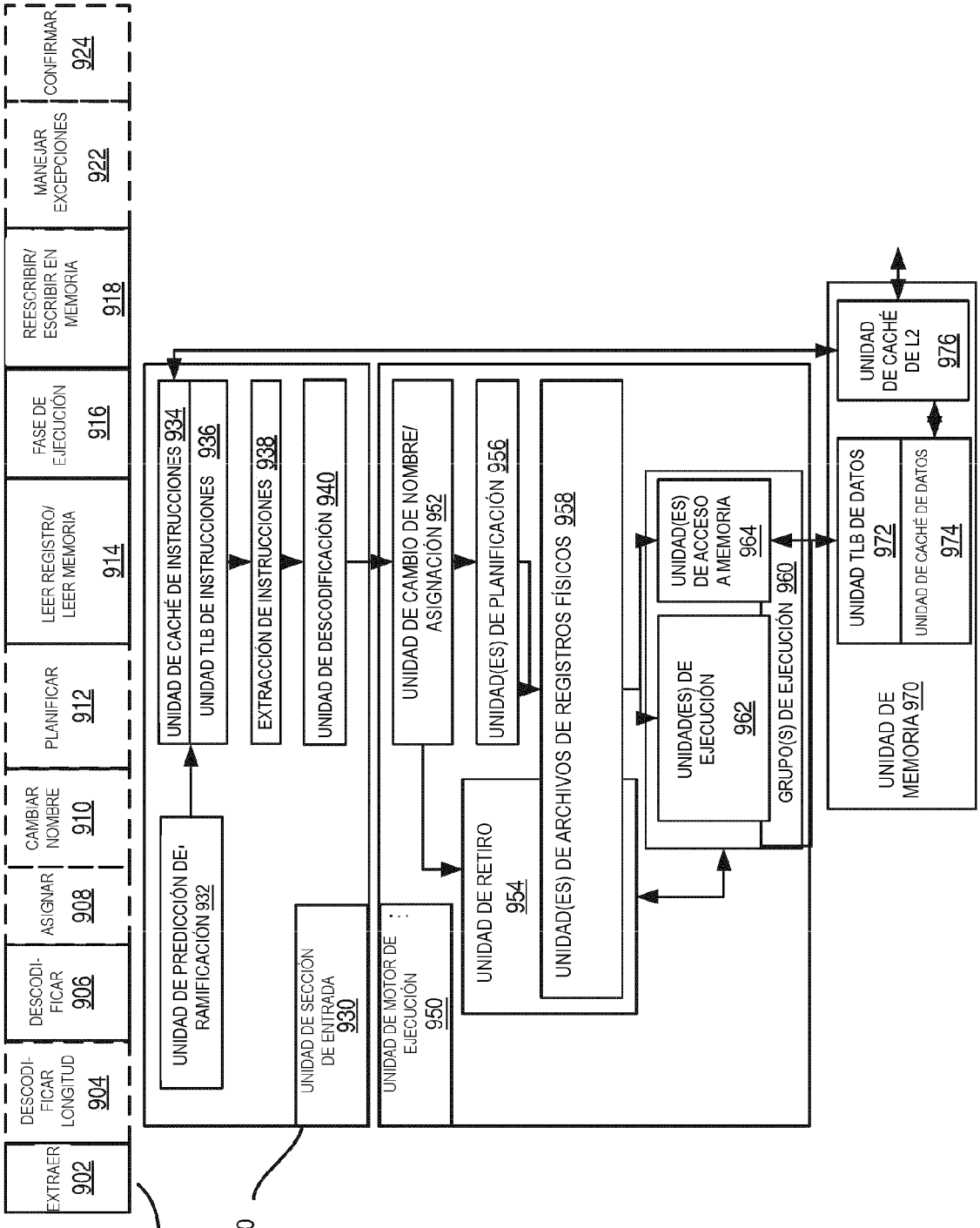

EXTRAER 902
DESCODIFICAR LONGITUD 904
DESCODIFICAR 906
ASIGNAR 908
CAMBIAR NOMBRE 910
PLANIFICAR 912
LEER REGISTRO/ LEER MEMORIA 914
FASE DE EJECUCIÓN 916
REESCRIBIR/ ESCRIBIR EN MEMORIA 918
MANEJAR EXCEPCIONES 922
CONFIRMAR 924
CANALIZACIÓN 900
NÚCLEO 990
UNIDAD DE CACHÉ DE INSTRUCCIONES 934
UNIDAD TLB DE INSTRUCCIONES 936
UNIDAD DE PREDICCIÓN DE RAMIFICACIÓN 932
UNIDAD DE SECCIÓN DE ENTRADA 930
EXTRACCIÓN DE INSTRUCCIONES 938
UNIDAD DE DESCODIFICACIÓN 940
