(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **3 019 657**

(51) Int. Cl.:

G06F 9/30 (2008.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **13.02.2019 E 19157044 (9)**

(97) Fecha y número de publicación de la concesión europea: **15.01.2025 EP 3547114**

(54) Título: **Acelerador de multiplicación de matrices densas-dispersas**

(30) Prioridad:

28.03.2018 US 201815938924

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
21.05.2025

(73) Titular/es:

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

(72) Inventor/es:

NARAYANAMOORTHY, SRINIVASAN;
SATISH, NADATHUR RAJAGOPALAN;
SUPRUN, ALEXEY y
JANIK, KENNETH J.

(74) Agente/Representante:

LEHMANN NOVO, María Isabel

DESCRIPCIÓN

Acelerador de multiplicación de matrices densas-dispersas

CAMPO DE LA INVENCIÓN

La presente divulgación pertenece al campo de la lógica de procesamiento, los microprocesadores y las arquitecturas de conjuntos de instrucciones asociadas y, más específicamente, a un acelerador para instrucciones matriciales densas-dispersas.

ANTECEDENTES

El aprendizaje profundo es una clase de algoritmos de aprendizaje automático. Las arquitecturas de aprendizaje profundo, como las redes neuronales profundas, se han aplicado en campos que incluyen visión por ordenador, reconocimiento de voz, procesamiento del lenguaje natural, reconocimiento de audio, filtrado de redes sociales, traducción automática, bioinformática y diseño de fármacos.

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

Las operaciones de multiplicación de matrices densas-dispersas (SDMM) son útiles en un contexto de aprendizaje profundo. Pero las arquitecturas tradicionales de conjuntos de instrucciones de CPU y GPU requieren entradas simétricas que tengan la misma densidad, lo que limita la capacidad de obtener una ventaja de rendimiento aprovechando la dispersión de una matriz de entrada dispersa.

MORAD AMIR ET AL, "Efficient Dense and Sparse Matrix Multiplication on GP-SIMD", 2014 24TH INTERNATIONAL WORKSHOP ON POWER AND TIMING MODELING, OPTIMIZATION AND SIMULATION (PATMOS), IEEE, 20140929, se refiere a la multiplicación eficiente de matrices densas y dispersas en GP-SIMD. El GP-SIMD es una arquitectura híbrida de computación SIMD de propósito general que elimina la sincronización mediante la computación en memoria, combinando el almacenamiento de datos y el procesamiento paralelo masivo.

SUMARIO

La presente invención se define en las reivindicaciones independientes 1 y 8. Las reivindicaciones dependientes definen realizaciones de las mismas.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

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

la **Figura 1** es un diagrama de bloques que ilustra componentes de procesamiento para ejecutar una instrucción de multiplicación de matrices densas-dispersas (SDMM), tal como una instrucción de red neuronal virtual (SDMM), según algunas realizaciones;

la **Figura 2A** es un diagrama de bloques que ilustra un flujo de datos para procesar una instrucción de multiplicación de matrices densas-dispersas (SDMM), según algunas realizaciones;

la **Figura 2B** es un diagrama de bloques que ilustra un flujo de datos para procesar una instrucción de multiplicación de matrices densas-dispersas (SDMM), según algunas realizaciones;

la **Figura 3** es un diagrama de bloques que ilustra circuitería de ejecución para procesar una instrucción de multiplicación de matrices densas-dispersas (SDMM), según algunas realizaciones;

la **Figura 4** es un diagrama de bloques que ilustra circuitería de ejecución para procesar una instrucción de multiplicación de matrices densas-dispersas (SDMM), según algunas realizaciones;

la **Figura 5** es un pseudocódigo que ilustra circuitería de ejecución para procesar una instrucción de multiplicación de matrices densas-dispersas (SDMM), según algunas realizaciones;

la **Figura 6** es un diagrama de flujo de un proceso que ilustra la ejecución de una instrucción de multiplicación de matrices densas-dispersas (SDMM) mediante un procesador, según algunas realizaciones;

la **Figura 7** es un diagrama de bloques que ilustra un formato de una instrucción de multiplicación de matrices densas-dispersas (SDMM), según algunas realizaciones;

las **Figuras 8A-8B** son diagramas de bloques que ilustran un formato de instrucciones genérico compatible con vectores y plantillas de instrucciones del mismo según algunas realizaciones;

la **Figura 8A** es un diagrama de bloques que ilustra un formato de instrucciones genérico compatible con vectores y plantillas de instrucciones de clase A del mismo según algunas realizaciones;

la **Figura 8B** es un diagrama de bloques que ilustra el formato de instrucciones genérico compatible con vectores y plantillas de instrucciones de clase B del mismo según algunas realizaciones;

la **Figura 9A** es un diagrama de bloques que ilustra un formato de instrucciones específico compatible con vectores ilustrativo, según algunas realizaciones;

la **Figura 9B** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico que componen el campo de código de operación completo según una realización de la invención;

la **Figura 9C** es un diagrama de bloques que ilustra los campos del formato de instrucción compatible con vectores específico que componen el campo de índice de registro según una realización de la invención;

la **Figura 9D** es un diagrama de bloques que ilustra los campos del formato de instrucciones específico compatible con vectores que constituye el campo de operación de aumento según una realización de la invención;

la **Figura 10** es un diagrama de bloques de una arquitectura de registro según una realización de la invención;

la **Figura 11A** es un diagrama de bloques que ilustra tanto una canalización ilustrativa en orden como una canalización ilustrativa de emisión/ejecución fuera de orden y de cambio de nombre de registro según algunas realizaciones;

la **Figura 11B** 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 fuera de orden y cambio de nombre de registro que se incluirá en un procesador según algunas realizaciones;

las **Figuras 12A-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 12A** 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), según algunas realizaciones;

la **Figura 12B** es una vista ampliada de parte del núcleo de procesador de la **Figura 12A** según algunas realizaciones;

la **Figura 13** 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 según algunas realizaciones;

las **Figuras 14-17** son diagramas de bloques de arquitecturas informáticas ilustrativas;

la **Figura 14** muestra un diagrama de bloques de un sistema según una realización de la presente invención;

la **Figura 15** es un diagrama de bloques de un primer sistema ilustrativo más específico según una realización de la presente invención;

la **Figura 16** es un diagrama de bloques de un segundo sistema ilustrativo más específico según una realización de la presente invención;

la **Figura 17** es un diagrama de bloques de un sistema en un chip (SoC) según una realización de la presente invención; y

la **Figura 18** 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 según algunas realizaciones.

DESCRIPCIÓN DETALLADA

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

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

Las realizaciones divulgadas maximizan el rendimiento de ejecución de una instrucción de multiplicación de matrices densas-dispersas (SDMM) que tiene entradas de precisión variable, tal como una instrucción de multiplicación de matrices de red neuronal virtual (VNN) que tiene una entrada de matriz dispersa y una entrada de matriz densa. Usando la circuitería divulgada en el presente documento, se espera que las instrucciones de SDMM divulgadas produzcan una ganancia de rendimiento con respecto a las instrucciones de multiplicación de matrices que tienen operandos simétricos.

Se espera que las realizaciones divulgadas, al no usar un circuito de multiplicación de matrices simétrico convencional, mejoren el rendimiento de instrucciones de SDMM de un ordenador por un factor proporcional a la dispersión de la matriz de entrada dispersa. Tal y como se usa en el presente documento, la dispersión se refiere a la proporción de elementos de la matriz que tienen valores cero o nulos. Por ejemplo, se espera que una instrucción de SDMM que opera en una matriz dispersa que tiene una dispersión de 0,125, teniendo solo 1/8 de los elementos valores distintos de cero, tenga un rendimiento 8 veces mayor. A modo de otro ejemplo, se espera que una instrucción de SDMM que opera en una matriz dispersa que tiene una dispersión de 0,05, teniendo solo 1/20 de los elementos valores distintos de cero, tenga un rendimiento 20 veces mayor.

En algunas realizaciones, las instrucciones de SDMM se ejecutan mediante una circuitería de ejecución que tiene carriles de procesamiento de SIMD usando una rejilla de circuitos multiplicación-suma fusionada (FMA). El ancho del carril de SIMD puede diferir en diferentes realizaciones. Por ejemplo, el ancho de carril de SIMD puede incluir cualquiera de 16 elementos, 32 elementos, 64 elementos y 128 elementos, y los elementos pueden ser cualquiera de 8 bits, 16 bits, 32 bits, 64 bits y 128 bits, sin limitación. Los carriles de SIMD se usan para ejecutar una instrucción en paralelo en múltiples elementos de datos.

En algunas realizaciones, se usan memorias de múltiples bancos para datos intermedios y almacenamiento de resultados mediante los múltiples carriles de SIMD. Por ejemplo, una circuitería de ejecución de SIMD que tiene 8 carriles de SIMD de 64 bits puede usar una memoria de 8 bancos.

La **Figura 1** es un diagrama de bloques que ilustra componentes de procesamiento para ejecutar una instrucción de multiplicación de matrices densas-dispersas (SDMM), tal como una instrucción de red neuronal virtual (SDMVNNI), que tiene entradas asimétricas según algunas realizaciones. Una instrucción de red neuronal virtual se aplica en un contexto de aprendizaje profundo y es un tipo de instrucción que puede beneficiarse de una multiplicación de matrices densas-dispersas. Existen otros tipos de aplicaciones que pueden beneficiarse de una instrucción de SDMM, tal como las nuevas instrucciones de campo Galois (GFNI). Por tanto, las instrucciones de SDMM divulgadas no pretenden limitarse a las instrucciones VNNI. Como se ilustra, el almacenamiento 101 almacena una instrucción de SDMM 103 para ser ejecutada.

La instrucción de SDMM 103 se extrae del almacenamiento 101 mediante el circuito de extracción 105. La instrucción de SDMM 107 extraída se decodifica mediante el circuito de decodificación 109. Por ejemplo, el circuito de decodificación 109 recibe la instrucción de SDMM 107 extraída del circuito de extracción 105. El formato de instrucciones de SDMM, como se describe más adelante y con respecto a las **Figuras 7-9,** tiene campos para especificar un código de operación, una matriz de salida densa, una matriz de origen densa y una matriz de origen dispersa que tiene una dispersión de elementos distintos de cero que son inferiores a uno. El circuito de decodificación 109 decodifica la instrucción de SDMM 107 extraída en una o más operaciones. En algunas realizaciones, esta decodificación incluye la generación de una pluralidad de microoperaciones que realizarán la circuitería de ejecución (tales como la circuitería de ejecución 117). El circuito de decodificación 109 también decodifica sufijos y prefijos de instrucciones (si se usan). La circuitería de ejecución 117 se describe e ilustra con más detalle más adelante, al menos con respecto a las **Figuras 2-6** y **Figuras 11-12,** incluidas más adelante.

En algunas realizaciones, el circuito de cambio de nombre de registros, asignación de registros y/o planificación 113 proporciona una o más de las siguientes funcionalidades: 1) cambiar el nombre de los valores de operandos lógicos a valores de operandos físicos (por ejemplo, una tabla de alias de registros en algunas realizaciones), 2) asignar bits de estado y banderas a la instrucción decodificada, y 3) planificar la instrucción de SDMM decodificada 111 para su

ejecución en la circuitería de ejecución 117 fuera de una agrupación de instrucciones (por ejemplo, usando una estación de reserva en algunas realizaciones).

Los registros (archivo de registro) y/o la memoria 115 almacenan datos como operandos de la instrucción de SDMM decodificada 111 para ser operados por la circuitería de ejecución 117. Los tipos de registro ilustrativos incluyen registros de máscara de escritura, registros de datos empaquetados, registros de propósito general y registros de coma flotante, como se describe e ilustra con mayor detalle a continuación, al menos con respecto a la **Figura 10.**

En algunas realizaciones, el circuito de reescritura 119 confirma el resultado de la ejecución de la instrucción de SDMM decodificada 111.

La **Figura 2A** es un diagrama de bloques que ilustra un flujo de datos para procesar una instrucción de multiplicación de matrices densas-dispersas (SDMM), según algunas realizaciones. Como se muestra, la instrucción de SDMM 202 tiene campos para especificar un código de operación 204 (SDMMVNNIW), una matriz de salida densa 206, una matriz de origen densa 210 y una matriz de origen dispersa 208. Como se muestra, la matriz de origen dispersa 212 especificada es lógicamente una matriz de M=3 filas por K=6 columnas que tiene una dispersión de elementos distintos de cero que es aproximadamente igual al 16,67 %. En otras palabras, solo el 16,67 % de los elementos de la matriz de origen dispersa tienen valores distintos de cero. El procesamiento de la instrucción de SDMM 202 según las realizaciones divulgadas en el presente documento mejora el rendimiento del procesador hasta seis veces, en comparación con el uso de un circuito de multiplicación de matrices simétrico convencional. Las realizaciones divulgadas evitan desperdiciar ciclos de procesamiento en elementos de valor cero de la matriz de origen dispersa 212 especificada. En algunas realizaciones, la dispersión de la matriz de origen dispersa 212 especificada está limitada a menos del 20 %. En el contexto de las redes neuronales virtuales, y como se muestra, las matrices de origen densas y dispersas pueden representar una matriz de activación y un vector de ponderaciones.

En algunas realizaciones, la matriz de origen dispersa 212 especificada es lógicamente una matriz de M por K, pero solo sus elementos distintos de cero se almacenan en la memoria en un formato de fila dispersa comprimida (CSR) o de columna dispersa comprimida (CSC), que en algunas realizaciones se prepara con antelación. Los formatos CSC y CSR se describen con más detalle más adelante, al menos con referencia a la **Figura 4.**

En algunas realizaciones, la instrucción de SDMM 202 especifica además un tamaño de elemento (en este caso, el código de operación 204 incluye un sufijo "W", que especifica **E**lementos de tamaño normal). El formato de la instrucción de SDMM se ilustra y describe con más detalle más adelante, al menos con respecto a las **Figuras 7-9.** En algunas realizaciones, una o más de las matrices identificadas se almacenan en registros, tales como en un archivo de registro de un procesador, por ejemplo, como se ilustra y analiza más adelante con referencia a la **Figura 10.** En algunas realizaciones, una o más de las matrices identificadas se almacenan en una ubicación de memoria.

Como se muestra, la matriz de origen dispersa 212 especificada es una matriz que lógicamente tiene M filas (igual a 3) y K columnas (igual a 6), con elementos distintos de cero en (0,0), (3,3) y (2,5). Por consiguiente, la matriz de origen dispersa 212 especificada tiene una dispersión de aproximadamente 16,67 %, y el procesamiento de la instrucción según las realizaciones divulgadas proporciona una mejora de hasta seis veces en el rendimiento del procesador. La matriz de origen densa 214 especificada tiene K filas (igual a 6) y N columnas (igual a 6). La matriz de salida densa 216 especificada se muestra con M filas y N columnas. Como se describe en el presente documento, las letras mayúsculas, M, N y K, se usan para hacer referencia a las dimensiones máximas de las matrices, mientras que las letras minúsculas, m, n y k, se usan como índices de las posiciones de los elementos dentro de las matrices.

En funcionamiento, como lo muestran los indicadores de flujo de datos 218, 220 y 222, para cada elemento distinto de cero en la fila m y columna k de la matriz de origen dispersa 212 especificada, la circuitería de ejecución genera un producto del elemento distinto de cero y todos los correspondientes elementos densos en la fila k de la matriz de origen densa 214 especificada. A continuación, la circuitería de ejecución acumula cada producto generado con valores anteriores de un elemento de salida correspondiente en la fila m de la matriz de salida densa 216 especificada. En algunas realizaciones, la circuitería de ejecución escribe la suma acumulada en los elementos correspondientes de la matriz de salida densa 216 especificada. En algunas realizaciones, la circuitería de ejecución escribe la suma acumulada en una memoria de bloc de notas (no mostrada) antes de escribir en la matriz de salida densa.

La **Figura 2B** es un diagrama de bloques que ilustra un flujo de datos para procesar una instrucción de multiplicación de matrices densas-dispersas (SDMM), según algunas realizaciones. Como se muestra, la instrucción de SDMM 252 tiene campos para especificar un código de operación 254 (SDMMVNNI), una matriz de salida densa 256, una matriz de origen densa 260 y una matriz de origen dispersa 258. La matriz de origen dispersa 262 especificada es lógicamente una matriz de M=4 por K=4, con elementos distintos de cero en la primera y tercera columnas. Por consiguiente, la matriz de origen dispersa 262 especificada tiene una dispersión de 0,5. En otras palabras, el 50 % de los elementos de la matriz de origen dispersa 262 especificada tienen valores distintos de cero. La instrucción de SDMM 252, procesada según las realizaciones divulgadas en el presente documento, mejora el rendimiento del procesador dos veces, en comparación con el uso de un circuito de multiplicación de matrices simétrico convencional. Las realizaciones divulgadas evitan desperdiciar ciclos de procesamiento en elementos de origen de valor igual a cero. En algunas realizaciones, la dispersión de la matriz de origen dispersa 262 especificada está limitada a menos del 10 %.

En algunas realizaciones, una o más de las matrices identificadas se almacenan en registros, tales como en un archivo de registro de un procesador, por ejemplo, como se ilustra y analiza más adelante con referencia a la **Figura 10.** En algunas realizaciones, una o más de las matrices de origen dispersas, de origen densas y de salida densas especificadas se almacenan en una ubicación de memoria.

Como se muestra, la matriz de origen dispersa 262 especificada es una matriz que lógicamente tiene M filas (igual a 4) y K columnas (igual a 4), con ocho elementos distintos de cero en la columna 0 y la columna 2. La matriz de origen densa 270 especificada tiene K filas (igual a 4) y N columnas (igual a 4). Como se describe en el presente documento, las letras mayúsculas, M, N y K, se usan para hacer referencia a las dimensiones máximas de las matrices, mientras que las letras minúsculas, m, n y k, se usan para referirse a índices de las posiciones del elemento dentro de las matrices.

En funcionamiento, según algunas realizaciones, para cada elemento distinto de cero en la fila m y la columna k de la matriz de origen dispersa 262 especificada, la circuitería de ejecución genera un producto del elemento distinto de cero y cada elemento correspondiente en la fila k y la columna {0, n-1} de la matriz de origen densa 264 especificada. En esta realización, como se muestra en la etapa 1, 266, cada uno de los elementos distintos de cero de la primera columna de la matriz de origen dispersa 262 especificada se multiplica por cada elemento en la primera fila correspondiente de la matriz de origen densa 264 especificada. Como se muestra en la etapa 2, 268, cada uno de los elementos distintos de cero de la tercera columna de la matriz de origen dispersa 262 especificada se multiplica por cada elemento en la tercera fila correspondiente de la matriz de origen densa 264 especificada. A continuación, la circuitería de ejecución acumula los productos generados en la etapa 1, 266, y la etapa 2, 268, con los valores anteriores del elemento de salida correspondiente en la fila m y la columna n de la matriz de salida densa 270 especificada. Para simplificar, en este caso no se muestran los valores anteriores de la matriz de salida, pero se supone que son cero. En algunas realizaciones, la circuitería de ejecución escribe las sumas acumuladas en los elementos correspondientes de la matriz de salida densa. En algunas realizaciones, la circuitería de ejecución escribe la suma acumulada en una memoria de bloc de notas antes de escribir en la matriz de salida densa.

La **Figura 3** es un diagrama de bloques que ilustra un circuito de ejecución para procesar una instrucción de multiplicación de matrices densas-dispersas (SDMM), según algunas realizaciones. Como se muestra, la instrucción de SDMM 302 tiene campos para especificar un código de operación 304 (SDMMVNNI), una matriz de salida densa 306, una matriz de origen dispersa 308 y una matriz de origen densa 310. Como se ilustra, la matriz de origen dispersa 312 especificada tiene una dispersión de elementos distintos de cero que es, a efectos de ilustración, de aproximadamente 0,07. En otras palabras, aproximadamente el 7 % de los elementos de la matriz de origen dispersa especificada tienen valores distintos de cero. El procesamiento de la instrucción de SDMM según las realizaciones divulgadas en el presente documento mejora el rendimiento del procesador aproximadamente 14 veces, en comparación con el uso de un circuito de multiplicación de matrices simétrico convencional. Las realizaciones divulgadas evitan desperdiciar ciclos de procesamiento en elementos de origen de valor igual a cero. En algunas realizaciones, la dispersión de la matriz de origen dispersa está limitada a menos del 10 %. En algunas realizaciones, una o más de las matrices identificadas se almacenan en registros, tales como en un archivo de registro de un procesador, por ejemplo, como se ilustra y analiza más adelante con referencia a la **Figura 10.** En algunas realizaciones, una o más de las matrices identificadas se almacenan en una ubicación de memoria.

Como se muestra, la matriz de origen dispersa 312 especificada es una matriz que tiene M filas (igual a 8) y K columnas (igual a 2), con un elemento distinto de cero en la fila 6 y la columna 1. La matriz de origen densa 314 especificada tiene K filas (igual a 2) y N columnas (igual a 6). La matriz de salida densa 316 especificada se muestra con M filas (igual a 8) y N columnas (igual a 6).

En funcionamiento, según algunas realizaciones, para cada elemento distinto de cero en la fila m y la columna k de la matriz de origen dispersa 312 especificada, la circuitería de ejecución genera un producto del elemento distinto de cero y cada elemento denso correspondiente en la fila k y la columna n de la matriz de origen densa 314 especificada. Como se muestra, el elemento distinto de cero en el elemento (6,1) de la matriz de origen dispersa 312 especificada se multiplica por cada elemento en la fila correspondiente 1 de la matriz de origen densa 314 especificada usando multiplicadores 318. La circuitería de ejecución genera a continuación una suma acumulada de los productos generados por los multiplicadores 318 y los valores anteriores de los elementos correspondientes de la matriz de salida densa 316 especificada usando sumadores/acumuladores 320. En este caso, no se muestran los valores anteriores de la matriz de salida, pero, por simplicidad, se supone que son cero. En algunas realizaciones, la circuitería de ejecución escribe las sumas acumuladas en los elementos correspondientes de la matriz de salida densa 316 especificada. En algunas realizaciones, la circuitería de ejecución escribe las sumas acumuladas en una memoria de bloc de notas 322 antes de escribir en la matriz de salida densa 316 especificada.

La circuitería de ejecución para ejecutar la instrucción de SDMM, según las realizaciones divulgadas, se ilustran y analizan con más detalle al menos con respecto a las **Figuras 4-6,** y las **Figuras 11-12.**

La **Figura 4** es un diagrama de bloques que ilustra un circuito de ejecución 411 para procesar una instrucción de multiplicación de matrices densas-dispersas (SDMM), según algunas realizaciones. Como se muestra, la instrucción de SDMM 402 tiene campos para especificar un código de operación 404 (SDMMVNNI), una matriz de salida densa

406, una matriz de origen dispersa 408 y una matriz de origen densa 410. En la realización ilustrada, las matrices de salida densa, de origen densa y de origen dispersa especificadas tienen dimensiones M=N=256 y K=512 de entradas de 2 bytes de precisión y la matriz de origen dispersa especificada tiene una dispersión de 0,125.

FORMATO DE FILA DISPERSA COMPRIMIDA/COLUMNA DISPERSA COMPRIMIDA

En algunas realizaciones, la matriz de origen dispersa especificada se almacena en formato disperso en la memoria 412, de tal manera que el circuito de ejecución 411 lee la matriz con formato disperso, pero solo almacena en memoria intermedia los valores distintos de cero en memorias intermedias dispersas 414.

Sin embargo, para evitar accesos innecesarios a la memoria y conservar el espacio de memoria, algunas realizaciones almacenan solo los elementos distintos de cero de la matriz de origen dispersa en la memoria 412 en formato de fila dispersa comprimida (CSR) o de columna dispersa comprimida (CSC). Con los formatos CSR y CSC, solo se almacenan los elementos distintos de cero de la matriz de origen dispersa, organizados en formato de fila principal o formato de columna principal, respectivamente. En algunas realizaciones, las matrices de origen dispersas con formato CSR o CSC se preparan en la memoria 412 antes de la operación mediante hardware o software especializado.

