(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 820 126**

(51) Int. CI.:

G06F 9/30 (2008.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **25.11.2015 PCT/US2015/062564**

(87) Fecha y número de publicación internacional: **30.06.2016 WO16105819**

(96) Fecha de presentación y número de la solicitud europea: **25.11.2015 E 15874023 (3)**

(97) Fecha y número de publicación de la concesión europea: **29.07.2020 EP 3238035**

(54) Título: **Método y equipo para realizar una reorganización de bits de un vector**

(30) Prioridad:

27.12.2014 US 201414583636

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
19.04.2021

(73) Titular/es:

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

(72) Inventor/es:

OULD-AHMED-VALL, ELMOUSTAPHA;
CORBAL, JESUS;
VALENTINE, ROBERT;
CHARNEY, MARK J.;
SOLE, GUILLEM y
ESPASA, ROGER

(74) Agente/Representante:

LEHMANN NOVO, María Isabel

DESCRIPCIÓN

Método y equipo para realizar una reorganización de bits de un vector

Antecedentes

Campo de la invención

Esta invención está relacionada, en general, con el campo de los procesadores de ordenador. Más en particular, la invención está relacionada con un método y un equipo para realizar una reorganización de los bits de un vector.

Descripción de la técnica asociada

Un conjunto de instrucciones, o una arquitectura de conjunto de instrucciones (ISA), es la parte de la arquitectura de un ordenador relacionada con la programación, incluyendo los tipos de datos, instrucciones, arquitectura de registros, modos de direccionamiento, arquitectura de memoria, gestión de interrupciones y excepciones nativos y entradas y salidas (E/S) externas. Se debería observar que el término “instrucción” se refiere en general en la presente solicitud a macro instrucciones – esto es instrucciones que se le proporcionan al procesador para su ejecución – en oposición a micro instrucciones o micro operaciones – que son el resultado de la decodificación de las macro instrucciones por parte de un decodificador del procesador. Las micro instrucciones o micro operaciones se pueden configurar para ordenarle a la unidad de ejecución del procesador que realice operaciones para implementar la lógica asociada con la macro instrucción.

La ISA es diferente de la microarquitectura, la cual es el conjunto de técnicas de diseño de procesadores utilizadas para implementar el conjunto de instrucciones. Procesadores con diferentes microarquitecturas pueden compartir un conjunto común de instrucciones. Por ejemplo, los procesadores Intel® Pentium 4, los procesadores Intel® Core™, y los procesadores de Advanced Micro Devices, Inc. de Sunnyvale, CA, implementan versiones prácticamente idénticas del conjunto de instrucciones x86 (con algunas extensiones que se han añadido con las versiones más nuevas), pero tienen diferentes diseños internos. Por ejemplo, la misma arquitectura de registros de ISA se puede implementar de formas diferentes en diferentes microarquitecturas utilizando técnicas bien conocidas, incluyendo registros físicos dedicados, uno o más registros físicos asignados dinámicamente utilizando un mecanismo para renombrar registros (por ejemplo, la utilización de una Tabla de Alias de Registros (RAT), una Memoria Intermedia de Reordenación (ROB y un fichero de registros de descarte). A menos que se especifique de otro modo, las expresiones arquitectura de registros, fichero de registros, y registro se utilizan en la presente solicitud para referirse a lo que es visible para el software/programador y la forma en la que las instrucciones especifican los registros. Cuando sea necesario diferenciarlos, se utilizará el adjetivo “lógico”, “de arquitectura”, o “visible por parte del software” para indicar registros/ficheros en la arquitectura de registros, mientras que se utilizarán diferentes adjetivos para denominar registros en una microarquitectura dada (por ejemplo, un registro físico, una memoria intermedia de reordenación, un registro de descarte, un clúster de registros).

Un conjunto de instrucciones incluye uno o más formatos de instrucción. Un formato de instrucción dado define varios campos (número de bits, localización de los bits) para especificar, entre otras cosas, la operación a realizar y el/los operando(s) sobre los que se va a realizar la operación. Algunos formatos de instrucción también se descomponen mediante la definición de plantillas de instrucción (o subformatos). Por ejemplo, se puede definir que las plantillas de instrucción de un formato de instrucción dado tengan diferentes subconjuntos de campos de formatos de instrucción (los campos incluidos típicamente se encuentran en el mismo orden, pero al menos algunos tienen diferentes posiciones de bit debido a que se incluyen menos campos) y/o se pueden definir para que tengan un campo dado interpretado de forma distinta. Una instrucción dada se expresa utilizando un formato de instrucción dado (y, si se define, en una de las plantillas de instrucción de dicho formato de instrucción) y especifica la operación y los operandos. Un flujo de instrucciones es una secuencia específica de instrucciones, en donde cada instrucción en la secuencia es la ocurrencia de una instrucción en un formato de instrucción (y, si se define, una dada de las plantillas de instrucción de dicho formato de instrucción).

El documento WO 2014/150913 A1 está relacionado con métodos y equipos para procesar de forma eficiente datos en varios formatos en una arquitectura de una instrucción y múltiples datos (“SIMD”) como, por ejemplo, un método para recuperar bits de un vector de bits en desplazamientos especificados con respecto a una base en una arquitectura SIMD.

Breve descripción de los dibujos

Se puede obtener una mejor comprensión de la presente invención a partir de la siguiente descripción detallada junto con los siguientes dibujos, en los que:

las FIG. 1A y 1B son diagramas esquemáticos que ilustran un formato genérico de instrucción adaptada para vectores y sus plantillas de instrucción de acuerdo con los modos de realización de la invención;

las FIG. 2A-D son diagramas esquemáticos que ilustran un formato específico de instrucción adaptada para vectores de ejemplo de acuerdo con los modos de realización de la invención;

la FIG. 3 es un diagrama esquemático de una arquitectura de registros de acuerdo con un modo de realización de la invención; y

la FIG. 4A es un diagrama esquemático que ilustra tanto un pipeline (secuencia) recuperar en orden, decodificar, descartar de ejemplo como un pipeline de cambio de nombre de registro, emisión/ejecución fuera de orden de ejemplo de acuerdo con los modos de realización de la invención;

la FIG. 4B es un diagrama esquemático que ilustra tanto un modo de realización de un núcleo de recuperar en orden, decodificar, descartar de ejemplo como un núcleo de arquitectura de cambio de nombre de registro, emisión/ejecución fuera de orden de ejemplo a incluir en un procesador de acuerdo con los modos de realización de la invención;

la FIG. 5A es un diagrama esquemático de un único núcleo de procesador junto con su conexión a una red de interconexión sobre el chip;

la FIG. 5B ilustra una vista ampliada de una parte del núcleo del procesador de la FIG. 5A de acuerdo con los modos de realización de la invención;

la FIG. 6 es un diagrama esquemático de un único núcleo de procesador y un procesador multinúcleo con un controlador de memoria y gráficos integrados de acuerdo con los modos de realización de la invención;

la FIG. 7 ilustra un diagrama esquemático de un sistema de acuerdo con un modo de realización de la presente invención;

la FIG. 8 ilustra un diagrama esquemático de un segundo sistema de acuerdo con un modo de realización de la presente invención;

la FIG. 9 ilustra un diagrama esquemático de un tercer sistema de acuerdo con un modo de realización de la presente invención;

la FIG. 10 ilustra un diagrama esquemático de un sistema en chip (SoC) de acuerdo con un modo de realización de la presente invención;

la FIG. 11 ilustra un diagrama esquemático que compara la utilización de un conversor de instrucciones software para convertir instrucciones binarias procedentes de un conjunto de instrucciones origen en instrucciones binarias en un conjunto de instrucciones objetivo de acuerdo con los modos de realización de la invención;

la FIG. 12 ilustra un procesador de ejemplo sobre el que se pueden implementar los modos de realización de la invención;

la FIG. 13 ilustra una lógica de reordenación de bits en un vector de acuerdo con un modo de realización de la invención; y

la FIG. 14 ilustra un método de acuerdo con un modo de realización de la invención.

Descripción detallada

En la siguiente descripción, con propósito explicativo, se describen en orden numerosos detalles específicos con el fin de proporcionar un conocimiento exhaustivo de los modos de realización de la invención descrita más abajo. Sin embargo, para una persona experimentada en la técnica será evidente que los modos de realización de la invención se pueden poner en práctica sin algunos de dichos detalles específicos. En otras instancias, las estructuras y dispositivos bien conocidos se muestran en forma de diagrama esquemático con el fin de evitar ocultar los principios subyacentes de los modos de realización de la invención.

Un conjunto de instrucciones incluye uno o más formatos de instrucción. Un formato de instrucción dado define varios campos (número de bits, localización de los bits) para especificar, entre otras cosas, la operación a realizar (opcode) y el/los operando(s) sobre los que realizar dicha operación. Algunos formatos de instrucción se descomponen también mediante la definición de plantillas de instrucción (o subformatos). Por ejemplo, se puede definir que las plantillas de instrucción de un formato de instrucción dado tengan diferentes subconjuntos de campos de formato de instrucción (los campos incluidos típicamente se encuentran en el mismo orden, pero al menos algunos tienen diferentes posiciones de bit debido a que se incluyen menos campos) y/o se pueden definir para que tengan un campo dado interpretado de forma distinta. De este modo, cada instrucción de una ISA se expresa utilizando un formato de instrucción dado (y, si se define, en una de las plantillas de instrucción dada de dicho formato de instrucción) e incluye campos para especificar la operación y los operandos. Por ejemplo, una instrucción ADD (sumar) de ejemplo tiene un opcode específico y un formato de instrucción que incluye un campo opcode para especificar dicho opcode y los

campos operando para seleccionar los operandos (origen1/destino y origen2); y la ocurrencia de esta instrucción ADD en un flujo de instrucciones tendrá contenidos específicos en los campos operando que seleccionan los operandos específicos. Existe, se ha liberado y/o publicado un conjunto de extensiones SIMD denominadas Extensiones Avanzadas de Vectores (AVX) (AVX1 y AVX2) y utilizando el código de esquema de Extensiones de Vector (VEX) (por ejemplo, ver el Manual de Arquitecturas Intel® 64 e IA-32 para Desarrolladores de Software, octubre de 2011; y ver la Referencia de Programación de Extensiones Avanzadas de Vectores de Intel®, junio de 2011).

Formatos de instrucción de ejemplo

Los modos de realización de la(s) instrucción/instrucciones descritos en la presente solicitud se pueden materializar en diferentes formatos. Adicionalmente, más abajo se detallan los sistemas, arquitecturas y pipelines de ejemplo. Los modos de realización de la(s) instrucción/instrucciones se pueden ejecutar en dichos sistemas, arquitecturas y pipelines, pero no se limitan a los que se detallan.

A. Formato Genérico de Instrucción Adaptada para vectores

Un formato de instrucción adaptada para vectores es un formato de instrucción que se ajusta para instrucciones de vectores (por ejemplo, existen ciertos campos específicos para operaciones vectoriales). Mientras que se describen modos de realización en los que se soportan tanto operaciones vectoriales como escalares utilizando el formato de instrucción adaptada para vectores, algunos modos de realización alternativos utilizan únicamente el formato de instrucción adaptada para vectores para operaciones vectoriales.

Las Figuras 1A-1B son diagramas esquemáticos que ilustran un formato genérico de instrucción adaptada para vectores y sus plantillas de instrucción de acuerdo con los modos de realización de la invención. La Figura 1A es un diagrama esquemático que ilustra un formato genérico de instrucción adaptada para vectores y sus plantillas de instrucción de clase A de acuerdo con los modos de realización de la invención; mientras que la Figura 1B es un diagrama esquemático que ilustra el formato genérico de instrucción adaptada para vectores y sus plantillas de instrucción de clase B de acuerdo con los modos de realización de la invención. Específicamente, para un formato genérico 100 de instrucción adaptada para vectores se definen plantillas de instrucción de clase A y clase B, incluyendo ambas plantillas 105 de instrucción sin acceso a memoria y plantillas 120 de instrucción con acceso a memoria. El término genérico en el contexto del formato de instrucción adaptada para vectores se refiere a un formato de instrucción que no está asociado a ningún conjunto de instrucciones específico.

Aunque se describirán modos de realización de la invención en los que el formato de instrucción adaptada para vectores soporta lo siguiente: una longitud (o tamaño) de operando de vector de 64 bytes con anchos (o tamaño) de elementos de datos de 32 bits (4 bytes) o 64 bits (8 bytes) (y de este modo, un vector de 64 bytes consiste bien en elementos de tamaño de 16 palabras dobles o, alternativamente, elementos de tamaño de 8 palabras cuádruples); una longitud (o tamaño) de operando de vector 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 de vector 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 de vector 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); algunos modos de realización alternativos pueden soportar tamaños de operandos de vector mayores, menores o diferentes (por ejemplo, operandos de vectores de 256 bytes) con anchos de elementos de datos mayores, menores o diferentes (por ejemplo, anchos de elementos de datos de 128 bits (16 bytes).

Las plantillas de instrucción de clase A en la Figura 1A incluyen: 1) dentro de las plantillas 105 de instrucción sin acceso a memoria se muestra una plantilla 110 de instrucción de operación de tipo de control de redondeo completo sin acceso a memoria y una plantilla 115 de instrucción de operación de tipo de transformación de datos sin acceso a memoria; y 2) dentro de las plantillas 120 de instrucción con acceso a memoria se muestra una plantilla 125 de instrucción temporal y una plantilla 130 de instrucción no temporal. Las plantillas de instrucción de clase B en la Figura 1B incluyen: 1) dentro de las plantillas 105 de instrucción sin acceso a memoria se muestra una plantilla 112 de instrucción de operación de tipo de control de redondeo parcial, control de máscara de escritura, sin acceso a memoria; y una plantilla 117 de instrucción de operación de tipo VSIZE, control de máscara de escritura, sin acceso a memoria; y 2) dentro de las plantillas 120 de instrucción con acceso a memoria se muestra una plantilla 127 de instrucción de control de máscara de escritura con acceso a memoria.

El formato genérico 100 de instrucción adaptada para vectores incluye los siguientes campos listados a continuación en el orden que se ilustra en las Figuras 1A-1B.

Campo 140 de formato – un valor específico (un valor de identificador de formato de instrucción) en este campo identifica unívocamente el formato de instrucción adaptada para vectores y, de este modo, las ocurrencias de las instrucciones en el formato de instrucción adaptada para vectores en flujos de instrucciones. Como tal, este campo es opcional en el sentido de que no existe la necesidad de un conjunto de instrucciones que disponga únicamente de un formato genérico de instrucción adaptada para vectores.

Campo 142 de operación base – su contenido distingue diferentes operaciones base.

Campo 144 de índice de registro – su contenido, directamente o mediante generación de direcciones, especifica las localizaciones de los operandos origen y destino, estén en registros o en memoria. Estos incluyen un número de bits suficiente para seleccionar N registros a partir de un fichero de registros P×Q (por ejemplo, 32×512, 16×128, 32×1024, 64×1024). Aunque en un modo de realización N puede tener hasta tres registros de origen y destino, algunos modos de realización alternativos pueden soportar más o menos registros de origen y destino (por ejemplo, pueden soportar hasta dos orígenes donde una de dichos orígenes también actúa como el destino, pueden soportar hasta tres orígenes donde una de dichos orígenes también actúa como el destino, puede soportar hasta dos orígenes y un destino).

Campo modificador 146 – su contenido diferencia ocurrencias de instrucciones en el formato genérico de instrucción con vectores que especifican acceso a memoria de aquellas que no; esto es, entre plantillas 105 de instrucción sin acceso a memoria y plantillas 120 de instrucción con acceso a memoria. Las operaciones con acceso a memoria leen y/o escriben en la jerarquía de la memoria (en algunos casos especificando las direcciones origen y/o destino utilizando los valores en los registros), mientras que las operaciones sin acceso a memoria no lo hacen (por ejemplo, el origen y el destino son registros). Aunque en un modo de realización este campo también selecciona entre tres formas diferentes de realizar los cálculos de direcciones de memoria, algunos modos de realización alternativos pueden soportar más, menos o diferentes formas de realizar el cálculo de las direcciones de memoria.

Campo 150 de operación de ampliación – su contenido diferencia cuál de una variedad de diferentes operaciones realizar además de la operación base. Este campo es específico del contexto. En un modo de realización de la invención, este campo se divide en un campo 168 de clase, un campo alfa 152 y un campo beta 154. El campo 150 de operación de ampliación permite realizar grupos de operaciones comunes en una única instrucción en lugar de 2, 3 ó 4 instrucciones.

Campo 160 de escala – su contenido permite escalar el contenido del campo índice para la generación de direcciones de memoria (por ejemplo, para la generación de direcciones que utilizan 2^{escala} * índice + base).

Campo 162A de desplazamiento – su contenido se utiliza como parte de la generación de direcciones de memoria (por ejemplo, para la generación de direcciones que utilizan 2^{escala} * índice + base + desplazamiento).

Campo 162B de factor de desplazamiento (obsérvese que la yuxtaposición del campo 162A de desplazamiento directamente sobre el campo 162B de factor de desplazamiento indica que se utiliza uno o el otro) – su contenido se utiliza como parte de la generación de direcciones; especifica un factor de desplazamiento que se va a escalar mediante el tamaño de un acceso de memoria (N) donde N es el número de bytes en la memoria de acceso (por ejemplo, para la generación de direcciones que utilizan 2^{escala} * índice + base + desplazamiento escalado). Los bits de orden bajo redundantes se ignoran y, por consiguiente, el contenido del campo de factor de desplazamiento se multiplica por el tamaño total (N) de operandos de memoria con el fin de generar el desplazamiento final a utilizar en el cálculo de una dirección efectiva. El hardware del procesador determina el valor de N en tiempo de ejecución basándose en el campo 174 opcode completo (descrito más adelante en la presente solicitud) y el campo 154C de manipulación de datos. El campo 162A de desplazamiento y el campo 162B de factor de desplazamiento son opcionales en el sentido de que no se utilizan para las plantillas 105 de instrucción sin acceso a memoria y/o diferentes modos de realización pueden implementar uno o ninguno de los dos.

Campo 164 de tamaño de elemento de datos – su contenido diferencia cuál de los tamaños de elementos de datos se va a utilizar (en algunos modos de realización para todas las instrucciones; en otros modos de realización únicamente para algunas de las instrucciones). Este campo es opcional en el sentido de que no es necesario si únicamente se soporta un tamaño de elemento de datos y/o los tamaños de los elementos de datos se definen utilizando algún aspecto de los opcode.

Campo 170 de máscara de escritura – su contenido controla, en base a una posición de elemento de datos, si dicha posición del elemento de datos en el operando vector de destino refleja el resultado de la operación base y una operación de ampliación. Las plantillas de instrucción de clase A soportan máscaras de escritura de mezcla, mientras que las plantillas de instrucción de clase B soportan tanto máscaras de escritura de mezcla como de puesta a cero. Cuando se mezclan, las máscaras de vector permiten proteger de actualizaciones cualquier conjunto de elementos en el destino durante la ejecución de cualquier operación (especificada por la operación base y la operación de ampliación); en otro modo de realización, se preserva el valor antiguo de cada elemento de destino en el que el bit correspondiente de la máscara tiene un 0. Por el contrario, las máscaras de puesta a cero de vector permiten que se pongan a cero cualquier configuración de elementos en el destino durante la ejecución de cualquier operación (especificada por la operación base y la operación de ampliación); en un modo de realización, un elemento de destino recibe el valor 0 cuando el bit correspondiente de la máscara tiene el valor 0. Un subconjunto de esta funcionalidad es la característica de controlar la longitud del vector de la operación que se está realizando (esto es, el alcance de los elementos que se están modificando, desde el primero hasta el último); sin embargo, no es necesario que los elementos que se están modificando sean consecutivos. Por lo tanto, el campo 170 de máscara de escritura permite operaciones parciales de vectores, incluyendo de carga, de almacenamiento, aritméticas, lógicas, etc. Aunque se describen modos de realización de la invención en los que el contenido del campo 170 de máscara de escritura selecciona uno de una serie de registros de máscara de escritura que contiene la máscara de escritura a utilizar (y por lo tanto el contenido del campo 170 de máscara de escritura identifica indirectamente que se va a realizar un

enmascaramiento), algunos modos de realización alternativos permiten en su lugar o adicionalmente que el contenido del campo 170 de máscara de escritura especifique directamente el enmascaramiento a realizar.

Campo 172 inmediato – su contenido permite la especificación de un valor inmediato. Este campo es opcional en el sentido de que no está presente en una implementación del formato genérico adaptado para vectores que no soporta valores inmediatos y no está presente en instrucciones que no utilizan un valor inmediato.

Campo 168 de clase – su contenido distingue entre diferentes clases de instrucciones. Haciendo referencia a las Figuras 1A-B, los contenidos de este campo seleccionan entre instrucciones de clase A y de clase B. En las Figuras 1A-B, se utilizan cuadrados con esquinas redondeadas para indicar que un valor específico está presente en un campo (por ejemplo, 168A de clase A y 168B de clase B para el campo 168 de clase, respectivamente, en las Figuras 1A-B).

