(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 779 033**

(51) Int. CI.:

G06F 9/455 (2008.01)
G06F 8/41 (2008.01)
G06F 8/52 (2008.01)
G06F 9/30 (2008.01)
G06F 9/34 (2008.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **15.11.2012 PCT/IB2012/056436**

(87) Fecha y número de publicación internacional: **19.09.2013 WO13136144**

(96) Fecha de presentación y número de la solicitud europea: **15.11.2012 E 12871580 (2)**

(97) Fecha y número de publicación de la concesión europea: **19.02.2020 EP 2769301**

(54) Título: **Transformar especificadores de instrucción no contiguos a especificadores de instrucción contiguos**

(30) Prioridad:

15.03.2012 US 201213421657

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
13.08.2020

(73) Titular/es:

INTERNATIONAL BUSINESS MACHINES CORPORATION (100.0%)
New Orchard Road
Armonk, New York 10504, US

(72) Inventor/es:

GSCHWIND, MICHAEL, KARL

(74) Agente/Representante:

ISERN JARA, Jorge

DESCRIPCIÓN

Transformar especificadores de instrucción no contiguos a especificadores de instrucción contiguos

Antecedentes

La invención se refiere, en general, a emulación en un entorno informático, y, en particular, a emulación de especificadores en instrucciones.

La emulación imita funciones en una arquitectura informática, denominada como una arquitectura objetivo. La arquitectura objetivo se diferencia de una arquitectura informática, denominada como una arquitectura de origen, para la que se definen las funciones. Por ejemplo, una instrucción escrita para la z/Architecture proporcionada por International Business Machines Corporation, Armonk, Nueva York, puede traducirse y representare como una o más instrucciones de una arquitectura diferente, tal como PowerPC, también ofrecida por International Business Machines Corporation, u otra arquitectura ofrecida por International Business Machines Corporation u otra compañía. Estas instrucciones traducidas realizan la misma o una función similar que la instrucción que se está traduciendo.

Hay diferentes tipos de emulación, que incluyen interpretación y traducción. Con interpretación, se leen datos que representan una instrucción, y a medida que se decodifica cada instrucción, se ejecuta. Cada instrucción se ejecuta cada vez que se hace referencia. Sin embargo, con traducción, también denominada como traducción binaria o recompilación, se traducen secuencias de instrucciones del conjunto de instrucciones de una arquitectura informática al conjunto de instrucciones de otra arquitectura informática.

Hay múltiples tipos de traducción, incluyendo traducción estática y traducción dinámica. En traducción estática, el código de una instrucción de una arquitectura se convierte a código que se ejecuta en la otra arquitectura sin ejecutar previamente el código. En contraste, en traducción dinámica, se ejecuta y se traduce al menos una sección del código, y el resultado se coloca en una caché para su ejecución posterior por un procesador de la arquitectura informática objetivo.

El documento US 2006/0195680 describe diversos formatos de instrucción de máquina que comprenden campos opcode no contiguos.

El documento US 7793081 describe un método para procesar código de instrucción que incluye procesar una instrucción de anchura fija de un conjunto de instrucciones de anchura fija usando un especificador de registro no contiguo de una especificación de registro no contigua. La instrucción de anchura fija incluye el especificador de registro no contiguo.