MEMORIAS INTERMEDIAS DISPERSAS 414

La matriz de origen dispersa especificada en la realización ilustrada es lógicamente una matriz de 256X512 que tiene 32 elementos distintos de cero por columna, es decir, con una dispersión de 0,125. La realización ilustrada usa el formato CSC para almacenar la matriz de origen dispersa en las memorias intermedias dispersas 414 y cada columna de la matriz dispersa de origen dispersa se divide en 8 bancos basándose en el índice de fila M. En funcionamiento, 32 elementos distintos de cero por columna de la matriz de origen dispersa se almacenan en 8 bancos de las memorias intermedias dispersas 414, distribuidos equitativamente entre ellos en el caso ideal. En algunas realizaciones, cada una de las entradas de la memoria intermedia dispersa usa hasta cinco bytes, que incluyen dos bytes para almacenar un valor de datos y hasta 3 bytes para especificar una posición de matriz del elemento dentro de la matriz de origen dispersa especificada. En algunas realizaciones, la matriz de origen dispersa especificada tiene lógicamente M=16 filas y K=16 columnas, y la posición de la matriz de 1 byte especifica un desplazamiento del elemento dentro de los 256 elementos. En algunas realizaciones, la matriz de origen dispersa especificada tiene M=16 filas y K=16 columnas y la posición de la matriz de 1 byte incluye un cuarteto para especificar la columna y un cuarteto para especificar la fila en la que se encuentra el elemento.

ARREGLO DE MULTIPLICADORES 416

En la realización ilustrada, el arreglo de multiplicadores tiene un tamaño de 8X32 donde las 8 filas del arreglo de multiplicadores están conectadas a los 8 bancos de las memorias intermedias dispersas 414 que proporcionan los 8 valores multiplicadores por ciclo necesarios para la multiplicación. 32 elementos de la fila k de la matriz de origen densa especificada que forman el multiplicando se difunden y se multiplican a través de 8 elementos de las memorias intermedias dispersas 414, realizando así 256 multiplicaciones por ciclo. El producto parcial de 32 elementos generado por banco se acumula a continuación en el arreglo de acumuladores 418. En algunas realizaciones, los 32 elementos de la matriz de origen densa especificada se almacenan en registros (no mostrados) antes de las multiplicaciones. En algunas realizaciones, como se muestra, los 32 elementos se introducen en el arreglo de multiplicadores 416 a medida que se cargan desde la memoria 412. En algunas realizaciones, el arreglo de multiplicadores 416 comprende una rejilla de unidades de hardware de multiplicación-suma-fusionada (FMA).

ARREGLO DE ACUMULADORES 418

Como se muestra, el arreglo de acumuladores 418 incluye 256 acumuladores divididos en 8 bancos, cada uno de ellos conectado al banco correspondiente de memorias intermedias dispersas 414 y el arreglo de multiplicadores 416. En las operaciones, los acumuladores (32 por banco) del arreglo de acumuladores 418 acumulan los productos generados por el arreglo de multiplicadores 416 con valores anteriores de los elementos correspondientes de la matriz de salida densa especificada por el campo de la matriz de salida densa 406 de la instrucción de SDMM 402.

CIRCUITO DE REDONDEO Y SATURACIÓN 420

En algunas realizaciones, los productos generados por el arreglo de multiplicadores 416 y acumulados por el arreglo de acumuladores 418 son resultados intermedios de alta precisión representados por al menos el doble de bits que los usados por los elementos de datos de las matrices especificadas. En algunas realizaciones, el circuito de redondeo y saturación 420 satura los resultados intermedios a un máximo predefinido y los redondea para que quepan dentro del número de bits de elementos de la matriz de salida densa especificados por el campo de matriz de salida densa 406 de la instrucción de SDMM 402, que en este caso es de 16 bits.

En el caso de la aritmética de coma flotante, el circuito de redondeo y saturación 420 puede redondear los resultados intermedios según el estándar de coma flotante IEEE 754, establecido en 1985 y actualizado en 2008 por el Instituto de Ingenieros Eléctricos y Electrónicos. El estándar de coma flotante IEEE 754 define las reglas de redondeo que se

aplicarán, incluyendo el redondeo al valor más cercano con empates a par, el redondeo al valor más cercano con empates lejos de cero, hacia cero, hacia infinito positivo y hacia infinito negativo. En algunas realizaciones, el circuito de redondeo y saturación 420 incluye un registro de control de redondeo accesible por software (no mostrado) para especificar la regla de redondeo que se aplicará.

En algunas realizaciones, cada una de las unidades de hardware de FMA en el arreglo de multiplicadores 416 realiza el redondeo por sí misma. En algunas realizaciones, cada una de las unidades de hardware de FMA del arreglo de multiplicadores 416 verifica la saturación y realiza la saturación por sí misma.

BLOC DE NOTAS 422

Como se muestra, el circuito de ejecución 411 incluye un bloc de notas 422 para almacenar resultados de ejecución intermedios. En algunas realizaciones, como en este caso, el bloc de notas 422 es una memoria de 32 kB, dividida en ocho (8) bancos, cada uno de ellos conectado uno a uno con la fila correspondiente del arreglo de multiplicadores y el banco del acumulador. En algunas realizaciones, todos los bancos del bloc de notas 422 se comunican con los bancos correspondientes del arreglo de acumuladores 418 en paralelo, generando una conexión de gran ancho de banda. El uso del formato CSC para la multiplicación, mediante el cual solo los elementos distintos de cero de la matriz de origen dispersa especificada se almacenan en memorias intermedias dispersas 414 y se suministran al arreglo de multiplicadores 416, evita la necesidad de usar un costoso circuito de dispersión de recopilación para recopilar y suministrar datos distintos de cero al arreglo de multiplicadores 416 y al arreglo de acumuladores 418.

En algunas realizaciones, como en este caso, el circuito de ejecución 411 utiliza uno o más carriles de procesamiento de instrucción única y múltiples datos (SIMD), por ejemplo, 8 carriles, para realizar una misma operación en múltiples elementos de datos al mismo tiempo. En algunas realizaciones, un carril de procesamiento de SIMD tiene un ancho de carril de 32 elementos y se usan 8 carriles de procesamiento de SIMD para realizar 256 operaciones en 256 elementos de datos.

En algunas realizaciones, dos o más carriles de procesamiento de SIMD operan simultáneamente y en paralelo. El número de carriles en un procesador de SIMD, así como el número de elementos asignados a cada carril, puede variar, sin limitación. Según algunas realizaciones, un carril de procesamiento de SIMD se implementa con un ancho de carril de 8 elementos, 16 elementos o 32 elementos, con anchos de elementos de 8 bits, 16 bits, 32 bits o 64 bits, sin limitación.

En algunas realizaciones, el circuito de ejecución 411 realiza la instrucción de SDMM 402 realizando operaciones de multiplicación-acumulación usando unidades de hardware de multiplicación-suma fusionada (FMA) para generar los productos de cada elemento distinto de cero de la matriz de origen dispersa especificada y cada uno de los elementos en una fila correspondiente de la matriz de origen densa especificada, y acumular productos con los valores anteriores de los elementos correspondientes de la matriz de salida densa especificada por el campo de matriz de salida densa 406 de la instrucción de SDMM 402.

Tal como se usa en el presente documento, el término "correspondiente" tiene una interpretación diferente en función de su contexto. En el contexto de generar los productos, los elementos correspondientes de la matriz de origen densa especificada por el campo de matriz de origen densa 410 que corresponden a cada elemento distinto de cero (m, k) de la matriz de origen dispersa especificada por el campo de matriz de origen dispersa 408 de la instrucción de SDMM 402 son los elementos correspondientes, (k, n), en una fila correspondiente, k, de la matriz de origen densa especificada por el campo de matriz de origen densa 410 de la instrucción de SDMM 402. En el contexto de acumular los productos con contenidos previos de la matriz de salida densa especificada por el campo de matriz de salida densa 406 de la instrucción de SDMM 402. Los elementos correspondientes de la matriz de salida densa especificada son aquellos en las ubicaciones (m, n).

En consecuencia, el circuito de ejecución 411, al ejecutar una instrucción de SDMM 402 que especifica una matriz de origen dispersa 408 que tiene una dispersión de 0,125, mejora 8 veces el rendimiento del procesador en el que está incorporado, en comparación con el uso de un circuito de multiplicación de matrices simétrico.

La circuitería de ejecución para ejecutar la instrucción de SDMM, según las realizaciones divulgadas, se ilustran y analizan con más detalle al menos con respecto a las **Figuras 3-6,** y las **Figuras 11-12.**

La **Figura 5** es un pseudocódigo que ilustra el funcionamiento de la circuitería de ejecución para procesar instrucciones que demandan una multiplicación de matrices, en este caso, una instrucción de red neuronal virtual (VNNI). Con fines ilustrativos, los pseudocódigos 502 y 504 ilustran la implementación de una instrucción de multiplicación de matrices VNNI en operandos de origen simétricos. Como se describe en el presente documento, las letras mayúsculas, M, N y K, se usan para hacer referencia a las dimensiones máximas de las matrices, mientras que las letras minúsculas, m, n y k, se usan para referirse a los índices de las posiciones del elemento dentro de las matrices. Tanto el pseudocódigo 502 como el pseudocódigo 504 ilustran la ejecución de una instrucción de VNNI en una matriz de origen A de M filas por K columnas (M x K) y en una matriz de origen B de K filas por N columnas (K x N) para generar resultados de una matriz de salida C de M filas por N columnas (M x N). Los pseudocódigos 506 y 508, por otro lado, ilustran la ejecución

de la instrucción de SDMM según las realizaciones divulgadas, en donde solo se procesan los elementos distintos de cero de la matriz de origen dispersa A, aumentando así el rendimiento del procesador en proporción a la dispersión de la matriz de origen A.

La circuitería de ejecución para ejecutar la instrucción de SDMM según las realizaciones divulgadas se ilustra y analiza al menos con respecto a las **Figuras 3-6,** y las **Figuras 11-12.**

La **Figura 6** es un diagrama de flujo de un proceso que ilustra la ejecución de una instrucción de multiplicación de matrices densas-dispersas (SDMM) mediante un procesador, según algunas realizaciones. En 602, el procesador extrae, del almacenamiento de código usando circuitería de extracción, la instrucción de multiplicación de matriz densa-dispersa que tiene campos para especificar un código de operación, una matriz de salida densa, una matriz de origen densa y una matriz de origen dispersa que tiene una dispersión de elementos distintos de cero, siendo la dispersión inferior a uno. La instrucción de SDMM extraída en 602 puede denominarse instrucción de SDMM asimétrica, en la medida en que un origen es una matriz dispersa y el otro origen es una matriz densa. En 604, el procesador decodifica, mediante una circuitería de decodificación, la instrucción de SDMM extraída. En 606, el procesador opcionalmente programa la ejecución de la instrucción de SDMM decodificada mediante un circuito de ejecución de SIMD. La operación 606 es opcional, como lo indica su borde discontinuo, en la medida en que la programación de la ejecución de la instrucción decodificada puede producirse en un momento diferente o no producirse en absoluto. En 608, el procesador ejecuta, mediante una circuitería de ejecución, la instrucción de SDMM decodificada, para cada elemento distinto de cero en la fila M y columna K de la matriz de origen dispersa, para generar un producto del elemento distinto de cero y cada elemento denso correspondiente en la fila K y columna N de la matriz de origen densa; y acumula cada producto generado con un valor anterior de un elemento de salida correspondiente en la fila M y columna N de la matriz de salida densa. En 610, el procesador opcionalmente confirma o retira la instrucción de SDMM ejecutada. La operación 610 es opcional, como lo indica su borde discontinuo, en la medida en que puede producirse en un momento diferente o no producirse en absoluto.

La circuitería de ejecución para ejecutar la instrucción de SDMM según las realizaciones divulgadas se ilustra y analiza al menos con respecto a las **Figuras 3-6,** y las **Figuras 11-12.**

La **Figura 7** es un formato ilustrativo de una instrucción de multiplicación de matrices densas dispersas (SDMM), según algunas realizaciones. Como se muestra, la instrucción de SDMM 700 incluye un código de operación 702 (SDMMVNNI*) y campos para especificar una matriz de salida densa 704, una matriz de origen densa 706 y una matriz de origen dispersa 708. La instrucción de SDMM 700 incluye además campos opcionales para especificar el tamaño del elemento 710 y las dimensiones M, N y K, 712, 714 y 716. En algunas realizaciones, una o más de las dimensiones 712, 714 y 716 opcionales se especifican mediante un registro específico de modelo (MSR) programable por software, que puede ser un MSR predeterminado. Como se muestra en la **Figura 7,** el tamaño de elemento 710 opcional puede estar especificado como parte del código de operación y las dimensiones M, N y K, 712, 714 y 716 pueden estar especificadas por un MSR, como parte del código de operación, o una combinación de los mismos. Se muestra que el código de operación 702 incluye un asterisco para indicar que, opcionalmente, puede incluir prefijos o sufijos adicionales para especificar comportamientos de instrucción. Por ejemplo, el código de operación 702 puede incluir un sufijo, tal como "B", "W", "D" o "Q", para especificar un tamaño de elemento de ocho, dieciséis, treinta y dos y sesenta y cuatro bits, respectivamente. Si la instrucción de SDMM 700 no especifica ninguno de los parámetros opcionales, se usan valores predeterminados. El formato de la instrucción de SDMM se ilustra y describe con más detalle más adelante con respecto a la **Figura 8A**, la **Figura 8B** y las **Figuras 9A-D**.

PREPARACIÓN ANTICIPADA DE LA MATRIZ DE ORIGEN DISPERSA COMPRIMIDA

Con referencia de nuevo a la **Figura 2B,** la matriz de origen dispersa comprimida (CSR o CSC) especificada en algunas realizaciones se prepara con antelación. Los formatos CSC y CSR se ilustran y describen con más detalle, al menos con referencia a la **Figura 4.**

En algunos ejemplos que no caen dentro del alcance de la presente invención, la instrucción de SDMM 700 se usa para hacer que el procesador prepare una matriz de origen dispersa en formato comprimido (ya sea, CSR o CSC, como se ilustra y describe con más detalle con respecto a la **Figura 4).** El código de operación 702 puede incluir un prefijo o un sufijo, tal como "PREP", para indicar al procesador que prepare una matriz de origen dispersa comprimida. El campo de la matriz de salida densa 704 especifica una dirección de memoria en la que almacenar la matriz de origen dispersa comprimida, y la matriz de origen dispersa 708 especifica una ubicación de memoria en la que se almacena un conjunto de datos dispersos, comprendiendo el conjunto de datos dispersos un bloque grande de memoria que tiene una dispersión de elementos válidos, estando la validez determinada por los propios valores de datos (por ejemplo, no válido siendo nulo, cero o por debajo de un valor umbral), o por un campo de control adjunto e indicando la validez de cada elemento. El tamaño del conjunto de datos dispersos es por el campo de matriz de origen densa 706.

En un ejemplo, la instrucción de SDMM 700 incluye un código de operación 702 que tiene un sufijo "PREP", especifica una matriz de salida densa 704 donde almacenar la matriz de origen dispersa condensada, usa el campo de la matriz de origen dispersa 708 para especificar dónde se encuentra un conjunto de datos dispersos que comprende decenas,

cientos, miles, millones o miles de millones de elementos de datos, con elementos no válidos que tienen valores nulos, y usa el campo de matriz de origen densa 706 para especificar un tamaño del conjunto de datos dispersos. En respuesta, el procesador carga los elementos de datos desde la ubicación de la matriz de origen dispersa 708 especificada, usa el campo de la matriz de origen densa 706 para determinar el tamaño del conjunto de datos dispersos, determina si cada elemento de datos es válido y escribe los elementos válidos, en formato comprimido (CSR o CSC) en la ubicación de la matriz de salida densa 704 especificada, que es una ubicación alineada con la línea de memoria caché. Por tanto, el procesador empaqueta los elementos válidos del conjunto de datos dispersos en formato comprimido (CSR o CSC) en la matriz de salida especificada, que puede servir como una matriz de origen dispersa para una instrucción de SDMM posterior.

En otro ejemplo, la instrucción de SDMM 700 incluye un código de operación 702 que tiene un sufijo "PREP", especifica una matriz de salida densa 704 donde almacenar la matriz de origen dispersa condensada, usa el campo de la matriz de origen dispersa 708 para especificar dónde se encuentra un conjunto de datos dispersos que comprende decenas, cientos, miles, millones o miles de millones de elementos de datos, incluyendo cada elemento campos de control que incluyen al menos un bit válido, y usa el campo de matriz de origen densa 706 para especificar un tamaño (es decir, un número de elementos) del conjunto de datos dispersos. En respuesta, el procesador carga los elementos de datos desde la ubicación de la matriz de origen dispersa 708 especificada, usa el campo de la matriz de origen densa 706 para determinar el tamaño del conjunto de datos dispersos, determina si cada elemento de datos es válido y escribe los elementos válidos, en formato comprimido (CSR o CSC) en la ubicación de la matriz de salida densa 704 especificada, que en algunas realizaciones es una ubicación alineada con la línea de memoria caché. Por tanto, el procesador empaqueta los elementos válidos del conjunto de datos dispersos en formato comprimido (CSR o CSC) en la matriz de salida especificada, que puede servir como una matriz de origen dispersa para una instrucción de SDMM posterior.

Un procesador que responde a una instancia de instrucción de SDMM 700 que tiene un código de operación con un sufijo "PREP" opera en segundo plano programando de manera oportunista la ejecución de sus cargas requeridas y almacena solo cuando el nivel de actividad del procesador está por debajo de un valor umbral. En algunas realizaciones, un procesador que responde a una instancia de instrucción de SDMM 700 que tiene un código de operación con un sufijo "PREP" genera una interrupción al finalizar la compresión del conjunto de datos dispersos especificado.

CONJUNTOS DE INSTRUCCIONES

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

FORMATOS DE INSTRUCCIÓN ILUSTRATIVOS

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

Formato de instrucciones genérico compatible con vectores

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

compatible con vectores, realizaciones alternativas usan únicamente operaciones vectoriales en el formato de instrucciones compatible con vectores.

Las **Figuras 8A-8B** son diagramas de bloques que ilustran un formato de instrucciones genérico compatible con vectores y plantillas de instrucciones del mismo según algunas realizaciones. La **Figura 8A** es un diagrama de bloques que ilustra un formato de instrucciones genérico compatible con vectores y plantillas de instrucciones de clase A del mismo, según algunas realizaciones; mientras que la **Figura 8B** es un diagrama de bloques que ilustra el formato de instrucciones genérico compatible con vectores y sus plantillas de instrucciones de clase B, según algunas realizaciones. Específicamente, un formato de instrucciones compatible con vectores genérico 800 para el cual se definen plantillas de instrucciones de clase A y clase B, las cuales incluyen plantillas de instrucciones sin acceso a memoria 805 y plantillas de instrucciones de acceso a memoria 820. El término genérico en el contexto del formato de instrucciones compatible con vectores se refiere a que el formato de instrucciones no está vinculado a un conjunto de instrucciones específico.

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

Las plantillas de instrucciones de clase A en la **Figura 8A** incluyen: 1) dentro de las plantillas de instrucciones sin acceso a memoria 805 se muestra una plantilla de instrucción de operación de tipo de control de redondeo completo, sin acceso a memoria 810 y una plantilla de instrucción de operación de tipo de transformada de datos, sin acceso a memoria 815; y 2) dentro de las plantillas de instrucciones de acceso a memoria 820 se muestra una plantilla de instrucción de acceso a memoria temporal 825 y una plantilla de instrucción de acceso a memoria no temporal 830. Las plantillas de instrucciones de clase B en la **Figura 8B** incluyen: 1) dentro de las plantillas de instrucciones sin acceso a memoria 805 se muestra una plantilla de instrucción de operación de tipo de control de redondeo parcial, control de máscara de escritura, sin acceso a memoria 812 y una plantilla de instrucción de operación de tipo de control de máscara de escritura, sin acceso a memoria, vsize 817; y 2) dentro de las plantillas de instrucciones de acceso a memoria 820 se muestra una plantilla de instrucción de acceso a memoria, control de máscara de escritura 827.

El formato de instrucciones genérico compatible con vectores 800 incluye los siguientes campos enumerados a continuación en el orden ilustrado en las **Figuras 8A-8B**.

Campo de formato 840 - un valor específico (un valor de identificador de formato de instrucciones) en este campo identifica de forma única el formato de instrucción compatible con vectores y, por lo tanto, las apariciones de instrucciones en el formato de instrucción compatible con vectores en los flujos de instrucciones. Como tal, este campo es opcional en el sentido de que no es necesario para un conjunto de instrucciones que solo tiene el formato de instrucciones genérico compatible con vectores.

Campo de operación base 842 - su contenido distingue diferentes operaciones base.

Campo de índice de registro 844 - su contenido, directamente o mediante la generación de direcciones, especifica las ubicaciones de los operandos de origen y destino, ya sea en registros o en memoria. Estos incluyen suficientes bits para seleccionar N registros de un archivo de registros PxQ (por ejemplo, 32x512, 16x128, 32x1024, 64x1024). Si bien en una realización N puede ser hasta tres registros de origen y uno de destino, realizaciones alternativas pueden soportar registros de origen y de destino (por ejemplo, pueden soportar hasta dos orígenes donde uno de estos orígenes también actúa como el destino, pueden soportar hasta tres orígenes donde uno de estos orígenes también actúa como el destino, puede soportar hasta dos orígenes y un destino).

Campo de modificador 846 - su contenido distingue las apariciones de instrucciones en el formato de instrucciones genérico de vector que especifican el acceso a memoria de aquellas que no lo hacen; es decir, entre las plantillas de instrucciones sin acceso a memoria 805 y las plantillas de instrucciones de acceso a memoria 820. Las operaciones de acceso a memoria leen y/o escriben en la jerarquía de memoria (especificando, en algunos casos, las direcciones de origen y/o de destino usando valores en registros), mientras que las operaciones sin acceso a memoria no lo hacen (por ejemplo, el origen y los destinos son registros). Si bien en una realización este campo también selecciona entre tres formas diferentes de realizar cálculos de direcciones de memoria, realizaciones alternativas pueden soportar más, menos o diferentes formas de realizar cálculos de direcciones de memoria.

Campo de operación de aumento 850 - su contenido distingue cuál de una diversidad de operaciones diferentes se realizará además de la operación de base. Este campo es específico del contexto. En una realización de la invención, este campo se divide en un campo de clase 868, un campo alfa 852 y un campo beta 854. El campo de operación de aumento 850 permite que se realicen grupos comunes de operaciones en una sola instrucción en lugar de 2, 3 o 4 instrucciones.

Campo de escala 860 - 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 862A - 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 862B (obsérvese que la yuxtaposición del campo de desplazamiento 862A directamente sobre el campo de factor de desplazamiento 862B 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 orden inferior se ignoran y, por consiguiente, el contenido del campo del factor de desplazamiento se multiplica por el tamaño total de los operandos de memoria (N) para generar el desplazamiento final que se usará en el cálculo de una dirección efectiva. El valor de N está determinado por el hardware del procesador en tiempo de ejecución basándose en el campo de código de operación completo 874 (descrito más adelante en el presente documento) y el campo de manipulación de datos 854C. El campo de desplazamiento 862A y el campo de factor de desplazamiento 862B son opcionales en el sentido de que no se usan para las plantillas de instrucciones sin acceso a memoria 805 y/o diferentes realizaciones pueden implementar únicamente uno o ninguno de los dos.

Campo de ancho de elemento de datos 864 - su contenido distingue cuál de muchos anchos de elemento 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 solo se soporta un ancho de elemento de datos y/o se soportan anchos de elementos de datos usando algún aspecto de los códigos de operación.

Campo de máscara de escritura 870 - su contenido controla, en función de la posición de cada elemento de datos, si esa posición del elemento de datos en el operando del vector de destino refleja el resultado de la operación base y la operación de aumento. Las plantillas de instrucciones de clase A soportan el enmascaramiento de escritura de fusión, mientras que las plantillas de instrucciones de clase B soportan el enmascaramiento de escritura tanto de fusión como de puesta a cero. Cuando se fusionan, las máscaras vectoriales permiten proteger de actualizaciones cualquier conjunto de elementos en el destino durante la ejecución de cualquier operación (especificada por la operación de base y la operación de aumento); en otra realización, conservando el valor antiguo de cada elemento del destino donde el bit de máscara correspondiente tiene un 0. Por el contrario, cuando se ponen a cero las máscaras vectoriales se permite 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 en 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 está realizando (es decir, el lapso de elementos que se están modificando, desde el primero hasta el último); sin embargo, no es necesario que los elementos que se modifican sean consecutivos. Por tanto, el campo de máscara de escritura 870 permite operaciones vectoriales parciales, incluyendo cargas, almacenamientos, aritméticas, lógicas, etc. Si bien se describen algunas realizaciones de la invención en las que el contenido del campo de máscara de escritura 870 selecciona uno de muchos registros de máscara de escritura que contiene la máscara de escritura que se va a usar (y, por tanto, el contenido del campo de máscara de escritura 870 identifica indirectamente ese enmascaramiento que se va a realizar), unas realizaciones alternativas en su lugar o adicionales permiten que el contenido del campo de escritura de máscara 870 especifique directamente el enmascaramiento a realizar.