Plantillas de Instrucción de Clase A

En el caso de plantillas 105 de instrucción sin acceso a memoria de clase A, el campo alfa 152 se interpreta como un campo RS 152A, cuyo contenido distingue cuál de los diferentes tipos de operación de ampliación se va a realizar (por ejemplo, 152A.1 de redondeo y 152A.2 de transformación de datos se especifican respectivamente para la operación 110 de tipo de redondeo sin acceso a memoria, y las plantillas 115 de instrucción de operación sin acceso a memoria), mientras que el campo beta 154 distingue cuál de las operaciones del tipo especificado se va a realizar. En las plantillas 105 de instrucción sin acceso a memoria no se encuentran presentes ni el campo 160 de escala, ni el campo 162A de desplazamiento ni el campo 162B de escala de desplazamiento.

Plantillas de Instrucción Sin Acceso a Memoria – Operación de Tipo de Control de Redondeo Completo

En la plantilla 110 de instrucción de operación de tipo de control de redondeo completo sin acceso a memoria, el campo beta 154 se interpreta como un campo 154A de control de redondeo, cuyo(s) contenido(s) proporciona(n) un redondeo estático. Aunque en los modos de realización de la invención descritos el campo 154A de control de redondeo incluye un campo 156 de supresión de todas las excepciones de coma flotante (SAE) y un campo 158 de control de operación de redondeo, algunos modos de realización alternativos pueden soportar que se puedan codificar ambos conceptos en el mismo campo o que únicamente tengan uno o el otro de estos conceptos/campos (por ejemplo, pueden tener únicamente el campo 158 de control de la operación de redondeo).

Campo SAE 156 – su contenido distingue si se deshabilita o no la notificación de eventos de excepción; cuando el contenido del campo SAE 156 indica que la supresión está habilitada, una instrucción dada no notifica ningún tipo de indicador de excepción de coma flotante y no activa ningún gestor de excepciones de coma flotante.

Campo 158 de control de operaciones de redondeo – su contenido distingue qué operación realizar de un grupo de operaciones de redondeo (por ejemplo, Redondeo hacia arriba, Redondeo hacia abajo, Redondeo a cero y Redondeo al más próximo). Así pues, el campo 158 de control de operaciones de redondeo permite el cambio de modo de redondeo por instrucción. En un modo de realización de la invención en el que el procesador incluye un registro de control para especificar los modos de redondeo, el contenido del campo 150 de control de operaciones de redondeo invalida el valor del registro.

Plantillas de Instrucción sin Acceso a Memoria – Operación de Tipo de Transformación de Datos

En la plantilla 115 de operación de tipo de transformación de datos sin acceso a memoria, el campo beta 154 se interpreta como campo 154B de transformación de datos, cuyo contenido distingue qué número de transformaciones de datos se van a realizar (por ejemplo, sin transformación de datos, reordenación, broadcast (adaptación)).

En el caso de la plantilla 120 de instrucción con acceso a memoria de clase A, el campo alfa 152 se interpreta como un campo 152B de indicación de desalojo, cuyo contenido distingue cual de las indicaciones de desalojo se va a utilizar (en la Figura 1A, el 152B.1 temporal y el 152B.2 no temporal se especifican respectivamente para la plantilla 125 de instrucción temporal con acceso a memoria y la plantilla 130 de instrucción no temporal con acceso a memoria), mientras que el campo beta 154 se interpreta como un campo 154C de manipulación de datos, cuyo contenido distingue cuál de un número de operaciones de manipulación de datos (también denominadas primitivas) se va a realizar (por ejemplo, sin manipulación; broadcast; conversión hacia arriba de una fuente; y conversión hacia abajo de un destino). Las plantillas 120 de instrucción con acceso a memoria incluyen el campo 160 de escala y, opcionalmente, el campo 162A de desplazamiento o el campo 162B de escala de desplazamiento.

Las instrucciones de memoria de vectores realizan cargas de vectores desde y almacenamientos de vectores a la memoria, con soporte de conversión. Al igual que sucede con las instrucciones de vectores normales, las instrucciones de memoria de vectores transfieren datos desde/hacia la memoria en forma de elementos de datos, los elementos que realmente se transfieren vienen dados por los contenidos de la máscara del vector que se selecciona como máscara de escritura

Plantillas de Instrucciones con Acceso a Memoria – Temporales

Los datos temporales son datos que probablemente se vuelvan a utilizar lo suficientemente pronto como para que resulte ventajoso realizar un almacenamiento temporal. Esto es, sin embargo, una recomendación, y diferentes procesadores pueden implementarlo de diferentes formas, incluyendo el ignorar completamente la recomendación.

Plantillas de Instrucciones con Acceso a Memoria – No temporales

Los datos no temporales son datos que no es probable que se vuelvan a utilizar lo suficientemente pronto como para que resulte ventajoso realizar un almacenamiento temporal en la caché de primer nivel y se debería asignar prioridad a su desalojo. Esto es, sin embargo, una recomendación, y diferentes procesadores pueden implementarlo de diferentes formas, incluyendo el ignorar completamente la recomendación.

Plantillas de Instrucción de Clase B

En el caso de las plantillas de instrucción de clase B el campo alfa 152 se interpreta como un campo 152C de control de máscara de escritura (Z), cuyo contenido distingue si el enmascaramiento de escritura controlado por el campo 170 de máscara de escritura debería ser una mezcla o una puesta a cero.

En el caso de plantillas 105 de instrucción sin acceso a memoria de clase B, parte del campo beta 154 se interpreta como un campo RL 157A, cuyo contenido distingue cuál de los diferentes tipos de operación de ampliación se va a realizar (por ejemplo, el 157A.1 de redondeo y el 157A.2 de longitud de vector (VSIZE) se especifican respectivamente para la plantilla 112 de instrucción de operación de tipo de control de redondeo parcial, control de máscara de escritura, sin acceso a memoria y la plantilla 117 de instrucción de operación de tipo VSIZE, control de máscara de escritura, sin acceso a memoria), mientras que el resto del campo beta 154 distingue cuál de las operaciones del tipo especificado se va a realizar. En las plantillas 105 de instrucción sin acceso a memoria, no se encuentran presentes ni el campo 160 de escala, ni el campo 162A de desplazamiento ni el campo 162B de escala de desplazamiento.

En la plantilla 110 de instrucción de operación de tipo de control de redondeo completo, sin acceso a memoria, el resto del campo beta 154 se interpreta como un campo 159A de operación de redondeo y se deshabilita la notificación de eventos de excepción (una instrucción dada no notifica ningún tipo de indicador de excepción de coma flotante y no activa ningún gestor de excepciones de coma flotante).

Campo 159A de control de operaciones de redondeo – al igual que en el caso del campo 158 de control de operaciones de redondeo, su contenido distingue cuál de un grupo de operaciones de redondeo realizar (por ejemplo, Redondeo hacia arriba, Redondeo hacia abajo, Redondeo a cero y Redondeo al más próximo). Así pues, el campo 159A de control de operaciones de redondeo permite el cambio del modo de redondeo para cada instrucción. En un modo de realización de la invención en la que el procesador incluye un registro de control para especificar los modos de redondeo, el contenido del campo 150 de control de operaciones de redondeo invalida el valor del registro.

En la plantilla 117 de instrucción de operación de tipo VSIZE, control de máscara de escritura, sin acceso a memoria, el resto del campo beta 154 se interpreta como un campo 159B de longitud de vector, cuyo contenido distingue la longitud del vector de datos sobre la que se va a actuar (por ejemplo, 128, 256 o 512 bytes).

En el caso de una plantilla 120 de instrucción con acceso a memoria de clase B, parte del campo beta 154 se interpreta como un campo 157B de broadcast, cuyo contenido distingue si se va a realizar o no una operación de manipulación de datos de tipo broadcast, mientras que el resto del campo beta 154 se interpreta como el campo 159B de longitud de vector. Las plantillas 120 de instrucción con acceso a memoria incluyen el campo 160 de escala y, opcionalmente, el campo 162A de desplazamiento o el campo 162B de escala de desplazamiento.

Teniendo en cuenta el formato genérico 100 de instrucción adaptada para vectores, se muestra un campo opcode completo 174 que incluye el campo 140 de formato, el campo 142 de operación base y el campo 164 de tamaño de elemento de datos. Aunque se muestra un modo de realización en el que el campo opcode completo 174 incluye todos estos campos, el campo opcode completo 174 no incluye todos los campos en los modos de realización que no los soportan a todos. El campo opcode completo 174 proporciona un código de operación (opcode).

En el formato genérico de instrucción adaptada para vectores el campo 150 de operación de ampliación, el campo 164 de tamaño de elemento de datos, y el campo 170 de máscara de escritura permiten que se puedan especificar estas características para cada instrucción.

La combinación del campo de máscara de escritura y el campo de tamaño de elemento de datos da lugar a tipos de instrucciones en las que permiten que se aplique la máscara en función de diferentes tamaños del elemento de datos.

Las diferentes plantillas de instrucción que se encuentran en la clase A y la clase B son beneficiosas en diferentes situaciones. En algunos modos de realización de la invención, diferentes procesadores o diferentes núcleos dentro de un procesador pueden soportar únicamente la clase A, únicamente la clase B, o ambas clases. Por ejemplo, un núcleo de propósito general de alto rendimiento con ejecución fuera de orden puede soportar únicamente la clase B, un núcleo destinado principalmente para cálculo de gráficos y/o científico (rendimiento) puede soportar únicamente la clase A, y un núcleo destinado a ambos puede soportar ambos (por supuesto, dentro del alcance de la invención se encuentra

un núcleo que tenga alguna combinación de plantillas e instrucciones de ambas clases pero no todas las plantillas e instrucciones de ambas clases). Además, un único procesador puede incluir múltiples núcleos, todos los cuales soportan la misma clase o en la que diferentes núcleos soportan diferentes clases. Por ejemplo, en un procesador con núcleos independientes de gráficos y de propósito general, uno de los núcleos de gráficos destinado principalmente para cálculo de gráficos y/o científico puede soportar únicamente la clase A, mientras que uno o más de los núcleos de propósito general pueden ser núcleos de propósito general de alto rendimiento con ejecución fuera de orden y un renombrado de registros destinado para cálculo de propósito general que soporte únicamente la clase B. Otro procesador que no tenga un núcleo de gráficos independiente, puede incluir uno o más núcleos de propósito general con ejecución en orden o fuera de orden que soporten tanto la clase A como la clase B. Por supuesto, en diferentes modos de realización de la invención, las características de una clase también se pueden implementar en la otra clase. Los programas escritos en un lenguaje de alto nivel se traducirían (por ejemplo, compilado al vuelo o compilado estáticamente) en una variedad de diferentes formas ejecutables, que incluyen: 1) una forma con únicamente instrucciones de la(s) clase(s) soportada(s) por el procesador de destino para su ejecución; o 2) una forma con rutinas alternativas escritas utilizando diferentes combinaciones de instrucciones de todas las clases y con un código de flujo de control que selecciona las rutinas a ejecutar en función de las instrucciones soportadas por el procesador que esté ejecutando el código en ese momento.

B. Formato Específico Ejemplar de Instrucción Adaptada para Vectores

La Figura 2 es un diagrama esquemático que ilustra un formato específico ejemplar de instrucción adaptada para vectores de acuerdo con algunos modos de realización de la invención. La Figura 2 muestra un formato específico 200 de instrucción adaptada para vectores que es específico en el sentido de que especifica la localización, tamaño, interpretación y orden de los campos, así como los valores para algunos de dichos campos. El formato específico 200 de instrucción adaptada para vectores se puede utilizar para ampliar el conjunto de instrucciones x86 y, por lo tanto, algunos de los campos son similares o iguales a los utilizados en el conjunto de instrucciones x86 existente y su extensión (por ejemplo, AVX). Este formato permanece consistente con el campo de codificación de prefijo, el campo de byte de opcode real, el campo R/M MOD, 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 1 que se corresponden con los campos de la Figura 2.

Se debería entender que, aunque los modos de realización de la invención se describen haciendo referencia al formato específico 200 de instrucción adaptada para vectores en el contexto del formato genérico 100 de instrucción adaptada para vectores a efectos ilustrativos, la invención no está limitada al formato específico 200 de instrucción adaptada para vectores excepto donde se reivindique. Por ejemplo, el formato genérico 100 de instrucción adaptada para vectores contempla una variedad de tamaños posibles para los distintos campos, mientras que el formato específico 200 de instrucción adaptada para vectores se muestra con campos de tamaños específicos. A modo de ejemplo específico, mientras que en el formato específico 200 de instrucción adaptada para vectores se ilustra el campo 164 de tamaño de elemento de datos como un campo de un bit, la invención no está limitada por ese motivo (esto es, el formato genérico 100 de instrucción adaptada para vectores contempla otros tamaños del campo 164 de tamaño de elemento de datos).

El formato genérico 100 de instrucción adaptada para vectores incluye los siguientes campos listados a continuación en el orden ilustrado en la Figura 2A.

Prefijo EVEX (Bytes 0-3) 202 – se codifica en forma de cuatro bytes.

Campo 140 de formato (EVEX Byte 0, bits [7:0]) – el primer byte (EVEX Byte 0) es el campo 140 de formato y contiene 0x62 (el valor único utilizado para diferenciar el formato de instrucción adaptada para vectores en un modo de realización de la invención).

Los bytes segundo a cuarto (EVEX Bytes 1-3) incluye una serie de campos de bit que proporcionan una capacidad específica.

Campo REX 205 (EVEX Byte 1, bits [7-5]) consiste en un campo de bit EVEX.R (EVEX Byte 1, bit [7] – R), un campo de bit EVEX.X (EVEX Byte 1, bit [6] – X) y un campo de bit EVEX.B (EVEX Byte 1, bit [5] – B). Los campos de bit EVEX.R, EVEX.X y EVEX.B proporcionan la misma funcionalidad que los campos de bit VEX correspondientes y se codifican utilizando la forma complemento a 1, esto es, 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 tal como es conocido en la técnica (rrr, xxx y bbb) de modo que se pueden formar Rrrr, Xxxx y Bbbb añadiendo EVEX.R, EVEX.X y EVEX.B.

Campo REX' 110 – esta es la primera parte del campo REX' 110 y es el campo de bit EVEX.R' (EVEX Byte 1, bit [4] – R') que se utiliza para codificar bien los 16 superiores o bien los 16 inferiores del conjunto extendido de 32 registros. En un modo de realización de la invención, este bit, junto con otros tal como se indica más abajo, se almacena en formato de bit invertido para distinguir (en el bien conocido modo x86 de 32 bits) de la instrucción BOUND, cuyo byte opcode real es 62, pero en el campo R/M MOD (descrito más abajo) no acepta el valor de 11 en el campo MOD; algunos modos de realización alternativos de la invención no almacenan esto y los otros bits indicados más abajo en

formato invertido. Se utiliza el valor 1 para codificar los 16 registros inferiores. En otras palabras, R'Rrrr se forma combinando EVEX.R', EVEX.R, y los otros RRR de otros campos.

Campo 215 de mapeo de opcode (EVEX Byte 1, bits [3:0] – mmmm) – su contenido codifica un byte opcode de cabecera implicado (0F, 0F 38 ó 0F 3).

Campo 164 de tamaño de elemento de datos (EVEX Byte 2, bit [7] – W) – se representa mediante la notación EVEX.W. EVEX.W se utiliza para definir la granularidad (tamaño) del tipo de datos (elementos de datos de 32 bits o elementos de datos de 64 bits).

EVEX.vvvv 220 (EVEX Byte 2, bits [6:3]-vvvv) – el papel de EVEX.vvvv puede incluir lo siguiente: 1) EVEX.vvvv codifica el primer operando de registro origen, especificado en forma invertida (complemento a 1) y es válido para instrucciones con dos o más operandos origen; 2) EVEX.vvvv codifica el operando de registro destino, especificado en forma complemento a 1 para ciertos desplazamientos de vector; o 3) EVEX.vvvv no codifica ningún operando, el campo es reservado y debería contener 1111b. Así pues, el campo 220 EVEX.vvvv codifica los 4 bits de orden bajo del primer especificador de registro origen almacenado en forma invertida (complemento a 1). Dependiendo de la instrucción, se utiliza un campo bit EVEX extra diferente para ampliar el tamaño especificador a 32 registros.

Campo 168 de clase EVEX.U (EVEX byte 2, bit [2]-U) – si EVEX.U = 0, indica clase A o EVEX.U0; si EVEX.U = 1 indica clase B o EVEX.U1.

Campo 225 de codificación de prefijo (EVEX byte 2, bits [1:0]-pp) – proporciona bits adicionales en el campo de operación base. Además de proporcionar soporte para las instrucciones SSE antiguas 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 únicamente 2 bits). En un modo de realización, con el fin de soportar las instrucciones SSE antiguas que utilizan un prefijo SIMD (66H, F2H, F3H) tanto en el formato antiguo como en el formato de prefijo EVEX, estos prefijos SIMD antiguos se codifican en el campo de codificación de prefijo SIMD; y en tiempo de ejecución se expanden en el prefijo SIMD antiguo antes de proporcionarse al PLA del decodificador (de forma que el PLA puede ejecutar tanto el formato antiguo como el EVEX de estas instrucciones antiguas sin modificación). Aunque instrucciones más modernas podrían utilizar directamente el contenido del campo de codificación del prefijo EVEX como una extensión del opcode, ciertos modos de realización se amplían de forma parecida por consistencia, pero permiten que estos prefijos SIMD antiguos especifiquen diferentes significados. Un modo de realización alternativo puede rediseñar el PLA para que soporte las codificaciones de prefijo SIMD de 2 bits y, de este modo, no requiere la ampliación.

Campo alfa 152 (EVEX byte 3, bit[7] – EH; también denominado EVEX.EH, EVEX.rs, EVEX.RL, EVEX.control de máscara de escritura, y EVEX.N; también ilustrado como α) – tal como se ha descrito previamente, este campo depende del contexto.

Campo beta 154 (EVEX byte 3, bits [6:4]-SSS, también denominado EVEX.s_{2-0}, EVEX.r_{2-0}, EVEX.rr1, EVEX.LL0, EVEX.LLB; también ilustrado como $\beta\beta\beta$) – tal como se ha descrito previamente, este campo depende del contexto.

Campo REX' 110 – esto es el resto del campo REX' y es el campo bit EVEX.V' (EVEX Byte 3, bit[3] – V') que se puede utilizar para codificar los 16 superiores o los 16 inferiores del conjunto extendido de 32 registros. Este bit se almacena en formato de bit invertido. Se utiliza el valor 1 para codificar los 16 registros inferiores. En otras palabras, V'VVVV se forma combinando EVEX.V' y EVEX.vvvv.

Campo 170 de máscara de escritura (EVEX byte 3, bits [2:0]-kkk) – su contenido especifica el índice de un registro en los registros de máscara de escritura tal como se ha descrito previamente. En un modo de realización de la invención, el valor específico EVEX.kkk=000 tiene un comportamiento especial que implica que no se utiliza ninguna máscara de escritura para la instrucción concreta (esto se puede implementar de diferentes formas incluyendo la utilización de una máscara de escritura cableada a todo unos o un hardware que evita el hardware de máscara).

Campo 230 de Opcode real (Byte 4) también se denomina byte opcode. En este campo se especifica parte del opcode.

Campo R/M MOD 240 (Byte 5) incluye el campo MOD 242, el campo Reg 244 y el campo R/M 246. Tal como se ha descrito previamente, el contenido del campo MOD 242 distingue entre operaciones con acceso a memoria y sin acceso a memoria. La función del campo Reg 244 se puede resumir en dos situaciones: codificar el operando de registro de destino o el operando de registro de origen, o tratarse como una extensión del opcode y no utilizarse para codificar ningún operando de instrucción. La función del campo R/M 246 puede incluir lo siguiente: codificar el operando de instrucción que referencia una dirección de memoria o codificar el operando de registro de destino o un operando de registro de origen.

Byte Escala, Índice, Base (SIB) (Byte 6) – Tal como se ha descrito previamente, el contenido del campo 150 de escala se utiliza para la generación de una dirección de memoria. SIB.xxx 254 y SIB.bbb 256 – los contenidos de estos campos se han mencionado previamente con respecto a los índices de registro Xxxx y Bbbb.

Campo 162A de desplazamiento (Bytes 7-10) – cuando el campo MOD 242 contiene 10, los bytes 7-10 son el campo 162A de desplazamiento, y funciona del mismo modo que el desplazamiento de 32 bits antiguo (disp32) y funciona en granularidad de byte.