Sumario

Se tratan las desventajas de la técnica anterior y se proporcionan ventajas a través del suministro de un producto de programa informático para transformar especificadores de instrucción de un entorno informático. El producto de programa informático incluye un medio de almacenamiento legible por ordenador legible por un circuito de procesamiento y que almacena instrucciones para su ejecución por el circuito de procesamiento para realizar un método que comprende: determinar a partir de una primera instrucción definida para una primera arquitectura informática que la primera instrucción incluye un especificador de operando de registro no contiguo que tiene una primera porción y una segunda porción, no contigua con la primera porción; obtener el especificador de operando de registro no contiguo a partir de la primera instrucción que comprende obtener la primera porción de un primer campo de la primera instrucción y la segunda porción de un segundo campo de la primera instrucción, el primer campo separado del segundo campo por al menos un campo intermedio, en donde una porción del opcode de la primera instrucción especifica el primer campo y el segundo campo usados para designar el especificador no contiguo; generar un especificador de operando de registro contiguo usando la primera porción y la segunda porción obtenidas de la primera instrucción, usando la generación una o más reglas almacenadas en memoria o almacenamiento externo dependiendo del formato de la instrucción especificado por el opcode de la primera instrucción; usar el especificador de operando de registro contiguo en lugar del especificador de operando de registro no contiguo para indicar un recurso a usar en la ejecución de una segunda instrucción, la segunda instrucción definida para una segunda arquitectura informática diferente de la primera arquitectura informática y emular una función de la primera instrucción; y ejecutar la segunda instrucción para emular la función de la primera instrucción, usando la ejecución el recurso indicado por el especificador de operando de registro contiguo sin tener más en cuenta el especificador de operando de registro no contiguo.

Los métodos y sistemas relacionados con uno o más aspectos de la presente invención se describen y reivindican también en el presente documento. Además, los servicios relacionados con uno o más aspectos de la presente invención se describen también y pueden reivindicarse en el presente documento.

Se realizan características y ventajas adicionales a través de las técnicas de la presente invención. Otras realizaciones y aspectos de la invención se describen en detalle en el presente documento y se consideran una parte de la invención

reivindicada.

Breve descripción de los dibujos

Las realizaciones de la presente invención se describirán ahora, a modo de ejemplo únicamente, con referencia a los dibujos adjuntos en los que:

La Figura 1 representa un ejemplo de un entorno informático para incorporar y usar uno o más aspectos de la presente invención;
La Figura 2 representa adicionalmente detalles de la memoria de la Figura 1, de acuerdo con un aspecto de la presente invención;
La Figura 3 representa una realización de una vista general de un proceso de emulación que emplea una o más de interpretación y traducción;
La Figura 4 representa un ejemplo de lógica asociada con el bloque de interpretación referenciado en la Figura 3;
La Figura 5 representa un ejemplo de lógica asociada con el bloque de traducción referenciado en la Figura 3;
La Figura 6 representa otra realización de una vista general de un proceso de emulación que emplea una o más de interpretación y traducción modificadas de acuerdo con un aspecto de la presente invención;
La Figura 7A representa un ejemplo de lógica asociada con el bloque de interpretación referenciado en la Figura 6, de acuerdo con un aspecto de la presente invención;
La Figura 7B representa una realización de la lógica para transformar un especificador no contiguo a un especificador contiguo, de acuerdo con un aspecto de la presente invención;
La Figura 8 representa un ejemplo de lógica asociada con el bloque de traducción referenciado en la Figura 6, de acuerdo con un aspecto de la presente invención;
La Figura 9A representa una realización de transformación de un especificador no contiguo en una instrucción de carga de vector de una arquitectura informática para un especificador contiguo en una instrucción indexada de vector de carga de otra arquitectura informática, de acuerdo con un aspecto de la presente invención;
La Figura 9B representa otro ejemplo de la transformación de la Figura 9A, que incluye la asignación de un registro particular al especificador contiguo, de acuerdo con un aspecto de la presente invención;
La Figura 10 representa un ejemplo de un fichero de registro, de acuerdo con un aspecto de la presente invención;
La Figura 11 representa un ejemplo de transformación de especificadores no contiguos a especificadores contiguos al asignar en memoria durante la emulación, de acuerdo con un aspecto de la presente invención;
La Figura 12 representa una realización de un producto de programa informático que incorpora uno o más aspectos de la presente invención;
La Figura 13 representa una realización de un sistema de ordenador de anfitrión para incorporar y usar uno o más aspectos de la presente invención;
La Figura 14 representa un ejemplo adicional de un sistema informático para incorporar y usar uno o más aspectos de la presente invención;
La Figura 15 representa otro ejemplo de un sistema informático que comprende una red informática para incorporar y usar uno o más aspectos de la presente invención;
La Figura 16 representa una realización de diversos elementos de un sistema informático para incorporar y usar uno o más aspectos de la presente invención;
La Figura 17A representa una realización de la unidad de ejecución del sistema informático de la Figura 16 para incorporar y usar uno o más aspectos de la presente invención;
La Figura 17B representa una realización de la unidad de ramal del sistema informático de la Figura 16 para incorporar y usar uno o más aspectos de la presente invención;
La Figura 17C representa una realización de la unidad de carga/almacén del sistema informático de la Figura 16 para incorporar y usar uno o más aspectos de la presente invención; y
La Figura 18 representa una realización de un sistema informático de anfitrión emulado para incorporar y usar uno o más aspectos de la presente invención.