Campo inmediato 872 - su contenido permite la especificación de un inmediato. Este campo es opcional en el sentido de que no está presente en una implementación del formato genérico compatible con vectores que no soporta inmediato y no está presente en instrucciones que no usan un inmediato.

Campo de clase 868 - su contenido distingue entre diferentes clases de instrucciones. Con referencia a las **Figuras 8A-B**, el contenido de este campo selecciona entre instrucciones de clase A y clase B. En las **Figuras 8A-B**, se usan cuadrados de esquinas redondeadas para indicar que un valor específico está presente en un campo (por ejemplo, clase A 868A y clase B 868B para el campo de clase 868 respectivamente en las **Figuras 8A-B**),

PLANTILLAS DE INSTRUCCIONES DE CLASE A

En el caso de las plantillas de instrucciones sin acceso a memoria de clase A 805, el campo alfa 852 se interpreta como un campo de RS 852A, cuyo contenido distingue cuál de los diferentes tipos de operación de aumento va a realizarse (por ejemplo, el redondeo 852A.1 y la transformada de datos 852A.2 se especifican respectivamente para las plantillas de instrucciones de operación de tipo de redondeo sin acceso a memoria 810 y operación de tipo de

transformada de datos sin acceso a memoria 815), mientras que el campo beta 854 distingue cuál de las operaciones del tipo especificado va a realizarse. En las plantillas de instrucciones sin acceso a memoria 805, no están presentes el campo de escala 860, el campo de desplazamiento 862A ni el campo de escala de desplazamiento 862B.

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

En la plantilla de instrucción de operación de tipo de control de redondeo completo sin acceso a memoria 810, el campo beta 854 se interpreta como un campo de control de redondeo 854A, cuyo contenido o contenidos proporcionan redondeo estático. Si bien en las realizaciones descritas de la invención, el campo de control de redondeo 854A incluye un campo de supresión 856 de todas las excepciones de coma flotante (SAE) y un campo de control de operación de redondeo 858, las realizaciones alternativas pueden soportar la codificación de ambos conceptos en el mismo campo o solo tener uno o el otro de estos conceptos/campos (por ejemplo, pueden tener solo el campo de control de operación de redondeo 858).

Campo de SAE 856 - su contenido distingue si se debe o no inhabilitar el informe de eventos de excepción; cuando el contenido del campo de SAE 856 indica que la supresión está habilitada, una instrucción dada no informa de ningún tipo de bandera de excepción de coma flotante y no genera ningún manejador de excepción de coma flotante.

Campo de control de operación de redondeo 858 - 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 al más cercano). Por tanto, el campo de control de operación de redondeo 858 permite el cambio del modo de redondeo por instrucción. En una realización de la invención donde un procesador incluye un registro de control para especificar modos de redondeo, el contenido del campo de control de operación de redondeo 850 anula ese valor de registro.

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

En la plantilla de instrucciones de operación de tipo de transformada de datos sin acceso a memoria 815, el campo beta 854 se interpreta como un campo de transformada de datos 854B, cuyo contenido distingue cuál de un número de transformadas de datos se va a realizar (por ejemplo, sin transformada de datos, mezcla, difusión).

En el caso de una plantilla de instrucciones de acceso a memoria 820 de clase A, el campo alfa 852 se interpreta como un campo de sugerencia de desalojo 852B, cuyo contenido distingue cuál de las sugerencias de desalojo se va a usar (en la **Figura 8A**, temporal 852B.1 y no temporal 852B.2 se especifican respectivamente para la plantilla de instrucciones de acceso a memoria temporal 825 y la plantilla de instrucciones de acceso a memoria no temporal 830), mientras que el campo beta 854 se interpreta como un campo de manipulación de datos 854C, cuyo contenido distingue cuál de un número de operaciones de manipulación de datos (también conocidas como primitivas) se va a realizar (por ejemplo, sin manipulación; difusión; conversión ascendente de un origen y conversión descendente de un destino). Las plantillas de instrucciones de acceso a memoria 820 incluyen el campo de escala 860 y, opcionalmente, el campo de desplazamiento 862A o el campo de escala de desplazamiento 862B.

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

PLANTILLAS DE INSTRUCCIONES DE ACCESO A MEMORIA - TEMPORALES

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

PLANTILLAS DE INSTRUCCIONES DE ACCESO A MEMORIA - NO TEMPORALES

Los datos no temporales son datos que es poco probable que se reutilicen lo suficientemente pronto como para beneficiarse del almacenamiento en caché en la 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 ignorando la sugerencia por completo.

PLANTILLAS DE INSTRUCCIONES DE CLASE B

En el caso de las plantillas de instrucciones de clase B, el campo alfa 852 se interpreta como un campo de control de máscara de escritura (Z) 852C, cuyo contenido distingue si el enmascaramiento de escritura controlado por el campo de máscara de escritura 870 debe ser una fusión o una puesta a cero.

En el caso de las plantillas de instrucciones sin acceso a memoria 805 de clase B, parte del campo beta 854 se interpreta como un campo RL 857A, cuyo contenido distingue cuál de los diferentes tipos de operación de aumento se va a realizar (por ejemplo, redondeo 857A.1) y la longitud del vector (VSIZE) 857A.2 se especifica respectivamente para la plantilla de instrucciones de la operación de tipo de control de redondeo parcial de control de máscara de escritura sin acceso a memoria 812 y la plantilla de instrucciones de la operación de tipo VSIZE de control de máscara de escritura sin acceso a memoria 817), mientras que el resto del campo beta 854 distingue cuál de las operaciones del tipo especificado se va a realizar. En las plantillas de instrucciones sin acceso a memoria 805, no están presentes el campo de escala 860, el campo de desplazamiento 862A ni el campo de escala de desplazamiento 862B.

En la plantilla de instrucciones de operación sin acceso a memoria, de control de máscara de escritura, tipo de control de redondeo parcial 810, el resto del campo beta 854 se interpreta como un campo de operación de redondeo 859A y el informe de eventos de excepción está inhabilitado (una instrucción dada no informa ninguna clase de bandera de excepción de coma flotante y no genera ningún manejador de excepción de coma flotante).

Campo de control de operación de redondeo 859A - al igual que el campo de control de operación de redondeo 858, 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 859A permite el cambio del modo de redondeo por instrucción. En una realización de la invención donde un procesador incluye un registro de control para especificar modos de redondeo, el contenido del campo de control de operación de redondeo 850 anula ese valor de registro.

En la plantilla de instrucciones de operación de tipo VSIZE 817, control de máscara de escritura, sin acceso a memoria, el resto del campo beta 854 se interpreta como un campo de longitud de vector 859B, cuyo contenido distingue cuál de varias longitudes de vector de datos se va a realizar (por ejemplo, 128, 256 o 512 bytes).

En el caso de una plantilla de instrucciones de acceso a memoria 820 de clase B, parte del campo beta 854 se interpreta como un campo de difusión 857B, cuyo contenido distingue si se va a realizar la operación de manipulación de datos de tipo difusión, mientras que el resto del campo beta 854 se interpreta como el campo de longitud de vector 859B. Las plantillas de instrucciones de acceso a memoria 820 incluyen el campo de escala 860 y, opcionalmente, el campo de desplazamiento 862A o el campo de escala de desplazamiento 862B.

Acerca del formato de instrucciones genérico compatible con vectores 800, se muestra un campo de código de operación completo 874 que incluye el campo de formato 840, el campo de operación de base 842 y el campo de ancho de elemento de datos 864. Si bien se muestra una realización donde el campo de código de operación completo 874 incluye estos campos, el campo de código de operación completo 874 incluye menos que estos campos en realizaciones que no los soportan todos. El campo de código de operación completo 874 proporciona el código de operación (opcode).

El campo de operación de aumento 850, el campo de anchura de elemento de datos 864 y el campo de máscara de escritura 870 permiten que estas características se especifiquen en una base por instrucción en el formato de instrucciones genérico compatible con vectores.

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

Las diversas plantillas de instrucciones que se encuentran dentro de la clase A y la clase B son beneficiosas en diferentes situaciones. En algunas realizaciones, diferentes procesadores o diferentes núcleos dentro de un procesador pueden soportar solo la clase A, solo la clase B o ambas clases. Por ejemplo, un núcleo fuera de orden de propósito general de alto rendimiento destinado a computación de propósito general puede soportar solo la clase B, un núcleo destinado principalmente a computación gráfica y/o científica (rendimiento) puede soportar solo la clase A, y un núcleo destinado a ambos puede soportar ambas (por supuesto, un núcleo que tiene alguna combinación de plantillas e instrucciones de ambas clases pero no todas las plantillas e instrucciones de ambas clases están dentro del alcance de la invención). Además, un solo procesador puede incluir varios núcleos, todos los cuales soportan la misma clase o en los que diferentes núcleos soportan clases diferentes. Por ejemplo, en un procesador con gráficos separados y núcleos de propósito general, uno de los núcleos de gráficos destinado principalmente a gráficos y/o computación científica puede soportar solo la clase A, mientras que uno o más de los núcleos de propósito general pueden ser núcleos de propósito general de alto rendimiento con ejecución fuera de orden y cambio de nombre de registro destinado a computación de propósito general que soportan solo clase B. Otro procesador que no tiene un núcleo de gráficos separado puede incluir uno más núcleos de propósito general en o fuera de orden que soportan tanto clase A como clase B. Por supuesto, las características de una clase también pueden implementarse en la otra clase en diferentes realizaciones. Los programas escritos en un lenguaje de alto nivel se colocarían (por ejemplo, compilados justo a tiempo o compilados estáticamente) en una variedad de formas ejecutables diferentes, incluyendo: 1) una forma que tiene solo instrucciones de la o las clases soportadas por el procesador de destino 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 soportadas por el procesador que actualmente está ejecutando el código.

FORMATO DE INSTRUCCIONES COMPATIBLE CON VECTORES ESPECÍFICO ILUSTRATIVO

La **Figura 9A** es un diagrama de bloques que ilustra un formato de instrucciones compatible con vectores específico ilustrativo, según algunas realizaciones. La **Figura 9A** muestra un formato de instrucciones compatible con vectores específico 900 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 instrucciones compatible con vectores específico 900 se puede usar para ampliar el conjunto de instrucciones x86 y, por lo tanto, algunos de los campos son similares o iguales a los usados en el conjunto de instrucciones x86 existente y su extensión (por ejemplo, AVX). Este formato sigue siendo coherente con el campo de codificación de prefijo, el campo de bytes de código de operación real, el campo MOD R/M, el campo SIB, el campo de desplazamiento y los campos inmediatos del conjunto de instrucciones x86 existente con extensiones. Se ilustran los campos de la **Figura 8** en los que se mapean los campos de la **Figura 9A**.

Se debe entender que, aunque algunas realizaciones se describen con referencia al formato de instrucciones compatible con vectores específico 900 en el contexto del formato de instrucciones genérico compatible con vectores 800 con fines ilustrativos, la invención no se limita al formato de instrucciones compatible con vectores específico 900 excepto donde se reivindique. Por ejemplo, el formato de instrucciones genérico compatible con vectores 800 contempla una variedad de tamaños posibles para los diversos campos, mientras que el formato de instrucciones compatible con vectores específico 900 se muestra como si tuviera campos de tamaños específicos. A modo de ejemplo específico, mientras que el campo de anchura de elemento de datos 864 se ilustra como un campo de un bit en el formato de instrucciones compatible con vectores específico 900, la invención no está limitada a esto (es decir, el formato de instrucciones genérico compatible con vectores 800 contempla otros tamaños del campo de anchura de elemento de datos 864).

El formato de instrucciones genérico compatible con vectores 800 incluye los siguientes campos enumerados a continuación en el orden ilustrado en la **Figura 9A.**

Prefijo EVEX (Bytes 0-3) 902 - está codificado en formato de cuatro bytes.

Campo de formato 840 (Byte 0 de EVEX, bits [7:0]) - el primer byte (Byte 0 de EVEX) es el campo de formato 840 y contiene 0x62 (el valor único usado para distinguir el formato de instrucciones compatible con vectores en una realización de la invención).

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

Campo REX 905 (Byte de EVEX 1, bits [7-5]) - consta de 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 857BEX byte 1, bit[5] - B). Los campos de bits EVEX.R, EVEX.X y EVEX.B proporcionan la misma funcionalidad que los campos de bits VEX correspondientes y se codifican en forma de complemento a 1, es decir, ZMM0 se codifica como 1111B, ZMM15 se codifica como 0000B. Otros campos de las instrucciones codifican los tres bits inferiores de los índices de registro como es conocido en la técnica (rrr, xxx y bbb), de modo que Rrrr, Xxxx y Bbbb pueden formarse sumando EVEX.R, EVEX.X, y EVEX.B.

Campo REX' 810 - esta es la primera parte del campo REX' 810 y es el campo de bit EVEX.R' (byte EVEX 1, bit [4] - R') que se usa para codificar los 16 superiores o los 16 inferiores del conjunto extendido de 32 registros. En una realización de la invención, este bit, junto con otros, como se indica más adelante, se almacena en formato de bits invertidos para distinguir (en el bien conocido modo x86 de 32 bits) de la instrucción BOUND, cuyo byte de código de operación real es 62, pero no acepta en el campo MOD R/M (descrito más adelante) el valor de 11 en el campo MOD; algunas realizaciones alternativas de la invención no almacenan este y los otros bits indicados más adelante en el formato invertido. Se usa un valor de 1 para codificar los 16 registros inferiores. En otras palabras, R'Rrrr se forma combinando EVEX.R', EVEX.R y el otro RRR de otros campos.

Campo de mapa de opcode 915 (byte EVEX 1, bits [3:0] - mmmm) - su contenido codifica un byte de código de operación principal implícito (0F, 0F 38 o 0F 3).

El campo de ancho de elemento de datos 864 (byte EVEX 2, bit [7] - W) está representado por 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 920 (EVEX Byte 2, bits [6:3]-vvvvv)- la función de EVEX.vvvv puede incluir lo siguiente: 1) EVEX.vvvv codifica el primer operando de registro de origen, especificado en forma invertida (complemento a 1) y es válido para instrucciones con 2 o más operandos de origen; 2) EVEX.vvvv codifica el operando de registro de destino, especificado en forma de complemento a 1 para ciertos desplazamientos de vector; o 3) EVEX.vvvv no codifica ningún operando, el campo está reservado y debe contener 1111b. Por lo tanto, el campo EVEX.vvvv 920 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 diferente adicional para ampliar el tamaño del especificador a 32 registros.

EVEX.U 868 Campo de clase (byte EVEX 2, bit [2]-U) - Si EVEX.U = 0, indica clase A o EVEX.U0; si EVEX.U = 1, indica clase B o EVEX.U1.

Campo de codificación de prefijo 925 (byte 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 el beneficio de compactar el prefijo SIMD (en lugar de requerir un byte para expresar el prefijo SIMD, el prefijo EVEX requiere solo 2 bits). En una realización, para soportar instrucciones SSE heredadas que usan un prefijo de SIMD (66H, F2H, F3H) tanto en el formato heredado como en el formato de prefijo EVEX, estos prefijos de SIMD heredados se codifican en el campo de codificación de prefijo de SIMD; y en el tiempo de ejecución se expanden en el prefijo de SIMD heredado antes de proporcionarse a la PLA del decodificador (para que la PLA pueda ejecutar tanto el formato heredado como el EVEX de estas instrucciones heredadas sin modificaciones). Aunque las instrucciones más nuevas podrían usar el contenido del campo de codificación del prefijo EVEX directamente como una extensión de código de operación, ciertas realizaciones se expanden de manera similar para lograr coherencia, pero permiten que estos prefijos SIMD heredados especifiquen significados diferentes. Una realización alternativa puede rediseñar la PLA para soportar las codificaciones de prefijo de SIMD de 2 bits y, por lo tanto, no requerir la expansión.

Campo alfa 852 (byte EVEX 3, bit [7] - EH; también conocido como EVEX.EH, EVEX.rs, EVEX.RL, EVEX.write mask control y EVEX.N; también ilustrado con α) - como se describió anteriormente, este campo es específico del contexto.

Campo beta 854 (byte 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.

Campo REX' 810 - este es el resto del campo REX' y es el campo de bit EVEX.V' (byte EVEX 3, bit [3] - V') que puede usarse para codificar los 16 superiores o los 16 inferiores del conjunto extendido de 32 registros. Este bit se almacena en formato de bits invertido. 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 870 (byte EVEX 3, bits [2:0]-kkk) - su contenido especifica el índice de un registro en los registros de máscara de escritura como se describió anteriormente. En una realización de la invención, el valor específico EVEX.kkk=000 tiene un comportamiento especial que implica que no se usa una máscara de escritura para la instrucción particular (esto se puede implementar de una diversidad de formas, incluyendo el uso de una máscara de escritura programada a todos o al hardware que sortea el hardware de enmascaramiento).

El campo código de operación real 930 (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 940 (byte 5) incluye el campo MOD 942, el campo Reg 944 y el campo R/M 946. Como se describió anteriormente, el contenido del campo MOD 942 distingue entre operaciones de acceso a memoria y operaciones sin acceso a memoria. La función del campo Reg 944 se puede resumir en dos situaciones: codificar el operando del registro de destino o un operando del registro de origen, o tratarse como una extensión de código de operación y no usarse para codificar un operando de instrucción. La función del campo R/M 946 puede incluir lo siguiente: codificar el operando de instrucción que hace referencia a una dirección de memoria, o codificar el operando del registro de destino o un operando del registro de origen.

Byte de escala, índice, base (SIB) (Byte 6) - como se describió anteriormente, el contenido el campo de escala 850 se usa para la generación de direcciones de memoria. SIB.xxx 954 y SIB.bbb 956 - el contenido de estos campos ha sido mencionado anteriormente con respecto a los índices de registro Xxxx y Bbbb.

Campo de desplazamiento 862A (Bytes 7-10) - cuando el campo MOD 942 contiene 10, los bytes 7-10 son el campo de desplazamiento 862A, y funciona igual que el desplazamiento de 32 bits heredado (disp32) y funciona con granularidad de byte.

Campo de factor de desplazamiento 862B (Byte 7) - cuando el campo MOD 942 contiene 01, el byte 7 es el campo de factor de desplazamiento 862B. 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. Como disp8 es de signo extendido, solo puede abordar desplazamientos entre -128 y 127 bytes; en términos de líneas de caché de 64 bytes, disp8 usa 8 bits que pueden configurarse solo en cuatro valores realmente útiles -128, - 64, 0 y 64; como a menudo se necesita un rango mayor, se usa disp32; sin embargo, disp32 requiere 4 bytes. A diferencia de disp8 y disp32, el campo de factor de desplazamiento 862B es una reinterpretación de disp8; cuando se usa el campo de factor de desplazamiento 862B, 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 promedio de instrucción (se usa un solo byte para el desplazamiento, pero con un rango mucho

mayor). Tal desplazamiento comprimido se basa en la suposición de que el desplazamiento efectivo es un múltiplo de la granularidad del acceso a memoria y, por lo tanto, no es necesario codificar los bits de bajo orden redundantes del desplazamiento de dirección. En otras palabras, el campo de factor de desplazamiento 862B sustituye el desplazamiento de 8 bits del conjunto de instrucciones x86 heredado. Por tanto, el campo de factor de desplazamiento 862B 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 ni en las longitudes de codificación, sino solo en la interpretación del valor de desplazamiento por parte del hardware (que necesita escalar el desplazamiento por el tamaño del operando de memoria para obtener un desplazamiento de dirección byte por byte). El campo inmediato 872 funciona como se ha descrito anteriormente.

CAMPO DE CÓDIGO DE OPERACIÓN COMPLETO

La **Figura 9B** es un diagrama de bloques que ilustra los campos del formato de instrucciones compatible con vectores específico 900 que componen el campo de código de operación completo 874 según una realización de la invención. Específicamente, el campo de código de operación completo 874 incluye el campo de formato 840, el campo de operación base 842 y el campo de anchura de elemento de datos (W) 864. El campo de operación base 842 incluye el campo de codificación de prefijo 925, el campo de mapa de código de operación 915 y el campo de código de operación real 930.

CAMPO DE ÍNDICE DE REGISTRO

La **Figura 9C** es un diagrama de bloques que ilustra los campos del formato de instrucciones compatible con vectores específico 900 que componen el campo de índice de registro 844 según una realización de la invención. Específicamente, el campo de índice de registro 844 incluye el campo REX 905, el campo REX' 910, el campo MODR/M.reg 944, el campo MODR/M.r/m 946, el campo VVVV 920, el campo xxx 954 y el campo bbb 956.

CAMPO DE OPERACIÓN DE AUMENTO

La **Figura 9D** es un diagrama de bloques que ilustra los campos del formato de instrucciones compatible con vectores específico que constituye el campo de operación de aumento 850 según una realización de la invención. Cuando el campo de clase (U) 868 contiene 0, significa EVEX.U0 (clase A 868A); cuando contiene 1, significa EVEX.U1 (clase B 868B). Cuando U=0 y el campo MOD 942 contiene 11 (lo que significa una operación sin acceso a memoria), el campo alfa 852 (byte de EVEX 3, bit [7] - EH) se interpreta como el campo de rs 852A. Cuando el campo de rs 852A contiene un 1 (redondeo 852A.1), el campo beta 854 (byte de EVEX 3, bits [6:4]-SSS) se interpreta como el campo de control de redondeo 854A. El campo de control de redondeo 854A incluye un campo de SAE de un bit 856 y un campo de operación de redondeo de dos bits 858. Cuando el campo rs 852A contiene un 0 (transformada de datos 852A.2), el campo beta 854 (byte de EVEX 3, bits [6:4]-SSS) se interpreta como un campo de transformada de datos de tres bits 854B. Cuando U=0 y el campo MOD 942 contiene 00, 01 o 10 (lo que significa una operación de acceso a memoria), el campo alfa 852 (EVEX byte 3, bit [7] - EH) se interpreta como el campo de sugerencia de expulsión (EH) 852B y el campo beta 854 (EVEX byte 3, bits [6:4]-SSS) se interpreta como un campo de manipulación de datos de tres bits 854C.

Cuando U=1, el campo alfa 852 (byte de EVEX 3, bit [7] - EH) se interpreta como el campo de control de máscara de escritura (Z) 852C. Cuando U=1 y el campo MOD 942 contiene 11 (lo que significa una operación sin acceso a memoria), parte del campo beta 854 (byte de EVEX 3, bit [4] - S_0) se **interpreta** como el campo de RL 857A; cuando contiene un 1 (redondeo 857A.1) el resto del campo beta 854 (byte de EVEX 3, bit [6-5] - S_{2-1}) se interpreta como el campo de operación de redondeo 859A, mientras que cuando el campo de RL 857A contiene un 0 (VSIZE 857.A2) el resto del campo beta 854 (byte de EVEX 3, bit [6-5] - S_{2-1}) se interpreta como el campo de longitud de vector 859B (byte de EVEX 3, bit [6-5] - L_{1-0}). Cuando U=1 y el campo MOD 942 contiene 00, 01 o 10 (lo que significa una operación de acceso a memoria), el campo beta 854 (byte de EVEX 3, bits [6:4] - SSS) se interpreta como el campo de longitud de vector 859B (byte de EVEX 3, bit [6-5] - L_{1-0}) y el campo de difusión 857B (byte de EVEX 3, bit [4] - B).

Arquitectura de registro ilustrativa

La **Figura 10** es un diagrama de bloques de una arquitectura de registro 1000 según una realización de la invención. En la realización ilustrada, hay 32 registros vectoriales 1010 que tienen 512 bits de ancho; estos registros se referencian como zmm0 a zmm31. Los 256 bits de orden inferior de los 16 registros zmm inferiores se superponen a 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 a los registros xmm0-15. El formato de instrucciones compatible con vectores específico 900 opera en estos archivos de registros superpuestos como se ilustra en las siguientes tablas.