UNIDAD DE MOTOR DE EJECUCIÓN 950
UNIDAD DE CAMBIO DE NOMBRE/ ASIGNACIÓN 952
UNIDAD DE RETIRO 954
UNIDAD(ES) DE PLANIFICACIÓN 956
UNIDAD(ES) DE ARCHIVOS DE REGISTROS FÍSICOS 958
UNIDAD(ES) DE EJECUCIÓN 962
UNIDAD(ES) DE ACCESO A MEMORIA 964
GRUPO(S) DE EJECUCIÓN 960
UNIDAD DE CACHÉ DE L2 976
UNIDAD TLB DE DATOS 972
UNIDAD DE CACHÉ DE DATOS 974
UNIDAD DE MEMORIA 970

FIG. 9A

FIG. 9B

FIG. 10B

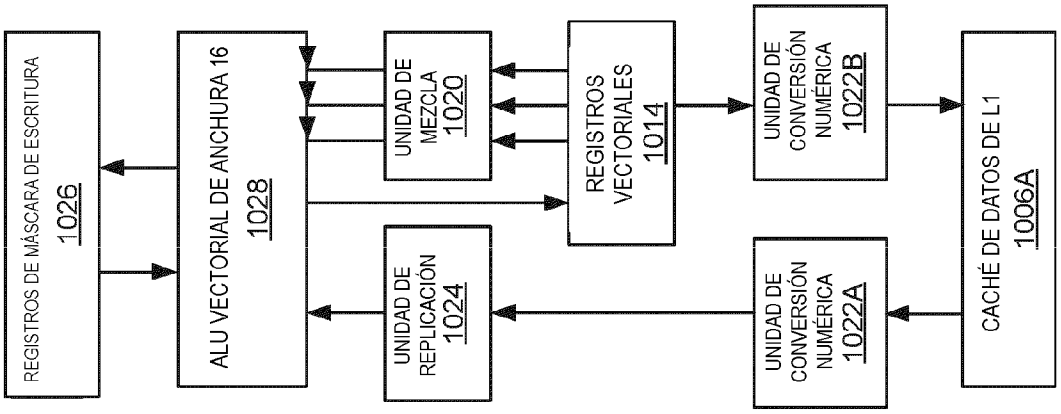
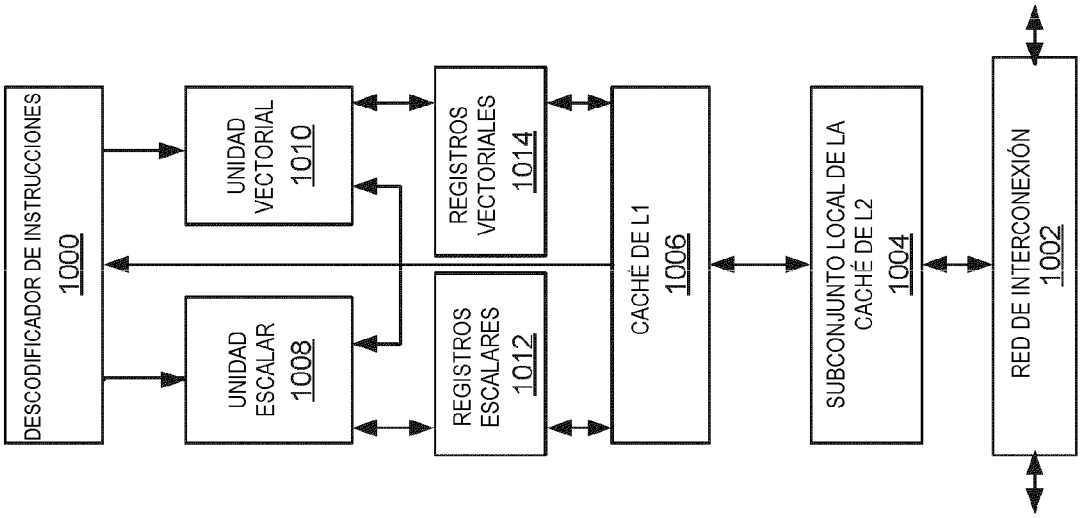