Descripción detallada

De acuerdo con un aspecto de la presente invención, se proporciona una técnica para facilitar la emulación de instrucciones que incluyen especificadores no contiguos. Un especificador no contiguo especifica un recurso de una instrucción, tal como un registro, usando múltiples campos de la instrucción. Por ejemplo, múltiples campos de la instrucción (por ejemplo, dos campos) incluyen bits que juntos designan un registro particular a usar por la instrucción.

En un aspecto particular de la invención, se proporciona una técnica para transformar especificadores no contiguos de instrucciones definidas en una arquitectura de sistema informático (por ejemplo, z/Architecture ofrecida por International Business Machines Corporation) a especificadores contiguos usables por instrucciones definidas en otra arquitectura de sistema informático (por ejemplo, la arquitectura PowerPC ofrecida por International Business Machines Corporation). Las instrucciones definidas en la otra arquitectura de sistema informático emulan las instrucciones definidas para la arquitectura de sistema informático.

Se describe una realización de un entorno informático que proporciona emulación con referencia a la Figura 1. En un ejemplo, un entorno 100 informático incluye, por ejemplo, una unidad 102 de procesamiento central nativa, una

memoria 104, y uno o más dispositivos de entrada/salida y/o interfaces 106 acoplados entre sí mediante, por ejemplo, uno o más buses 108 y/u otras conexiones. Como ejemplos, el entorno 100 informático puede incluir un procesador PowerPC, un servidor pSeries o un servidor xSeries ofrecido por International Business Machines Corporation, Armonk, Nueva York; un HP Superdome con procesadores Intel Itanium II ofrecido por Hewlett Packard Co., Palo Alto, California; y/u otras máquinas basadas en arquitecturas ofrecidas por International Business Machines Corporation, Hewlett Packard, Intel, Oracle u otros.

La unidad 102 de procesamiento central nativa incluye uno o más registros 110 nativos, tales como uno o más registros de fin general y/u uno o más registros de fin especial usados durante el procesamiento en el entorno. Estos registros incluyen información que representa el estado del entorno en cualquier punto en el tiempo particular.

Además, la unidad 102 de procesamiento central nativa ejecuta instrucciones y código que se almacenan en la memoria 104. En un ejemplo particular, la unidad de procesamiento central ejecuta el código 112 del emulador almacenado en la memoria 104. Este código posibilita que el entorno de procesamiento configurado en una arquitectura emule otra arquitectura. Por ejemplo, el código 112 del emulador permite que máquinas basadas en arquitecturas distintas de la z/Architecture, tal como procesadores PowerPC, servidores pSeries, servidores xSeries, servidores HP Superdome u otros, emulen la z/Architecture y ejecuten software e instrucciones desarrolladas basándose en la z/Architecture.

Se describen detalles adicionales con respecto al código 112 emulador con referencia a la Figura 2. Las instrucciones 200 de invitado comprenden instrucciones de software (por ejemplo, instrucciones de máquina) que se desarrollaron para ejecutarse en una arquitectura distinta de la CPU 102 nativa. Por ejemplo, las instrucciones 200 de invitado pueden haberse diseñado para ejecutarse en un procesador de z/Architecture, pero en su lugar, se están emulando en la CPU 102 nativa, que puede ser, por ejemplo, un procesador PowerPC u otro tipo de procesador. En un ejemplo, el código 112 emulador incluye una unidad 202 de captura de instrucción para obtener una o más instrucciones 200 de invitado de la memoria 104, y para proporcionar opcionalmente almacenamiento en memoria intermedia local para las instrucciones obtenidas. También incluye una rutina 204 de traducción de instrucción para determinar el tipo de instrucción de invitado que se ha obtenido y para traducir la instrucción de invitado en una o más instrucciones 206 nativas correspondientes. Esta traducción incluye, por ejemplo, identificar la función a realizarse por la instrucción de invitado (por ejemplo, mediante el opcode) y elegir la instrucción o instrucciones nativas para realizar esa función.