Longitud del vector ajustable	Clase	Operaciones	Registros
Plantillas de instrucciones que no incluyen el campo de longitud de vector 859B	A (Figura 8A; U=0)	810, 815, 825, 830	registros zmm (la longitud de vector es de 64 bytes)
	B (Figura 8B; U=1)	812	registros zmm (la longitud de vector es de 64 bytes)
Plantillas de instrucciones que incluyen el campo de longitud de vector 859B	B (Figura 8B; U=1)	817, 827	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 859B

En otras palabras, el campo de longitud de vector 859B selecciona entre una longitud máxima y una o más longitudes más cortas, donde cada una de tales longitudes más cortas es la mitad de la longitud de la longitud precedente; y las plantillas de instrucciones sin el campo de longitud de vector 859B operan en la longitud de vector máxima. Además, en una realización, las plantillas de instrucciones de clase B del formato de instrucciones compatible con vectores específico 900 operan en datos de coma flotante de precisión sencilla/doble escalar o empaquetados y datos de número entero escalar o empaquetados. Las operaciones escalares son operaciones realizadas en la posición del elemento de datos de orden más bajo en un registro zmm/ymm/xmm; las posiciones de los elementos de datos de orden superior se dejan igual que antes de la instrucción o se ponen a cero dependiendo de la realización.

Registros de máscara de escritura 1015 - 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 1015 tienen un tamaño de 16 bits. Como se describió anteriormente, en una realización de la invención, el registro de máscara vectorial k0 no se puede usar como máscara de escritura; cuando la codificación que normalmente indicaría k0 se usa para una máscara de escritura, selecciona una máscara de escritura cableada de 0xFFFF, inhabilitando efectivamente la máscara de escritura para esa instrucción.

Registros de propósito general 1025 - en la realización ilustrada, hay dieciséis registros de propósito general de 64 bits que se usan junto con los modos de direccionamiento x86 existentes para direccionar operandos de memoria. A estos registros se hace referencia con los nombres RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP y R8 a R15.

Archivo de registro de pila de coma flotante escalar (pila x87) 1045, en el que se asigna un alias al archivo de registro plano de número entero empaquetado MMX 1050, en la realización ilustrada, la pila x87 es una pila de ocho elementos usada para realizar operaciones escalares de coma flotante en datos de coma flotante de 32/64/80 bits usando la extensión del conjunto de instrucciones x87; mientras que los registros MMX se usan para realizar operaciones con datos de números enteros empaquetados de 64 bits, así como para contener operandos para algunas operaciones realizadas entre los registros MMX y XMM.

Realizaciones alternativas pueden usar registros más anchos o más estrechos. Además, las realizaciones alternativas pueden usar más, menos o diferentes archivos de registro y 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 en orden de propósito general destinado a computación de propósito general; 2) un núcleo fuera de orden de propósito general de alto rendimiento destinado a computación de propósito general; 3) un núcleo de propósito especial destinado principalmente a gráficos y/o computación científica (rendimiento). Las implementaciones de diferentes procesadores pueden incluir: 1) una CPU que incluye uno o más núcleos en orden de propósito general destinados a computación de propósito general y/o uno o más núcleos fuera de orden de propósito general destinados a computación de propósito general; y 2) un coprocesador que incluye uno o más núcleos de propósito especial destinados principalmente a gráficos y/o científica (de rendimiento). Estos diferentes procesadores 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 matriz separada en el mismo paquete que una CPU; 3) el coprocesador en la misma matriz que una CPU (en cuyo caso, dicho coprocesador a veces se denomina lógica de propósito especial, como gráficos integrados y/o lógica científica (de rendimiento), o como núcleos de propósito especial); y 4) un sistema en un chip que puede incluir en la misma matriz la CPU descrita (a veces 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 DEL NÚCLEO EN ORDEN Y FUERA DE ORDEN

La **Figura 11A** es un diagrama de bloques que ilustra tanto una canalización ilustrativa en orden como una canalización ilustrativa de emisión/ejecución fuera de orden y de cambio de nombre de registro según algunas realizaciones. La **Figura 11B** 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 fuera de orden y cambio de nombre de registro que se incluirá en un procesador según algunas realizaciones. Los recuadros con líneas continuas de las **Figuras 11A-B** ilustran la canalización en orden y el núcleo en orden, mientras que la adición opcional de los recuadros con líneas discontinuas ilustra la canalización y núcleo 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 11A,** una canalización de procesador 1100 incluye una etapa de extracción 1102, una etapa de decodificación de longitud 1104, una etapa de decodificación 1106, una etapa de asignación 1108, una etapa de cambio de nombre 1110, una etapa de planificación (también conocida como despacho o emisión) 1112, una etapa de lectura de registro/lectura de memoria 1114, una etapa de ejecución 1116, una etapa de reescritura/escritura de memoria 1118, una etapa de manejo de excepciones 1122 y una etapa de confirmación 1124.

La **Figura 11B** muestra el núcleo de procesador 1190 que incluye una unidad de sección de entrada 1130 acoplada a una unidad de motor de ejecución 1150, y ambas están acopladas a una unidad de memoria 1170. El núcleo 1190 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 1190 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 1130 incluye una unidad de predicción de ramificación 1132 acoplada a una unidad de memoria caché de instrucciones 1134, que está acoplada a una memoria intermedia de traducción anticipada (TLB) de instrucciones 1136, que está acoplada a una unidad de extracción de instrucciones 1138, que está acoplada a una unidad de decodificación 1140. La unidad de decodificación 1140 (o decodificador) puede decodificar instrucciones y generar como salida una o más microoperaciones, puntos de entrada de microcódigo, microinstrucciones, otras instrucciones u otras señales de control, que se decodifican a partir de, o que de otro modo reflejan o se derivan de, las instrucciones originales. La unidad de decodificación 1140 puede implementarse 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 1190 incluye una ROM de microcódigo u otro medio que almacena microcódigo para determinadas macroinstrucciones (por ejemplo, en la unidad de decodificación 1140 o, de otro modo, dentro de la unidad de sección de entrada 1130). La unidad de decodificación 1140 está acoplada a una unidad de cambio de nombre/asignación 1152 en la unidad de motor de ejecución 1150.

La unidad de motor de ejecución 1150 incluye la unidad de cambio de nombre/asignación 1152 acoplada a una unidad de retiro 1154 y un conjunto de una o más unidades de planificador 1156. La o las unidades de planificador 1156 representan cualquier número de planificadores diferentes, incluidas estaciones de reserva, ventana de instrucción central, etc. La o las unidades de planificador 1156 están acopladas a la o las unidades de archivos de registro físico 1158. Cada una de las unidades de archivo o archivos de registro físico 1158 representa uno o más archivos de registro físico, diferentes de los que almacenan uno o más tipos de datos diferentes, tales como entero escalar, coma flotante escalar, entero empaquetado, coma flotante empaquetado, entero vectorial, coma flotante vectorial, estado (por ejemplo, un puntero de instrucción que es la dirección de la siguiente instrucción a ejecutar), etc. En una realización, la unidad de archivo o archivos de registros físicos 1158 comprende una unidad de registros vectoriales, una unidad de registros de máscara y una unidad de registros escalares. Estas unidades de registro pueden proporcionar registros vectoriales arquitectónicos, registros de máscara vectorial y registros de propósito general. La(s) unidad(es) de archivo(s) de registros físicos 1158 se superponen con la unidad de retiro 1154 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 1154 y la unidad o unidades de archivo o archivos de registro físico 1158 se acoplan a la agrupación o agrupaciones de ejecución 1160. La o las agrupaciones de ejecución 1160 incluyen un conjunto de una o más unidades de ejecución 1162 y un conjunto de una o más unidades de acceso a memoria 1164. Las unidades de ejecución 1162 pueden realizar diversas operaciones (por ejemplo, desplazamientos, suma, resta, multiplicación) y sobre diversos tipos de datos (por ejemplo, coma flotante escalar, número entero empaquetado, coma flotante empaquetada, número entero vectorial, coma flotante vectorial). Si bien algunas realizaciones pueden incluir un número de unidades de ejecución especializadas a funciones específicas o conjuntos de funciones, otras realizaciones pueden incluir únicamente una unidad de ejecución o múltiples unidades de ejecución que realizan todas las funciones. La unidad o unidades de planificador 1156, la unidad o unidades de archivo o archivos de registro físico 1158 y la agrupación o agrupación de ejecución 1160 se muestran como posiblemente varios porque ciertas realizaciones crean canalizaciones separadas para ciertos tipos de datos/operaciones (por ejemplo, una canalización de número entero escalar, una canalización de coma flotante escalar/número entero empaquetado/coma flotante empaquetado/número entero vectorial/coma flotante vectorial y/o una canalización de acceso a memoria, cada una de las cuales tiene su propia unidad de planificador, unidad de

archivo o archivos de registro físico y/o agrupación de ejecución - y en el caso de una canalización de acceso a memoria separada, se implementan ciertas realizaciones en las que únicamente la agrupación de ejecución de esta canalización tiene la unidad o unidades de acceso a memoria 1164). También se debe entender que, cuando se usan canalizaciones separadas, una o más de estas canalizaciones pueden ser de emisión/ejecución fuera de orden y el resto en orden.

El conjunto de unidades de acceso a memoria 1164 está acoplado a la unidad de memoria 1170, que incluye una unidad TLB de datos 1172 acoplada a una unidad de caché de datos 1174 acoplada a una unidad de caché de nivel 2 (L2) 1176. En una realización ilustrativa, las unidades de acceso a memoria 1164 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 TLB de datos 1172 en la unidad de memoria 1170. La unidad de caché de instrucciones 1134 está acoplada además a una unidad de caché de nivel 2 (L2) 1176 en la unidad de memoria 1170. La unidad de caché L2 1176 está acoplada a uno o más niveles de caché y eventualmente a una memoria principal.

A modo de ejemplo, la arquitectura de núcleo de ejecución/emisión fuera de orden y cambio de nombre de registro ilustrativo puede implementar la canalización 1100 de la siguiente manera: 1) la extracción de instrucciones 1138 realiza las etapas de extracción y decodificación de longitud 1102 y 1104; 2) la unidad de decodificación 1140 realiza la etapa de decodificación 1106; 3) la unidad de cambio de nombre/asignación 1152 realiza la etapa de asignación 1108 y la etapa de cambio de nombre 1110; 4) la o las unidades de planificador 1156 realizan la etapa de planificador 1112; 5) la o las unidades de archivos de registro físico 1158 y la unidad de memoria 1170 realizan la etapa de lectura de registro/lectura de memoria 1114; la agrupación de ejecución 1160 realiza la etapa de ejecución 1116; 6) la unidad de memoria 1170 y la o las unidades de archivos de registro físico 1158 realizan la etapa de reescritura/escritura de memoria 1118; 7) varias unidades pueden estar involucradas en la etapa de manejo de excepciones 1122; y 8) la unidad de retiro 1154 y la o las unidades de archivos de registro físico 1158 realizan la etapa de confirmación 1124.

El núcleo 1190 puede soportar uno o más conjuntos de instrucciones (por ejemplo, el conjunto de instrucciones x86 (con algunas extensiones que se han agregado con versiones más nuevas); el conjunto de instrucciones MIPS de MIPS Technologies de Sunnyvale, CA; el conjunto de instrucciones ARM (con extensiones adicionales opcionales como NEON) de ARM Holdings de Sunnyvale, CA), incluidas las instrucciones descritas en el presente documento. En una realización, el núcleo 1190 incluye lógica para soportar una extensión de conjunto de instrucciones de datos empaquetados (por ejemplo, AVX1, AVX2), permitiendo así que las operaciones usadas por muchas aplicaciones multimedia se realicen usando datos empaquetados.

Debe entenderse que el núcleo puede soportar multiprocesamiento (ejecución de dos o más conjuntos paralelos de operaciones o subprocesos) y puede hacerlo de diversas maneras, incluyendo multiprocesamiento en porciones de tiempo, multiprocesamiento simultáneo (donde un solo núcleo físico proporciona un núcleo lógico para cada uno de los subprocesos que ese núcleo físico está multiprocesando simultáneamente) o una combinación de estos (por ejemplo, obtención y decodificación en porciones de tiempo y multiprocesamiento simultáneo a partir de entonces, tal como en la tecnología Intel® Hyperthreading).

Si bien el cambio de nombre de registro se describe en el contexto de la ejecución fuera de orden, debe entenderse que el cambio de nombre de registro puede usarse en una arquitectura en orden. Si bien la realización ilustrada del procesador también incluye unidades de caché de instrucciones y datos 1134/1174 separadas y una unidad de caché L2 1176 compartida, realizaciones alternativas pueden tener una única caché interna tanto para instrucciones como para datos, como, por ejemplo, una caché interna de Nivel 1 (L1), o múltiples niveles de caché interna. En algunas realizaciones, el sistema puede incluir una combinación de una caché interna y una caché externa que es externa al núcleo y/o al procesador. Alternativamente, toda la caché puede ser externa al núcleo y/o al procesador.

ARQUITECTURA DE NÚCLEO EN ORDEN ILUSTRATIVA ESPECÍFICA

Las **Figuras 12A-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. Los bloques lógicos se comunican a través de una red de interconexión de gran ancho de banda (por ejemplo, una red de anillo) con cierta lógica de función fija, interfaces de E/S de memoria y otra lógica de E/S necesaria, de acuerdo con la aplicación.

La circuitería de ejecución para ejecutar la instrucción de SDMM según las realizaciones divulgadas se ilustra y analiza al menos con respecto a las **Figuras 3-6,** y las **Figuras 11-12.**

La **Figura 12A** es un diagrama de bloques de un único núcleo de procesador, junto con su conexión a la red de interconexión integrada 1202 y con su subconjunto local de la caché de Nivel 2 (L2) 1204, según algunas realizaciones. En una realización, un decodificador de instrucciones 1200 soporta el conjunto de instrucciones x86 con una extensión de conjunto de instrucciones de datos empaquetados. Una caché L1 1206 permite accesos de baja latencia a la memoria caché en las unidades escalares y vectoriales. Si bien en una realización (para simplificar el diseño), una unidad escalar 1208 y una unidad vectorial 1210 usan conjuntos de registros separados (respectivamente, registros escalares 1212 y registros vectoriales 1214) y los datos transferidos entre ellos se escriben en la memoria y a

continuación se vuelven a leer desde una caché de nivel 1 (L1) 1206, realizaciones alternativas pueden usar un enfoque diferente (por ejemplo, usar un único conjunto de registros o incluir una ruta de comunicaciones que permita que los datos se transfieran entre los dos archivos de registro sin tener que escribirse ni volverse a leer).

El subconjunto local de la caché L2 1204 es parte de una caché L2 global que se divide en subconjuntos locales separados, uno por núcleo de procesador. Cada núcleo de procesador tiene una ruta de acceso directo a su propio subconjunto local de la caché L2 1204. Los datos leídos por un núcleo de procesador se almacenan en su subconjunto de caché L2 1204 y se puede acceder a ellos rápidamente, en paralelo con otros núcleos de procesador que acceden a sus propios subconjuntos de caché L2 locales. Los datos escritos por un núcleo de procesador se almacenan en su propio subconjunto de caché L2 1204 y se restablece de otros subconjuntos, si es necesario. La red de anillo garantiza la coherencia de los datos compartidos. La red de anillo es bidireccional para permitir que agentes, tales como núcleos de procesador, cachés L2 y otros bloques lógicos, se comuniquen entre sí dentro del chip. Cada ruta de datos del anillo tiene 1012 bits de ancho por dirección.

La **Figura 12B** es una vista ampliada de parte del núcleo de procesador de la **Figura 12A** según algunas realizaciones. La **Figura 12B** incluye una caché de datos L1 1206A parte de la caché L1 1204, así como más detalles con respecto a la unidad vectorial 1210 y los registros vectoriales 1214. Específicamente, la unidad vectorial 1210 es una unidad de procesamiento vectorial (VPU) de ancho 16 (véase la UAL 1228 de ancho 16), que ejecuta una o más de instrucciones de números enteros, flotantes de precisión simple y flotantes de precisión doble. La VPU soporta la conversión de entradas de registro con la unidad de conversión 1220, la conversión numérica con las unidades de conversión numérica 1222A-B y la replicación con la unidad de replicación 1224 en la entrada de memoria. Los registros de máscara de escritura 1226 permiten predecir escrituras vectoriales resultantes.

La circuitería de ejecución para ejecutar la instrucción de SDMM según las realizaciones divulgadas se ilustra y analiza al menos con respecto a las **Figuras 3-6,** y las **Figuras 11-12.**

La **Figura 13** es un diagrama de bloques de un procesador 1300 que puede tener más de un núcleo, puede tener un controlador de memoria integrado y puede tener gráficos integrados, según algunas realizaciones. Los recuadros con líneas continuas en la **Figura 13** ilustran un procesador 1300 con un único núcleo 1302A, un agente de sistema 1310, un conjunto de una o más unidades de controlador de bus 1316, mientras que, la adición opcional de los recuadros con líneas discontinuas ilustra un procesador alternativo 1300 con múltiples núcleos 1302A-N, un conjunto de una o más unidad o unidades de controlador de memoria integrada 1314 en la unidad de agente de sistema 1310 y lógica de propósito especial 1308.

Por tanto, diferentes implementaciones del procesador 1300 pueden incluir: 1) una CPU con la lógica de propósito especial 1308 que es una lógica (que puede incluir uno o más núcleos) de gráficos y/o temas científicos integrados (rendimiento), y siendo los núcleos 1302A-N uno o más núcleos de propósito general (por ejemplo, núcleos en orden de propósito general, núcleos fuera de orden de propósito general, una combinación de los dos); 2) un coprocesador con núcleos 1302A-N que son un gran número de núcleos de propósito especial destinados principalmente a gráficos y/o científica (rendimiento); y 3) un coprocesador con los núcleos 1302A-N que son un gran número de núcleos en orden de propósito general. Por tanto, el procesador 1300 puede ser un procesador de propósito general, coprocesador o procesador de propósito especial, tal como, por ejemplo, un procesador de red o comunicación, motor de compresión, procesador de gráficos, GPGPU (unidad de procesamiento de gráficos de propósito general), un coprocesador de alto rendimiento de muchos núcleos integrados (MIC) (que incluye 30 o más núcleos), procesador integrado o similar. El procesador puede implementarse en uno o más chips. El procesador 1300 puede ser parte y/o puede implementarse en uno o más sustratos usando cualquiera de muchas tecnologías de proceso, tales como, por ejemplo, BiCMOS, CMOS o NMOS.

La jerarquía de memoria incluye uno o más niveles de caché dentro de los núcleos, un conjunto o una o más unidades de caché compartidas 1306 y memoria externa (no mostrada) acoplada al conjunto de unidades de controlador de memoria integrada 1314. El conjunto de unidades de caché compartida 1306 puede incluir una o más cachés de nivel medio, tales como nivel 2 (L2), nivel 3 (L3), nivel 4 (L4), u otros niveles de caché, una caché de último nivel (LLC), y/o combinaciones de las mismas. Si bien en una realización una unidad de interconexión basada en anillo 1312 interconecta la lógica de gráficos integrada 1308 (la lógica de gráficos integrada 1308 es un ejemplo y también se denomina en el presente documento lógica de propósito especial), el conjunto de unidades de caché compartida 1306 y la unidad de agente del sistema 1310/unidad o unidades de controlador de memoria integrada 1314, realizaciones alternativas pueden usar cualquier número de técnicas bien conocidas para interconectar dichas unidades. En una realización, se mantiene la coherencia entre una o más unidades de caché 1306 y núcleos 1302-A-N.

En algunas realizaciones, uno o más de los núcleos 1302A-N son capaces de realizar multiprocesamiento. El agente de sistema 1310 incluye aquellos componentes que coordinan y operan los núcleos 1302A-N. La unidad de agente de sistema 1310 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 1302A-N y la lógica de gráficos integrados 1308. La unidad de visualización es para controlar una o más pantallas conectadas externamente.

Los núcleos 1302A-N pueden ser homogéneos o heterogéneos en términos de conjunto de instrucciones de arquitectura; es decir, dos o más de los núcleos 1302A-N pueden ser capaces de ejecutar el mismo conjunto de instrucciones, mientras que otros pueden ejecutar solo un subconjunto de ese conjunto de instrucciones o un conjunto de instrucciones diferente.

ARQUITECTURAS INFORMÁTICAS ILUSTRATIVAS

Las **Figuras 14-17** son diagramas de bloques de arquitecturas informáticas ilustrativas. Otros diseños y configuraciones de sistemas conocidos en las técnicas para computadoras portátiles, de escritorio, PC de mano, asistentes digitales personales, estaciones de trabajo de ingeniería, servidores, dispositivos de red, concentradores de red, conmutadores, procesadores integrados, procesadores de señales digitales (DSP), dispositivos gráficos, dispositivos de videojuegos, decodificadores, microcontroladores, teléfonos móviles, reproductores multimedia portátiles, dispositivos de mano y varios otros dispositivos electrónicos también son adecuados. En general, son generalmente adecuados una gran variedad de sistemas o dispositivos electrónicos capaces de incorporar un procesador y/u otra lógica de ejecución como la descrita en el presente documento.

Haciendo referencia ahora a la **Figura 14**, se muestra un diagrama de bloques de un sistema 1400 según una realización de la presente invención. El sistema 1400 puede incluir uno o más procesadores 1410, 1415, que están acoplados a un concentrador controlador 1420. En una realización, el concentrador controlador 1420 incluye un concentrador controlador de memoria gráfica (GMCH) 1490 y un concentrador de entrada/salida (IOH) 1450 (que pueden estar en chips separados); el GMCH 1490 incluye controladores de memoria y gráficos a los que están acoplados la memoria 1440 y un coprocesador 1445; el IOH 1450 acopla dispositivos de entrada/salida (E/S) 1460 al GMCH 1490. Como alternativa, uno o ambos controladores de memoria y gráficos están integrados dentro del procesador (como se describe en el presente documento), la memoria 1440 y el coprocesador 1445 están acoplados directamente al procesador 1410 y al concentrador controlador 1420 en un único chip con el IOH 1450.

La naturaleza opcional de los procesadores adicionales 1415 se indica en la **Figura 14** con líneas discontinuas. Cada procesador 1410, 1415 puede incluir uno o más de los núcleos de procesamiento descritos en el presente documento y puede ser alguna versión del procesador 1300.

La memoria 1440 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 controlador 1420 se comunica con el o los procesadores 1410, 1415 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 1495.

En una realización, el coprocesador 1445 es un procesador de propósito especial, tal como, por ejemplo, un procesador MIC de alto rendimiento, un procesador de red o comunicación, motor de compresión, procesador de gráficos, GPGPU, procesador integrado o similar. En una realización, el concentrador controlador 1420 puede incluir un acelerador de gráficos integrado.

Puede haber varias diferencias entre los recursos físicos 1410, 1415 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 1410 ejecuta instrucciones que controlan operaciones de procesamiento de datos de un tipo general. Dentro de las instrucciones pueden estar incorporadas instrucciones de coprocesador. El procesador 1410 reconoce estas instrucciones de coprocesador como de un tipo que debería ser ejecutado por el coprocesador adjunto 1445. En consecuencia, el procesador 1410 emite estas instrucciones de coprocesador (o señales de control que representan instrucciones de coprocesador) en un bus de coprocesador u otra interconexión, al coprocesador 1445. El o los coprocesadores 1445 aceptan y ejecutan las instrucciones de coprocesador recibidas.

Con referencia ahora a la **Figura 15**, se muestra un diagrama de bloques de un primer sistema 1500 ilustrativo más específico según una realización de la presente invención. Como se muestra en la **Figura 15**, el sistema multiprocesador 1500 es un sistema de interconexión punto a punto e incluye un primer procesador 1570 y un segundo procesador 1580 acoplados a través de una interconexión punto a punto 1550. Cada uno de los procesadores 1570 y 1580 puede ser alguna versión del procesador 1300. En una realización de la invención, los procesadores 1570 y 1580 son respectivamente los procesadores 1410 y 1415, mientras que el coprocesador 1538 es el coprocesador 1445. En otra realización, los procesadores 1570 y 1580 son respectivamente el procesador 1410 y el coprocesador 1445.

Se muestran los procesadores 1570 y 1580 incluyendo las unidades de controlador de memoria integrada (IMC) 1572 y 1582, respectivamente. El procesador 1570 también incluye como parte de sus unidades de controlador de bus las interfaces punto a punto (P-P) 1576 y 1578; de manera similar, el segundo procesador 1580 incluye las interfaces P-P 1586 y 1588. Los procesadores 1570, 1580 pueden intercambiar información a través de una interfaz punto a punto (P-P) 1550 usando circuitos de interfaz P-P 1578, 1588. Como se muestra en la **Figura 15**, los IMC 1572 y 1582 acoplan los procesadores a las respectivas memorias, en concreto, una memoria 1532 y una memoria 1534, que pueden ser porciones de la memoria principal conectadas localmente a los respectivos procesadores.