FIG. 10A



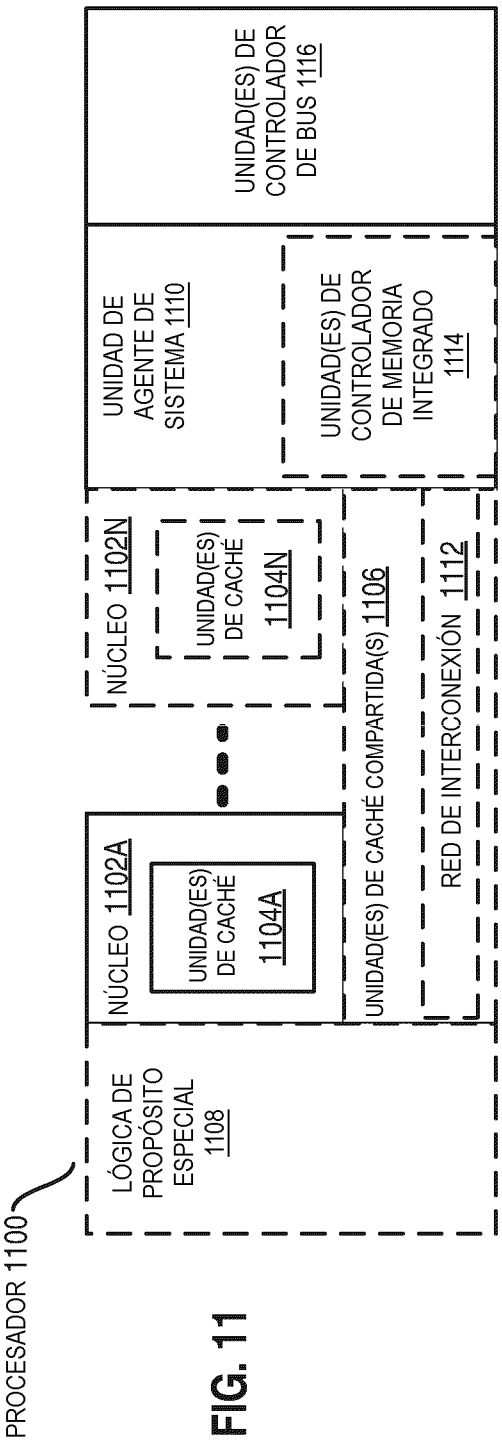

PROCESADOR 1100
LÓGICA DE PROPÓSITO ESPECIAL 1108
NÚCLEO 1102A
UNIDAD(ES) DE CACHÉ 1104A
NÚCLEO 1102N
UNIDAD(ES) DE CACHÉ 1104N
UNIDAD(ES) DE CACHÉ COMPARTIDA(S) 1106
RED DE INTERCONEXIÓN 1112
UNIDAD DE AGENTE DE SISTEMA 1110
UNIDAD(ES) DE CONTROLADOR DE MEMORIA INTEGRADO 1114
UNIDAD(ES) DE CONTROLADOR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