Campo 162B de factor de desplazamiento (Byte 7) – cuando el campo MOD 242 contiene 01, el byte 7 es el campo 162B de factor de desplazamiento. La localización de este campo es la misma que la del desplazamiento de 8 bits (disp8) del conjunto de instrucciones x86 antiguo, el cual funciona en granularidad de byte. Como disp8 es de signo extendido, únicamente pude direccionar desplazamientos de bytes entre -128 y 127; en términos de líneas de caché de 64 bytes, disp8 utiliza 8 bits que solo pueden adoptar realmente 4 valores útiles: -128, -64, 0 y 64; como en general se necesita un rango mayor, se utiliza disp32; sin embargo, disp32 requiere 4 bytes. En comparación con disp8 y disp32, el campo 162B de factor de desplazamiento es una reinterpretación de disp8; cuando se utiliza el campo 162B de factor de desplazamiento, se determina el desplazamiento real mediante 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 la instrucción (un único byte utilizado para el desplazamiento, pero con un rango mucho mayor). Dicho desplazamiento comprimido se basa en la asunción de que el desplazamiento efectivo es múltiplo de la granularidad del acceso a memoria y, por lo tanto, no es necesario codificar los bits de orden bajo redundantes del desplazamiento de dirección. En otras palabras, el campo 162B de factor de desplazamiento sustituye el desplazamiento de 8 bits del conjunto de instrucciones x86 antiguo. Así pues, el campo 162B de factor de desplazamiento se codifica del mismo modo que el 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 pasa a disp8*N. En otras palabras, no existen cambios en las reglas de codificación ni en las longitudes de codificación sino únicamente en la interpretación del valor de desplazamiento mediante hardware (que necesita escalar el desplazamiento por el tamaño del operando de memoria para obtener un desplazamiento de dirección en términos de byte).

El campo 172 inmediato opera tal como se ha descrito previamente.

Campo Opcode Completo

La Figura 2B es un diagrama esquemático que ilustra los campos del formato específico 200 de instrucción adaptada para vectores que forma el campo opcode completo 174 de acuerdo con un modo de realización de la invención. Específicamente, el campo opcode completo 174 incluye el campo 140 de formato, el campo 142 de operación base, y el campo 164 de tamaño (W) de elemento de datos. El campo 142 de operación base incluye el campo 225 de codificación de prefijo, el campo 215 de mapeo de opcode y el campo 230 de opcode real.

Campo de Índice de Registro

La Figura 2C es un diagrama esquemático que ilustra los campos del formato específico 200 de instrucción adaptada para vectores que constituye el campo 144 de índice de registro de acuerdo con un modo de realización de la invención. Específicamente, el campo 144 de índice de registro incluye el campo REX 205, el campo REX' 210, el campo MODR/M.reg 244, el campo MODR/M.r/m 246, el campo VVVV 220, el campo xxx 254 y el campo bbb 256.

Campo de Operación de ampliación

La Figura 2D es un diagrama esquemático que ilustra los campos del formato específico 200 de instrucción adaptada para vectores que constituyen el campo 150 de operación de ampliación de acuerdo con un modo de realización de la invención. Cuando el campo 168 de clase (U) contiene 0, significa EVEX.U0 (168A clase A); cuando contiene 1, significa EVEX.U1 (168B clase B). Cuando U=0 y el campo MOD 242 contiene 11 (indicando una operación sin acceso a memoria), el campo alfa 152 (EVEX byte 3, bit [7] – EH) se interpreta como el campo rs 152A. Cuando el campo rs 152A contiene 1 (152A.1 de redondeo), el campo beta 154 (EVEX byte 3, bits [6:4] – SSS) se interpreta como el campo 154A de control de redondeo. El campo 154A de control de redondeo incluye el campo SAE 156 de un bit y un campo 158 de operación de redondeo de dos bits. Cuando el campo rs 152A contiene un 0 (152A.2 de transformación de datos), el campo beta 154 (EVEX byte 3, bits [6:4] – SSS) se interpreta como un campo 154B de transformación de datos de tres bits. Cuando U=0 y el campo MOD 242 contiene 00, 01 ó 10 (indicando una operación con acceso a memoria), el campo alfa 152 (EVEX byte 3, bit [7] – EH) se interpreta como el campo 152B de indicación de desalojo (EH) y el campo beta 154 (EVEX byte 3, bits [6:4] – SSS) se interpreta como un campo 154C de manipulación de datos de tres bits.

Cuando U=1, el campo alfa 152 (EVEX byte 3, bit [7] – EH) se interpreta como el campo 152C de control de máscara de escritura (Z). Cuando U=1 y el campo MOD 242 contiene 11 (indicando una operación sin acceso a memoria), parte del campo beta 154 (EVEX byte 3, bit [4] – S_0) se interpreta como el campo RL 157A; cuando contiene 1 (157A.1 de redondeo) el resto del campo beta 154 (EVEX byte 3, bit [6-5] – S_{2-1}) se interpreta como el campo 159A de operación de redondeo, mientras que cuando el campo RL 157A contiene un 0 (157.A2 VSIZE) el resto del campo beta 154 (EVEX byte 3, bit [6-5] – S_{2-1}) se interpreta como el campo 159B de longitud de vector (EVEX byte 3, bit [6-5] – L_{1-0}). Cuando U=1 y el campo MOD 242 contiene 00, 01 ó 10 (indicando una operación con acceso a memoria), el campo beta 154 (EVEX byte 3, bits [6:4] – SSS) se interpreta como el campo 159B de longitud de vector (EVEX byte 3, bit [6-5] – L_{1-0}) y el campo 157B de broadcast (EVEX byte 3, bit [4] – B).

C. Arquitectura de Registros de Ejemplo

La Figura 3 es un diagrama esquemático de una arquitectura 300 de registros de acuerdo con un modo de realización de la invención. En el modo de realización ilustrado, existen 32 registros 310 de vectores que tienen un tamaño de 512 bits; estos registros se denominan zmm0 a zmm31. Los 256 bits de orden inferior de los 16 registros zmm se superponen en los registros ymm0-16. Los 128 bits de orden inferior de los 16 registros zmm inferiores (los 128 bits de orden inferior de los registros ymm) se superponen en los registros xmm0-15. El formato específico 200 de instrucción adaptada para vectores opera sobre este fichero de registros solapados tal como se ilustra en las siguientes tablas.

Longitud Ajustable De Vector	Clase	Operaciones	Registros
Plantillas de instrucción que no incluyen el campo 159B de longitud de vector	A (Figura 1A; U=0)	110, 115, 125, 130	Registros zmm (la longitud de vector es 64 bytes)
	B (Figura 1B; U=1)	112	Registros zmm (la longitud de vector es 64 bytes)
Plantillas de instrucción que incluyen el campo 159B de longitud de vector	B (Figura 1B; U=1)	117, 127	Registros zmm, ymm, o xmm (la longitud de vector es 64 bytes, 32 bytes, o 16 bytes) dependiendo del campo 159B de longitud de vector

En otras palabras, el campo 159B de longitud de vector selecciona entre una longitud máxima y una o más longitudes distintas más cortas, donde cada una de dichas longitudes más cortas es la mitad de la longitud de la longitud anterior; y las plantillas de instrucción sin el campo 159B de longitud de vector operan sobre la longitud de vector máxima. Además, en un modo de realización, las plantillas de instrucción de clase B del formato específico 200 de instrucción adaptada para vectores operan sobre datos de coma flotante de precisión simple/doble empaquetados o escalares y datos enteros empaquetados o escalares. Las operaciones escalares son operaciones realizadas sobre la posición de elementos de datos de menor orden en un registro zmm/ymm/xmm; las posiciones de elementos de datos de orden superior bien se dejan igual que antes de la instrucción o se ponen a cero, dependiendo del modo de realización.

Registros 315 de máscara de escritura – en el modo de realización ilustrado, existen 8 registros de máscara de escritura (k0 a k7), cada uno de 64 bits de tamaño. En un modo de realización alternativo, los registros 315 de máscara de escritura tienen un tamaño de 16 bits. Tal como se ha descrito previamente, en un modo de realización de la invención, el registro k0 de máscara de vector no se puede utilizar como máscara de escritura; cuando se utiliza la codificación que normalmente indicaría k0 para una máscara de escritura, se selecciona la máscara de escritura cableada 0xFFFF, deshabilitando efectivamente el enmascaramiento de escritura para dicha instrucción.

Registros 325 de propósito general – en el modo de realización ilustrado, existen dieciséis registros de propósito general de 64 bits que se utilizan junto con los modos de direccionamiento de x86 existentes para direccionar operandos en memoria. Estos registros se referencian mediante los nombres RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP y R8 a R15.

El fichero 345 de registros de la pila de coma flotante escalar (pila x87), abreviado en el fichero 350 de registros planos de enteros empaquetados MMX – en el modo de realización ilustrado, la pila x87 es una pila de ocho elementos utilizada para realizar operaciones escalares de coma flotante sobre datos de coma flotante de 32/64/80 bits utilizando la extensión del conjunto de instrucciones x87; mientras que los registros MMX se utilizan para realizar operaciones sobre datos enteros empaquetados de 64 bits, así como para mantener operandos para algunas operaciones realizadas entre los registros MMX y XMM.

Algunos modos de realización alternativos de la invención pueden utilizar registros más grandes o más pequeños. Adicionalmente, algunos modos de realización alternativos de la invención pueden utilizar más, menos o diferentes ficheros de registros y registros.

D. Arquitecturas del Núcleo, Procesadores y Arquitecturas de Ordenadores de Ejemplo

Se pueden implementar núcleos de procesadores de diferentes formas, para diferentes propósitos y en diferentes procesadores. Por ejemplo, las implementaciones de dichos núcleos pueden incluir: 1) un núcleo de ejecución en orden de propósito general para cálculos de propósito general; 2) un núcleo de ejecución fuera de orden de propósito general de alto rendimiento destinado a cálculos de propósito general; y 3) un núcleo de propósito especial destinado principalmente para cálculos de gráficos y/o científicos (rendimiento). Las implementaciones de los diferentes procesadores pueden incluir: 1) una CPU que incluya uno o más núcleos de ejecución en orden de propósito general destinados para cálculos de propósito general y/o uno o más núcleos de ejecución fuera de orden de propósito general destinado a cálculos de propósito general; y 2) un coprocesador que incluye uno o más núcleos de propósito especial destinados principalmente para cálculos de gráficos y/o científicos (rendimiento). Dichos diferentes procesadores conducen a diferentes arquitecturas del sistema del ordenador, que pueden incluir: 1) el coprocesador en un chip independiente de la CPU; 2) el coprocesador en una oblea independiente en el mismo paquete que la CPU; 3) el coprocesador en la misma oblea que la CPU (en cuyo caso, dicho coprocesador se denomina algunas veces lógica de propósito especial como, por ejemplo, lógica de gráficos y/o científica integrados (rendimiento), o núcleos de propósito especial); y 4) un sistema en chip que puede incluir en la misma oblea la CPU descrita (algunas veces denominada núcleo(s) de aplicaciones o procesador(es) de aplicaciones), el coprocesador descrito más arriba y funcionalidad adicional. A continuación, se describen arquitecturas de núcleos de ejemplo, seguido por arquitecturas de procesadores y ordenadores de ejemplo.

La Figura 4A es un diagrama esquemático que ilustra tanto un pipeline de ejecución en orden de ejemplo como un pipeline de emisión/ejecución fuera de orden con renombrado de registros de ejemplo de acuerdo con algunos modos de realización de la invención. La Figura 4B es un diagrama esquemático que ilustra tanto un modo de realización de ejemplo de un núcleo de arquitectura con ejecución en orden como un núcleo de arquitectura de emisión/ejecución fuera de orden con renombrado de registros para ser incluidos en un procesador de acuerdo con los modos de realización de la invención. Las cajas con líneas continuas en las Figuras 4A-B ilustran un pipeline de ejecución en orden y un núcleo de ejecución en orden, mientras que la adición opcional de las cajas con líneas discontinuas ilustra un pipeline y un núcleo de emisión/ejecución fuera de orden con renombrado de registros. Dado que el aspecto de ejecución en orden es un subconjunto del aspecto de ejecución fuera de orden, se describirá el aspecto fuera de orden.

En la Figura 4A, un flujo 400 de procesador incluye una etapa 402 de recuperación, una etapa 404 de decodificación de longitud, una etapa 406 de decodificación, una etapa 408 de asignación, una etapa 410 de renombrado, una etapa 412 de planificación (también denominada de entrega o emisión), una etapa 414 de lectura de registro/lectura de memoria, una etapa 416 de ejecución, una etapa 418 de devolución/escritura en memoria, una etapa 422 de gestión de excepciones y una etapa 424 de confirmación.

La Figura 4B muestra un núcleo 490 de procesador que incluye una unidad frontal 430 acoplada a una unidad 450 de motor de ejecución, y ambas están acopladas a una unidad 470 de memoria. El núcleo 490 puede ser un núcleo de computación con un conjunto reducido de instrucciones (RISC), un núcleo de computación con un conjunto complejo de instrucciones (CISC), un núcleo con palabras de instrucción muy largas (VLIW), o un tipo de núcleo híbrido o alternativo. Como todavía una opción adicional, el núcleo 490 puede ser un núcleo de propósito especial como, por ejemplo, un núcleo de red o comunicación, un módulo de compresión, un núcleo coprocesador, un núcleo de unidad de procesamiento de gráficos de computación de propósito general (GPGPU), un núcleo de gráficos, etc.

La unidad frontal 430 incluye una unidad 432 de predicción de salto acoplada a una unidad 434 de caché de instrucciones, la cual está acoplada a un buffer (memoria intermedia) lookaside 436 de traducción adelantada (TLB) de instrucciones, el cual está acoplado a una unidad 438 de recuperación de instrucciones, la cual está acoplada a una unidad 440 de decodificación. La unidad 440 de decodificación (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, las cuales se decodifican a partir de, o reflejan de otra forma, o se derivan de, las instrucciones originales. La unidad 440 de decodificación se puede implementar utilizando varios mecanismos diferentes. Algunos ejemplos de mecanismos apropiados incluyen, pero no se limitan a, tablas de búsqueda, implementaciones hardware, tablas de lógica programable (PLA), memorias de solo lectura (ROM) de microcódigo, etc. En un modo de realización, el núcleo 490 incluye una ROM de microcódigo u otro medio que almacena microcódigo para ciertas macroinstrucciones (por ejemplo, en una unidad 440 de decodificación o de otro modo dentro de la unidad frontal 430). La unidad 440 de decodificación está acoplada a una unidad 452 de renombrado/asignación en la unidad 450 de motor de ejecución.

La unidad 450 de motor de ejecución incluye la unidad 452 de renombrado/asignación acoplada a una unidad 454 de descarte y un conjunto de una o más unidades 456 planificadoras. La(s) unidad(es) 456 planificadora(s) representa(n) cualquier número de diferentes planificadores, incluyendo estaciones de reserva, ventana central de instrucciones, etc. La(s) unidad(es) 456 planificadora(s) está(n) acoplada(s) a la(s) unidad(es) 458 de fichero(s) de registros físicos. Cada una de las unidades 458 de fichero(s) de registros físicos representa uno o más ficheros de registros físicos, cada uno de los cuales almacena uno o más tipos de datos diferentes como, por ejemplo, un entero escalar, una coma flotante escalar, un entero empaquetado, una coma flotante empaquetada, un vector de enteros, un vector de comas flotantes, un estado (por ejemplo, un puntero de instrucciones que es la dirección de la siguiente instrucción a ejecutar),

etc. En un modo de realización, la unidad 458 de fichero(s) de registros físicos comprende una unidad de registros de vectores, una unidad de registros de máscaras de escritura y una unidad de registros de escalares. Estas unidades de registros pueden proporcionar registros de vectores de arquitectura, registros de máscaras de vectores y registros de propósito general. La(s) unidad(es) 458 de fichero(s) de registros físicos se puede(n) superponer mediante la unidad 454 de descarte para ilustrar varios modos en los que se puede implementar el renombrado de registros y la ejecución fuera de orden (por ejemplo, utilizando un/varios buffer(es) de reordenación y un/varios fichero(s) de registros de descarte; utilizando un/varios fichero(s) de futuro, un/varios buffer(es) históricos, y un/varios fichero(s) de registros de descarte; utilizando mapas de registros y un conjunto de registros; etc.). La unidad 454 de descarte y la unidad 458 de fichero(s) de registros físicos están acoplada al/a los clúster(es) (grupo) 460 de ejecución. El/los clúster(es) 460 de ejecución incluye(n) un conjunto de una o más unidades 462 de ejecución y un conjunto de una o más unidades 464 de acceso a memoria. Las unidades 462 de ejecución pueden realizar varias operaciones (por ejemplo, desplazamientos, adición, sustracción, multiplicación) y sobre varios tipos de datos (por ejemplo, coma flotante escalar, entero empaquetado, coma flotante empaquetada, vector de enteros, vector de comas flotantes). Aunque algunos modos de realización pueden incluir un número de unidades de ejecución dedicadas a funciones o conjuntos de funciones específicas, otros modos de realización pueden incluir únicamente una unidad de ejecución o múltiples unidades de ejecución que realizan todas las funciones. La(s) unidad(es) 456 planificadora(s), la(s) unidad(es) 458 de fichero(s) de registros físicos, y el/los clúster(es) 460 de ejecución se muestran como posiblemente varios porque ciertos modos de realización crean pipelines independientes para ciertos tipos de datos/operaciones (por ejemplo, un pipeline de enteros escalares, un pipeline de comas flotantes escalares/enteros escalares/comas flotantes empaquetadas/vectores de enteros/vectores de coma flotante, y/o un pipeline de acceso a memoria cada uno de los cuales tiene su propia unidad planificadora, unidad de fichero(s) de registros físicos y/o clúster de ejecución – y en el caso de un pipeline de acceso a memoria independiente, se implementan ciertos modos de realización de modo que únicamente el clúster de ejecución de este pipeline tiene la(s) unidad(es) 464 de acceso a memoria). También se debería entender que cuando se utilizan pipelines independientes, uno o más de estos pipelines pueden ser de emisión/ejecución fuera de orden y el resto en orden.

El conjunto de unidades 464 de acceso a memoria está acoplado a la unidad 470 de memoria, la cual incluye una unidad TLB 472 acoplada a una unidad 474 de caché de datos acoplada a una unidad 476 de caché de nivel 2 (L2). En un modo de realización de ejemplo, las unidades 464 de acceso a memoria pueden incluir una unidad de carga, una unidad de almacenamiento de direcciones y una unidad de almacenamiento de datos, cada una de las cuales está acoplada a la unidad TLB 472 en la unidad 470 de memoria. La unidad 434 de caché de instrucciones está acoplada, además, a una unidad 476 de caché de nivel 2 (L2) en la unidad 470 de memoria. La unidad 476 de caché L2 está acoplada a una o más cachés de otros niveles y eventualmente a la memoria principal.

A modo de ejemplo, la arquitectura de núcleo emisión/ejecución fuera de orden con renombrado de registros de ejemplo puede implementar el pipeline 400 del siguiente modo: 1) la recuperación 438 de instrucciones ejecuta las etapas 402 y 404 de recuperación y decodificación de longitud; 2) la unidad 440 de decodificación ejecuta la etapa 406 de decodificación; 3) la unidad 452 de renombrado/asignación ejecuta la etapa 408 de asignación y la etapa 410 de renombrado; 4) la(s) unidad(es) 456 de planificación ejecutan la etapa 412 de planificación; 5) la(s) unidad(es) 458 de fichero(s) de registros físicos y la unidad 470 de memoria ejecutan la etapa 414 de lectura de registros/lectura de memoria; el clúster 460 de ejecución ejecuta la etapa 416 de ejecución; 6) la unidad 470 de memoria y la(s) unidad(es) 458 de fichero(s) de registros físicos ejecutan la etapa 418 de devolución/escritura en memoria; 7) varias unidades están implicadas en la etapa 422 de gestión de excepciones; y 8) la unidad 454 de descarte y la(s) unidad(es) 458 de fichero(s) de registros físicos ejecutan la etapa 424 de confirmación.

El núcleo 490 puede soportar uno o más conjuntos de instrucciones (por ejemplo, el conjunto de instrucciones x86 (con algunas extensiones que se han añadido en versiones posteriores); el conjunto de instrucciones MIPS de MIPS Technologies de Sunnyvale, CA; el conjunto de instrucciones ARM (con extensiones opcionales adicionales como, por ejemplo, NEON) de ARM Holdings de Sunnyvale, CA), incluyendo las instrucciones descritas en la presente solicitud. En un modo de realización, el núcleo 490 incluye lógica para soportar una extensión del conjunto de instrucciones de datos empaquetados (por ejemplo, AVX1, AVX2), permitiendo de este modo que las operaciones utilizadas por muchas aplicaciones multimedia se ejecuten utilizando datos empaquetados.

Se debería entender que el núcleo puede soportar múltiples hilos (ejecución de dos o más conjuntos de operaciones o hilos en paralelo), y puede hacerlo de diferentes modos incluyendo múltiples hilos por ranuras de tiempo, múltiples hilos simultáneos (donde un único núcleo físico proporciona un núcleo lógico para cada uno de los hilos que se ejecutan simultáneamente con múltiples hilos), o una combinación de los mismos (por ejemplo, recuperación en ranuras de tiempo y decodificación y ejecución simultánea con múltiples hilos a partir de ese momento como, por ejemplo, en la tecnología Hyperthreading de Intel®).