Además, el emulador 112 incluye una rutina 210 de control de emulación para provocar que se ejecuten las instrucciones nativas. La rutina 210 de control de emulación puede provocar que la CPU 102 nativa ejecute una rutina de instrucciones nativas que emulan una o más instrucciones de invitado previamente obtenidas, y en la conclusión de tal ejecución, devolver el control a la rutina de captura de instrucción para emular la obtención de la siguiente instrucción de invitado o un grupo de instrucciones de invitado. La ejecución de las instrucciones 206 nativas puede incluir cargar datos en un registro de la memoria 104; almacenar datos de vuelta a la memoria de un registro; o realizar algún tipo de operación aritmética o lógica, como se determina por la rutina de traducción.

Cada rutina se implementa, por ejemplo, en software, que se almacena en memoria y se ejecuta por la unidad 102 de procesamiento central nativa. En otros ejemplos, una o más de las rutinas u operaciones se implementan en firmware, hardware, software o alguna combinación de los mismos. Los registros del procesador emulado pueden emularse usando los registros 110 de la CPU nativa o usando ubicaciones en la memoria 104. En las realizaciones, las instrucciones 200 de invitado, instrucciones 206 nativas y el código 112 emulador pueden residir en la misma memoria o pueden distribuirse entre diferentes dispositivos de memoria.

Como se usa en el presente documento, el firmware incluye, por ejemplo, el microcódigo, el milicode y/o macrocódigo del procesador. Incluye, por ejemplo, las instrucciones de nivel de hardware y/o estructuras de datos usadas en la implementación de código máquina de nivel superior. En una realización, incluye, por ejemplo, código propietario que se entrega típicamente como microcódigo que incluye software confiable o microcódigo específico para el hardware subyacente y controla el acceso del sistema operativo al sistema hardware.

En un ejemplo, una instrucción 200 de invitado que se obtiene, traduce y ejecuta, es una o más de las instrucciones descritas en el presente documento. La instrucción, que es de una arquitectura (por ejemplo, la z/Architecture) se captura de memoria, se traduce y representa como una secuencia de instrucciones 206 nativas de otra arquitectura (por ejemplo, PowerPC, pSeries, xSeries, Intel, etc.). Estas instrucciones nativas se ejecutan a continuación.

Se describen detalles adicionales con respecto a la emulación con referencia a las Figuras 3-5. En particular, la Figura 3 representa una realización de una vista general de un proceso de emulación que emplea una o más de interpretación y traducción; la Figura 4 representa una realización de la lógica asociada con interpretación referenciada en la Figura 3 (Técnica 2000); y la Figura 5 representa una realización de la lógica asociada con traducción binaria referenciada en la Figura 3 (Técnica 3000). En este ejemplo particular, las instrucciones escritas para la z/Architecture se están traduciendo a instrucciones PowerPC. Sin embargo, las mismas técnicas son aplicables para emulación de la z/Architecture a otras arquitecturas objetivo; de otras arquitecturas de origen a la arquitectura PowerPC; y/o de otras arquitecturas de origen a otras arquitecturas objetivo.