FIG. 11

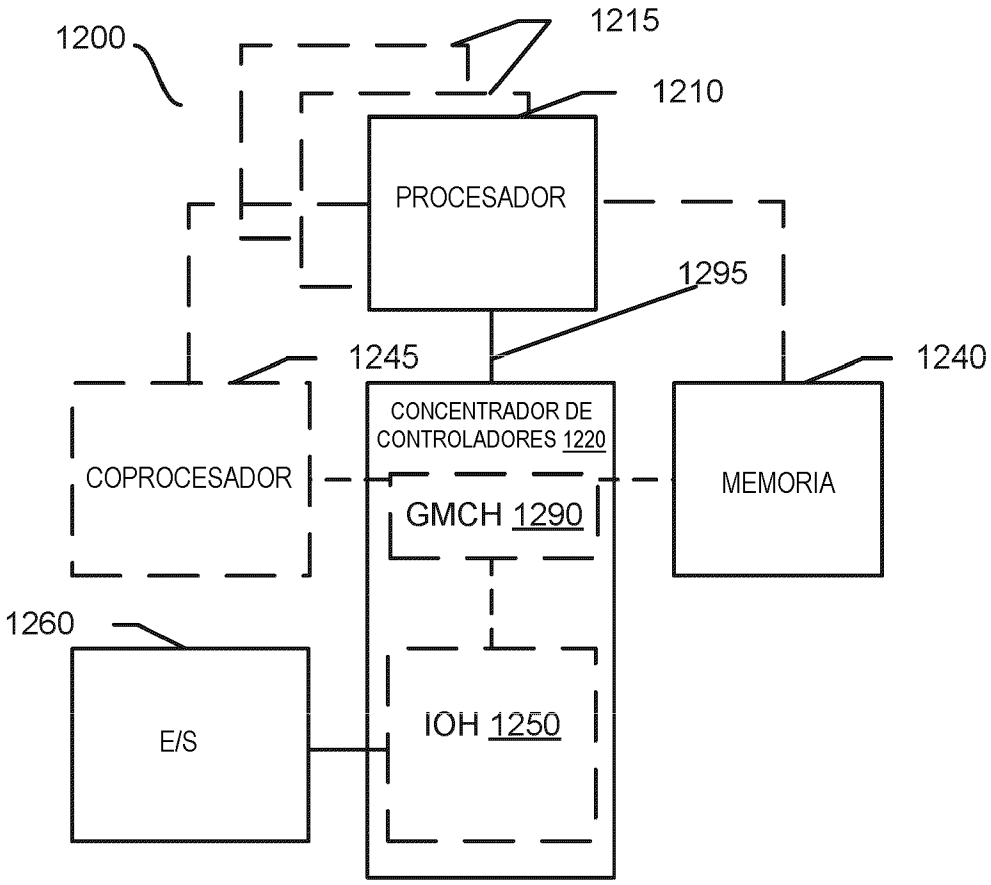

1200
1215
1210
PROCESADOR
1295
1245
1240
CONCENTRADOR DE CONTROLADORES 1220
COPROCESADOR
MEMORIA
GMCH 1290
1260
E/S
IOH 1250