Aunque el renombrado de registros se describe en el contexto de una ejecución fuera de orden, se debería entender que el renombrado de registros se puede utilizar en una arquitectura de ejecución en orden. Aunque el modo de realización del procesador ilustrado también incluye unidades 434/474 de caché de instrucciones y datos y una unidad 476 de caché L2 compartida, algunos modos de realización alternativos 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 algunos modos de realización, el sistema puede incluir una combinación de caché interna y caché externa

que es externa al núcleo y/o procesador. Alternativamente, todas las cachés pueden ser externas al núcleo y/o al procesador.

Las Figuras 5A-B ilustran un diagrama esquemático de una arquitectura de núcleo de ejecución en orden de ejemplo más específica, cuyo núcleo sería uno de varios bloques lógicos (incluyendo otros núcleos del mismo tipo y/o diferentes tipos) en un chip. Los bloques lógicos se comunican a través de una red de interconexión de alto ancho de banda (por ejemplo, una red en anillo) con alguna lógica de función fija, interfaces de memoria de E/S, y otra lógica de E/S necesaria, en función de la aplicación.

La Figura 5A es un diagrama esquemático de un único procesador, junto con su conexión a la red 502 de interconexión sobre la oblea y con su subconjunto local de la cache 504 de Nivel 2 (L2), de acuerdo con los modos de realización de la invención. En un modo de realización, un decodificador 500 de instrucciones soporta el conjunto de instrucciones x86 con una extensión del conjunto de instrucciones de datos empaquetados. Una cache L1 506 permite accesos de baja latencia a la memoria caché en las unidades de escalares y de vectores. Mientras que en un modo de realización (para simplificar el diseño), una unidad 508 de escalares y una unidad 510 de vectores utilizan conjuntos de registros independientes (respectivamente, los registros 512 de escalares y los registros 514 de vectores) y los datos transferidos entre ellos se escriben en memoria y a continuación se leen de nuevo desde una caché 506 de nivel 1 (L1), algunos modos de realización alternativos de la invención pueden utilizar una estrategia distinta (por ejemplo, utilizar un único conjunto de registros o incluir una ruta de comunicación que permita que los datos se transfieran entre los dos ficheros de registros sin escribirse y volverse a leer).

El subconjunto local de la caché L2 504 es parte de una caché L2 global que se divide en subconjuntos locales independientes, una por núcleo de procesador. Cada núcleo de procesador tiene una ruta de acceso directo a su propio subconjunto local de la caché L2 504. Los datos leídos por un núcleo de procesador son almacenados en su subconjunto de caché L2 504 y se puede acceder rápidamente a ellos en paralelo con otros núcleos de procesadores que acceden a sus propios subconjuntos locales de caché L2. Los datos escritos por un núcleo de procesador se almacenan en su propio subconjunto de caché L2 504 y se descarga de otros subconjuntos si es necesario. La red en anillo asegura coherencia para los datos compartidos. La red en anillo es bidireccional y permite que agentes como, por ejemplo, los núcleos de procesador, las cachés L2 y otros bloques lógicos se comuniquen entre sí dentro del chip. Cada ruta de datos en el anillo tiene un ancho de 1012 bits por sentido.

La Figura 5B es una vista ampliada de parte del núcleo de procesador de la Figura 5A de acuerdo con los modos de realización de la invención. La Figura 5B incluye una parte 506A de la caché L1 de datos, una parte de la caché L1 504, así como más detalles en relación con la unidad 510 de vectores y los registros 514 de vectores. Específicamente, la unidad 510 de vectores es una unidad de procesamiento vectorial (VPU) de ancho 16 (véase la ALU 528 de ancho 16), que ejecuta una o más instrucciones de enteros, coma flotante de precisión simple y coma flotante de precisión doble. La VPU soporta reordenar las entradas de registros con la unidad 520 de reordenación, conversión numérica con las unidades 522A-B de conversión numérica y replicación con la unidad 524 de replicación en la entrada de memoria. Los registros 526 de máscara de escritura permiten indicar las escrituras de los vectores resultantes.

La Figura 6 es un diagrama esquemático de un procesador 600 que puede tener más de un núcleo, puede tener un controlador de memoria integrado y puede tener gráficos integrados de acuerdo con los modos de realización de la invención. Las cajas de líneas continuas de la Figura 6 ilustran un procesador 600 con un único núcleo 602A, un agente 610 de sistema, un conjunto de una o más unidades 616 controladoras de bus, mientras que la adición opcional de las cajas de líneas discontinuas ilustra un procesador 600 alternativo con múltiples núcleos 602A-N, un conjunto de una o más unidades 614 controladoras de memoria en el agente 610 de sistema, y una lógica 608 de propósito especial.

Así pues, diferentes implementaciones del procesador 600 pueden incluir: 1) una CPU con la lógica 608 de propósito especial que integra lógica de gráficos y/o científica (rendimiento) (que puede incluir uno o más núcleos), y siendo los núcleos 602A-N uno o más núcleos de propósito general (por ejemplo, núcleos de ejecución en orden de propósito general, núcleos de ejecución fuera de orden de propósito general, o una combinación de ambos); 2) un coprocesador con los núcleos 602A-N siendo estos un gran número de núcleos de propósito especial destinados principalmente para gráficos y/o científico (rendimiento); y 3) un coprocesador con los núcleos 602A-N siendo estos un gran número de núcleos de ejecución en orden de propósito general. Así pues, el procesador 600 puede ser un procesador de propósito general, un coprocesador o un procesador de propósito especial como, por ejemplo, un procesador de red o comunicación, un módulo de compresión, un procesador gráfico, una GPGPU (unidad de procesamiento gráfico de propósito general), un coprocesador de muchos núcleos integrados de alto rendimiento (MIC) (incluyendo 30 o más núcleos), un procesador embebido, etc. El procesador se puede implementar sobre uno o más chips. El procesador 600 puede ser una parte de y/o se puede implementar sobre uno o más sustratos utilizando cualquiera de una serie de tecnologías de procesamiento como, por ejemplo, BiCMOS, CMOS o NMOS.

La jerarquía de memoria incluye uno o más niveles de caché dentro de los núcleos, un conjunto de una o más unidades 606 de caché compartida y una memoria externa (no se muestra) acoplada al conjunto de unidades 614 de controlador de memoria integrada. El conjunto de unidades 606 de caché compartida puede incluir uno o más cachés de nivel medio como, por ejemplo, nivel 2 (L2), nivel 3 (L3), nivel 4 (L4) u otros niveles de caché, un último nivel de caché (LLC), y/o combinaciones de los mismos. Aunque en un modo de realización una unidad 612 de interconexión basada

en anillo interconecta la lógica 608 de gráficos integrada, el conjunto de unidades 606 de caché compartida y la unidad 610 de agente de sistema/las(s) unidad(es) 614 de controlador de memoria integrada, algunos modos de realización alternativos pueden utilizar cualquier número de técnicas bien conocidas para interconectar dichas unidades. En un modo de realización, se mantiene la coherencia entre una o más unidades 606 de caché y núcleos 602A-N.

En algunos modos de realización, uno o más de los núcleos 602A-N son capaces de ejecución multihilo. El agente 610 de sistema incluye aquellos componentes que coordinan y operan los núcleos 602A-N. La unidad 610 de agente de sistema 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 602A-N y la lógica 608 de gráficos integrada. La unidad de visualización se utiliza para gestionar una o más pantallas conectadas externamente.

Los núcleos 602A-N pueden ser homogéneos o heterogéneos en términos de conjunto de instrucciones de la arquitectura; esto es, dos o más de los núcleos 602A-N pueden ser capaces de ejecutar el mismo conjunto de instrucciones, mientras que otros pueden ser capaces de ejecutar únicamente un subconjunto de dicho conjunto de instrucciones o un conjunto de instrucciones diferente.

Las Figuras 7-10 son diagramas esquemáticos de arquitecturas de ordenador de ejemplo. También son apropiados otros diseños y configuraciones de sistema conocidos en la técnica para portátiles, ordenadores de sobremesa, PC de mano, asistentes personales digitales, estaciones de trabajo de ingeniería, servidores, dispositivos de red, concentradores de red, conmutadores, procesadores embebidos, procesadores de señales digitales (DSP), dispositivos gráficos, dispositivos de vídeo juegos, decodificadores, microcontroladores, teléfonos móviles, reproductores de medios portátiles, dispositivos portátiles y varios dispositivos electrónicos distintos. En general, también son apropiados una enorme variedad de sistemas y dispositivos electrónicos capaces de incorporar un procesador y/u otra lógica de ejecución como la divulgada en la presente solicitud.

Haciendo ahora referencia a la Figura 7, se muestra un diagrama esquemático de un sistema 700 de acuerdo con un modo de realización de la presente invención. El sistema 700 puede incluir uno o más procesadores 710, 715, los cuales están acoplados a un concentrador controlador 720. En un modo de realización el concentrador controlador 720 incluye un concentrador controlador de memoria y gráficos (GMCH) 790 y un Concentrador de Entrada/Salida (IOH) 750 (que pueden estar en chips independientes); el GMCH 790 incluye controladores de memoria y gráficos a los que se acopla una memoria 740 y un coprocesador 745; el IOH 750 tiene acoplados dispositivos de entrada/salida (E/S) 760 al GMCH 790. Alternativamente, uno o ambos de los controladores de memoria y gráficos se integran dentro del procesador (como se ha descrito en la presente solicitud), la memoria 740 y el coprocesador 745 se acoplan directamente al procesador 710, y el concentrador controlador 720 en un único chip con el IOH 750.

En la Figura 7 se representa la naturaleza opcional de los procesadores 715 adicionales utilizando líneas discontinuas. Cada uno de los procesadores 710, 715 puede incluir uno o más de los núcleos de procesamiento descritos en la presente solicitud y pueden ser algunas de las versiones del procesador 600.

La memoria 740 puede ser, por ejemplo, una memoria dinámica de acceso aleatorio (DRAM), una memoria de cambio de fase (PCM), o una combinación de ambas. Para al menos un modo de realización, el concentrador controlador 720 se comunica con el/los procesador(es) 710, 715 a través de un bus multipunto como, por ejemplo, un bus frontal (FSB), una interfaz punto a punto como, por ejemplo, QuickPath Interconnect (Interconexión de Ruta Rápida) (QPI), o una conexión 795 similar.

En un modo de realización, el coprocesador 745 es un procesador de propósito especial como, por ejemplo, un procesador MIC de alto rendimiento, un procesador de red o comunicación, un módulo de compresión, un procesador gráfico, una GPGPU, un procesador embebido, etc. En un modo de realización el concentrador controlador 720 puede incluir un acelerador gráfico integrado.

Puede existir una variedad de diferencias entre los recursos físicos 710, 715 en términos del espectro de métricas de mérito incluyendo características de arquitectura, microarquitectura, térmicas, consumo de energía, etc.

En un modo de realización, el procesador 710 ejecuta instrucciones que controlan operaciones de procesamiento de datos de tipo general. Embebidas dentro de las instrucciones se pueden encontrar instrucciones de coprocesador. El procesador 710 reconoce estas instrucciones de coprocesador como de un tipo que debería ser ejecutado por el coprocesador 745 anexo. En consecuencia, el procesador 710 le envía al coprocesador 745 estas instrucciones de coprocesador (o señales de control que representan las instrucciones de coprocesador) sobre un bus de coprocesador u otra interconexión. El/Los coprocesador(es) 745 aceptan y ejecutan las instrucciones de coprocesador recibidas.

Haciendo ahora referencia a la Figura 8, se muestra un diagrama esquemático de un primer sistema 800 más específico de ejemplo de acuerdo con un modo de realización de la presente invención. Tal como se muestra en la Figura 8, el sistema 800 multiprocesador es un sistema de interconexión punto a punto, e incluye un primer procesador 870 y un segundo procesador 880 acoplados a una interconexión punto a punto 850. Cada uno de los procesadores 870 y 880 puede ser alguna de las versiones del procesador 600. En un modo de realización de la invención, los procesadores 870 y 880 son, respectivamente, los procesadores 710 y 715, mientras que el coprocesador 838 es el

coprocesador 745. En otro modo de realización, los procesadores 870 y 880 son, respectivamente, el procesador 710 y el coprocesador 745.

Se muestra que los procesadores 870 y 880 incluyen unidades 872 y 882 de controlador de memoria integrada (IMC), respectivamente. El procesador 870 también incluye como parte de sus unidades de controlador de bus interfaces 876 y 878 punto a punto (P-P); del mismo modo, el segundo procesador 880 incluye interfaces 886 y 888 P-P. Los procesadores 870, 880 pueden intercambiar información a través de una interfaz 850 punto a punto (P-P) utilizando los circuitos 878, 888 de interfaz P-P. Tal como se muestra en la Figura 8, los IMC 872 y 882 acoplan los procesadores a las memorias respectivas, denominadas memoria 832 y memoria 834, que pueden ser partes de la memoria principal conectada localmente a los procesadores respectivos.

Los procesadores 870 y 880 pueden cada uno intercambiar información con un chipset 890 a través de interfaces 852, 854 P-P individuales utilizando los circuitos 876, 894, 886, 898 de interfaz punto a punto. El chipset 890 puede opcionalmente intercambiar información con el coprocesador 838 a través de una interfaz 839 de alto rendimiento. En un modo de realización, el coprocesador 838 es un procesador de propósito especial como, por ejemplo, un procesador MIC de alto rendimiento, un procesador de red o comunicación, un módulo de compresión, un procesador gráfico, una GPGPU, un procesador embebido, etc.

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

El chipset 890 puede estar acoplado a un primer bus 816 a través de una interfaz 896. En un modo de realización, el primer bus 816 puede ser un bus de Interconexión de Componentes Periféricos (PCI) o un bus como, por ejemplo, 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 en este aspecto.

Tal como se muestra en la Figura 8, se pueden acoplar varios dispositivos 814 de E/S al primer bus 816, junto con un puente 818 de bus que acopla el primer bus 816 con un segundo bus 820. En un modo de realización, al primer bus 816 se acoplan uno o más procesadores 815 adicionales como, por ejemplo, coprocesadores, procesadores MIC de alto rendimiento, GPGPU, aceleradores (por ejemplo, aceleradores gráficos o unidades de procesamiento de señales digitales (DSP)), matrices de puertas programables en campo o cualquier otro procesador. En un modo de realización, el segundo bus 820 puede ser un bus de baja cantidad de pines (LPC). En un modo de realización se pueden acoplar varios dispositivos a este segundo bus 820 incluyendo, por ejemplo, un teclado y/o ratón 822, dispositivos 827 de comunicación y una unidad 828 de almacenamiento como, por ejemplo, una unidad de disco u otro dispositivo de almacenamiento masivo que pueda almacenar instrucciones/código y datos 830. Además, a este segundo bus 820 se puede acoplar una E/S de sonido 824. Obsérvese que son posibles otras arquitecturas. Por ejemplo, en lugar de la arquitectura punto a punto de la Figura 8, un sistema puede implementar un bus multipunto u otra arquitectura similar.

Haciendo referencia ahora a la Figura 9, se muestra un diagrama esquemático de un segundo sistema 900 más específico de ejemplo de acuerdo con un modo de realización de la presente invención. Los elementos similares en las Figuras 8 y 9 utilizan los mismos números de referencia, y en la Figura 9 se han omitido ciertos aspectos de la Figura 8 con el fin de evitar esconder otros aspectos de la Figura 9.

La Figura 9 ilustra que los procesadores 870 y 880 pueden incluir una memoria integrada y una lógica de control de E/S ("CL") 872 y 882, respectivamente. Así pues, la CL 872, 882 incluye unidades controladoras de memoria integradas y una lógica de control de E/S. La Figura 9 ilustra que no existen únicamente las memorias 832, 834 acopladas a la CL 872, 882, sino que a la lógica de control 872, 882 también se le acoplan dispositivos 914 de E/S. Los dispositivos de E/S antiguos 915 se acoplan al chipset 890.

Haciendo referencia ahora a la Figura 10, se muestra un diagrama esquemático de un SoC 1000 de acuerdo con un modo de realización de la presente invención. Los elementos similares en la Figura 6 utilizan los mismos números de referencia. Además, las cajas con líneas discontinuas son características opcionales en SoC más avanzados. En la Figura 10, una/varias unidad(es) 1002 de interconexión está(n) acoplada(s) a: un procesador 1010 de aplicaciones que incluye un conjunto de uno o más núcleos 202A-N y una/varias unidad(es) 606 de caché compartida; una unidad 610 de agente de sistema; una/varias unidad(es) 616 controladoras de bus; una/varias unidad(es) 614 de controlador de memoria integrada; un conjunto de uno o más coprocesadores 1020 que pueden incluir una lógica de gráficos integrada, un procesador de imágenes, un procesador de audio y un procesador de vídeo; una unidad 1030 de memoria estática de acceso aleatorio (SRAM); una unidad 1032 de acceso directo a memoria (DMA); y una unidad 1040 de visualización para conectar una o más pantallas externas. En un modo de realización, el/los coprocesador(es) 1020 incluyen un procesador de propósito especial como, por ejemplo, un procesador de red o comunicación, un módulo de compresión, una GPGPU, un procesador MIC de alto rendimiento, un procesador embebido, etc.

Los modos de realización de los mecanismos divulgados en la presente solicitud se pueden implementar mediante hardware, software, firmware o una combinación de dichas formas de implementación. Los modos de realización de la invención se pueden implementar como programas de ordenador o código de programa que se ejecuta en sistemas

programables que comprenden al menos un procesador, un sistema de almacenamiento (incluyendo memoria volátil y no volátil y/o elementos de almacenamiento), al menos un dispositivo de entrada y al menos un dispositivo de salida.

El código de programa como, por ejemplo, el código 830 que se ilustra en la Figura 8, consiste en instrucciones de entrada para realizar las funciones descritas en la presente solicitud y generar información de salida. La información de salida se puede enviar a uno o más dispositivos de salida, de forma conocida. Para el propósito de esta solicitud, un sistema de procesamiento incluye cualquier sistema que tenga un procesador como, por ejemplo: un procesador de señales digitales (DSP), un microcontrolador, un circuito integrado de aplicación específica (ASIC) o un microprocesador.

El código de programa se puede implementar en un lenguaje de programación de alto nivel procedimental u orientado a objetos para comunicarse con un sistema de procesamiento. Si se desea, el código de programa también se puede implementar en ensamblador o lenguaje de máquina. De hecho, los mecanismos descritos en la presente solicitud no se limitan en alcance a ningún lenguaje de programación concreto. En cualquier caso, el lenguaje puede ser un lenguaje compilado o interpretado.

Uno o más aspectos de al menos un modo de realización se puede implementar mediante instrucciones representativas almacenadas en un medio legible por una máquina que representa diferente lógica dentro del procesador, las cuales cuando son leídas por una máquina hacen que la máquina aplique una lógica para implementar las técnicas descritas en la presente solicitud. Dichas representaciones, conocidas como “núcleos IP” se pueden almacenar en un medio tangible legible por una máquina y suministrarse a varios clientes o instalaciones de fabricación para cargarlas en las máquinas fabricadas que realmente constituyen la lógica o el procesador.

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

De acuerdo con ello, algunos modos de realización de la invención también incluyen medios de almacenamiento tangibles no transitorios legibles por una máquina que contienen instrucciones o que contienen datos de diseño como, por ejemplo, el Lenguaje de Descripción de Hardware (HDL), que define características de estructuras, circuitos, equipos, procesadores y/o sistemas descritas en la presente solicitud. Dichos modos de realización también se pueden denominar productos de programa.

En algunos casos, se puede utilizar un conversor de instrucciones para convertir una instrucción desde un conjunto de instrucciones origen a un conjunto de instrucciones objetivo. Por ejemplo, el conversor de instrucciones puede traducir (por ejemplo, utilizando una traducción binaria estática, una traducción binaria dinámica con compilación dinámica), transformar, emular o convertir de otra forma una instrucción en una o más instrucciones distintas para ser procesadas por el núcleo. El conversor de instrucciones se puede implementar mediante software, hardware, firmware o una combinación de los mismos. El conversor de instrucciones puede encontrarse dentro del procesador, fuera del procesador o una parte dentro y otra parte fuera del procesador.