Haciendo referencia a la Figura 3, durante la emulación, se obtiene e interpreta una instrucción, denominada como la instrucción X, como se describe en detalle adicional con referencia a la Figura 4, ETAPA 300. Se actualizan diversas estadísticas relacionadas con la instrucción interpretada, ETAPA 302, y a continuación, el procesamiento continúa a la siguiente instrucción, que se vuelve la instrucción X en la lógica, ETAPA 304. Se realiza una determinación en cuanto a si esta siguiente instrucción tiene un punto de entrada previamente traducido, CONSULTA 306. Si no, se realiza una determinación adicional en cuanto a si esta siguiente instrucción ha sido vista por N (por ejemplo, 15) veces, CONSULTA 308. Es decir, se ha visto esta instrucción con la suficiente frecuencia para optimizar la ejecución, por ejemplo, realizando compilación justo a tiempo (JIT) del código, que proporciona un punto de entrada para su uso posterior. Si esta función no se ha observado N veces, tal como 15 veces, a continuación, el procesamiento continúa con la ETAPA 300. De lo contrario, el procesamiento continúa formando un grupo de instrucciones y traduciendo el grupo de instrucciones para una arquitectura a otra arquitectura, ETAPA 310. Un ejemplo de realización de esta traducción se describe con referencia a la Figura 5. Posterior a la formación y traducción del grupo, el grupo se ejecuta, ETAPA 312, y el procesamiento continúa a la ETAPA 304.

Volviendo a CONSULTA 306, si hay un punto de entrada traducido existente para la instrucción, el procesamiento continúa con la ejecución del grupo en el punto de entrada, ETAPA 312.

Se describen detalles adicionales para interpretar una instrucción (Técnica 2000) con referencia a la Figura 4. Inicialmente, se lee una instrucción en el siguiente contador de programa (PC), ETAPA 400. Esta instrucción se analiza, y se extraen los campos de opcode, registro e inmediatos, ETAPA 402. A continuación, se realiza un ramal al código que emula el comportamiento que corresponde al opcode extraído, ETAPA 404. El código emulado se realiza a continuación, ETAPA 406.

Se describen detalles adicionales con respecto a traducir instrucciones en un grupo (Técnica 3000) con referencia a la Figura 5. Inicialmente, se lee una instrucción en un grupo de instrucciones predefinido, ETAPA 500. En un ejemplo, el grupo puede formarse usando una diversidad de maneras. De acuerdo con una realización, se forma un grupo para abarcar una única ruta de ejecución a lo largo de la ruta más probable. En otra realización, se forma un grupo para abarcar una de las rutas de ejecución anteriores, o la ruta de ejecución actual, basándose en el estado de la arquitectura emulada. En otra realización, se supone que no se toman todos los ramales. En otra realización más, están incluidas múltiples rutas en un grupo, tal como todas las rutas que empiezan desde el punto de inicio del grupo. En otra realización, todas las instrucciones hasta, y que incluyen el primer ramal, se añaden a un grupo (es decir, un grupo corresponde a una pieza de línea recta de código también conocido comúnmente como un "bloque básico"). En cada realización, ha de realizarse una decisión en cuanto a cuándo y dónde finalizar un grupo. En una realización, se finaliza un grupo después de un número fijo de instrucciones. En otra realización, el grupo se finaliza después de que una probabilidad acumulativa de alcanzar una instrucción está por debajo de un umbral dado. En algunas realizaciones, se detiene un grupo inmediatamente cuando se ha alcanzado una condición de detención. En otro conjunto de las realizaciones, se detiene un grupo únicamente en un "punto de detención" bien definido, por ejemplo, una instrucción definida, una alineación de inicio de grupo específica u otra condición.

Posteriormente, la instrucción se analiza, y se extraen los campos de opcode, registro e inmediatos de la instrucción, ETAPA 502. A continuación, se proporciona una representación interna de la información extraída, ETAPA 504. Esta representación interna es un formato de la información extraída que se usa por el procesador (por ejemplo, compilador o traductor) para optimizar la decodificación, asignación de registro y/u otras tareas asociadas con la traducción de la instrucción.

Además, se realiza una determinación en cuanto a si hay otra instrucción en el grupo a traducirse, CONSULTA 506. En caso afirmativo, a continuación, el procesamiento continúa con la ETAPA 500. De lo contrario, el procesamiento continúa con la optimización de la representación interna, ETAPA 508, asignando uno o más registros del grupo de instrucciones, ETAPA 510, y generando código que emula las instrucciones en el grupo, ETAPA 512.