FIG. 12

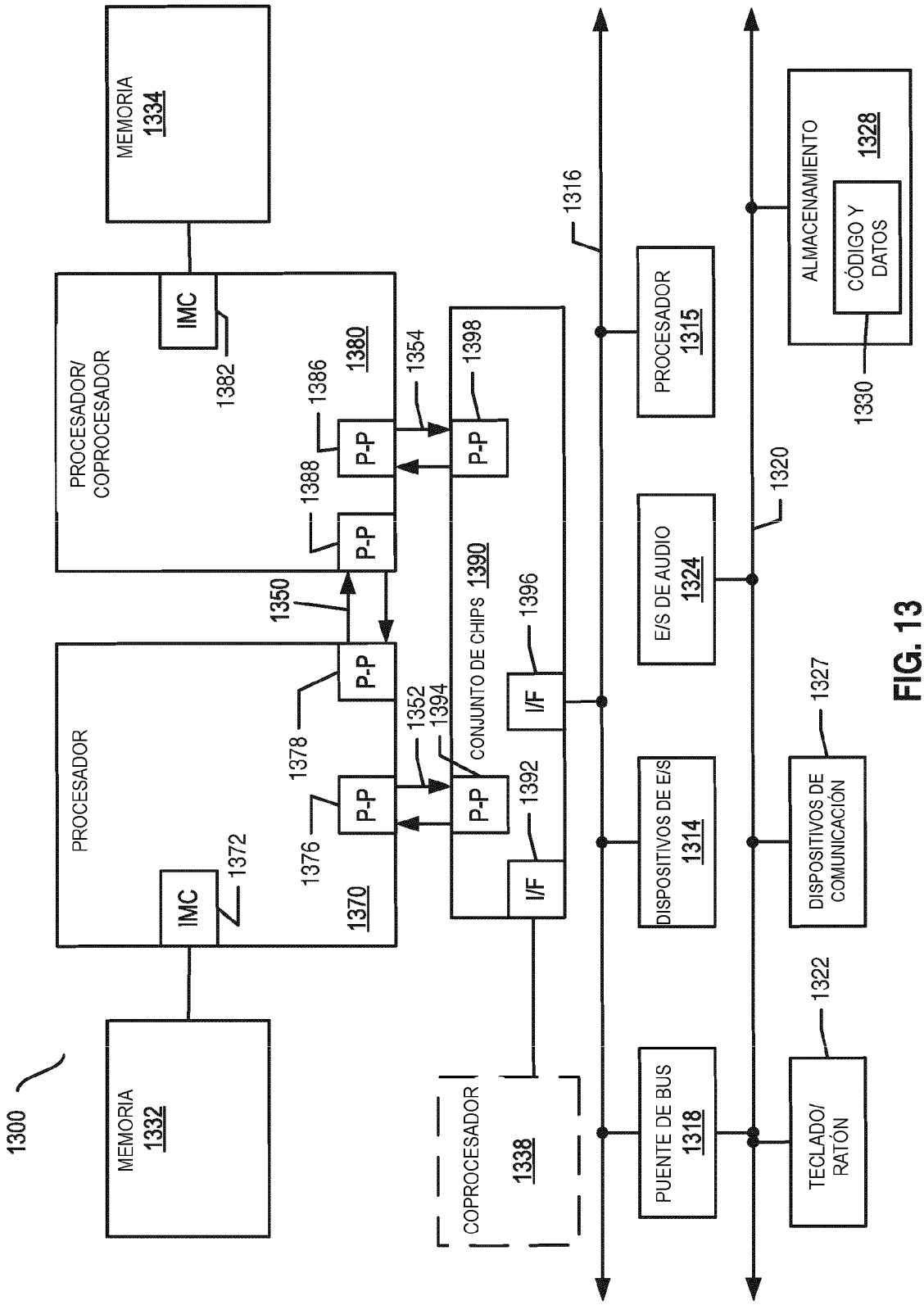

1300
MEMORIA 1332
MEMORIA 1334
PROCESADOR
IMC
1372
1370
PROCESADOR/ COPROCESADOR
IMC
1382
1380
P-P
1376
1378
1386
1388
1350
1352
1354
1394
CONJUNTO DE CHIPS 1390
1392
1396
1398
I/F
COPROCESADOR 1338
1316
PUENTE DE BUS 1318
DISPOSITIVOS DE E/S 1314
PROCESADOR 1315
E/S DE AUDIO 1324
1320
TECLADO/ RATÓN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