La Figura 11 es un diagrama esquemático que compara la utilización de un conversor de instrucciones software para convertir instrucciones binarias en un conjunto de instrucciones origen a instrucciones binarias en un conjunto de instrucciones objetivo de acuerdo con algunos modos de realización de la invención. En el modo de realización ilustrado, el conversor de instrucciones es un conversor de instrucciones software, aunque, alternativamente, el conversor de instrucciones se puede implementar en software, firmware, hardware o varias combinaciones de los mismos. La Figura 11 muestra un programa en lenguaje 1102 de alto nivel que se puede compilar utilizando un compilador x86 1104 para generar código binario x86 1106 que un procesador puede ejecutar nativamente con al menos un núcleo 1116 del conjunto de instrucciones x86. El procesador con al menos un núcleo 1116 del conjunto de instrucciones x86 representa cualquier procesador que pueda realizar sustancialmente las mismas funciones que un procesador Intel con al menos un núcleo del conjunto de instrucciones x86 mediante compatibilidad ejecutando o al menos procesando (1) una parte sustancial del conjunto de instrucciones del núcleo del conjunto de instrucciones x86 de Intel o (2) versiones de aplicaciones de código objeto u otro software cuyo propósito es ejecutarse en un procesador Intel con al menos un núcleo del conjunto de instrucciones x86, con el fin de conseguir sustancialmente el mismo resultado que un procesador Intel con al menos un núcleo del conjunto de instrucciones x86. El compilador x86 1104 representa un compilador que se puede utilizar para generar código binario x86 1106 (por ejemplo, código objeto) que puede, con o sin un proceso de enlace adicional, ejecutarse en el procesador con al menos un núcleo 1116 del conjunto de instrucciones x86. De forma análoga, la Figura 11 muestra que el programa en lenguaje 1102 de alto nivel se puede compilar utilizando un compilador 1108 de un conjunto de instrucciones alternativo para generar código binario 1110 de un conjunto de instrucciones alternativo que un procesador puede ejecutar de forma nativa sin al menos un núcleo 1114 del conjunto de instrucciones x86 (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 conversor 1112 de instrucciones se utiliza para convertir el código binario x86 1106 en código que el procesador pueda ejecutar de forma nativa sin un núcleo 1114 del conjunto de instrucciones x86. Este código convertido probablemente no sea el mismo que el código binario 1110 del conjunto de instrucciones alternativo porque es difícil hacer un conversor de instrucciones capaz de esto; sin embargo, el código convertido cumplirá el funcionamiento general y estará formado por instrucciones del conjunto de instrucciones alternativo. Por lo tanto, el conversor 1112 de instrucciones representa software, firmware, hardware o una combinación de los mismos que, mediante emulación, simulación o cualquier otro proceso, permite que un procesador u otro dispositivo electrónico que no tenga un procesador o núcleo del conjunto de instrucciones x86 ejecute código binario x86 1106.

Método y Equipo para Realizar una Reorganización de Bits de Vectores

A continuación, se describe una instrucción de reorganización de bits de un vector que realiza una reorganización de bits utilizando un primer operando origen como control y un segundo operando origen como datos. Esta instrucción se puede utilizar para implementar de forma eficiente varias rutinas de manipulación de bits. Mediante un ejemplo y sin suponer una limitación, se puede utilizar para implementar una permutación de bits variable suponiendo un aumento de velocidad de hasta 8x con respecto a las implementaciones VEX o EVEX actuales.

Tal como se ilustra en la Figura 12, un procesador 1255 de ejemplo sobre el que se pueden implementar los modos de realización de la invención incluye un conjunto de registros de propósito general (GPR) 1205, un conjunto de registros 1206 de vectores y un conjunto de registros 1207 de máscaras. En un modo de realización, se empaquetan múltiples elementos de datos vectoriales en cada registro 1206 de vectores que puede tener un tamaño de 512 bits para almacenar dos valores de 256 bits, cuatro valores de 128 bits, ocho valores de 64 bits, dieciséis valores de 32 bits, etc. Sin embargo, los principios subyacentes de la invención no están limitados a un tamaño/tipo concreto de datos vectoriales. En un modo de realización, los registros 1207 de máscaras incluyen ocho registros de máscaras de operandos de 64 bits utilizados para realizar operaciones de selección de bit sobre los valores almacenados en los registros 1206 de vectores (por ejemplo, implementados como registros k0-k7 de máscaras descritos más arriba). Sin embargo, los principios subyacentes de la invención no están limitados a un tamaño/tipo concreto del registro de máscaras.

Por simplicidad, en la Figura 12, se ilustran los detalles de un único núcleo de procesador (“Núcleo 0”). Sin embargo, se entenderá que cada uno de los núcleos que se muestran en la Figura 12 puede tener el mismo conjunto de lógica que el Núcleo 0. Por ejemplo, cada uno de los núcleos puede incluir una caché 1212 de Nivel 1 (L1) y caché 1211 de Nivel 2 (L2) dedicadas al almacenamiento temporal de instrucciones y datos de acuerdo con una política de gestión de caché especificada. La caché L1 1212 incluye una caché 1220 de instrucciones independiente para almacenar instrucciones y una caché 1221 de datos independiente para almacenar datos. Las instrucciones y datos almacenados dentro de las distintas cachés del procesador se gestionan con la granularidad de líneas de caché que pueden ser de tamaño fijo (por ejemplo, de una longitud de 64, 128, 512 Bytes). Cada núcleo en este modo de realización de ejemplo tiene una unidad 1210 de recuperación de instrucciones para recuperar instrucciones de la memoria principal 1200 y/o una caché 1216 de Nivel 3 (L3) compartida; una unidad 1220 de decodificación para decodificar las instrucciones (por ejemplo, decodificar instrucciones de programa en micro operaciones o “μops”); una unidad 1240 de ejecución para ejecutar las instrucciones; y una unidad 1250 de respuesta para descartar las instrucciones y responder con los resultados.

La unidad 1210 de recuperación de instrucciones incluye varios componentes bien conocidos incluyendo un puntero 1203 a la próxima instrucción para almacenar la dirección de la instrucción siguiente a recuperar de la memoria 1200 (o una de las cachés); un buffer de traducción adelantada (ITLB) 1204 de instrucciones para almacenar un mapeo de las direcciones de instrucciones virtuales a físicas utilizadas para mejorar la velocidad de traducción de direcciones; una unidad 1202 de predicción de saltos para predecir de forma especulativa direcciones de salto de las instrucciones; y varios buffer destino de salto (BTB) 1201 para almacenar direcciones de salto y direcciones de destino. Una vez recuperadas, las instrucciones se encaminan a continuación al resto de etapas del pipeline de instrucciones que incluyen la unidad 1230 de decodificación, la unidad 1240 de ejecución y la unidad 1250 de reescritura. Aquellos con un conocimiento normal en la técnica entienden bien la estructura y función de cada una de estas unidades y no se describirá aquí en detalle con el fin de evitar esconder los aspectos pertinentes de los diferentes modos de realización de la invención.

En un modo de realización, la unidad 1230 de decodificación incluye una lógica 1231 de decodificación de reorganización de bits de vectores para decodificar las instrucciones de reorganización de bits de un vector descritas en la presente solicitud (por ejemplo, en secuencias de microoperaciones en un modo de realización) y la unidad 1240 de ejecución incluye una lógica 1241 de ejecución de reorganización de bits de vectores para ejecutar las instrucciones.

Tal como se ha mencionado, en un modo de realización, la instrucción de reorganización de bits de un vector realiza una selección de varios bits utilizando un primer origen como control y un segundo origen como datos e incluyendo los resultados en un registro de destino. En un modo de realización, cada uno de los bits del destino se identifica del segundo origen utilizando 6 bits de control del primer origen.

La Figura 13 ilustra un modo de realización de ejemplo que incluye un primer registro de origen, SRC2, para almacenar los bits de control, un segundo registro de origen, SRC3, para almacenar los datos de origen, y un registro de destino, DST, para almacenar los resultados de la instrucción de reorganización de bits de un vector. En un modo de realización, SRC3 comprende ocho líneas 0-7 de datos de 64 bits empaquetados en un registro de un vector de 512 bits; SRC2 comprende ocho conjuntos de 8 bytes de control 1300-1302, 1307 también empaquetados en un registro de un vector de 512 bits; y DST comprende un registro 1320 de máscara de 64 bits. Sin embargo, tal como se ha mencionado previamente, los principios subyacentes de la invención no están limitados a cualquier tamaño/tipo concreto de operandos o registros. Obsérvese que por simplicidad en la Figura 13 únicamente se muestra una porción de los datos almacenados en SRC3 y los bits de control almacenados en SRC2.

En funcionamiento, cada uno de los bytes de cada conjunto de 8 bytes de control identifica un bit concreto dentro de su línea de 64 bits correspondiente. Por lo tanto, cada uno de los 8 bytes de control 1300 identifica un bit dentro de la Línea 0; cada uno de los 8 bytes de control 1301 identifica un bit dentro de la Línea 1, etc. En un modo de realización, únicamente 6 de los 8 bits en cada byte de control 1300-1302, 1307 se utiliza para identificar un bit dentro de SRC3 (seis bits son suficientes ya que $2^6 = 64$). Los dos bits restantes se pueden ignorar.

En un modo de realización, estos 6 bits de cada uno de los 8 bytes de control 1300-1307 se aplican a una lógica de selección 1310-1312, 1317 (por ejemplo, un conjunto de multiplexores) para seleccionar los 8 bits de cada línea. Así pues, la lógica 1310 de selección selecciona 8 bits de la Línea 0; la lógica 1311 de selección selecciona 8 bits de la Línea 1; la lógica 1312 de selección selecciona 8 bits de la Línea 2; y la lógica 1317 de selección selecciona 8 bits de la Línea 7 (tal como se ha mencionado, por simplicidad no se muestran las Líneas 3-6 y la lógica de selección asociada).

El resultado final es que de SRC3 se leen ocho conjuntos de 8 bits. En un modo de realización, los ocho conjuntos de 8 bits se concatenan dentro de DST para formar un valor 1320 de máscara de 64 bits. Una vez se ha formado, el valor 1320 de máscara de 64 bits se puede utilizar para operaciones de selección posteriores.

En la Figura 14 se ilustra un método de acuerdo con un modo de realización de la invención. El método se puede ejecutar en el contexto de las arquitecturas descritas más arriba, pero no está limitado a ninguna arquitectura de sistema específica.

En 1401, se recupera la instrucción de reorganización de bits de un vector de la memoria del sistema o se lee de la caché (por ejemplo, una caché L1, L2 o L3). En 1402, en respuesta a la decodificación/ejecución de la instrucción de reorganización de bits de un vector, los datos del vector de entrada a ser reorganizados se almacenan en un primer registro de origen. Tal como se ha mencionado, en un modo de realización, el primer registro de origen es un registro de un vector de 512 bits y los datos del vector comprenden ocho líneas de 64 bits de datos. En 1403, en un segundo registro origen se almacenan los datos de control necesarios para realizar la reorganización de bits del vector, tal como se ha mencionado, y puede ser otro registro de un vector de 512 bits.

En 1404, se identifica un conjunto de bits de cada línea en el primer registro origen utilizando los conjuntos de bits de control asociados en el segundo registro origen. Tal como se ha mencionado, en un modo de realización, se proporcionan ocho bytes de control para cada línea y se utilizan 6 bits de cada byte de control para identificar un bit de la línea correspondiente. El resultado final es que del primer registro origen se leen ocho conjuntos de 8 bits. Por último, en 1405, se concatenan los conjuntos de bits dentro de un registro de máscara de destino. En el modo de realización descrito más arriba, por ejemplo, se concatenan los ocho conjuntos de 8 bits para formar un valor de máscara de 64 bits.

En un modo de realización, para una implementación codificada EVEX, el primer operando origen, el segundo operando origen y el operando destino son todos registros ZMM. En un modo de realización la instrucción de reorganización de bits de un vector tiene la siguiente forma (donde DEST es el destino, SRC2 comprende el origen que contiene los datos de control y SRC3 comprende el origen que contiene los datos a reorganizar:

```
VPSHUFBITQMB DEST, SRC2, SRC3
```

El siguiente pseudocódigo proporciona una representación de las operaciones realizadas de acuerdo con un modo de realización de la invención:

```
VSHUFBITQMB DEST, SRC2, SRC3
(KL, VL) = (16, 128), (32, 256), (64, 512)
FOR i:= 0 to KL/8 - 1; Qword
        FOR j:= 0 to 7; Byte
                IF EVEX.b AND SRC3 *is memory*
                THEN
                        Data:= SRC3.qword[0];
```

```
                    ELSE
                        Data:= SRC3.qword[i];
                    m:= SRC2.qword[i].byte[j] & 0x3F
                    k1[i*8+j]:=Data.bit[m]
                END FOR
            END FOR
            k1[MAX_KL-1 :KL]:=0
```

Así pues, asumiendo que KL = 64 y VL = 512, el bucle FOR externo (de índice i) se utiliza para seleccionar cada una de las diferentes líneas de 64 bits (Qword) y el bucle FOR interior (de índice j) se utiliza para seleccionar los 8 bits en cada línea utilizando los valores de control especificados en los bytes de control. La sentencia IF con "EVEX.b AND SRC3 *is memory*" indica que si el bit "b" está activado en el campo de bit EVEX (utilizado típicamente para broadcast de origen, control de redondeo (combinado con L'L) o excepciones de supresión) y si los datos de origen se leen de la memoria del sistema, entonces se copia un único valor origen de 64 bits a todas las líneas (para (KL, VL) = (64, 512)). En caso contrario, la palabra cuádruple (qword)/línea a utilizar se selecciona basándose en el valor actual de i (Data:= SRC3.qword[i]). Además, se realiza una operación AND del valor 0x3F con el valor índice SRC2.qword[i].byte[j] ya que únicamente se utilizan 6 bits de cada byte de control para identificar el valor de bit dentro de cada línea de 64 bits (esto es, la operación AND con 0x3F elimina los dos bits superiores).

En la memoria descriptiva precedente, se han descrito los modos de realización haciendo referencia a sus modos de realización específicos de ejemplo. Sin embargo, será evidente que se pueden realizar varias modificaciones y cambios a los mismos sin apartarse del alcance más amplio de la invención tal como se describe en las reivindicaciones adjuntas. En consecuencia, la memoria descriptiva y los dibujos se deben tomar en sentido ilustrativo en lugar de restrictivo.

Algunos modos de realización de la invención pueden incluir varios pasos, los cuales se han descrito más arriba. Los pasos se pueden materializar en instrucciones ejecutables por una máquina que pueden ser utilizadas para que un procesador de propósito general o propósito especial ejecute los pasos. Alternativamente, estos pasos pueden ejecutarse mediante componentes hardware específicos que contienen lógica cableada para ejecutar los pasos, o mediante cualquier combinación de componentes de ordenador programados y componentes hardware a medida.

Tal como se ha descrito en la presente solicitud, las instrucciones se pueden referir a configuraciones específicas de hardware como, por ejemplo, circuitos integrados de aplicación específica (ASIC) configurados para realizar ciertas operaciones o con una funcionalidad predeterminada, o instrucciones software almacenadas en una memoria en forma de medio no transitorio legible por un ordenador. Así pues, las técnicas que se muestran en la Figuras se pueden implementar utilizando código y datos almacenados, ejecutado en uno o más dispositivos electrónicos (por ejemplo, una estación final, un elemento de red, etc.). Dichos dispositivos electrónicos almacenan y comunican (internamente y/o con otros dispositivos electrónicos sobre una red) código y datos utilizando medios legibles por una máquina u ordenador como, por ejemplo, medios de almacenamiento no transitorios legibles por una máquina u ordenador (por ejemplo, discos magnéticos; discos ópticos; memoria de acceso aleatorio; memoria de solo lectura; dispositivos de memoria flash; memoria de cambio de fase) y medios de comunicación transitorios legibles por una máquina u ordenador (por ejemplo, señales eléctricas, ópticas, acústicas u otras formas de señales propagadas – por ejemplo ondas de portadora, señales infrarrojas, señales digitales, etc.). Además, dichos dispositivos electrónicos incluyen típicamente un conjunto de uno o más procesadores acoplados a uno o más componentes distintos como, por ejemplo, uno o más dispositivos de almacenamiento (medios de almacenamiento no transitorios legibles por una máquina), dispositivos de entrada/salida de usuario (por ejemplo, un teclado, una pantalla táctil y/o una pantalla), y conexiones de red. El acoplamiento del conjunto de procesadores y otros componentes se realiza típicamente mediante uno o más buses y puentes (también denominados controladores de bus). El dispositivo de almacenamiento y las señales que transportan el tráfico de red representan, respectivamente, uno o más medios de almacenamiento legibles por una máquina y medios de comunicación legibles por una máquina. Así pues, el dispositivo de almacenamiento de un dispositivo electrónico dado almacena típicamente código y/o datos para su ejecución en el conjunto de uno o más procesadores de dicho dispositivo electrónico. Por supuesto, una o más partes de un modo de realización de la invención se pueden implementar utilizando diferentes combinaciones de software, firmware y/o hardware. A lo largo de esta descripción detallada, con el propósito de explicación, se han descrito numerosos detalles específicos con el fin de proporcionar una amplia comprensión de la presente invención. Sin embargo, será evidente para alguien experimentado en la técnica que la invención se puede poner en práctica sin algunos de dichos detalles específicos. En ciertos casos, algunas estructuras y funciones bien conocidas no se han descrito en gran detalle con el fin de evitar oscurecer la materia objeto de la presente invención. En consecuencia, el alcance de la invención se debe evaluar en términos de las siguientes reivindicaciones.

REIVINDICACIONES

1. Un procesador que comprende:

un decodificador para decodificar una instrucción de reorganización de bits de un vector, comprendiendo la instrucción de reorganización de bits de un vector un primer operando de origen, un segundo operando de origen y un operando de destino;

un primer registro de vector identificado por el primer operando de origen para almacenar una pluralidad de elementos de datos de origen;

un segundo registro de vector identificado por el segundo operando de origen para almacenar una pluralidad de elementos de control, correspondiendo cada uno de los elementos de control a uno diferente de una pluralidad de elementos de datos de origen en el primer registro de vector y comprendiendo una pluralidad de campos de bit, correspondiendo cada uno de los campos de bit a una única posición de bit en un registro de máscara de destino identificado por el operando de destino, y sirviendo además cada uno de los campos de bit para identificar exactamente un bit del elemento de datos de origen correspondiente para copiarse a la posición única de bit correspondiente en el registro de máscara de destino; y

una lógica de reordenación de bits de vector para leer los campos de bits del segundo registro de vector y, para cada uno de los campos de bit, identificar exactamente un bit de los elementos de datos de origen y copiar como consecuencia únicamente el bit identificado del elemento de datos de origen a una única posición de bit correspondiente al campo de bit en el registro de máscara de destino.

2. El procesador como el de la reivindicación 1 en donde la lógica de reordenación de bits de vector comprende uno o más multiplexores para seleccionar un conjunto de bits de cada uno de los elementos de datos de origen de acuerdo con los campos de bit de los elementos de control.

3. El procesador como el de la reivindicación 1 en donde cada uno de los elementos de datos de origen comprende un elemento de datos de 64 bits y en donde cada campo de bit comprende al menos 6 bits para identificar un bit de cada uno de los elementos de datos de 64 bits.

4. El procesador como el de la reivindicación 3 en donde cada uno de los campos de bit comprende un byte de control y en donde los 6 bits son para ser seleccionados de cada uno de los bytes de control para identificar cada bit de cada uno de los elementos de datos de 64 bits.

5. El procesador como el de la reivindicación 4 en donde ocho bits de cada uno de los elementos de datos es para ser seleccionado utilizando los ocho bytes de control.

6. El procesador como el de la reivindicación 5 en donde los bits de cada elemento de datos son para ser concatenados dentro del registro de máscara de destino.

7. El procesador como el de la reivindicación 6 en donde el primer registro de vector es para almacenar ocho de los elementos de datos de 64 bits y en donde el registro de máscara de destino es para registrar ocho valores de 8 bits correspondientes seleccionados de los ocho elementos de datos de 64 bits.

8. El procesador como el de la reivindicación 7 en donde los bits dentro del registro de máscara son para ser utilizados para realizar operaciones de masking (selección) para una o más instrucciones posteriores ejecutadas por el procesador.

9. El procesador como el de la reivindicación 1 en donde la lógica de reordenación de bits de vector es para operar como respuesta a la instrucción de reorganización de bits de un vector decodificada por la lógica de decodificación y ejecutada por la lógica de ejecución en el procesador.

10. Un método que comprende:

decodificar una instrucción de reorganización de bits de un vector, comprendiendo la instrucción de reorganización de bits de un vector un primer operando de origen, un segundo operando de origen y un operando de destino;

almacenar una pluralidad de elementos de datos de origen en un primer registro de vector identificado por el primer operando de origen;

almacenar una pluralidad de elementos de control en un segundo registro de vector identificado por el segundo operando de origen, correspondiendo cada uno de los elementos de control a uno diferente de una pluralidad de elementos de datos de origen en el primer registro de vector y comprendiendo una pluralidad de campos de bit, correspondiendo cada uno de los campos de bit a una única posición de bit en un registro de máscara de destino

identificado por el operando de destino, y sirviendo además cada uno de los campos de bit para identificar exactamente un bit del elemento de datos de origen correspondiente para copiarse a la posición única de bit correspondiente en el registro de máscara de destino; y

leer los campos de bits del segundo registro de vector y, para cada uno de los campos de bit, identificar exactamente un bit de los elementos de datos de origen correspondientes y copiar como consecuencia únicamente el bit identificado del elemento de datos de origen a una única posición de bit correspondiente al campo de bit en el registro de máscara de destino.

11. El método como el de la reivindicación 10 que comprende, además, seleccionar un conjunto de bits de cada uno de los elementos de datos de origen con uno o más multiplexores de acuerdo con los campos de bit en cada uno de los elementos de control.