Cada uno de los procesadores 1570, 1580 puede intercambiar información con un conjunto de chips 1590 a través de interfaces P-P 1552, 1554 individuales usando circuitos de interfaz punto a punto 1576, 1594, 1586, 1598. El conjunto de chips 1590 puede intercambiar opcionalmente información con el coprocesador 1538 a través de una interfaz de alto rendimiento 1592. En una realización, el coprocesador 1538 es un procesador de propósito especial, tal como, por ejemplo, un procesador MIC de alto rendimiento, un procesador de red o comunicación, motor de compresión, procesador de gráficos, GPGPU, procesador integrado o similar.

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

El conjunto de chips 1590 se puede acoplar a un primer bus 1516 a través de una interfaz 1596. En una realización, el primer bus 1516 puede ser un bus de interconexión de componentes periféricos (PCI), o un bus tal como un bus PCI Express u otro bus de interconexión de E/S de tercera generación, aunque el alcance de la presente invención no está limitado de esa manera.

Como se muestra en la **Figura 15**, varios dispositivos de E/S 1514 pueden estar acoplados al primer bus 1516, junto con un puente de bus 1518 que acopla el primer bus 1516 a un segundo bus 1520. En una realización, uno o más procesadores adicionales 1515, tales como coprocesadores, procesadores MIC de alto rendimiento, GPGPU, aceleradores (tales como, por ejemplo, aceleradores de gráficos o unidades de procesamiento de señales digitales (DSP)), conjuntos de puertas programación en campo o cualquier otro procesador, están acoplados al primer bus 1516. En una realización, el segundo bus 1520 puede ser un bus de recuento bajo de pines (LPC). Se pueden acoplar diversos dispositivos a un segundo bus 1520 que incluye, por ejemplo, un teclado y/o ratón 1522, dispositivos de comunicaciones 1527 y una unidad de almacenamiento 1528 tal como una unidad de disco u otro dispositivo de almacenamiento masivo que puede incluir instrucciones/códigos y datos 1530, en una realización. Además, se puede acoplar una E/S de audio 1524 al segundo bus 1520. Obsérvese que son posibles otras arquitecturas. Por ejemplo, en lugar de la arquitectura punto a punto de la **Figura 15,** un sistema puede implementar un bus multipunto u otra arquitectura similar.

Con referencia ahora a la **Figura 16**, se muestra un diagrama de bloques de un segundo sistema 1600 ilustrativo más específico según una realización de la presente invención. Elementos similares en las **Figuras 15 y 16** llevan números de referencia similares y determinados aspectos de la **Figura 15** se han omitido de la **Figura 16** para evitar complicar otros aspectos de la **Figura 16**.

La **Figura 16** ilustra que los procesadores 1570, 1580 pueden incluir memoria integrada y lógica de control de E/S ("CL") 1572 y 1582, respectivamente. Así, las CL 1572, 1582 incluyen unidades de controlador de memoria integradas e incluyen lógica de control de E/S. La **Figura 16** ilustra que no solo las memorias 1532, 1534 están acopladas a la CL 1572, 1582, sino que también los dispositivos de E/S 1614 también están acoplados a la lógica de control 1572, 1582. Los dispositivos de E/S heredados 1615 están acoplados al conjunto de chips 1590.

Con referencia ahora a la **Figura 17**, se muestra un diagrama de bloques de un SoC 1700 según una realización de la presente invención. Los elementos similares en la **Figura 13** llevan números de referencia iguales. Además, los recuadros con línea discontinua son características opcionales en los SoC más avanzados. En la **Figura 17**, una unidad o unidades de interconexión 1702 están acopladas a: un procesador de aplicaciones 1710 que incluye un conjunto de uno o más núcleos 1302A-N, que incluyen unidades de caché 1304A-N y una unidad o unidades de caché compartida 1306; una unidad de agente de sistema 1310; una unidad o unidades de controlador de bus 1316; una unidad o unidades de controlador de memoria integrada 1314; un conjunto o uno o más coprocesadores 1720 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) 1730; una unidad de acceso directo a memoria (DMA) 1732; y una unidad de visualización 1740 para acoplarse a una o más pantallas externas. En una realización, el o los coprocesadores 1720 incluyen un procesador de propósito especial, tal como, por ejemplo, un procesador de red o de comunicaciones, un motor de compresión, GPGPU, un procesador MIC de alto rendimiento, un procesador integrado o similar.

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

El código de programa, tal como el código 1530 ilustrado en la **Figura 15**, 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 manera conocida. Para los fines de esta solicitud,

un sistema de procesamiento incluye cualquier sistema que tenga un procesador, tal como, por ejemplo; un procesador de señal digital (DSP), un microcontrolador, un circuito integrado de aplicación específica (ASIC) o un microprocesador.

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

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

Dichos medios de almacenamiento legibles por máquina pueden incluir, sin limitación, disposiciones tangibles no transitorias de artículos fabricados o formados por una máquina o dispositivo, incluidos medios de almacenamiento tales como discos duros, cualquier otro tipo de disco, incluidos disquetes, discos ópticos, memorias de solo lectura de discos compactos (CD-ROM), discos compactos regrabables (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 dinámica (DRAM), memorias de acceso aleatorio estática (SRAM), memorias de solo lectura programables y borrables (EPROM), memorias flash, memorias de solo lectura programables y borrables eléctricamente (EEPROM), memoria de cambio de fase (PCM), tarjetas magnéticas u ópticas o cualquier otro tipo de medio adecuado para almacenar instrucciones electrónicas.

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

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

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

La **Figura 18** 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 según algunas realizaciones. En la realización ilustrada, el convertidor de instrucciones es un convertidor de instrucciones de software, aunque alternativamente el convertidor de instrucciones puede implementarse en software, firmware, hardware o varias combinaciones de los mismos. La **Figura 18** muestra que un programa en un lenguaje de alto nivel 1802 puede compilarse usando un compilador x86 1804 para generar código binario x86 1806 que puede ejecutarse de forma nativa mediante un procesador con al menos un núcleo de conjunto de instrucciones x86 1816. El procesador con al menos un núcleo de conjunto de instrucciones x86 1816 representa cualquier procesador que puede realizar sustancialmente las mismas funciones que un procesador Intel con al menos un núcleo de conjunto de instrucciones x86 ejecutando o procesando de otro modo (1) una porción sustancial del conjunto de instrucciones del núcleo de conjunto de instrucciones Intel x86 o (2) versiones de código objeto de aplicaciones u otro software que tiene como objetivo ejecutarse en un procesador Intel con al menos un núcleo de conjunto de instrucciones x86, para lograr sustancialmente el mismo resultado que un procesador Intel con al menos un núcleo de conjunto de instrucciones x86. El compilador x86 1804 representa un compilador que puede funcionar para generar código binario x86 1806 (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 1816. De manera similar, la **Figura 18** muestra que el programa en el lenguaje de alto nivel 1802 puede compilarse usando un compilador de conjunto de instrucciones alternativo 1808 para generar un código binario de conjunto de instrucciones alternativo 1810 que puede ejecutarse de forma nativa por un procesador sin al menos un núcleo de conjunto de instrucciones x86 1814 (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 1812 se usa para convertir el código binario x86 1806 en un código que el procesador puede ejecutar de forma nativa sin un núcleo de conjunto de instrucciones x86 1814. No es probable que este código convertido sea el mismo que el código binario de conjunto de instrucciones alternativo 1810 porque es difícil crear un convertidor de instrucciones capaz de hacer esto; sin embargo, el código convertido realizará la

operación general y estará compuesto por instrucciones de conjunto de instrucciones alternativo. Así, el convertidor de instrucciones 1812 representa un software, firmware, hardware o una combinación de los mismos que, mediante emulación, simulación o cualquier otro proceso, permite a un procesador u otro dispositivo electrónico que no disponga de un procesador o núcleo de conjunto de instrucciones x86 para ejecutar el código binario x86 1806.

REIVINDICACIONES

1. Un procesador para ejecutar una instrucción de multiplicación de matrices densas-dispersas, SDMM (103, 202, 700), que comprende:

circuitería de extracción (105) para extraer, del almacenamiento de código (101), la instrucción SDMM (103, 202, 700) que tiene campos para especificar un código de operación (204), una matriz de salida densa (206, 216), una matriz de origen densa (210, 214), y una matriz de origen dispersa (208, 212) que tiene una dispersión de elementos distintos de cero, siendo la dispersión menor que uno, estando la matriz de origen dispersa (212) en un formato comprimido, incluyendo el formato comprimido solo elementos distintos de cero de la matriz de origen dispersa (212), estando representado cada elemento distinto de cero por un valor de datos y una ubicación de matriz, y estando especificadas las dimensiones (712, 714, 716) de la matriz de origen densa (214) y la matriz de origen dispersa (212) por la instrucción SDMM (103, 202, 700) o un registro;
circuitería de decodificación (109) para decodificar la instrucción SDMM extraída; y
circuitería de ejecución (117), en respuesta a la instrucción SDMM decodificada, para cada elemento distinto de cero en la fila M y la columna K de la matriz de origen dispersa especificada (212):
generar un producto del elemento distinto de cero y cada elemento denso correspondiente en la fila K y la columna N de la matriz de origen densa especificada (214); y
generar una suma acumulada de cada producto generado y un valor anterior de un elemento de salida correspondiente en la fila M y la columna N de la matriz de salida densa especificada (216).

2. El procesador de la reivindicación 1, en donde la dispersión es de 0,125 o menos.

3. El procesador de una cualquiera de las reivindicaciones 1-2, en donde la instrucción SDMM (103, 202, 700) especifica además un tamaño de elementos de datos de la matriz de origen dispersa especificada (212), la matriz de origen densa especificada (214) y la matriz de salida densa especificada (216), siendo el tamaño uno de 8 bits, 16 bits, 32 bits, 64 bits y 128 bits, especificándose el tamaño bien como un operando de la instrucción o como parte del código de operación.

4. El procesador de una cualquiera de las reivindicaciones 1-3, en donde la instrucción SDMM (103, 202, 700) especifica además un formato de elementos de datos de la matriz de origen dispersa especificada (212), la matriz de origen densa especificada (214), y la matriz de salida densa especificada (216), siendo el formato uno de punto fijo, coma flotante de precisión simple, coma flotante de precisión doble, coma flotante de precisión extendida y coma flotante de precisión doble extendida.

5. El procesador de una cualquiera de las reivindicaciones 1-4, en donde la circuitería de ejecución (117) genera los productos usando multiplicadores que tienen un mismo tamaño que los elementos de datos de la matriz de origen dispersa especificada (212), la matriz de origen densa especificada (214) y la matriz de salida densa especificada (216), genera las sumas acumuladas usando acumuladores que tienen el doble del tamaño de los elementos de datos de la matriz de origen dispersa especificada (212) y la matriz de origen densa especificada (214), y realiza la saturación y el redondeo en cada una de las sumas acumuladas antes de escribir la suma acumulada saturada y redondeada en la matriz de salida densa especificada (216).

6. El procesador de una cualquiera de las reivindicaciones 1-5, en donde la circuitería de ejecución (117) comprende un circuito de ejecución de instrucción única y datos múltiples, SIMD, que comprende una pluralidad de multiplicadores y una pluralidad de acumuladores para ejecutar la SDMM decodificada en una pluralidad de elementos distintos de cero de la matriz de origen dispersa especificada (212) en paralelo, comprendiendo la pluralidad de elementos distintos de cero uno de 16, 32, 64 y 128 elementos.

7. El procesador de una cualquiera de las reivindicaciones 1-6, comprende además un circuito de lectura de memoria para leer la matriz de origen dispersa especificada (212) y la matriz de origen densa especificada (214), y escribir los resultados en una memoria, en donde la matriz de origen dispersa especificada (212) se almacena en la memoria en el formato comprimido, y en donde el circuito de lectura de memoria lee la matriz de origen dispersa especificada (212) en el formato comprimido.

8. Un método para ejecutar una instrucción de multiplicación de matrices densas-dispersas, SDMM (103, 202, 700), que comprende:

extraer del almacenamiento de código, usando circuitería de extracción (105), la instrucción SDMM (103, 202, 700) que tiene campos para especificar un código de operación (204), una matriz de salida densa (206, 216), una matriz de origen densa (210, 214), y una matriz de origen dispersa (208, 212) que tiene una dispersión de elementos distintos de cero, siendo la dispersión menor que uno, estando la matriz de origen dispersa (212) en un formato comprimido, incluyendo el formato comprimido solo elementos distintos de cero de la matriz de origen dispersa (212), estando representado cada elemento distinto de cero por un valor de datos y una ubicación de matriz, y estando especificadas las dimensiones (712, 714, 716) de la matriz de origen densa (214) y la matriz de origen dispersa (212) por la instrucción SDMM (103, 202, 700) o un registro;

decodificar, mediante circuitería de decodificación (109), la instrucción SDMM extraída (103, 202, 700); y
ejecutar, mediante circuitería de ejecución (117), la instrucción SDMM decodificada, para cada elemento distinto de cero en la fila M y la columna K de la matriz de origen dispersa especificada (212):
generar un producto del elemento distinto de cero y cada elemento denso correspondiente en la fila K y la columna N de la matriz de origen densa especificada (214); y
generar una suma acumulada de cada producto generado y un valor anterior de un elemento de salida correspondiente en la fila M y la columna N de la matriz de salida densa especificada (216).

9. El método de la reivindicación 8, en donde la instrucción SDMM (103, 202, 700) especifica además un tamaño de elementos de datos de la matriz de origen dispersa especificada (212), la matriz de origen densa especificada (214), y la matriz de salida densa especificada (216), siendo el tamaño uno de 8 bits, 16 bits, 32 bits, 64 bits y 128 bits, especificándose el tamaño bien como operando de la instrucción o como parte del código de operación.

10. El método de una cualquiera de las reivindicaciones 8-9, en donde la circuitería de ejecución (117) genera los productos usando multiplicadores que tienen el mismo tamaño que los elementos de datos de la matriz de origen dispersa especificada (212), la matriz de origen densa especificada (214) y la matriz de salida densa especificada (216), genera las sumas acumuladas usando acumuladores que tienen el doble del tamaño de los elementos de datos de la matriz de origen dispersa especificada (212), la matriz de origen densa especificada (214) y la matriz de salida densa especificada (216), y realiza la saturación y el redondeo en cada una de las sumas acumuladas antes de escribir la suma acumulada saturada y redondeada en la matriz de salida densa especificada (216).

11. El método de una cualquiera de las reivindicaciones 8-10, en donde la circuitería de ejecución (117) comprende un circuito de ejecución de instrucción única y datos múltiples, SIMD, que comprende una pluralidad de multiplicadores y una pluralidad de acumuladores para ejecutar la SDMM decodificada en una pluralidad de elementos distintos de cero de la matriz de origen dispersa especificada (212) en paralelo, comprendiendo la pluralidad de elementos distintos de cero uno de 16, 32, 64 y 128 elementos.

12. El método de la reivindicación 11, que comprende además: leer la matriz de origen dispersa especificada y la matriz de origen densa especificada por medio de un circuito de lectura de memoria, y escribir los resultados en una memoria; en donde la matriz de origen dispersa especificada se almacena en una memoria en el formato comprimido, y en donde la matriz de origen dispersa especificada se lee por medio del circuito de lectura de memoria en el formato comprimido; y en donde el formato comprimido es un formato de fila dispersa comprimida, CSR, que almacena los elementos distintos de cero en orden mayor de fila, o bien un formato de columna dispersa comprimida, CSC, que almacena los elementos distintos de cero en orden mayor de columna.

13. Almacenamiento legible por máquina que incluye instrucciones legibles por máquina para, cuando se ejecuta en el procesador de cualquiera de las reivindicaciones 1 a 7, ejecutar el método de cualquiera de las reivindicaciones 8-12.

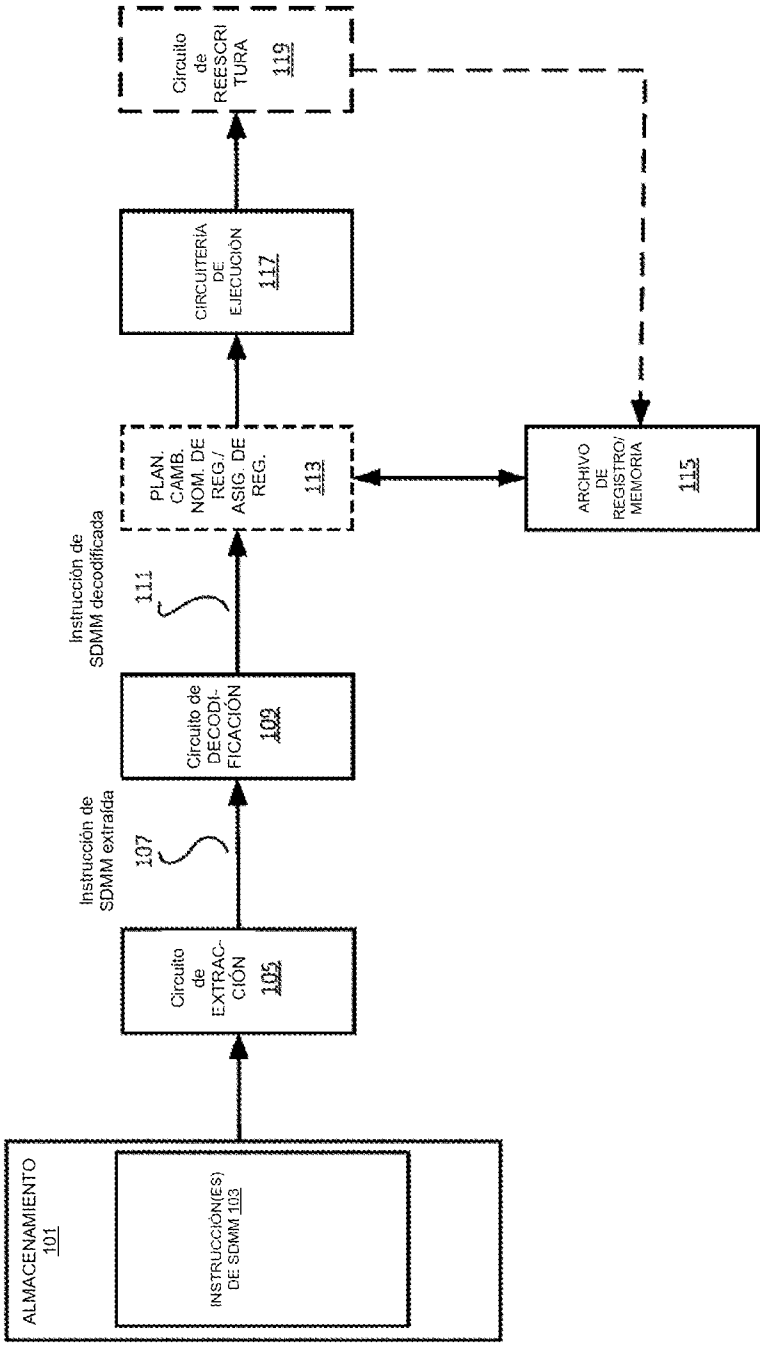

ALMACENAMIENTO
101
INSTRUCCIÓN(ES) DE SDMM 103
Circuito de EXTRAC-CIÓN 105
Instrucción de SDMM extraída
107
Circuito de DECODI-FICACIÓN 109
Instrucción de SDMM decodificada
111
PLAN. CAMB. NOM. DE REG./ ASIG. DE REG. 113
CIRCUITERÍA DE EJECUCIÓN 117
Circuito de REESCRI TURA 119
ARCHIVO DE REGISTRO/ MEMORIA 115

FIG. 1

Instrucción de multiplicación de matriz densa-dispersa 202 SDMM			
Código de operación^ SDMMVNNIW 204	**Matriz de salida densa** 206	**Matriz de origen dispersa** 208	**Matriz de origen densa** 210

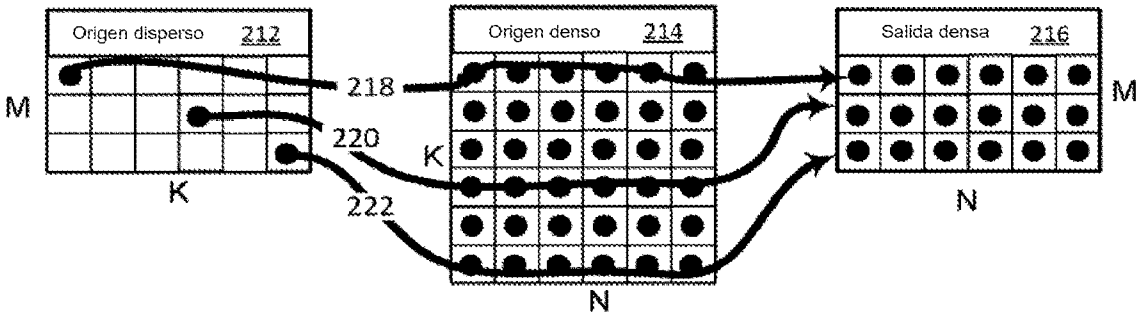


FIG. 2A

Instrucción de multiplicación de matriz densa-dispersa 252 SDMM			
Código de operación^ SDMMVNNI 254	**Matriz de salida densa** 256	**Matriz de origen dispersa** 258	**Matriz de origen densa** 260

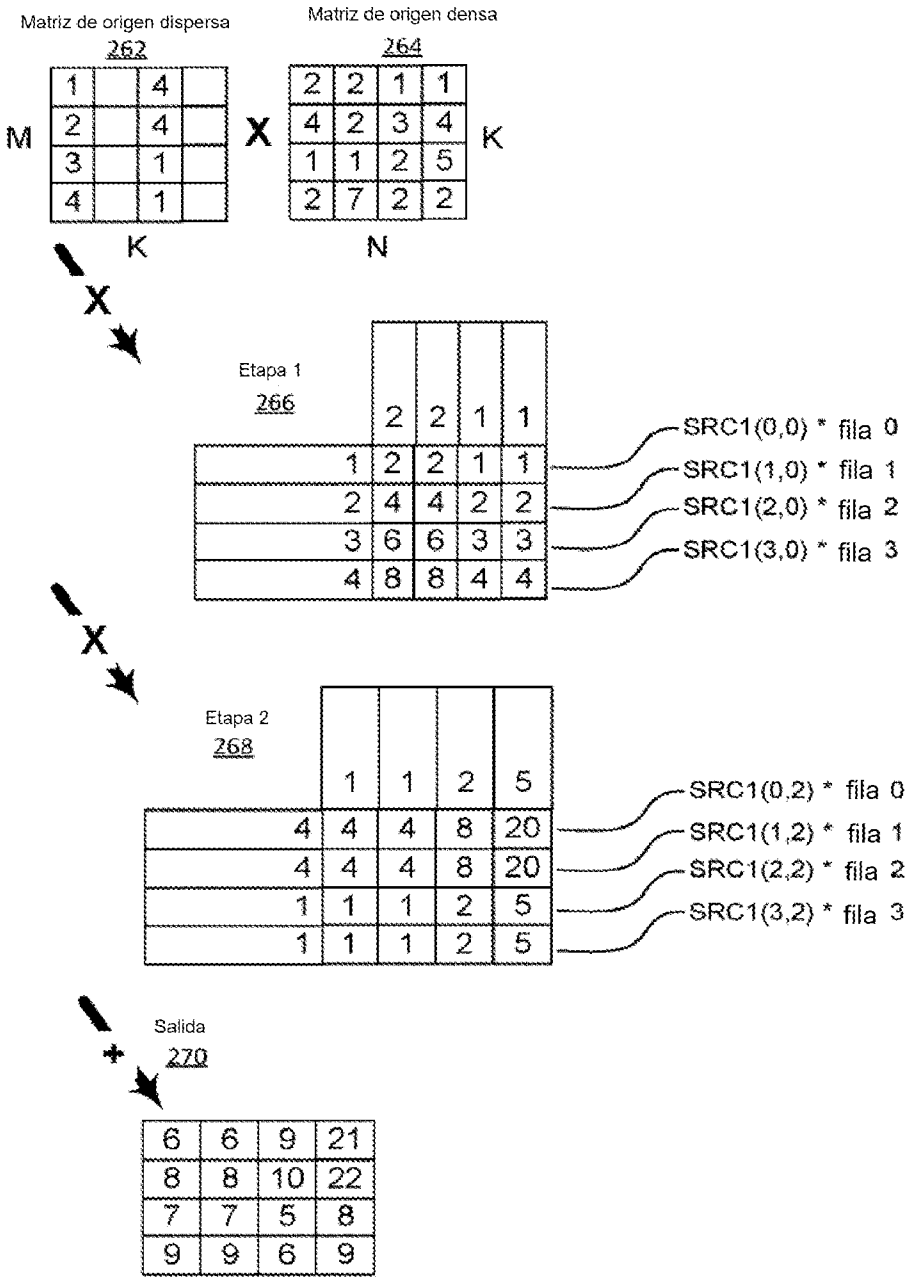


FIG. 2B

Instrucción de multiplicación de matriz densa-dispersa 302 SDMM			
Código de operación^ SDMMVNNI 304	**Matriz de salida densa** 306	**Matriz de origen dispersa** 308	**Matriz de origen densa** 310