FIG. 13

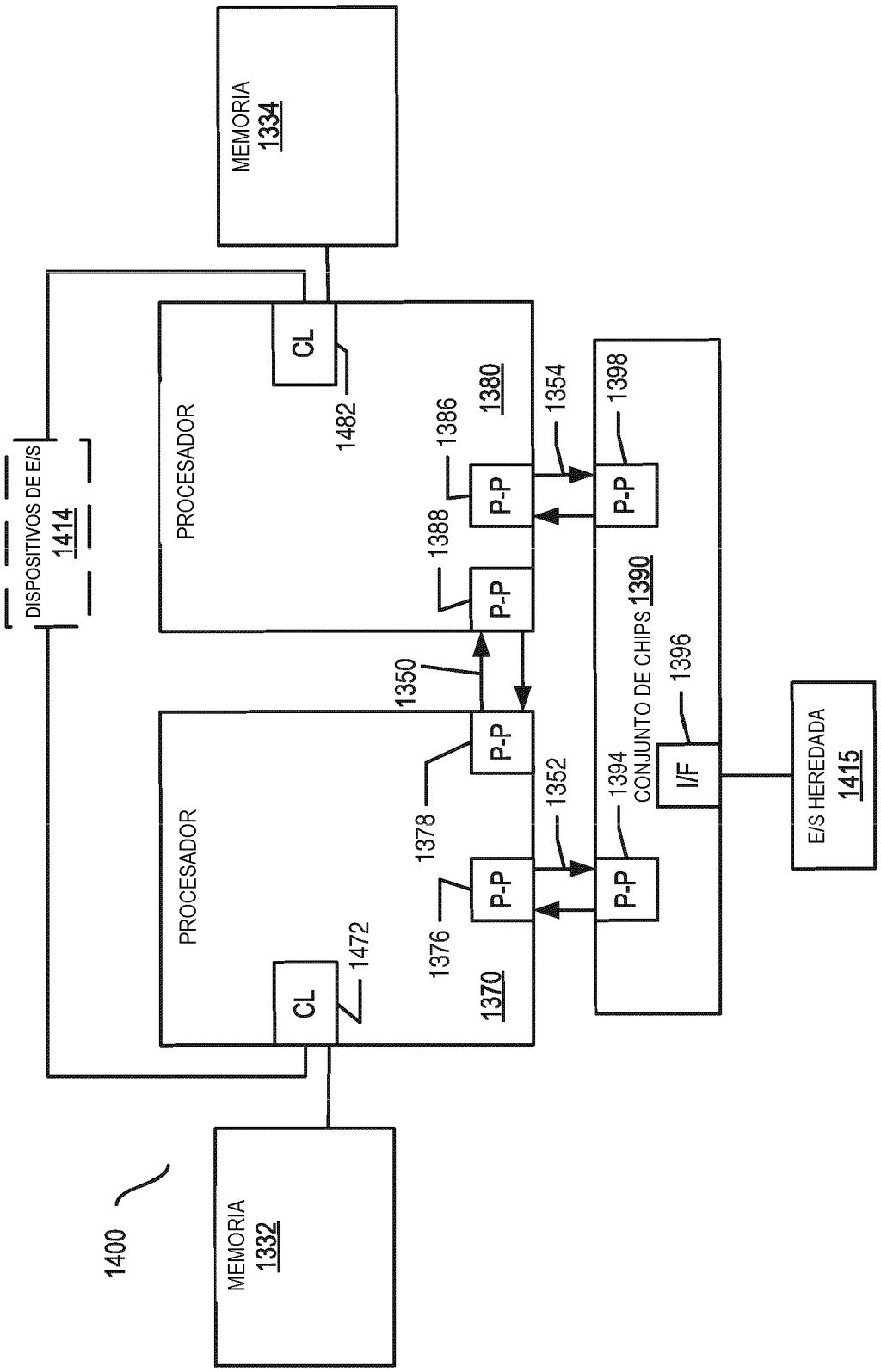

1400
MEMORIA
1332
MEMORIA
1334
DISPOSITIVOS DE E/S
1414
PROCESADOR
CL
1472
1482
1370
1380
P-P
1376
1378
1386
1388
1350
1352
1354
1394
CONJUNTO DE CHIPS 1390
1396
1398
I/F
E/S HEREDADA
1415