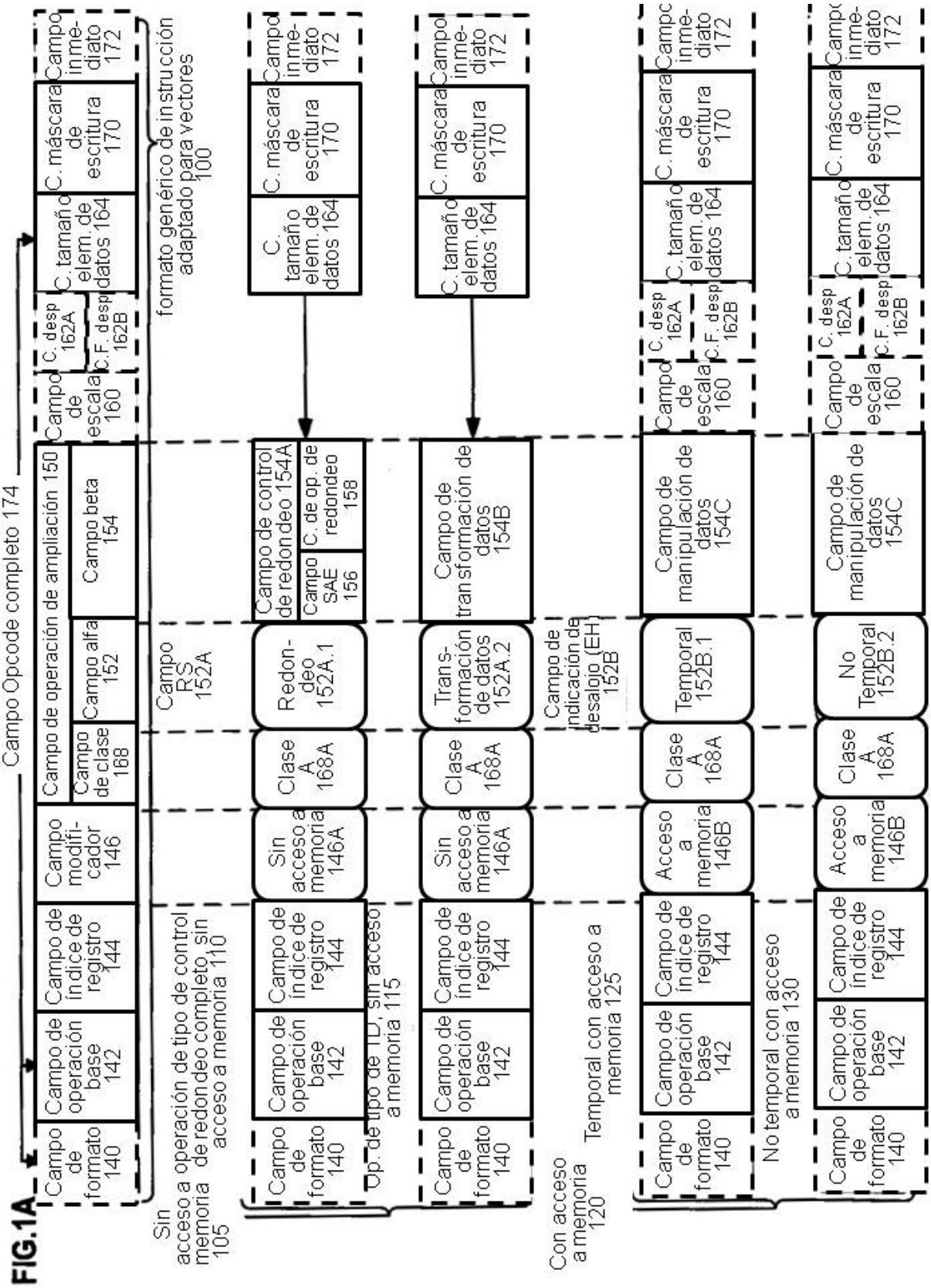
12. El método como el de la reivindicación 10 en donde cada uno de los elementos de datos de origen comprende un elemento de datos de 64 bits y en donde cada uno de los campos de bit comprende al menos 6 bits para identificar un bit de cada uno de los elementos de datos de 64 bits.

13. El método como el de la reivindicación 12 en donde cada uno de los campos de bit comprende un byte de control y en donde los 6 bits son para ser seleccionados de cada uno de los bytes de control para identificar cada bit de cada uno de los elementos de datos de 64 bits.

14. El método como el de la reivindicación 13 en donde ocho bits de cada uno de los elementos de datos son para ser seleccionados utilizando ocho de los bytes de control.

15. El método como el de la reivindicación 14 en donde los bits de cada elemento de datos son para ser concatenados dentro del registro de máscara de destino.

FIG.1A


Campo Opcode completo 174
Campo de formato 140
Campo de operación base 142
Campo de índice de registro 144
Campo modificador 146
Campo de operación de ampliación 150
Campo de clase 168
Campo alfa 152
Campo beta 154
Campo de escala 160
C. desp 162A
C.F. desp 162B
C. tamaño elem. de datos 164
C. máscara de escritura 170
Campo inmediato 172
formato genérico de instrucción adaptado para vectores 100
Sin acceso a memoria 105
Sin acceso a operación de tipo de control de redondeo completo, sin acceso a memoria 110
Campo RS 152A
Sin acceso a memoria 146A
Clase A 168A
Redondeo 152A.1
Campo de control de redondeo 154A
Campo SAE 156
C. de op. de redondeo 158
Op. de tipo de TD, sin acceso a memoria 115
Transformación de datos 152A.2
Campo de transformación de datos 154B
Campo de indicación de desalojo (EH) 152B
Con acceso a memoria 120
Temporal con acceso a memoria 125
Acceso a memoria 146B
Temporal 152B.1
Campo de manipulación de datos 154C
No temporal con acceso a memoria 130
No Temporal 152B.2

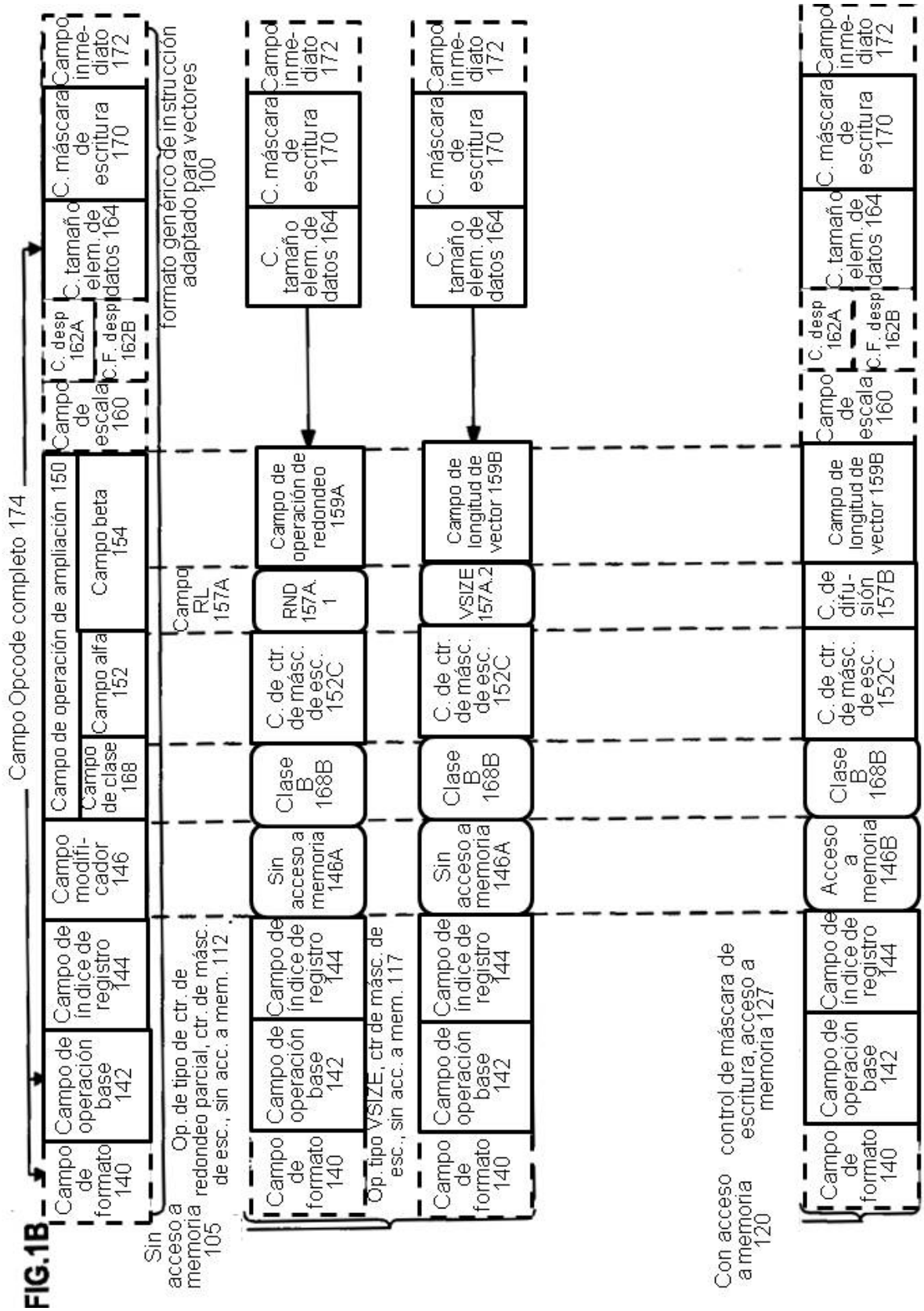


FIG.1B
Campo Opcode completo 174
Campo de formato 140
Campo de operación base 142
Campo de índice de registro 144
Campo modificador 146
Campo de operación de ampliación 150
Campo de clase 168
Campo alfa 152
Campo beta 154
Campo de escala 160
C. desp 162A
C.F. desp 162B
C. tamaño elem. de datos 164
C. máscara de escritura 170
Campo inmediato 172
Sin acceso a memoria 105
formato genérico de instrucción adaptado para vectores 100
Op. de tipo de ctr. de redondeo parcial, ctr. de másc. de esc., sin acc. a mem. 112
Campo RL 157A
Sin acceso a memoria 146A
Clase B 168B
C. de ctr. de másc. de esc. 152C
RND 157A.1
Campo de operación de redondeo 159A
Op. tipo VSIZE, ctr de másc. de esc., sin acc. a mem. 117
VSIZE 157A.2
Campo de longitud de vector 159B
Con acceso a memoria 120
control de máscara de escritura, acceso a memoria 127
Acceso a memoria 146B
C. de difusión 157B

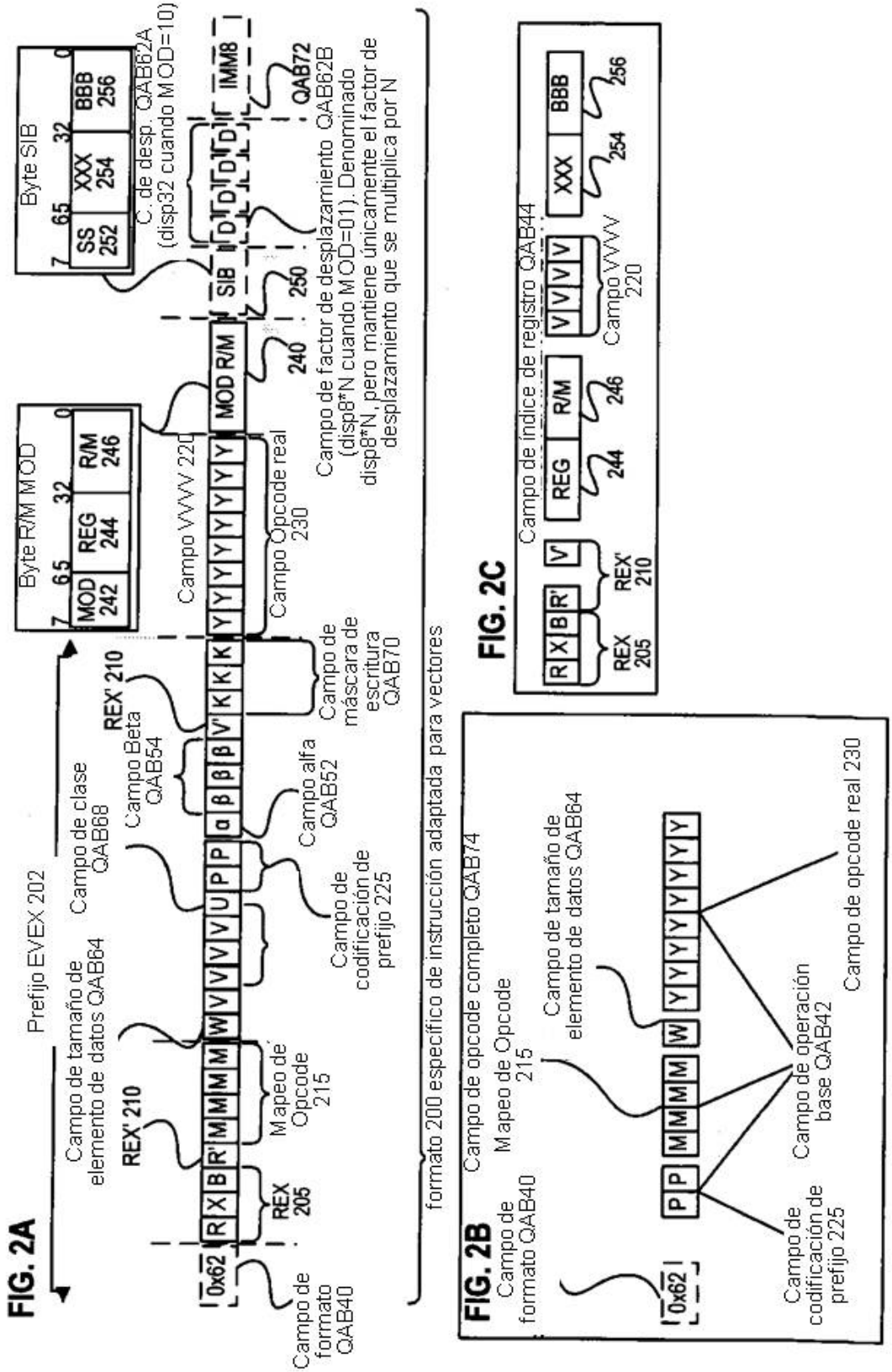
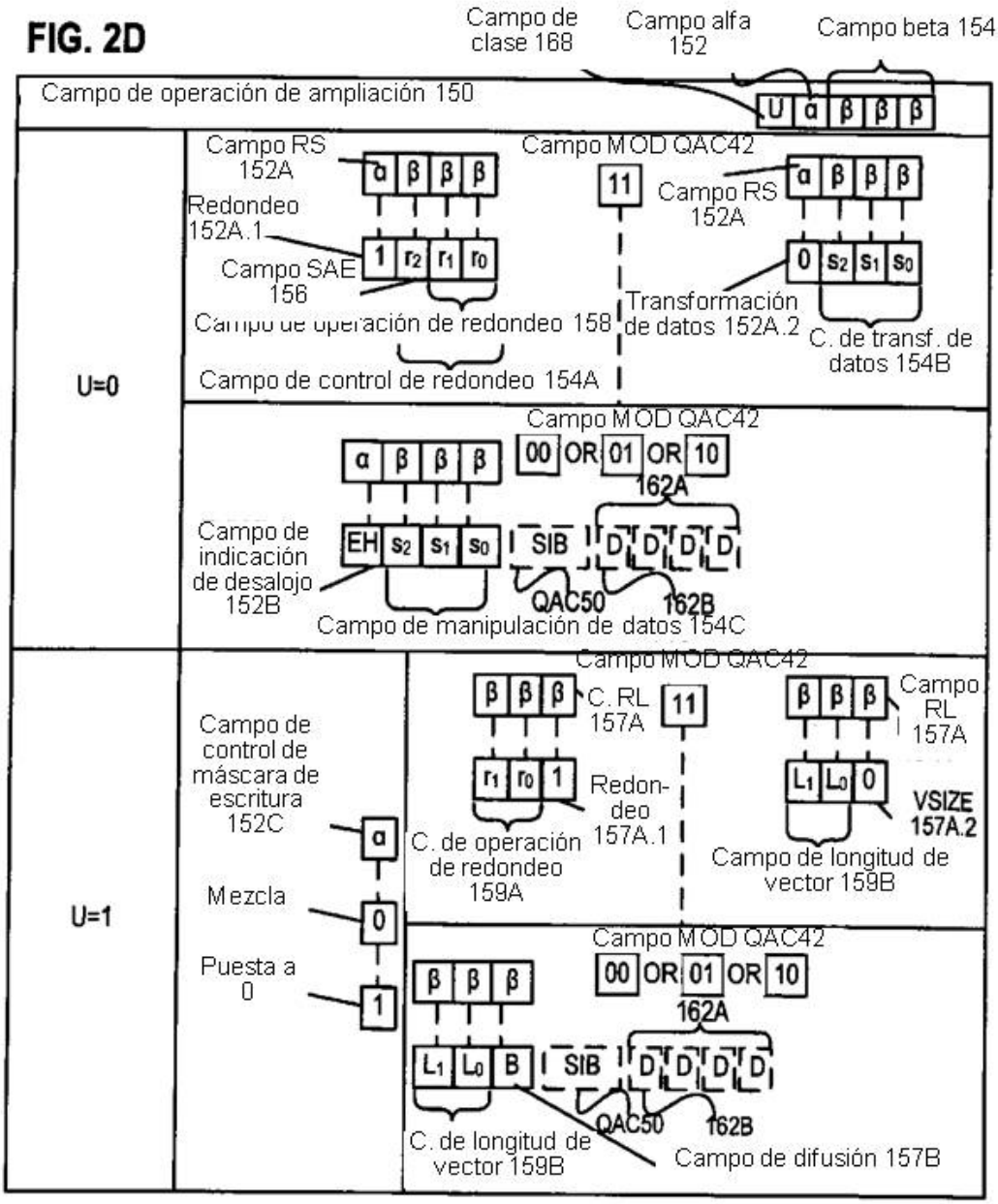


FIG. 2A
Prefijo EVEX 202
Campo de tamaño de elemento de datos QAB64
Campo de clase QAB68
REX' 210
Campo Beta QAB54
Campo alfa QAB52
Campo de codificación de prefijo 225
Campo de máscara de escritura QAB70
Campo VVVV 220
Campo Opcode real 230
Campo de factor de desplazamiento QAB62B (disp8*N cuando MOD=01). Denominado disp8*N, pero mantiene únicamente el factor de desplazamiento que se multiplica por N
C. de desp. QAB62A (disp32 cuando MOD=10)
QAB72
IMM8
Byte R/M MOD
Byte SIB
Campo de formato QAB40
REX 205
Mapeo de Opcode 215
formato 200 específico de instrucción adaptada para vectores
FIG. 2B
Campo de opcode completo QAB74
Campo de formato QAB40
Mapeo de Opcode 215
Campo de tamaño de elemento de datos QAB64
Campo de codificación de prefijo 225
Campo de operación base QAB42
Campo de opcode real 230
FIG. 2C
Campo de índice de registro QAB44
REX 205
REX' 210
Campo VVVV 220

FIG. 2D


Campo de clase 168
Campo alfa 152
Campo beta 154
Campo de operación de ampliación 150
U α β β β
U=0
Campo RS 152A
α β β β
Redondeo 152A.1
1 r2 r1 r0
Campo SAE 156
Campo de operación de redondeo 158
Campo de control de redondeo 154A
Campo MOD QAC42
11
Campo RS 152A
α β β β
0 s2 s1 s0
Transformación de datos 152A.2
C. de transf. de datos 154B
Campo MOD QAC42
α β β β
00 OR 01 OR 10
162A
Campo de indicación de desalojo 152B
EH s2 s1 s0
SIB
D D D D
QAC50
162B
Campo de manipulación de datos 154C
U=1
Campo de control de máscara de escritura 152C
α
Mezcla
0
Puesta a 0
1
Campo MOD QAC42
β β β
C. RL 157A
11
r1 r0 1
Redon-deo 157A.1
C. de operación de redondeo 159A
β β β
Campo RL 157A
L1 L0 0
VSIZE 157A.2
Campo de longitud de vector 159B
Campo MOD QAC42
β β β
00 OR 01 OR 10
162A
L1 L0 B
SIB
D D D D
QAC50
162B
C. de longitud de vector 159B
Campo de difusión 157B