Mientras que los procedimientos de interpretación y traducción anteriores proporcionan emulación de una instrucción definida en una arquitectura para una o más instrucciones definidas en otra arquitectura, pueden realizarse avances en la emulación de instrucciones que usan especificadores no contiguos. Por ejemplo, de acuerdo con un aspecto de la presente invención, se proporcionan mejoras en las técnicas de emulación para tratar la situación en la que se designa un operando de registro de una instrucción por múltiples campos de la instrucción.

Un tipo de instrucción que usa especificadores no contiguos son instrucciones de vector que son parte de una instalación de vector, proporcionada de acuerdo con un aspecto de la presente invención. En muchas instrucciones de vector, el campo de registro no incluye todos los bits necesarios para designar un registro para que se use por la instrucción, sino que, en su lugar, se usa otro campo junto con el campo de registro para designar un registro. Este otro campo se denomina en el presente documento como un campo RXB.

El campo de RXB, también denominado como el bit de extensión de registro es, por ejemplo, un campo de cuatro bits (bits 0-3) que incluye el bit más significativo para cada uno de los operandos designados de registro de vector de una instrucción de vector. Los bits para las designaciones de registro no especificadas por la instrucción han de reservarse y establecerse a cero.

En un ejemplo, los bits RXB se definen como sigue:

0 - Bit más significativo para la primera designación de registro de vector de la instrucción.

1 - Bit más significativo para la segunda designación de registro de vector de la instrucción, si la hubiera.

2 - Bit más significativo para la tercera designación de registro de vector de la instrucción, si la hubiera.

3 - Bit más significativo para la cuarta designación de registro de vector de la instrucción, si la hubiera.

Cada bit se establece a cero o uno, por ejemplo, mediante el ensamblador, dependiendo del número de registro. Por ejemplo, para los registros 0-15, el bit se establece a 0; para los registros 16-31, el bit se establece a 1, etc.

En una realización, cada bit de RXB es un bit de extensión para una ubicación particular en una instrucción que incluye uno o más registros de vector. Por ejemplo, en una o más instrucciones de vector, el bit 0 del RXB es un bit de extensión para la ubicación 8-11, que se asigna a, por ejemplo, V_1; el bit 1 de RXB es un bit de extensión para la ubicación 12-15, que se asigna a, por ejemplo, V_2; y así sucesivamente.

En una realización adicional, el campo de RXB incluye bits adicionales, y se usa más de un bit como una extensión para cada vector o ubicación.

De acuerdo con un aspecto de la presente invención, se proporcionan técnicas para transformar especificadores de operando no contiguos en especificadores contiguos. Una vez transformados, los especificadores contiguos se usan sin tener en cuenta los especificadores no contiguos.

Se describe una realización de la lógica para emular instrucciones que usan especificadores no contiguos con referencia a las Figuras 6-8. En particular, la Figura 6 representa una vista general de un proceso de emulación que incluye una o más de interpretación y traducción de instrucciones que incluyen especificadores no contiguos; la Figura 7A representa una realización de interpretación (Técnica 6000), que incluye la interpretación de especificadores no contiguos; la Figura 7B representa una realización de transformación de un especificador no contiguo a un especificador contiguo; y la Figura 8 representa una realización de traducción (Técnica 7000), que incluye la traducción de especificadores no contiguos.

Haciendo referencia de manera inicial a la Figura 6, se proporciona una vista general de un proceso de emulación. Esta vista general es similar a la vista general proporcionada en la Figura 3, excepto la ETAPA 600 que usa la Técnica 6000 descrita con referencia a la Figura 7A, en lugar de la Técnica 2000 referenciada en la ETAPA 300; y la ETAPA 610 usa la Técnica 7000 descrita con referencia a la Figura 8, en lugar de la Técnica 3000 referenciada en la ETAPA 310. Puesto que se ha descrito la vista general anteriormente con referencia a la Figura 3, no se repite en este punto; en su lugar, el análisis continúa a la lógica de la Figura 7A.