FIG. 14

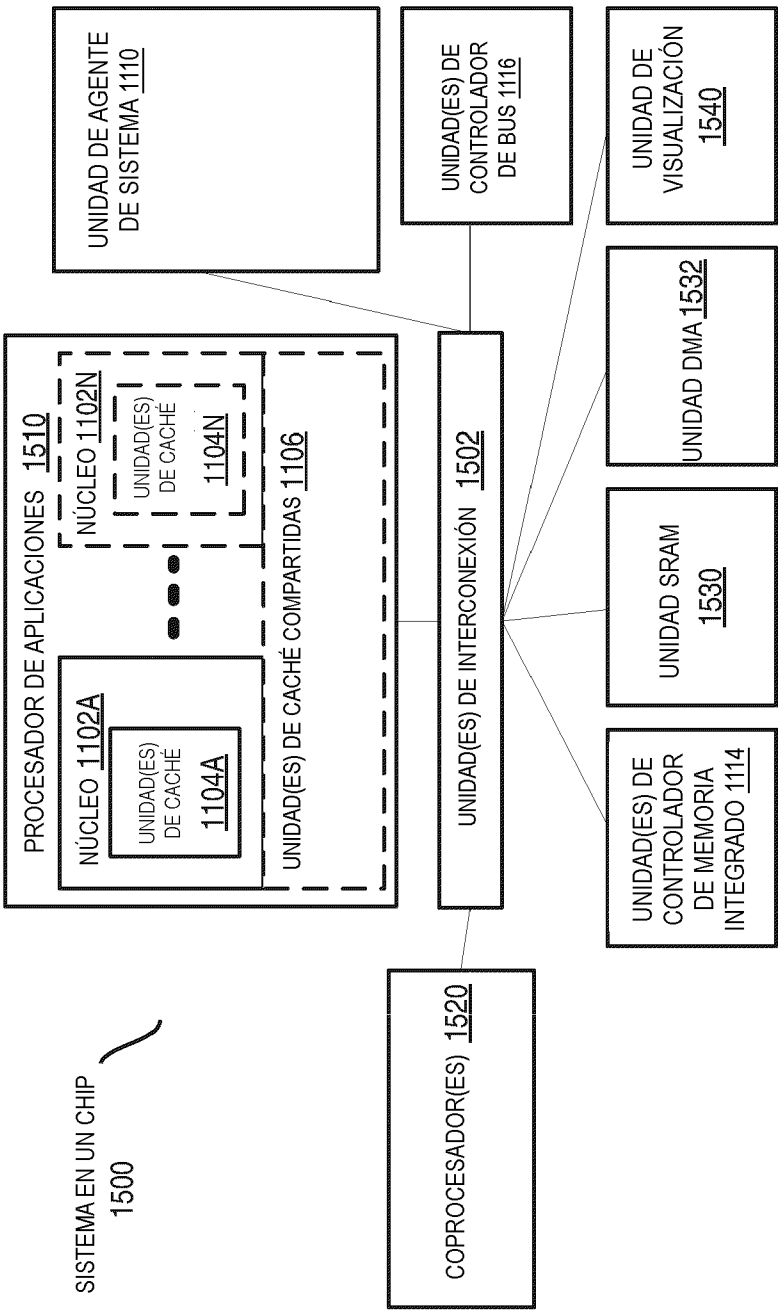

SISTEMA EN UN CHIP
1500
PROCESADOR DE APLICACIONES 1510
NÚCLEO 1102A
UNIDAD(ES) DE CACHÉ 1104A
NÚCLEO 1102N
UNIDAD(ES) DE CACHÉ 1104N
UNIDAD(ES) DE CACHÉ COMPARTIDAS 1106
UNIDAD DE AGENTE DE SISTEMA 1110
UNIDAD(ES) DE CONTROLADOR DE BUS 1116
UNIDAD DE VISUALIZACIÓN 1540
UNIDAD DMA 1532
UNIDAD SRAM 1530
UNIDAD(ES) DE CONTROLADOR DE MEMORIA INTEGRADO 1114
UNIDAD(ES) DE INTERCONEXIÓN 1502
COPROCESADOR(ES) 1520