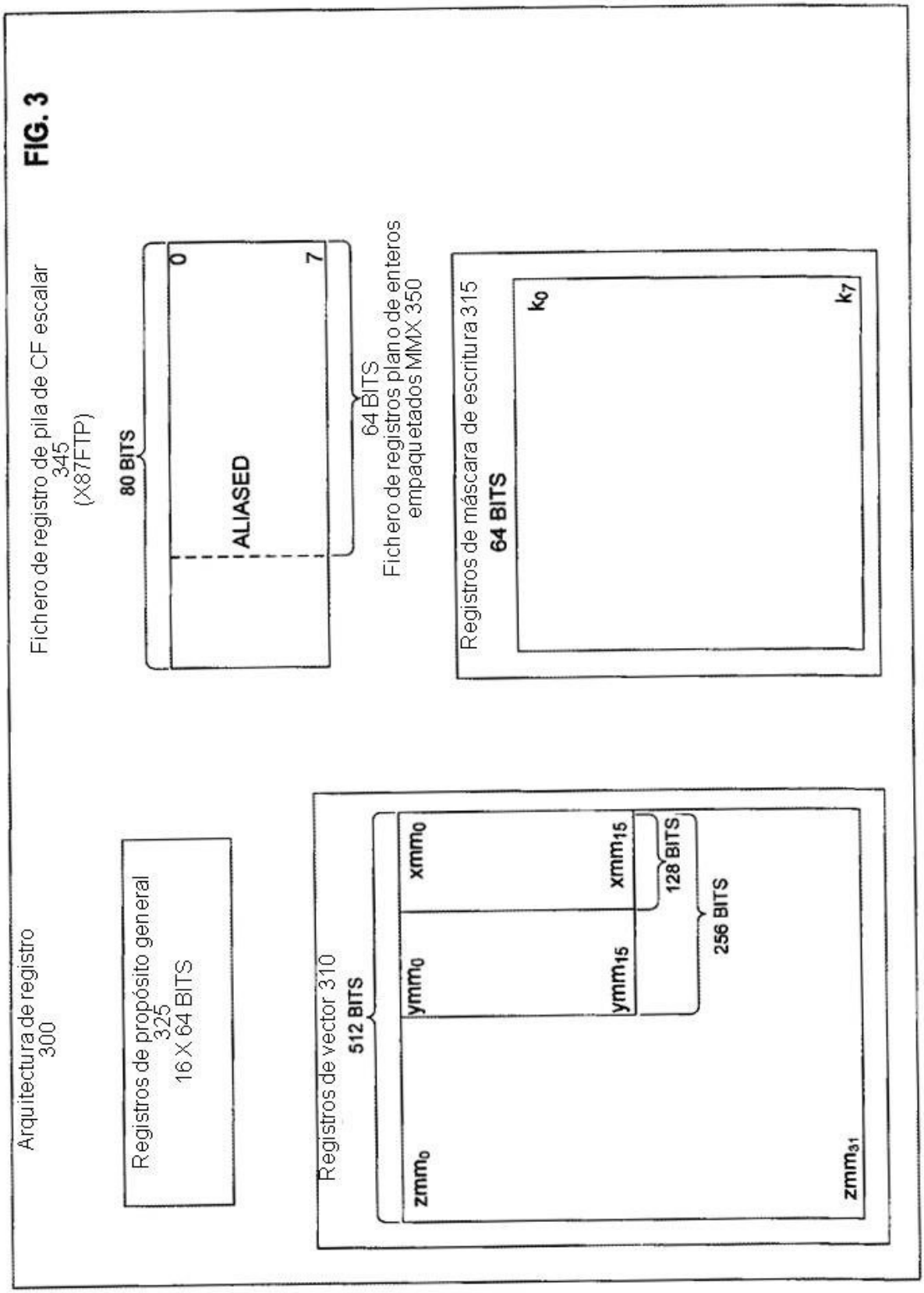

Arquitectura de registro
300
Registros de propósito general
325
16 X 64 BITS
Registros de vector 310
512 BITS
zmm0
ymm0
xmm0
ymm15
xmm15
128 BITS
256 BITS
zmm31
Fichero de registro de pila de CF escalar
345
(X87FTP)
80 BITS
0
ALIASED
7
64 BITS
Fichero de registros plano de enteros
empaquetados MMX 350
Registros de máscara de escritura 315
64 BITS
k0
k7

FIG. 3

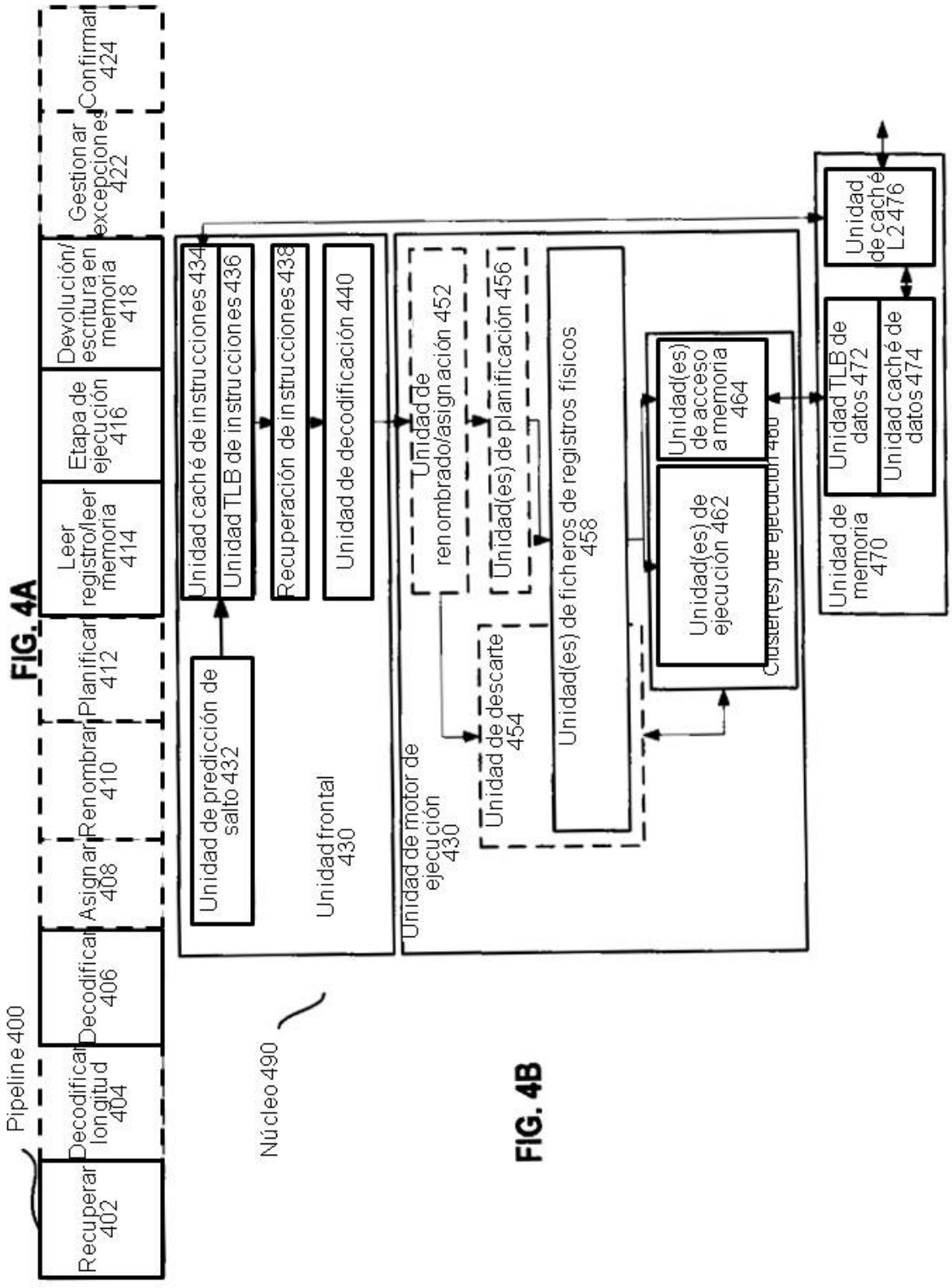

Pipeline 400
Recuperar 402
Decodificar longitud 404
Decodificar 406
Asignar 408
Renombrar 410
Planificar 412
Leer registro/leer memoria 414
Etapa de ejecución 416
Devolución/ escritura en memoria 418
Gestionar excepciones 422
Confirmar 424
Unidad caché de instrucciones 434
Unidad TLB de instrucciones 436
Recuperación de instrucciones 438
Unidad de decodificación 440
Unidad de predicción de salto 432
Unidad frontal 430
Unidad de renombrado/asignación 452
Unidad(es) de planificación 456
Unidad de descarte 454
Unidad(es) de ficheros de registros físicos 458
Unidad(es) de ejecución 462
Unidad(es) de acceso a memoria 464
Clúster(es) de ejecución 460
Unidad de motor de ejecución 430
Unidad de memoria 470
Unidad TLB de datos 472
Unidad caché de datos 474
Unidad de caché L2 476
Núcleo 490

FIG. 4A

FIG. 4B

FIG. 5A

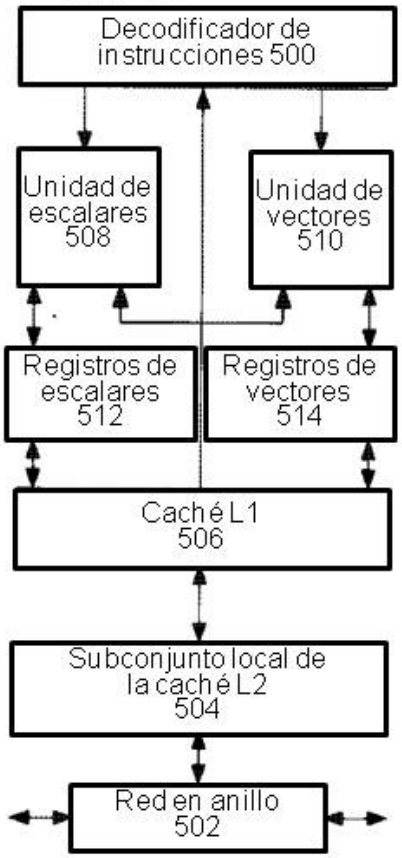
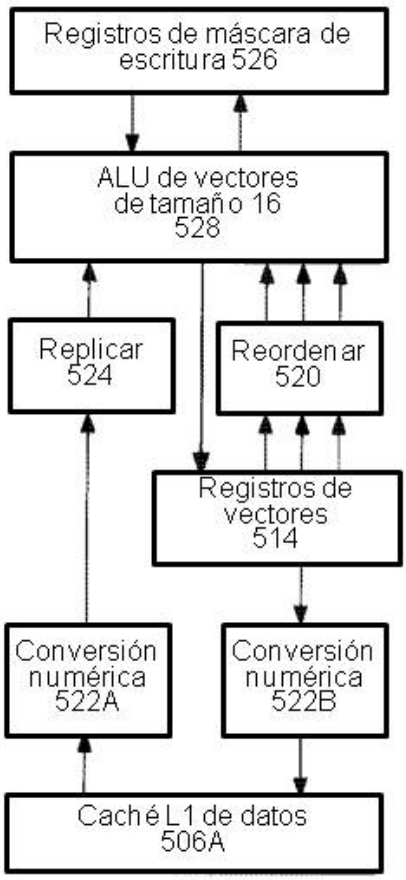

Decodificador de instrucciones 500
Unidad de escalares 508
Unidad de vectores 510
Registros de escalares 512
Registros de vectores 514
Caché L1 506
Subconjunto local de la caché L2 504
Red en anillo 502

FIG. 5B


Registros de máscara de escritura 526
ALU de vectores de tamaño 16 528
Replicar 524
Reordenar 520
Registros de vectores 514
Conversión numérica 522A
Conversión numérica 522B
Caché L1 de datos 506A

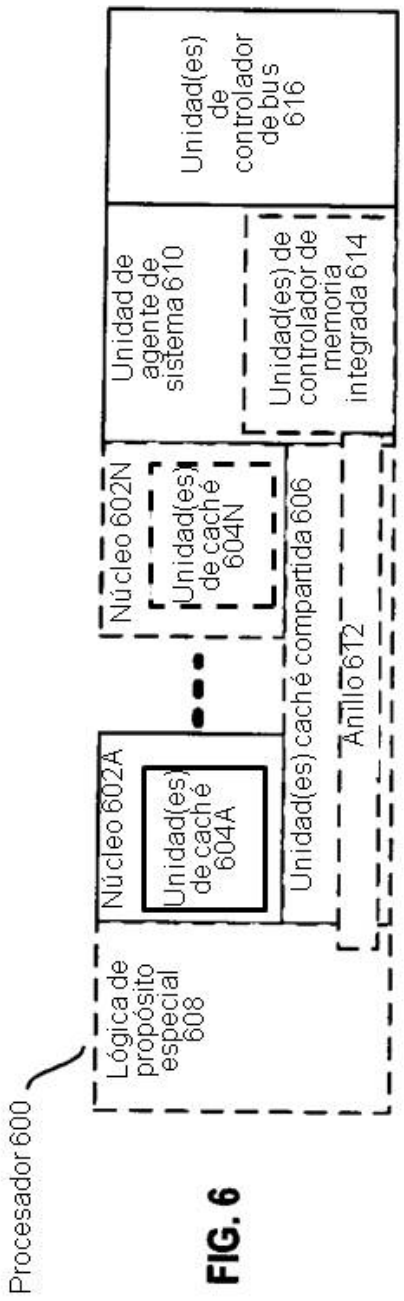

Procesador 600
Lógica de propósito especial 608
Núcleo 602A
Unidad(es) de caché 604A
Núcleo 602N
Unidad(es) de caché 604N
Unidad(es) caché compartida 606
Anillo 612
Unidad de agente de sistema 610
Unidad(es) de controlador de memoria integrada 614
Unidad(es) de controlador de bus 616

FIG. 6

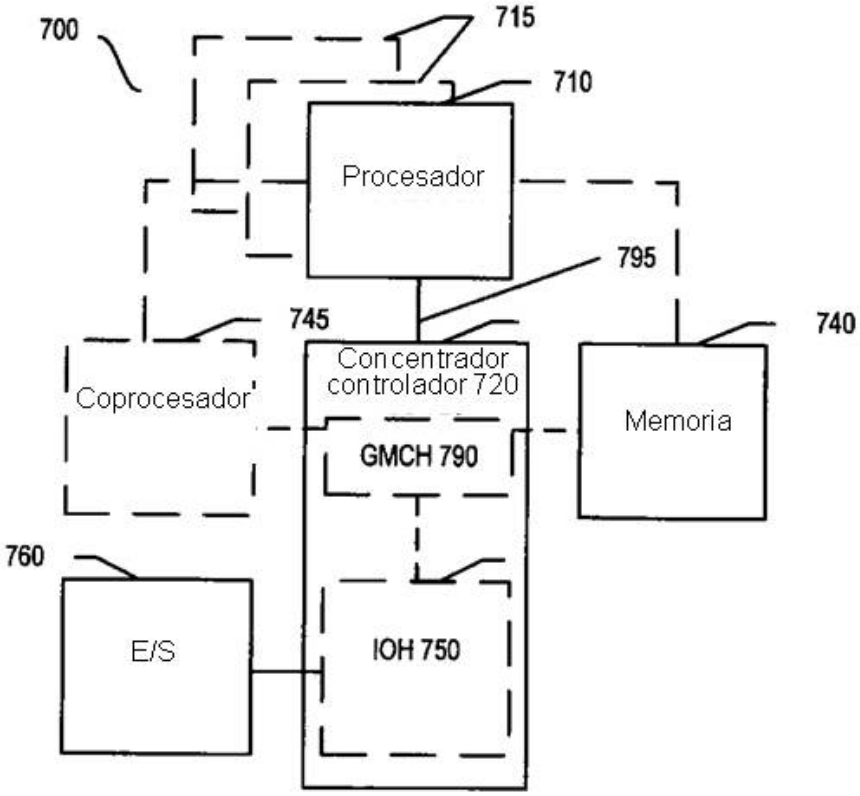

700
715
710
Procesador
795
745
740
Concentrador controlador 720
Coprocesador
Memoria
GMCH 790
760
E/S
IOH 750

FIG. 7

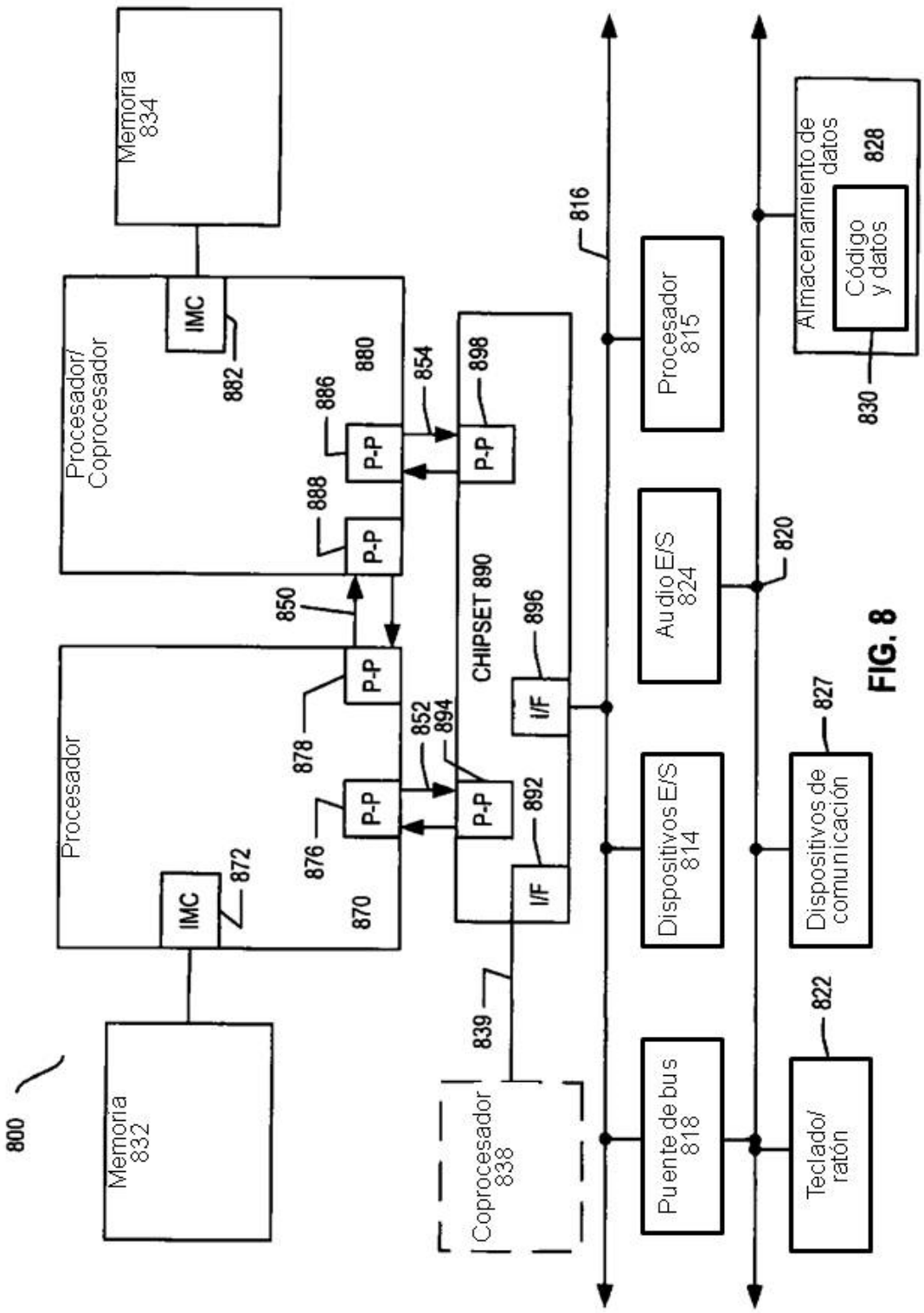

800
Memoria 832
Memoria 834
Procesador
IMC
872
870
876
878
P-P
852
894
892
I/F
839
Coprocesador 838
Procesador/ Coprocesador
IMC
882
880
886
888
P-P
854
898
850
CHIPSET 890
896
816
Procesador 815
Audio E/S 824
820
Dispositivos E/S 814
Puente de bus 818
822
Dispositivos de comunicación
827
Teclado/ ratón
Almacenamiento de datos
828
Código y datos
830