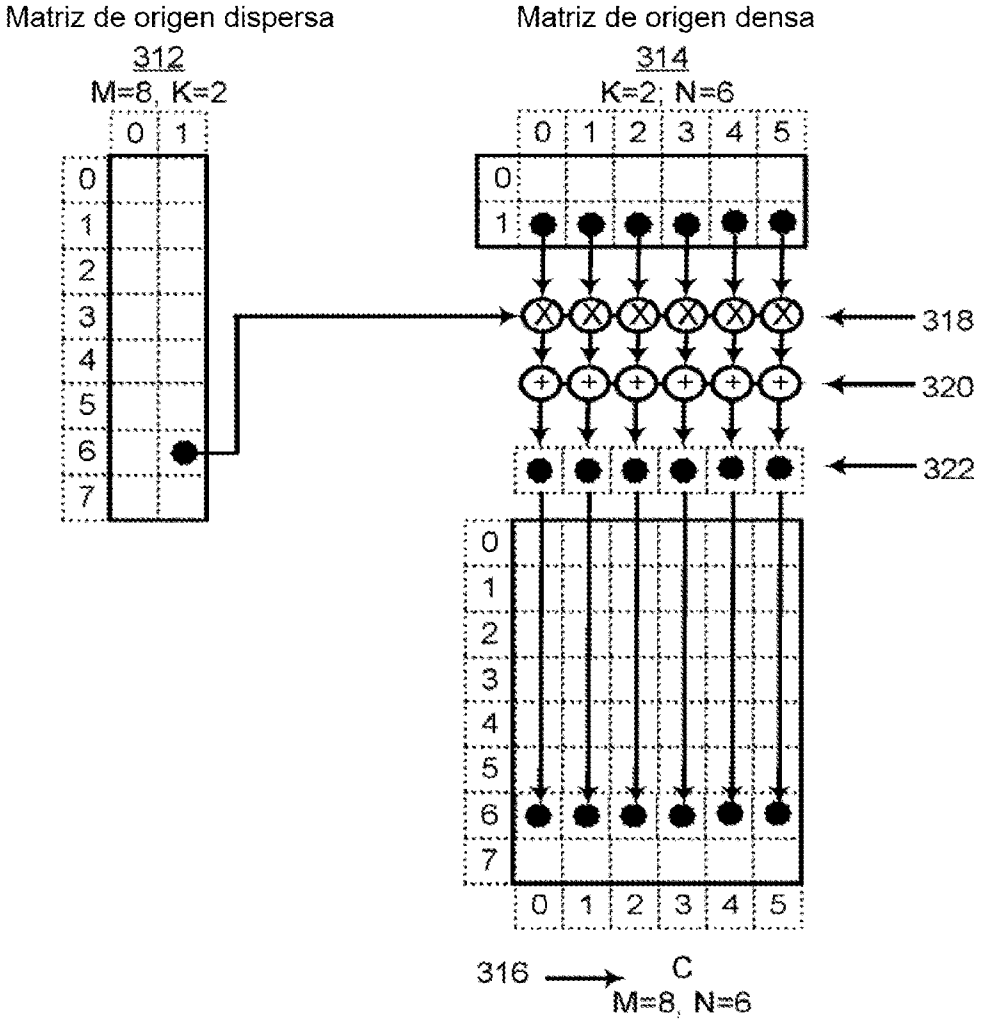


FIG. 3

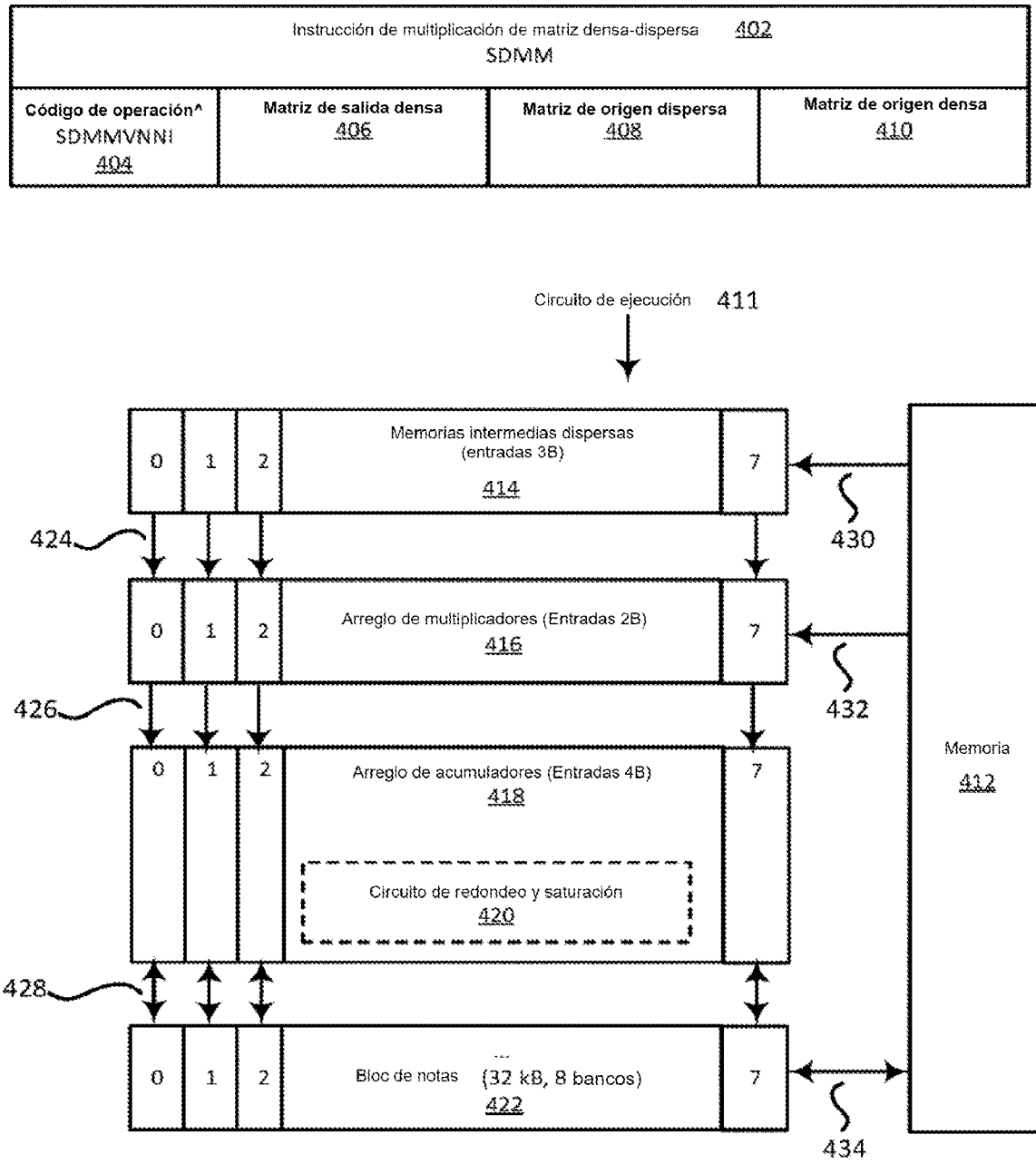

Instrucción de multiplicación de matriz densa-dispersa 402
SDMM
Código de operación^
SDMMVNNI
404
Matriz de salida densa
406
Matriz de origen dispersa
408
Matriz de origen densa
410
Circuito de ejecución 411
0
1
2
Memorias intermedias dispersas
(entradas 3B)
414
7
430
424
Arreglo de multiplicadores (Entradas 2B)
416
432
426
Arreglo de acumuladores (Entradas 4B)
418
Circuito de redondeo y saturación
420
Memoria
412
428
Bloc de notas (32 kB, 8 bancos)
422
434

FIG. 4

```
502
VNNI(C, A. B) {                    // C(mxn), A(M x K), B(K x N)
    para (k en 0 a K) {                    // para cada caso de flujo
        para (m en 0 a M) {                // para cada una de M filas de A
            para (n en 0 a N) {            // para cada una de N columnas de B
                C(m, n) += A(m,k) * B(k,n);
}   }   }   }

504
VNNI(C, A, B) {                    // C(mxn), A(M x K), B(K x N)
    para ((m, k) en (0,0)-(M,K)) {         // para cada elemento de A en (m, k)
        para (n en 0 a N) {                // para cada columna de B en fila correspondiente
            C(m, n) += A(m,k) * B(k,n);
}   }   }

506
SDMMVNNIW(C, A, B) {                   // C(M x N), A(M x K), B(K x N)
    para (cada A(m, k)) distinto de cero   {   // Usar CSR o CSC para encontrar
        para (n en 0 a N) {                // para cada columna de B en fila correspondiente
            C(m, n)[31:0] += A(m,k)[15:0] * B(k,n){15:0];
}   }   }

508
SDMMVNNI(C, A, B) {                    // C(M x N), A(M x K), B(K x N)
    para (cada A(m, k)) distinto de cero   {   // Usar CSR o CSC para encontrar
        para (n en 0 a N) {                // para cada columna de B en fila correspondiente
            C(m, n)[15:0] += A(m,k)[15:0] * B(k,n){15:0];
}   }   }
```

FIG. 5

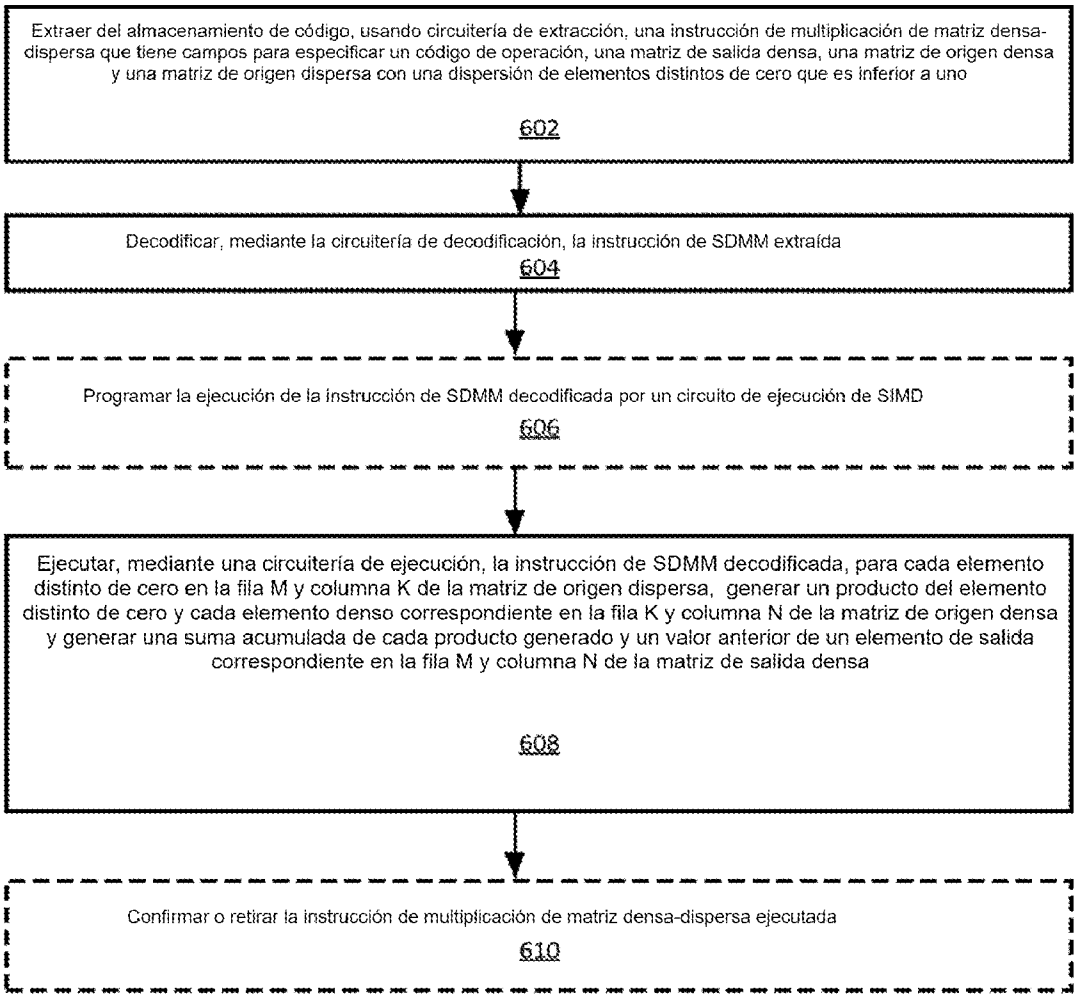

Extraer del almacenamiento de código, usando circuitería de extracción, una instrucción de multiplicación de matriz densa-dispersa que tiene campos para especificar un código de operación, una matriz de salida densa, una matriz de origen densa y una matriz de origen dispersa con una dispersión de elementos distintos de cero que es inferior a uno
602
Decodificar, mediante la circuitería de decodificación, la instrucción de SDMM extraída
604
Programar la ejecución de la instrucción de SDMM decodificada por un circuito de ejecución de SIMD
606
Ejecutar, mediante una circuitería de ejecución, la instrucción de SDMM decodificada, para cada elemento distinto de cero en la fila M y columna K de la matriz de origen dispersa, generar un producto del elemento distinto de cero y cada elemento denso correspondiente en la fila K y columna N de la matriz de origen densa y generar una suma acumulada de cada producto generado y un valor anterior de un elemento de salida correspondiente en la fila M y columna N de la matriz de salida densa
608
Confirmar o retirar la instrucción de multiplicación de matriz densa-dispersa ejecutada
610

FIG. 6

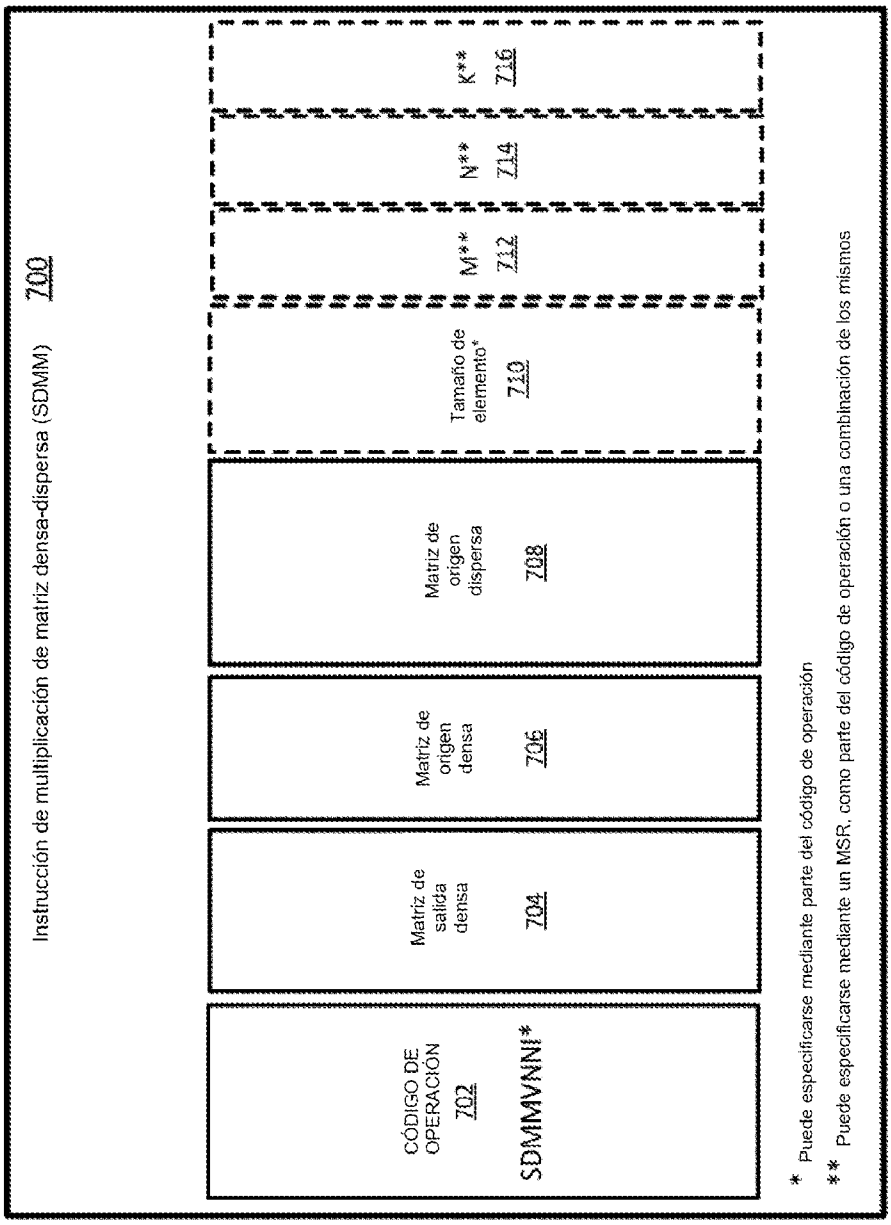

Instrucción de multiplicación de matriz densa-dispersa (SDMM) 700
CÓDIGO DE OPERACIÓN 702
SDMMVNNI*
Matriz de salida densa 704
Matriz de origen densa 706
Matriz de origen dispersa 708
Tamaño de elemento* 710
M** 712
N** 714
K** 716
* Puede especificarse mediante parte del código de operación
** Puede especificarse mediante un MSR, como parte del código de operación o una combinación de los mismos

FIG. 7

FIG.8A

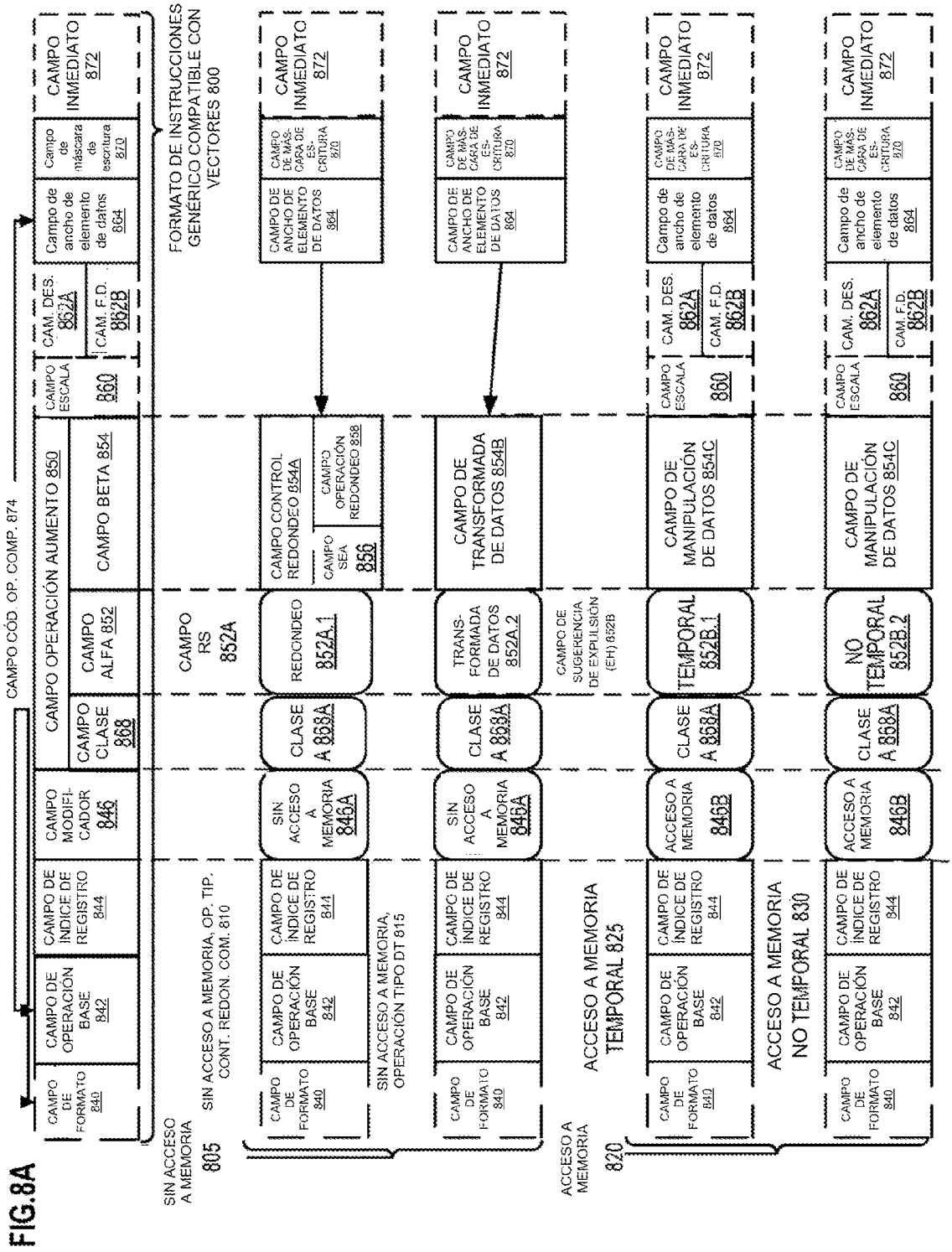
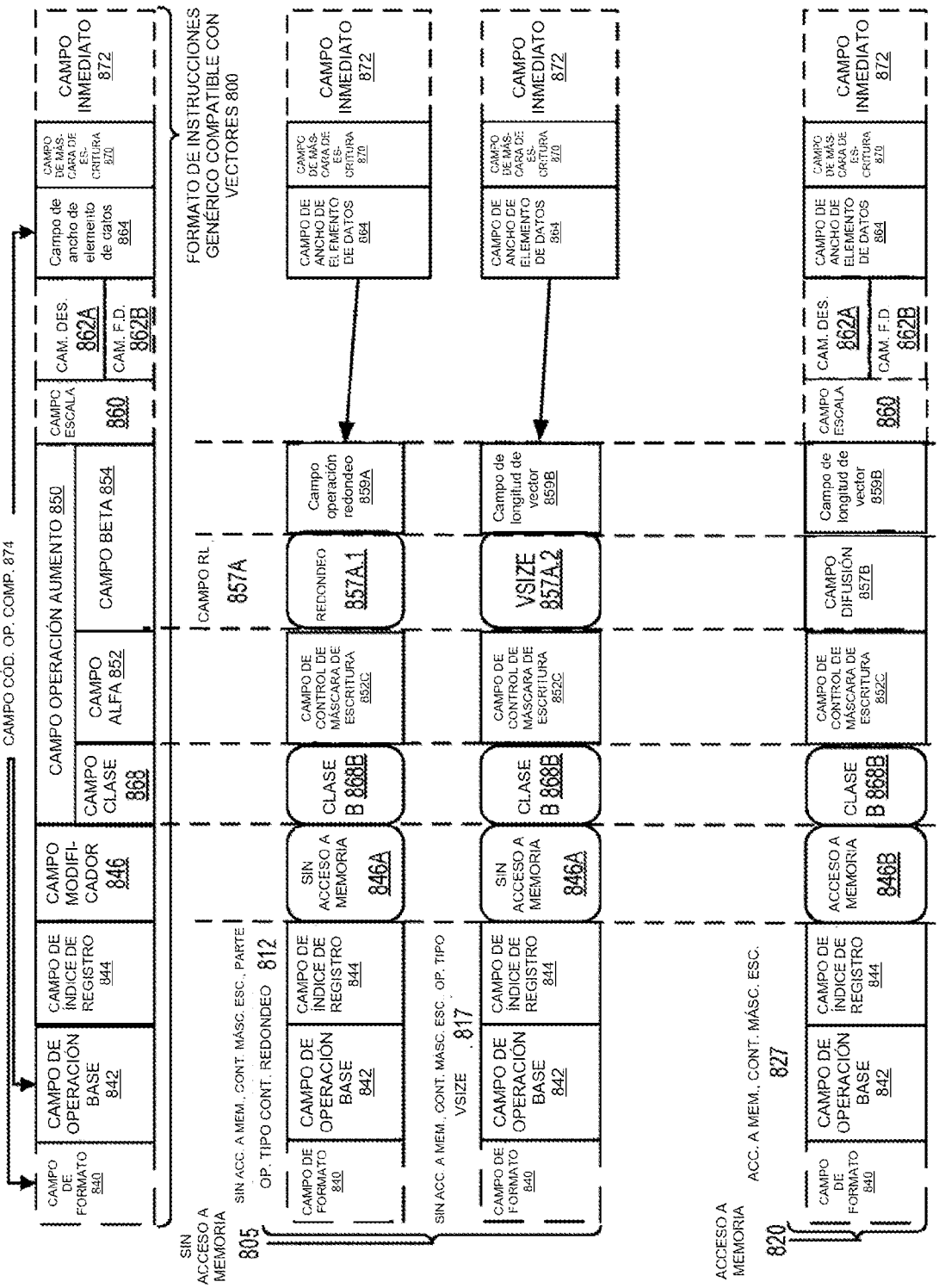