FIG. 15

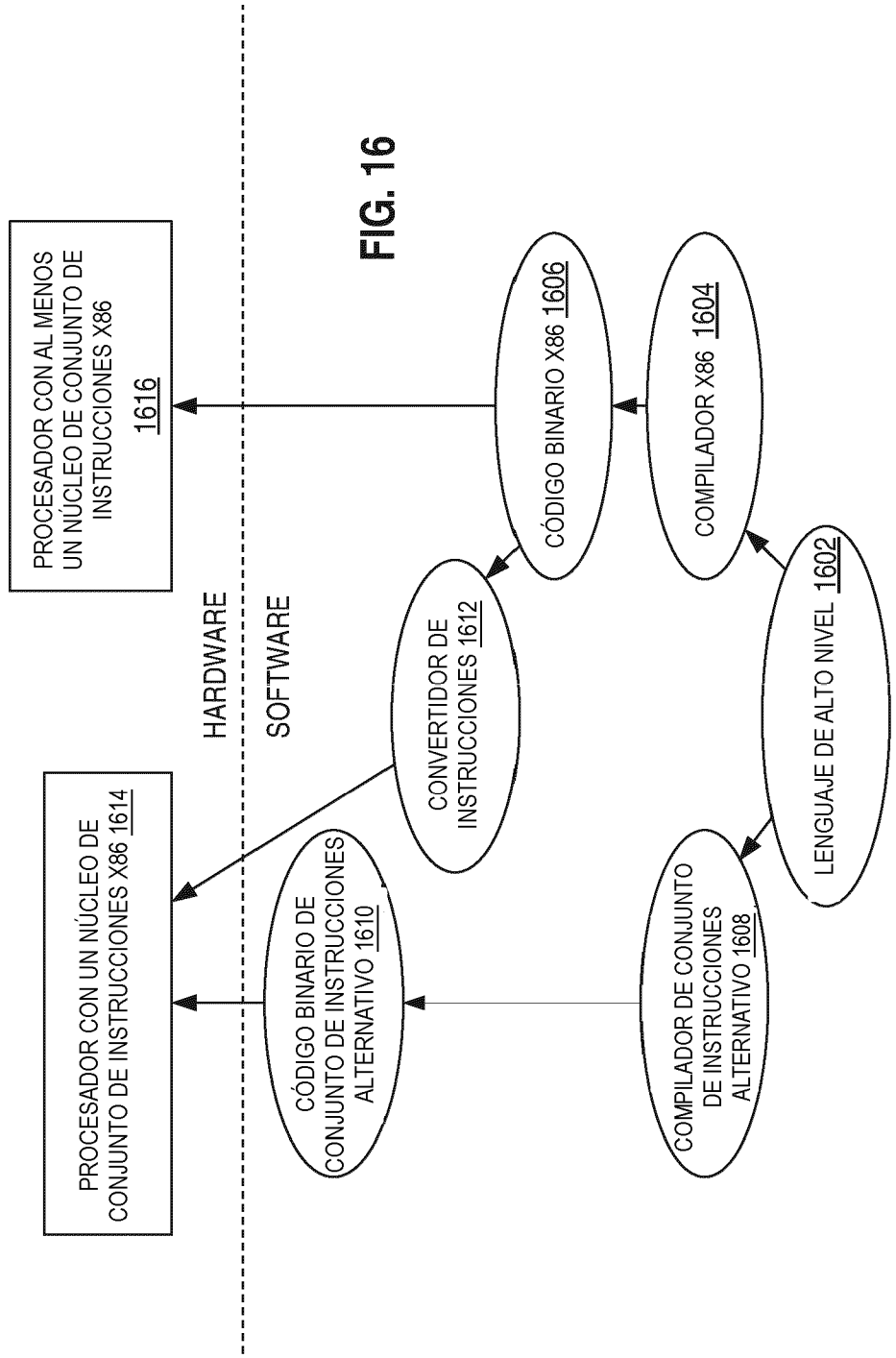

PROCESADOR CON AL MENOS UN NÚCLEO DE CONJUNTO DE INSTRUCCIONES X86 1616
HARDWARE
SOFTWARE
PROCESADOR CON UN NÚCLEO DE CONJUNTO DE INSTRUCCIONES X86 1614
CÓDIGO BINARIO DE CONJUNTO DE INSTRUCCIONES ALTERNATIVO 1610
CONVERTIDOR DE INSTRUCCIONES 1612
CÓDIGO BINARIO X86 1606
COMPILADOR DE CONJUNTO DE INSTRUCCIONES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

FIG. 16