FIG. 8

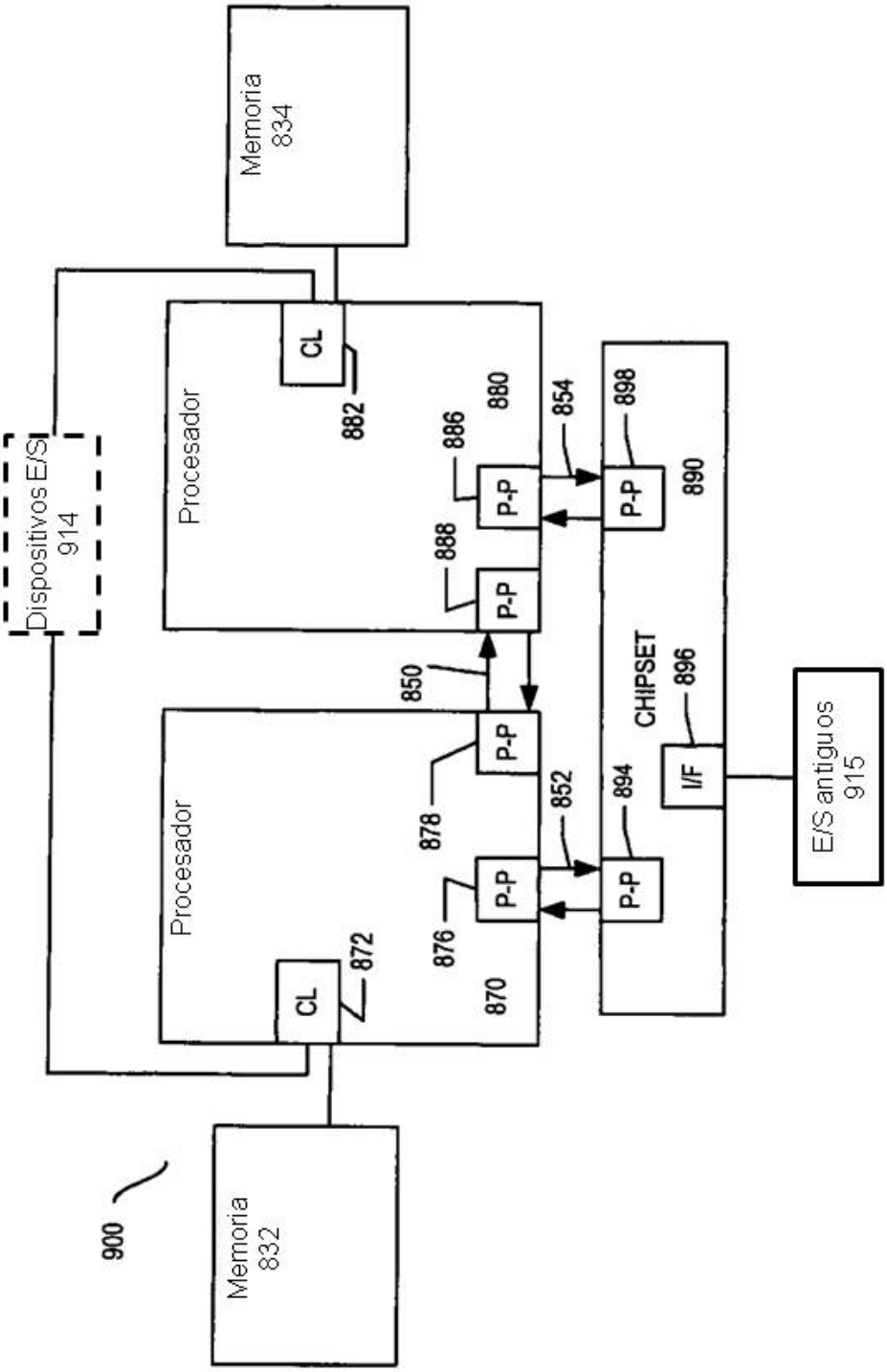

900
Memoria 832
Memoria 834
Dispositivos E/S 914
Procesador
CL
872
Procesador
CL
882
870
876
878
880
886
888
850
852
854
CHIPSET
890
894
896
898
P-P
I/F
E/S antiguos 915

FIG. 9

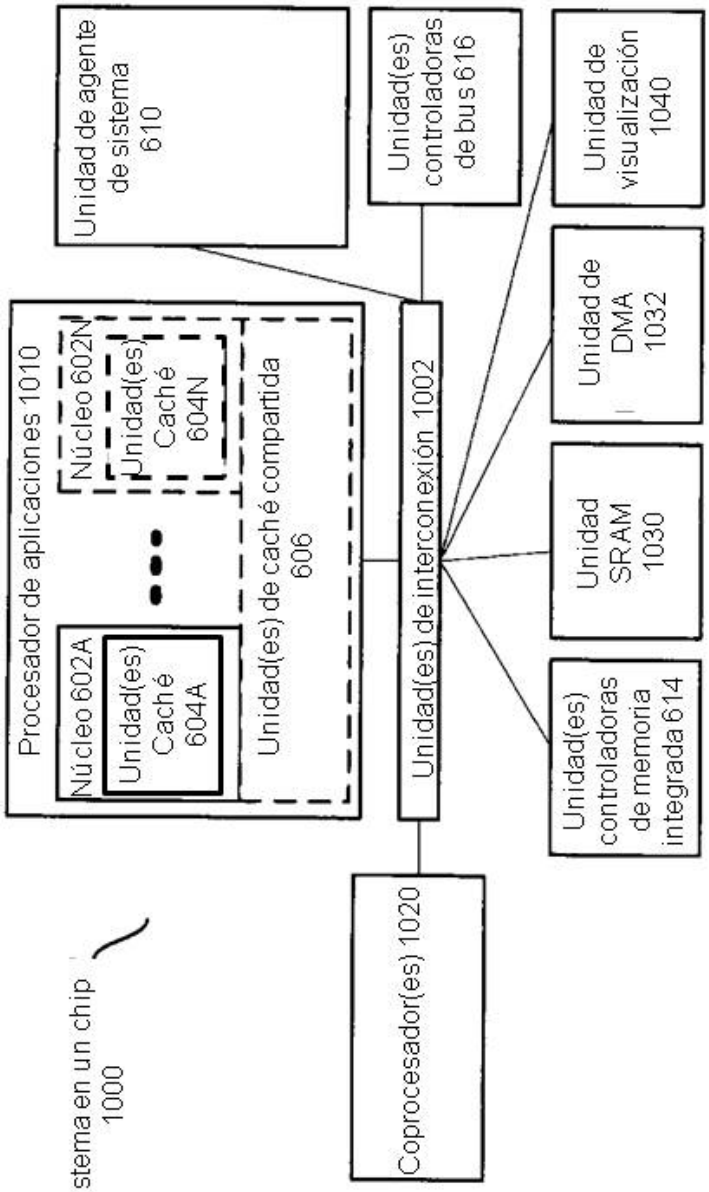

Sistema en un chip
1000
Procesador de aplicaciones 1010
Núcleo 602A
Unidad(es) Caché 604A
Núcleo 602N
Unidad(es) Caché 604N
Unidad(es) de caché compartida 606
Unidad de agente de sistema 610
Unidad(es) controladoras de bus 616
Unidad de visualización 1040
Unidad de DMA 1032
Unidad SRAM 1030
Unidad(es) controladoras de memoria integrada 614
Unidad(es) de interconexión 1002
Coprocesador(es) 1020

FIG. 10

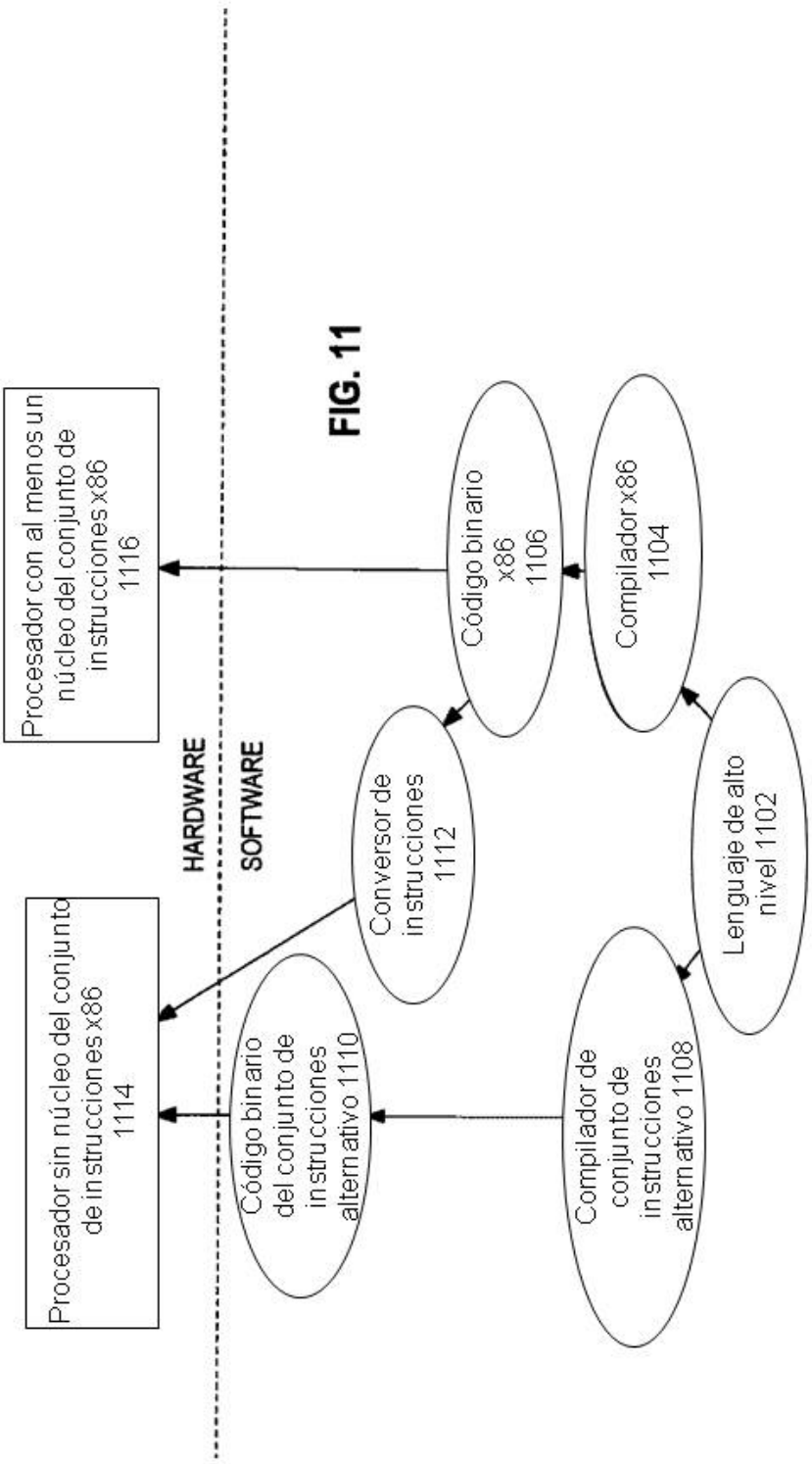

Procesador con al menos un núcleo del conjunto de instrucciones x86 1116
HARDWARE
SOFTWARE
Procesador sin núcleo del conjunto de instrucciones x86 1114
Código binario x86 1106
Compilador x86 1104
Conversor de instrucciones 1112
Lenguaje de alto nivel 1102
Código binario del conjunto de instrucciones alternativo 1110
Compilador de conjunto de instrucciones alternativo 1108

FIG. 11

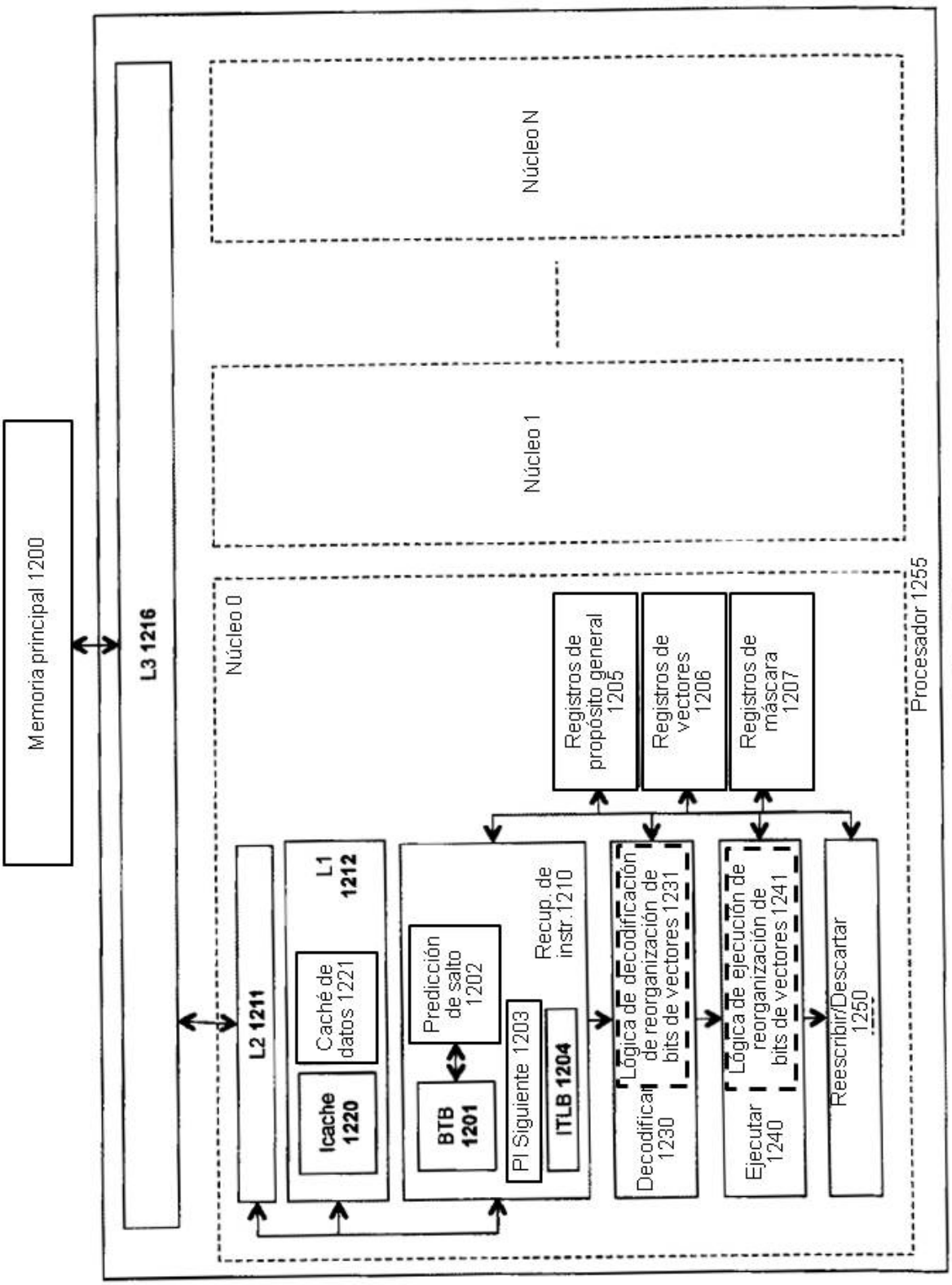

Memoria principal 1200
L3 1216
Núcleo N
Núcleo 1
Núcleo 0
Procesador 1255
L2 1211
L1 1212
Caché de datos 1221
Icache 1220
Predicción de salto 1202
BTB 1201
PI Siguiente 1203
ITLB 1204
Recup. de instr.1210
Registros de propósito general 1205
Registros de vectores 1206
Registros de máscara 1207
Lógica de decodificación de reorganización de bits de vectores 1231
Decodificar 1230
Lógica de ejecución de reorganización de bits de vectores 1241
Ejecutar 1240
Reescribir/Descartar 1250

Fig. 12

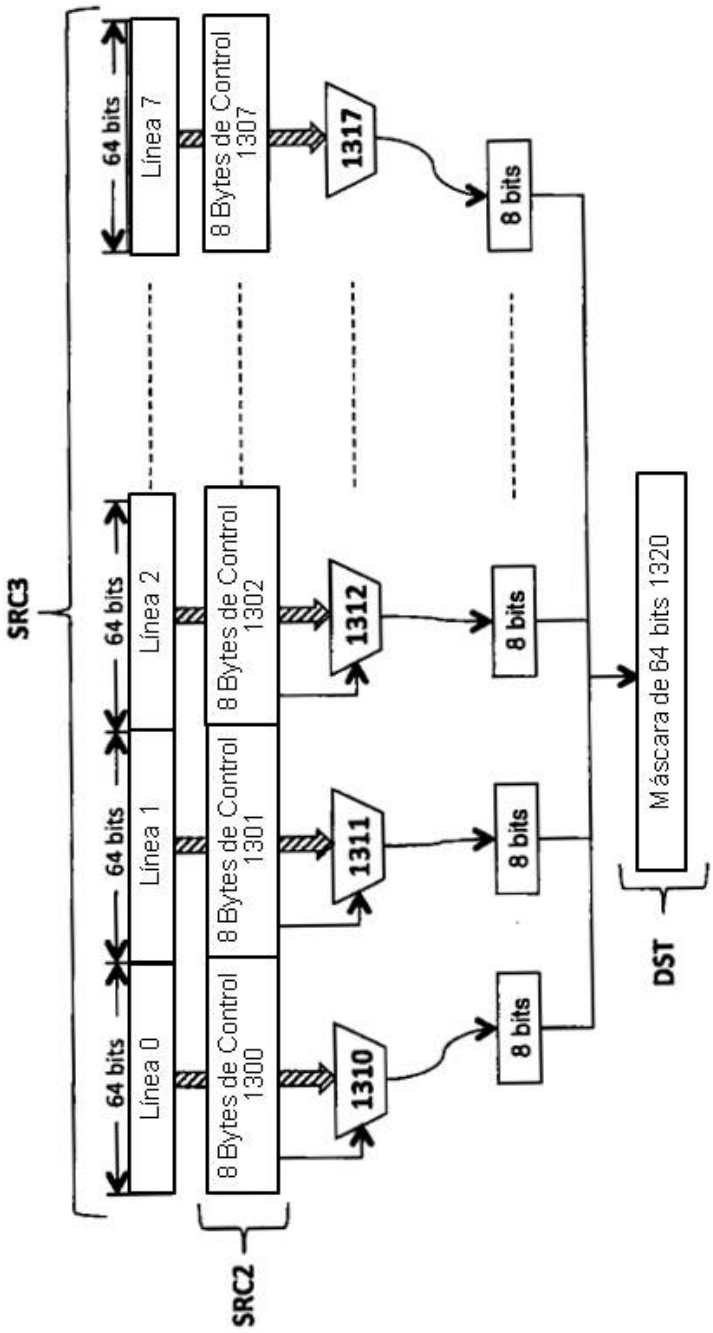

SRC3
64 bits
Línea 0
64 bits
Línea 1
64 bits
Línea 2
64 bits
Línea 7
8 Bytes de Control 1300
8 Bytes de Control 1301
8 Bytes de Control 1302
8 Bytes de Control 1307
SRC2
1310
1311
1312
1317
8 bits
8 bits
8 bits
8 bits
DST
Máscara de 64 bits 1320

Fig. 13

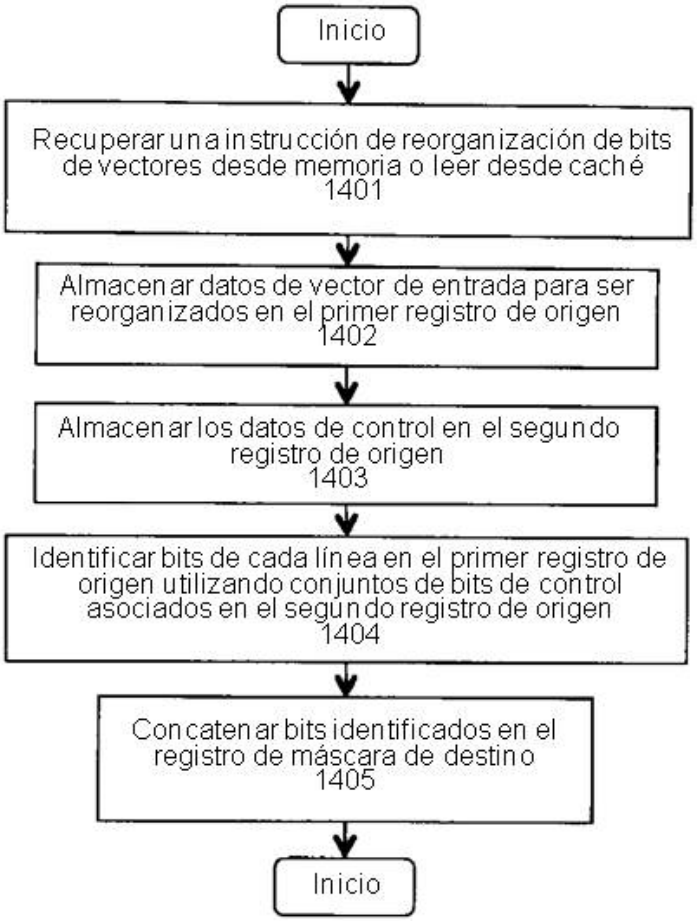

Inicio
Recuperar una instrucción de reorganización de bits de vectores desde memoria o leer desde caché
1401
Almacenar datos de vector de entrada para ser reorganizados en el primer registro de origen
1402
Almacenar los datos de control en el segundo registro de origen
1403
Identificar bits de cada línea en el primer registro de origen utilizando conjuntos de bits de control asociados en el segundo registro de origen
1404
Concatenar bits identificados en el registro de máscara de destino
1405
Inicio

Fig. 14