FORMATO DE INSTRUCCIONES GENÉRICO COMPATIBLE CON VECTORES 800
CAMPO CÓD. OP. COMP. 874
SIN ACCESO A MEMORIA 805
SIN ACCESO A MEMORIA, OP. TIP., CONT. REDON. COM. 810
SIN ACCESO A MEMORIA, OPERACIÓN TIPO DT 815
ACCESO A MEMORIA 820
ACCESO A MEMORIA TEMPORAL 825
ACCESO A MEMORIA NO TEMPORAL 830
CAMPO DE FORMATO 840
CAMPO DE OPERACIÓN BASE 842
CAMPO DE ÍNDICE DE REGISTRO 844
CAMPO MODIFICADOR 846
SIN ACCESO A MEMORIA 846A
ACCESO A MEMORIA 846B
CAMPO OPERACIÓN AUMENTO 850
CAMPO CLASE 858
CLASE A 858A
CLASE A 858A
CLASE A 858A
CAMPO ALFA 852
CAMPO BETA 854
CAMPO RS 852A
REDONDEO 852A.1
TRANSFORMADA DE DATOS 852A.2
TEMPORAL 852B.1
NO TEMPORAL 852B.2
CAMPO CONTROL REDONDEO 854A
CAMPO SEA 856
CAMPO OPERACIÓN REDONDEO 858
CAMPO DE TRANSFORMADA DE DATOS 854B
CAMPO DE MANIPULACIÓN DE DATOS 854C
CAMPO DE SUGERENCIA DE EXPULSIÓN (EH) 852B
CAMPO ESCALA 860
CAM. DES. 862A
CAM. F.D. 862B
Campo de ancho de elemento de datos 864
CAMPO DE ANCHO DE ELEMENTO DE DATOS 864
Campo de máscara de escritura 870
CAMPO DE MÁSCARA DE ESCRITURA 870
CAMPO INMEDIATO 872

FIG. 8B


FORMATO DE INSTRUCCIONES GENÉRICO COMPATIBLE CON VECTORES 800
CAMPO CÓD. OP. COMP. 874
SIN ACCESO A MEMORIA
805
SIN ACC. A MEM., CONT. MÁSC. ESC., PARTE OP. TIPO CONT. REDONDEO 812
SIN ACC. A MEM., CONT. MASC. ESC., OP. TIPO VSIZE 817
ACCESO A MEMORIA
820
ACC. A MEM., CONT. MÁSC. ESC. 827
CAMPO DE FORMATO 840
CAMPO DE OPERACIÓN BASE 842
CAMPO DE ÍNDICE DE REGISTRO 844
CAMPO MODIFICADOR 846
SIN ACCESO A MEMORIA 846A
ACCESO A MEMORIA 846B
CAMPO OPERACIÓN AUMENTO 850
CAMPO CLASE 868
CLASE B 868B
CAMPO ALFA 852
CAMPO BETA 854
CAMPO DE CONTROL DE MÁSCARA DE ESCRITURA 852C
CAMPO ESCALA 860
CAM. DES. 862A
CAM. F.D. 862B
Campo de ancho de elemento de datos 864
CAMPO DE MÁS-CARA DE ES-CRITURA 870
CAMPO INMEDIATO 872
CAMPO RL 857A
REDONDEO 857A.1
VSIZE 857A.2
Campo operación redondeo 859A
Campo de longitud de vector 859B
CAMPO DIFUSIÓN 857B
CAMPO DE ANCHO DE ELEMENTO DE DATOS 864

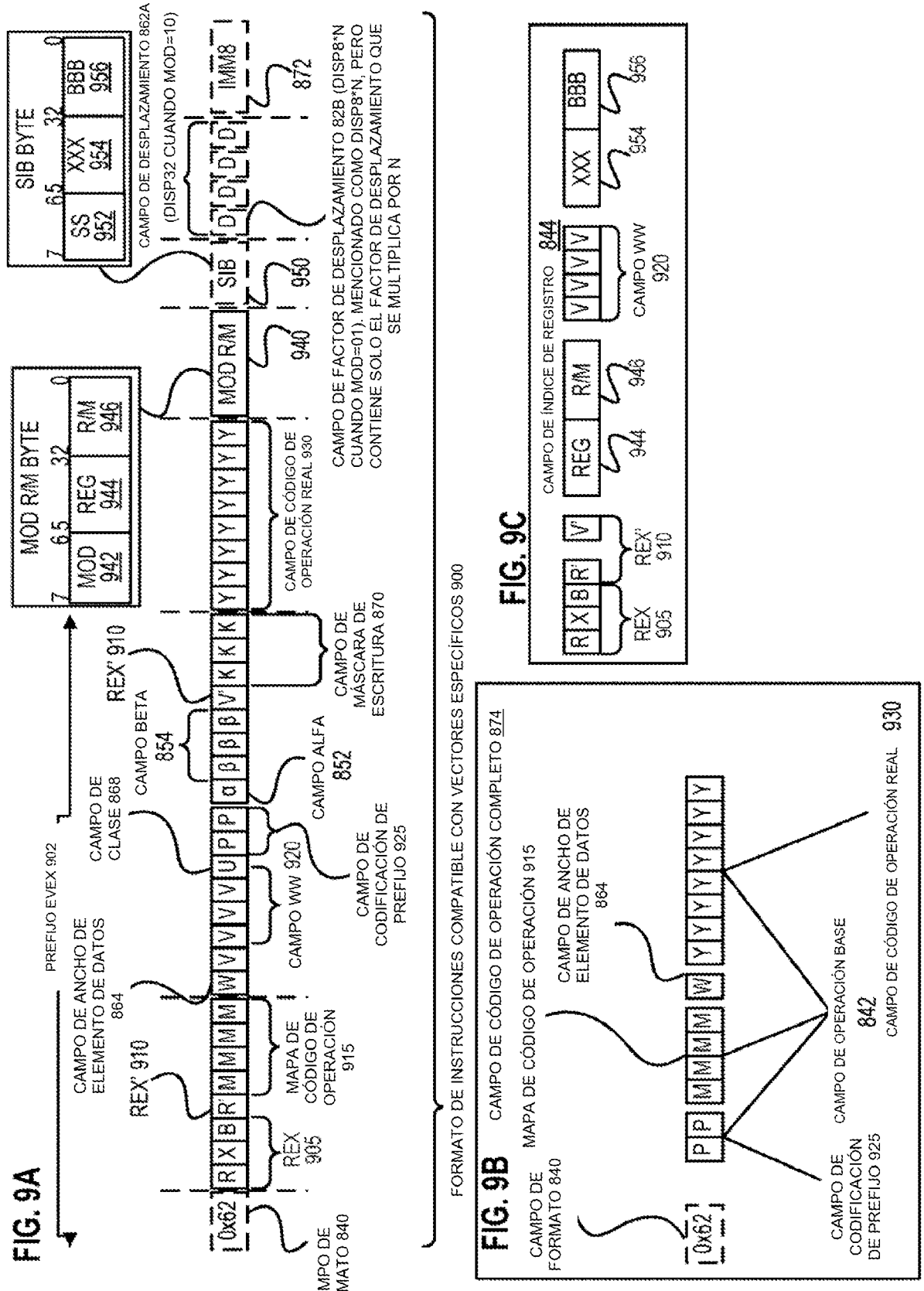


FIG. 9A
PREFIJO EVEX 902
CAMPO DE ANCHO DE ELEMENTO DE DATOS 864
CAMPO DE CLASE 868
CAMPO BETA 854
REX' 910
CAMPO WW 920
CAMPO ALFA 852
CAMPO DE CODIFICACIÓN DE PREFIJO 925
CAMPO DE MÁSCARA DE ESCRITURA 870
CAMPO DE CÓDIGO DE OPERACIÓN REAL 930
CAMPO DE FACTOR DE DESPLAZAMIENTO 82B (DISP8*N CUANDO MOD=01). MENCIONADO COMO DISP8*N, PERO CONTIENE SOLO EL FACTOR DE DESPLAZAMIENTO QUE SE MULTIPLICA POR N
CAMPO DE DESPLAZAMIENTO 862A
(DISP32 CUANDO MOD=10)
MOD R/M BYTE
MOD 942
REG 944
R/M 946
SIB BYTE
SS 952
XXX 954
BBB 956
CAMPO DE ORMATO 840
REX 905
REX' 910
MAPA DE CÓDIGO DE OPERACIÓN 915
FORMATO DE INSTRUCCIONES COMPATIBLE CON VECTORES ESPECÍFICOS 900
0x62
MOD R/M
SIB
872
940
950
IMM8
FIG. 9B
CAMPO DE CÓDIGO DE OPERACIÓN COMPLETO 874
CAMPO DE FORMATO 840
MAPA DE CÓDIGO DE OPERACIÓN 915
CAMPO DE ANCHO DE ELEMENTO DE DATOS 864
CAMPO DE CODIFICACIÓN DE PREFIJO 925
CAMPO DE OPERACIÓN BASE 842
CAMPO DE CÓDIGO DE OPERACIÓN REAL 930
FIG. 9C
CAMPO DE ÍNDICE DE REGISTRO 844
REX 905
REX' 910
V'
REG 944
R/M 946
CAMPO WW 920
XXX 954
BBB 956

FIG. 9D

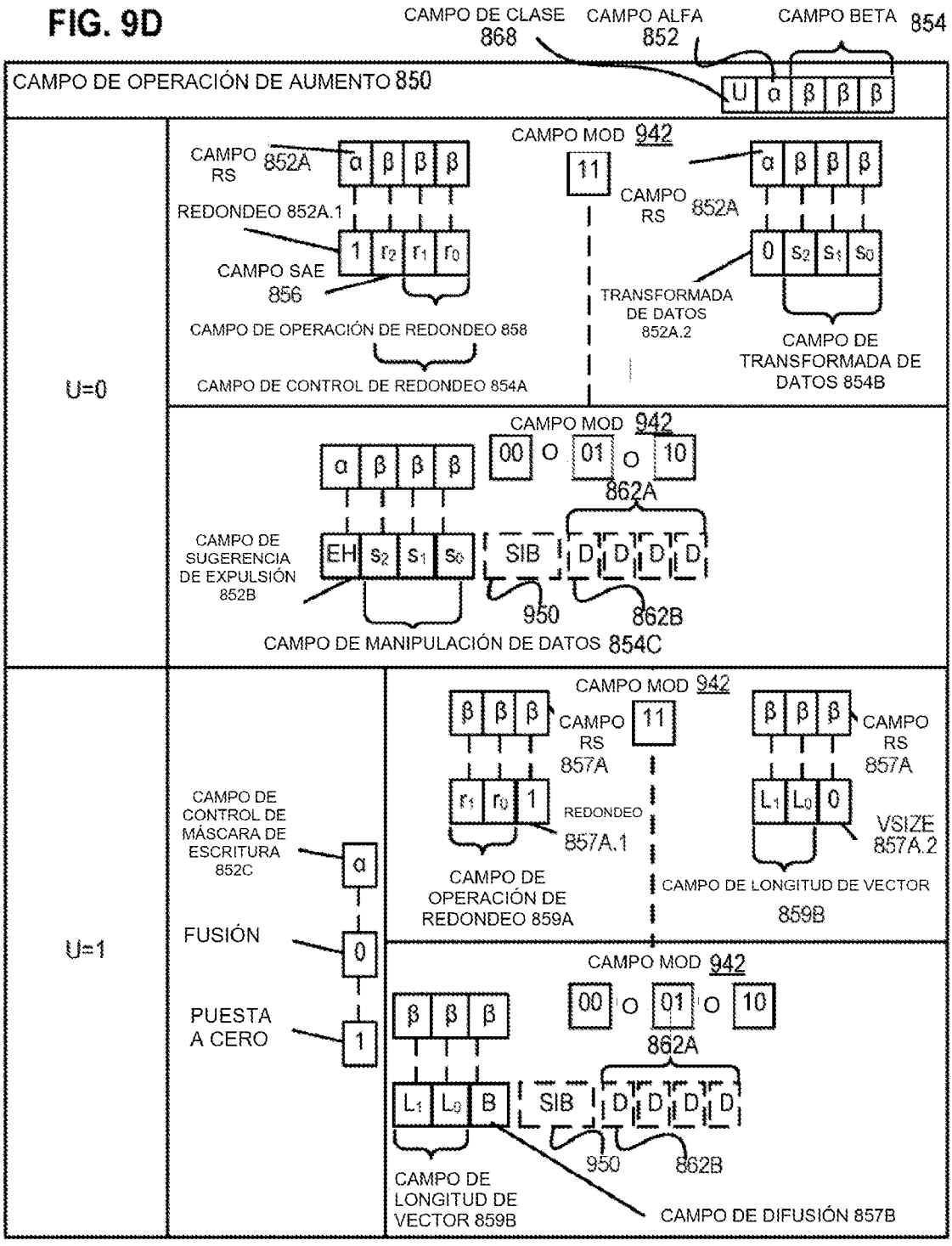
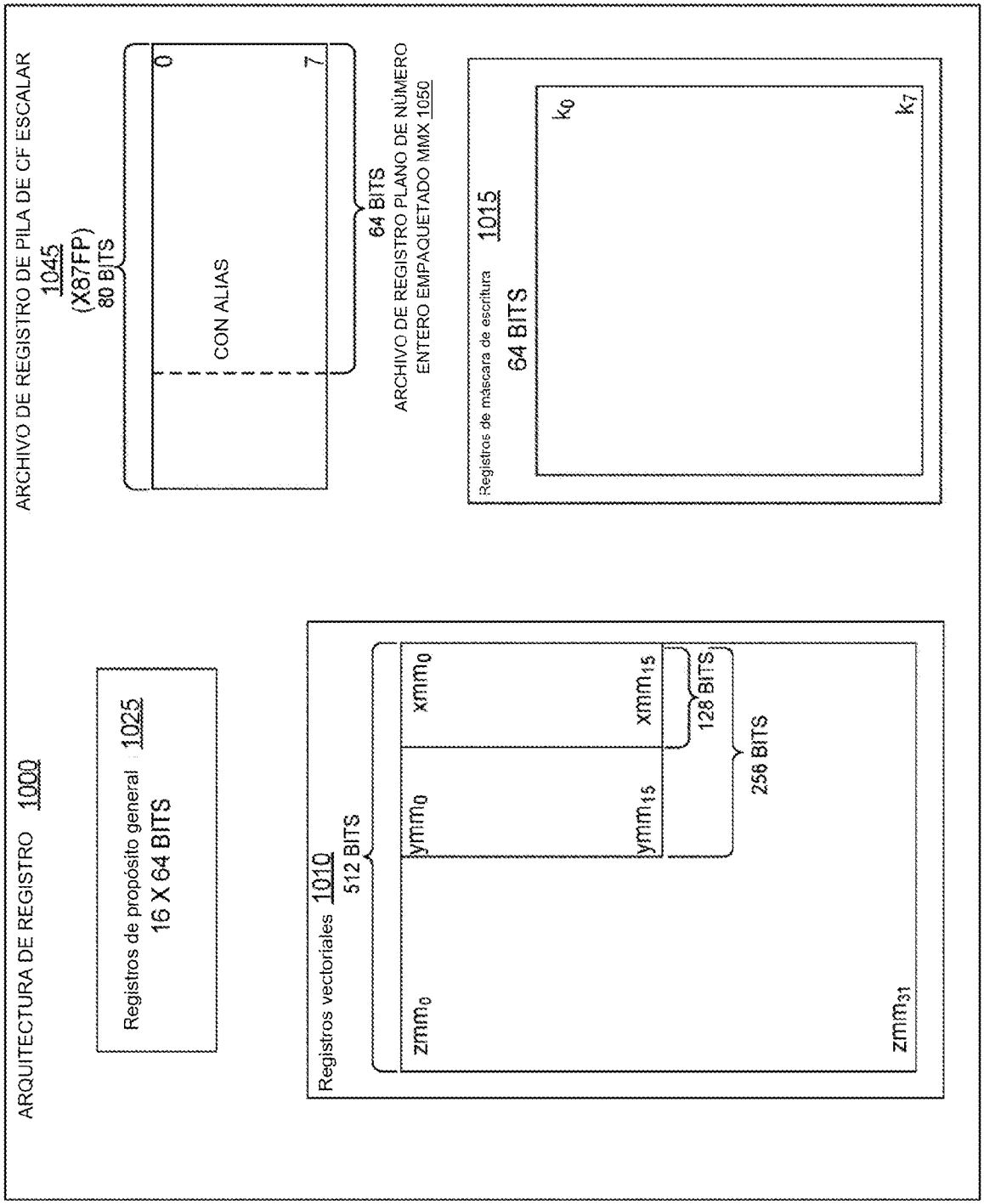

CAMPO DE CLASE 868
CAMPO ALFA 852
CAMPO BETA 854
CAMPO DE OPERACIÓN DE AUMENTO 850
U α β β β
U=0
CAMPO RS 852A
REDONDEO 852A.1
CAMPO SAE 856
CAMPO DE OPERACIÓN DE REDONDEO 858
CAMPO DE CONTROL DE REDONDEO 854A
CAMPO MOD 942
11
TRANSFORMADA DE DATOS 852A.2
CAMPO DE TRANSFORMADA DE DATOS 854B
00 O 01 O 10
862A
CAMPO DE SUGERENCIA DE EXPULSIÓN 852B
EH SIB D D D D
950
862B
CAMPO DE MANIPULACIÓN DE DATOS 854C
U=1
CAMPO DE CONTROL DE MÁSCARA DE ESCRITURA 852C
FUSIÓN
PUESTA A CERO
CAMPO RS 857A
REDONDEO 857A.1
CAMPO DE OPERACIÓN DE REDONDEO 859A
VSIZE 857A.2
CAMPO DE LONGITUD DE VECTOR 859B
L1 L0 B
CAMPO DE LONGITUD DE VECTOR 859B
CAMPO DE DIFUSIÓN 857B

FIG. 10


ARQUITECTURA DE REGISTRO 1000
Registros de propósito general 1025
16 X 64 BITS
ARCHIVO DE REGISTRO DE PILA DE CF ESCALAR 1045
(X87FP)
80 BITS
0
7
CON ALIAS
64 BITS
ARCHIVO DE REGISTRO PLANO DE NÚMERO ENTERO EMPAQUETADO MMX 1050
Registros de máscara de escritura 1015
64 BITS
k0
k7
Registros vectoriales 1010
512 BITS
zmm0
ymm0
xmm0
ymm15
xmm15
128 BITS
256 BITS
zmm31

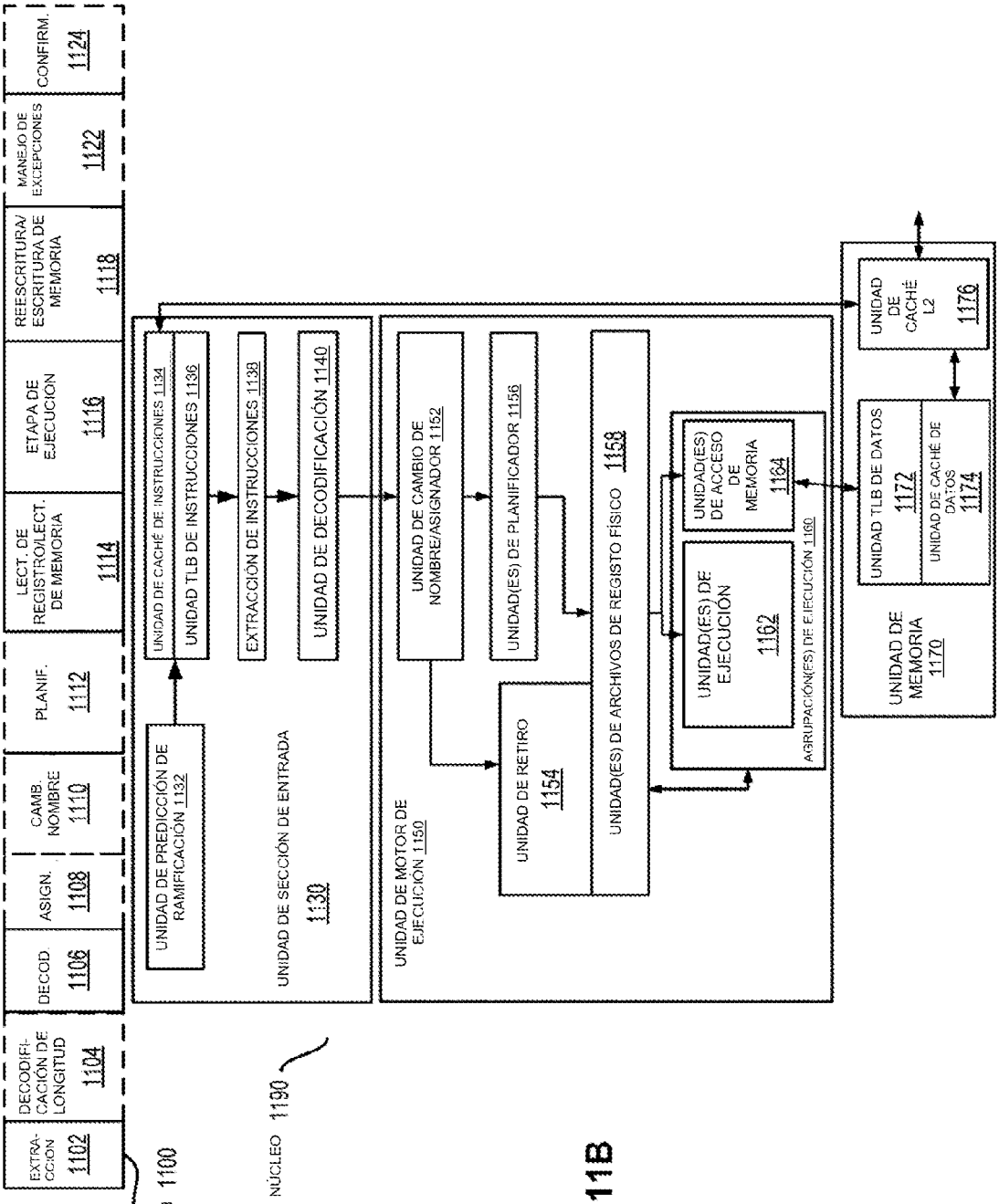


FIG. 11A

FIG. 11B

FIG. 12B

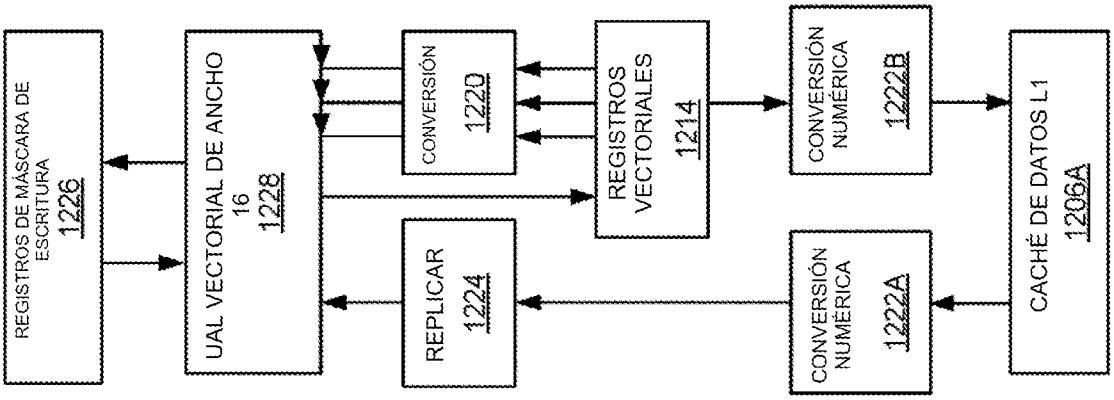
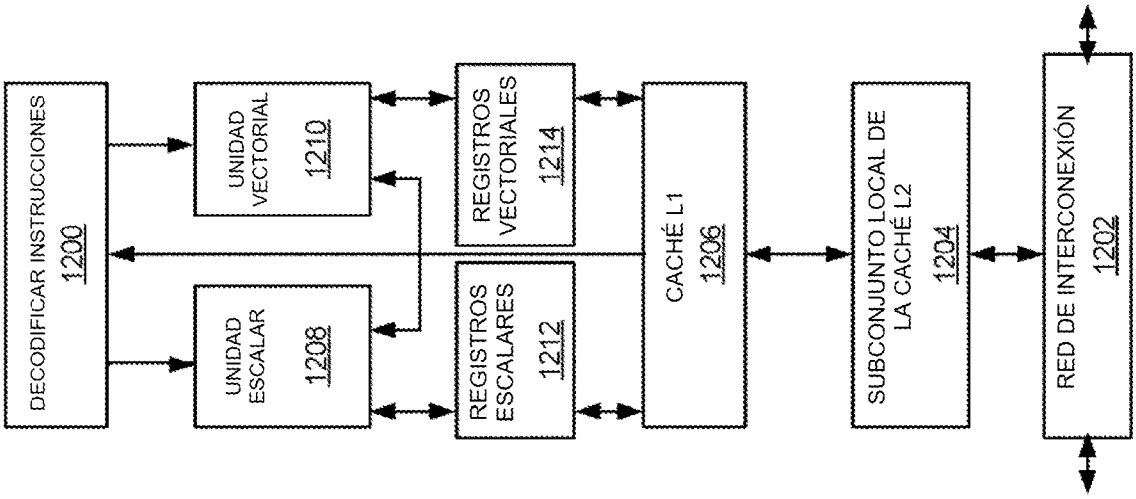


FIG. 12A



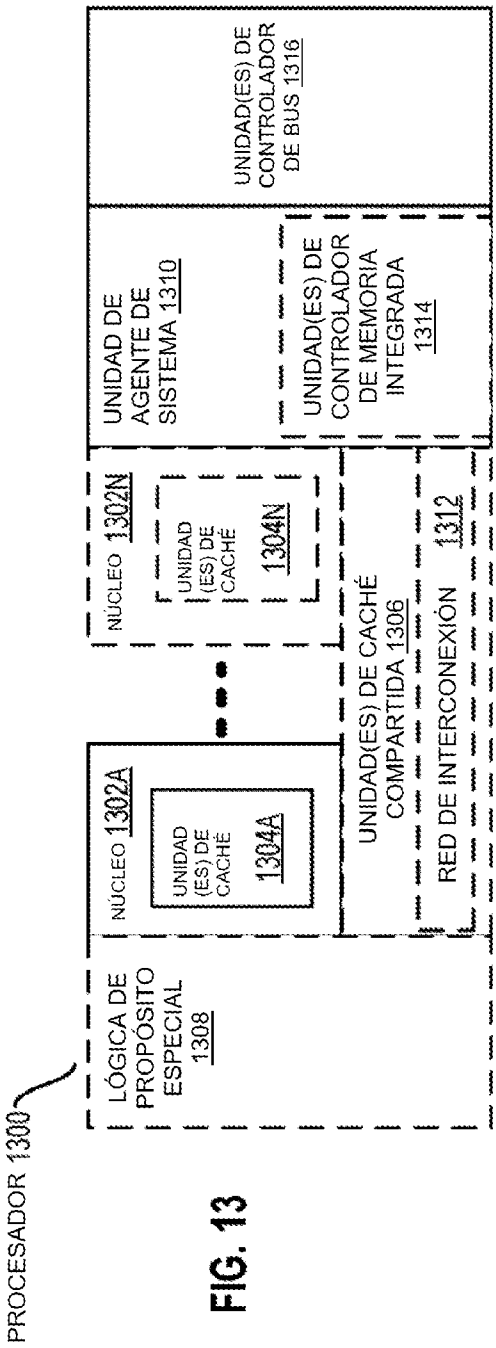

PROCESADOR 1300
LÓGICA DE PROPÓSITO ESPECIAL 1308
NÚCLEO 1302A
UNIDAD (ES) DE CACHÉ 1304A
NÚCLEO 1302N
UNIDAD (ES) DE CACHÉ 1304N
UNIDAD(ES) DE CACHÉ COMPARTIDA 1306
RED DE INTERCONEXIÓN 1312
UNIDAD DE AGENTE DE SISTEMA 1310
UNIDAD(ES) DE CONTROLADOR DE MEMORIA INTEGRADA 1314
UNIDAD(ES) DE CONTROLADOR DE BUS 1316

FIG. 13

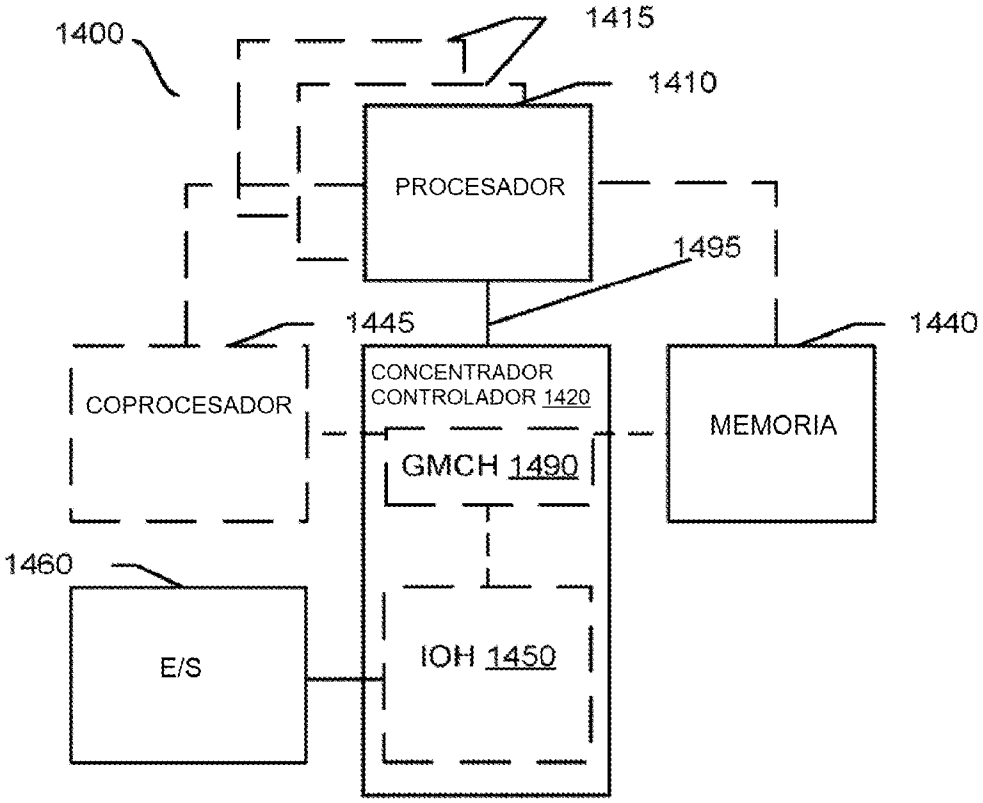

1400
1415
1410
PROCESADOR
1495
1445
1440
COPROCESADOR
CONCENTRADOR
CONTROLADOR 1420
MEMORIA
GMCH 1490
1460
E/S
IOH 1450

FIG. 14

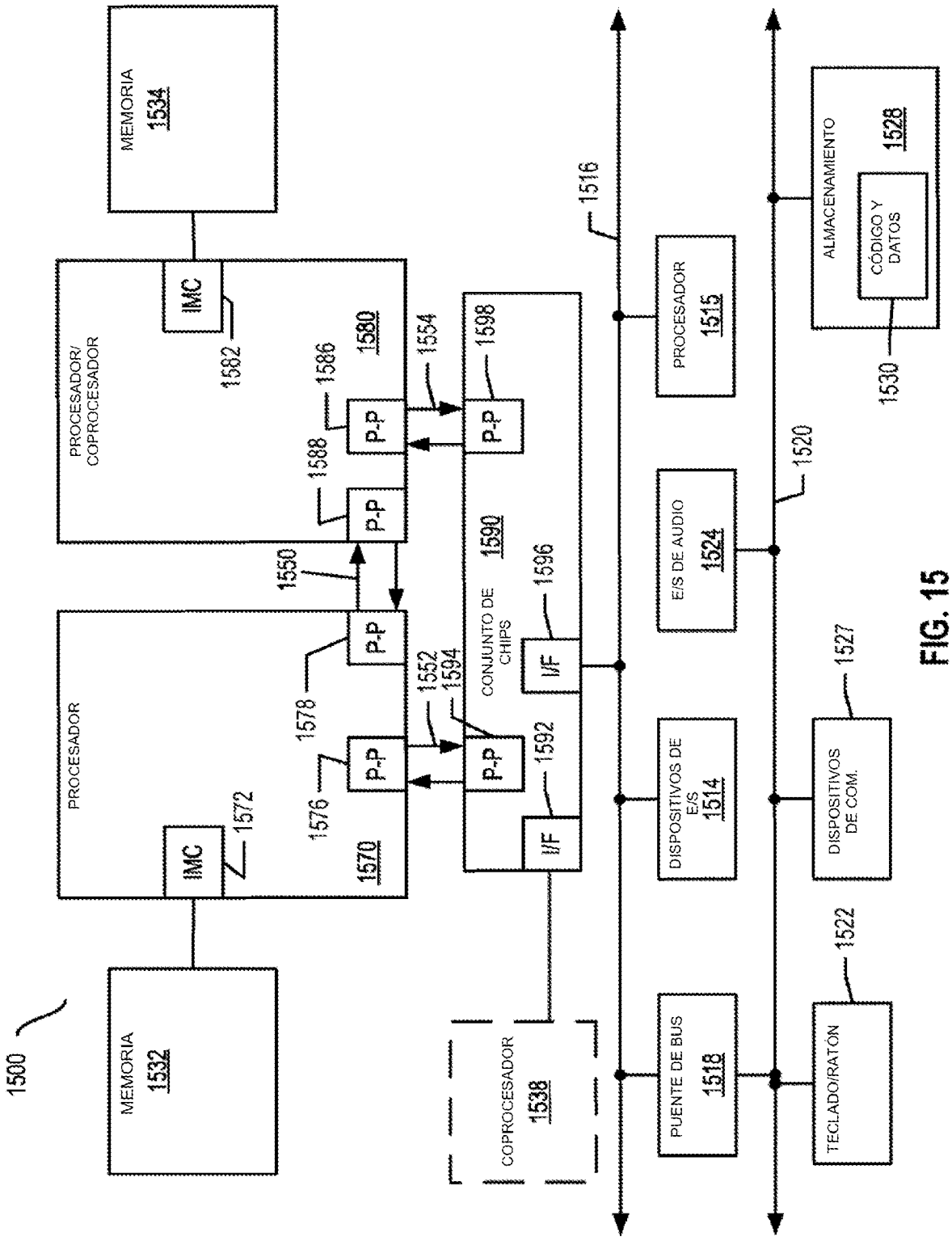

1500
MEMORIA
1532
MEMORIA
1534
IMC
1572
IMC
1582
PROCESADOR
PROCESADOR/
COPROCESADOR
1570
1580
P-P
1576
1578
1586
1588
1550
1552
1554
1590
CONJUNTO DE CHIPS
1592
1594
1596
1598
I/F
I/F
COPROCESADOR
1538
1516
PUENTE DE BUS
1518
DISPOSITIVOS DE E/S
1514
E/S DE AUDIO
1524
PROCESADOR
1515
1520
TECLADO/RATÓN
1522
DISPOSITIVOS DE COM.
1527
ALMACENAMIENTO
1528
CÓDIGO Y DATOS
1530

FIG. 15

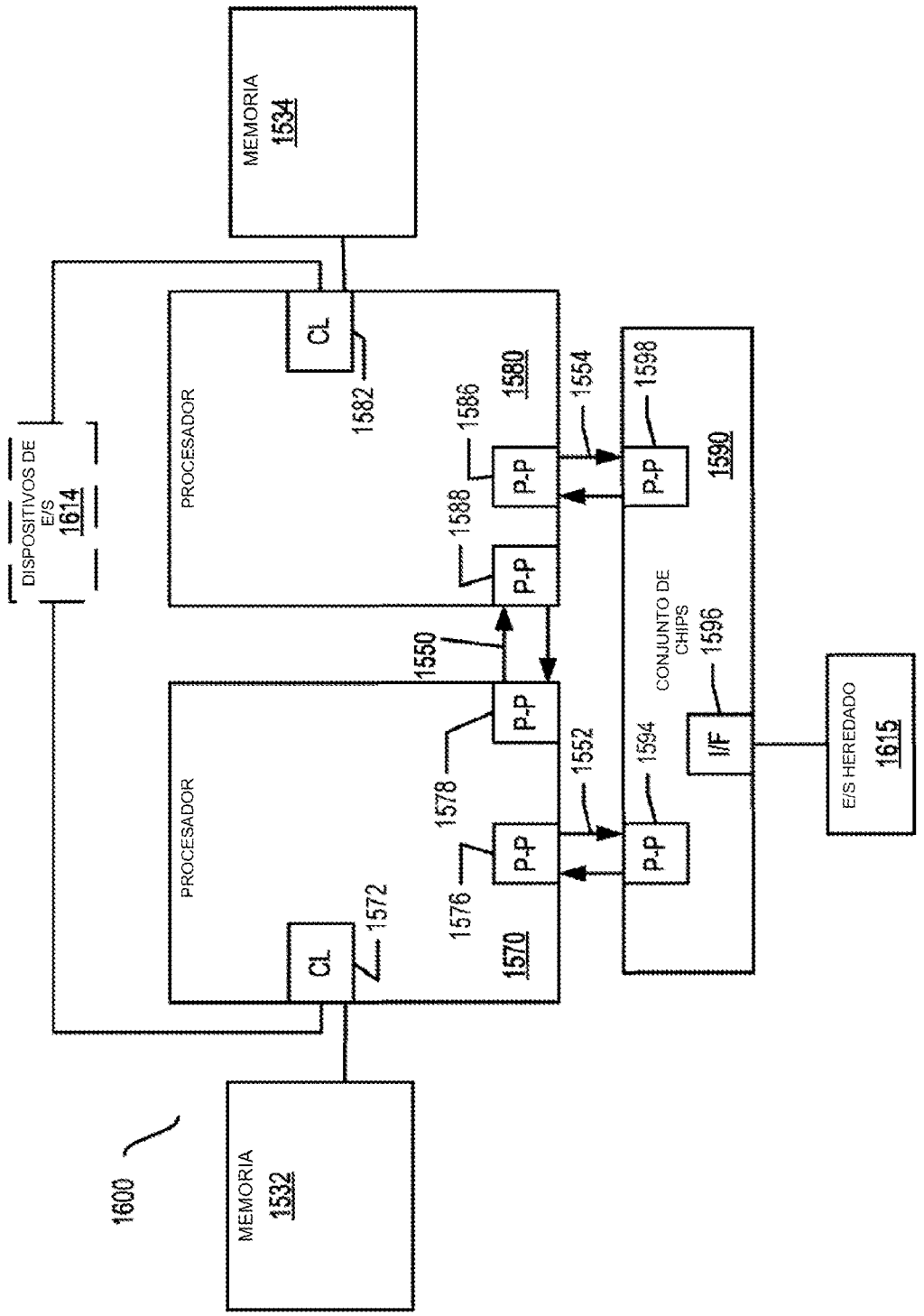

1600
MEMORIA
1532
MEMORIA
1534
DISPOSITIVOS DE E/S
1614
PROCESADOR
1570
CL
1572
1576
1578
1580
CL
1582
1586
1588
1550
1552
1554
P-P
CONJUNTO DE CHIPS
1590
1594
1596
1598
I/F
E/S HEREDADO
1615

FIG. 16

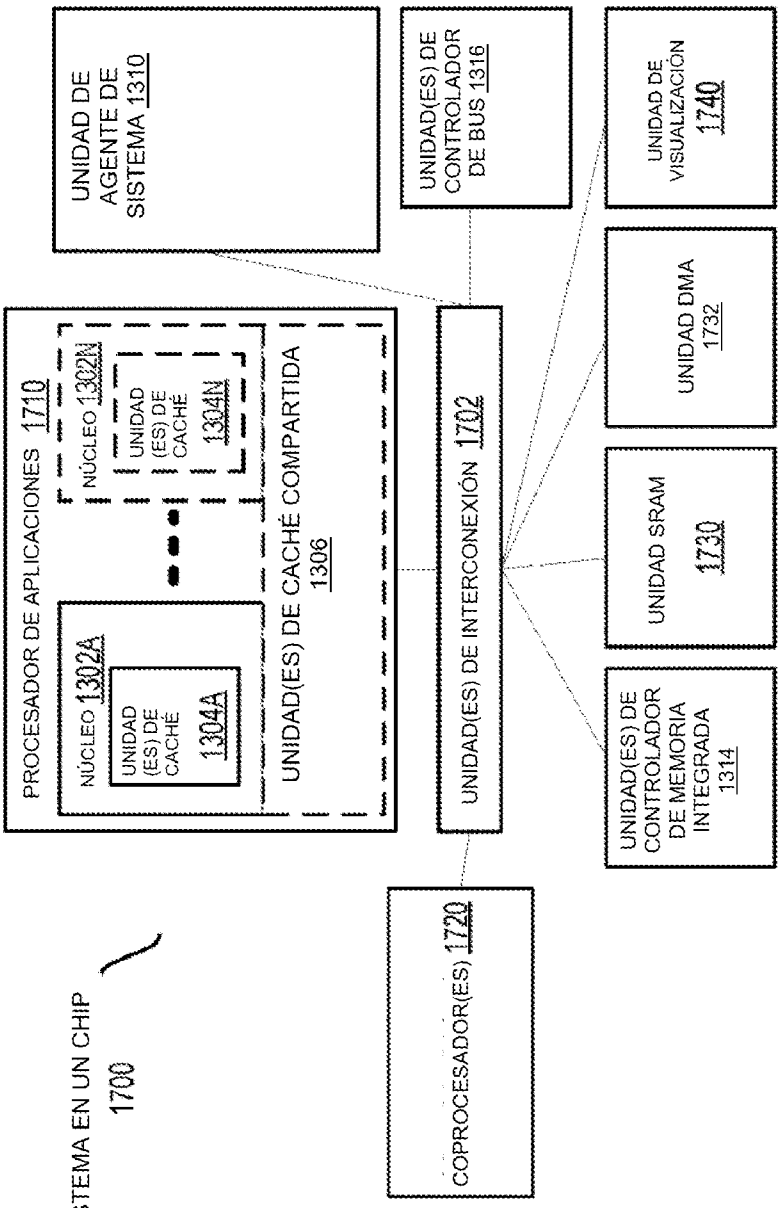

SISTEMA EN UN CHIP
1700
PROCESADOR DE APLICACIONES 1710
NÚCLEO 1302A
UNIDAD (ES) DE CACHÉ 1304A
NÚCLEO 1302N
UNIDAD (ES) DE CACHÉ 1304N
UNIDAD(ES) DE CACHÉ COMPARTIDA 1306
UNIDAD DE AGENTE DE SISTEMA 1310
UNIDAD(ES) DE CONTROLADOR DE BUS 1316
UNIDAD DE VISUALIZACIÓN 1740
UNIDAD DMA 1732
UNIDAD SRAM 1730
UNIDAD(ES) DE CONTROLADOR DE MEMORIA INTEGRADA 1314
UNIDAD(ES) DE INTERCONEXIÓN 1702
COPROCESADOR(ES) 1720

FIG. 17

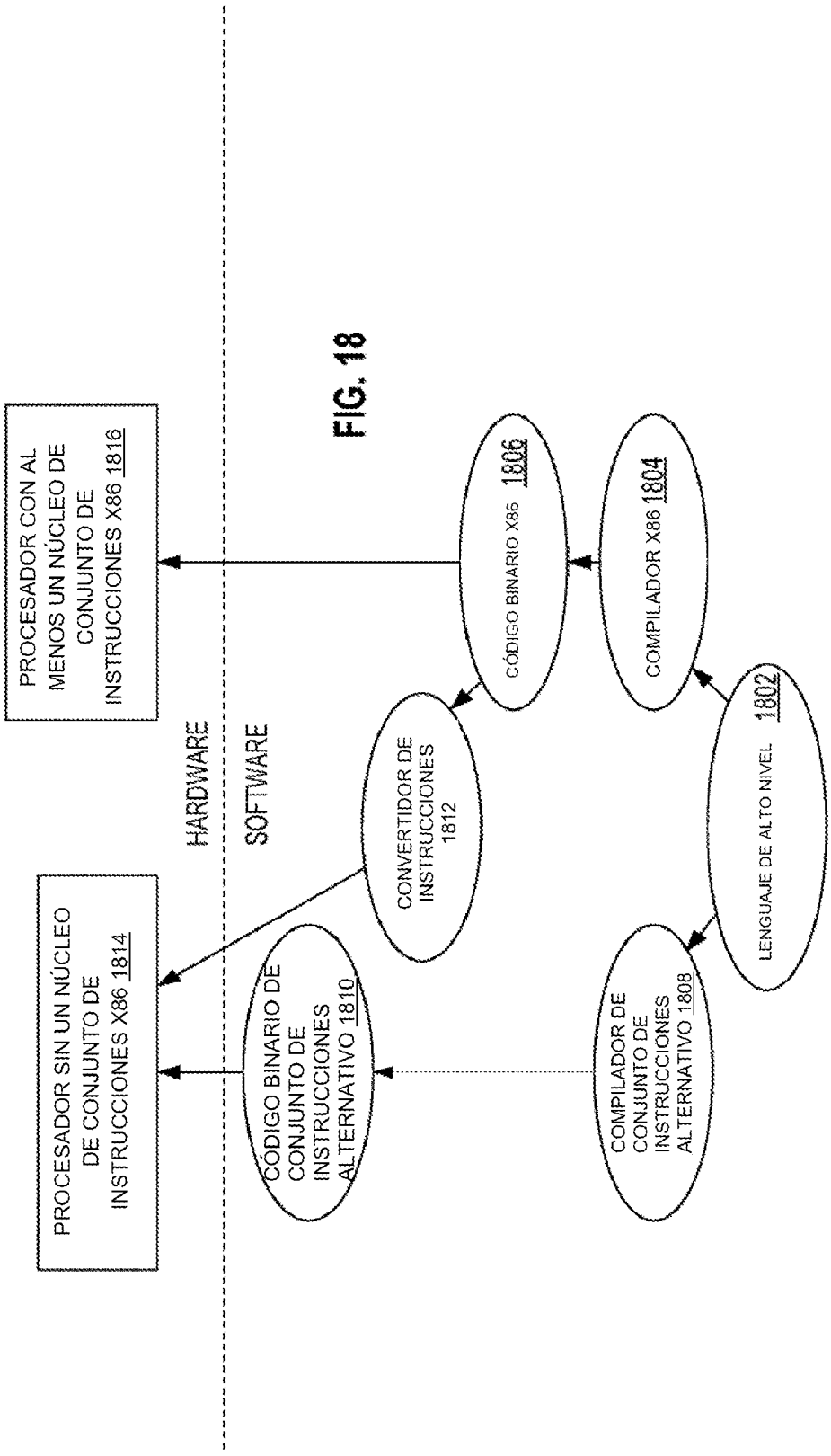

PROCESADOR CON AL MENOS UN NÚCLEO DE CONJUNTO DE INSTRUCCIONES X86 1816
PROCESADOR SIN UN NÚCLEO DE CONJUNTO DE INSTRUCCIONES X86 1814
HARDWARE
SOFTWARE
CÓDIGO BINARIO X86 1806
CONVERTIDOR DE INSTRUCCIONES 1812
CÓDIGO BINARIO DE CONJUNTO DE INSTRUCCIONES ALTERNATIVO 1810
COMPILADOR X86 1804
LENGUAJE DE ALTO NIVEL 1802
COMPILADOR DE CONJUNTO DE INSTRUCCIONES ALTERNATIVO 1808

FIG. 18