Haciendo referencia a la Figura 7A, las etapas 700, 702, 704 y 706 son similares a las ETAPAS 400, 402, 404 y 406, respectivamente, de la Figura 4 y, por lo tanto, no se describen de nuevo; sin embargo, se describen las etapas 703 y 705. Con la ETAPA 703, de acuerdo con un aspecto de la presente invención, se genera un especificador contiguo (también denominado en el presente documento como un índice contiguo) desde un especificador no contiguo. Se describen detalles adicionales con respecto a la generación de un especificador contiguo de un especificador no contiguo con referencia a la Figura 7B.

Haciendo referencia a la Figura 7B, en una realización, inicialmente, se obtiene el especificador no contiguo, ETAPA 750. Esto incluye, por ejemplo, determinar a partir del opcode que la instrucción tiene un especificador no contiguo y determinar qué campos de la instrucción se usan para designar el especificador no contiguo. Por ejemplo, una porción del opcode especifica un formato de la instrucción y este formato indica al procesador que la instrucción tiene al menos un especificador no contiguo y especifica adicionalmente los campos usados para designar el especificador no contiguo. Estos campos se leen a continuación para obtener los datos (por ejemplo, bits) en estos campos. Por ejemplo, en muchas instrucciones de vector, la ubicación 8-11 de la instrucción (por ejemplo, V_1) especifica una pluralidad de bits (por ejemplo, 4) usados para designar un registro de vector, y un campo RXB de la instrucción incluye uno o más bits adicionales usados para designar el registro de vector particular. Estos bits se obtienen en esta etapa.

Posterior a obtener el especificador no contiguo (por ejemplo, los bits del campo de registro V_1 y el bit o bits de RXB), se usan una o más reglas para combinar las porciones del especificador no contiguo para crear el especificador contiguo, ETAPA 752. La una o más reglas dependen de, por ejemplo, el formato de la instrucción como se especifica por el opcode de la instrucción. En un ejemplo particular en el que el opcode indica un campo RXB, la una o más reglas incluyen usar el o los bits de RXB asociados con el operando de registro como el bit o bits más significativos para los bits especificados en el campo de registro. Por ejemplo, el campo de RXB tiene, en una realización, 4 bits y cada bit corresponde a un operando de registro. Por ejemplo, el bit 0 corresponde al primer operando de registro, el bit 1 corresponde al segundo operando de registro, y así sucesivamente. Así, se extrae el bit que corresponde al

operando de registro y se usa para formar el especificador contiguo. Por ejemplo, si se especifica el binario 0010 en el primer campo de registro de operando y se especifica el binario 1000 en el campo de RXB, el valor del bit asociado con el primer operando, bit 0, en este ejemplo, se concatena a 0010. Por lo tanto, el especificador contiguo es 10010 (registro 18), en este ejemplo.

El especificador contiguo generado se usa a continuación como si fuera el especificador proporcionado en la instrucción, ETAPA 754.

Posteriormente, volviendo a la Figura 7A, se realiza un ramal para codificar que emula el comportamiento que corresponde al opcode, ETAPA 704. Además, se usa el índice contiguo para gestionar el recurso de arquitectura homogeneizado sin tener en cuenta el especificador no contiguo, ETAPA 705. Es decir, se usa el especificador de registro contiguo, como si no hubiera especificador contiguo. Cada especificador contiguo indica un registro a usar por el código de emulación. Posteriormente, se realiza el código de emulación, ETAPA 706.

Se describen detalles adicionales con respecto a la traducción que incluyen la transformación de especificadores no contiguos a especificadores contiguos (denominado como Técnica 7000) con referencia a la Figura 8. En una realización, las etapas 800, 802, 804, 806, 808, 810 y 812 son similares a las ETAPAS 500, 502, 504, 506, 508, 510, y 512, respectivamente, de la Figura 5 y, por lo tanto, no se describen en este punto con referencia a la Figura 8. Sin embargo, de acuerdo con un aspecto de la presente invención, se realizan etapas adicionales para transformar un especificador no contiguo de una instrucción de una arquitectura de origen a un especificador contiguo de una instrucción de una arquitectura objetivo. La instrucción de la arquitectura objetivo emula una función de la instrucción de la arquitectura de origen.

Por ejemplo, en la ETAPA 803, se genera un especificador contiguo de un especificador no contiguo. Como se ha descrito con referencia a la Figura 7B, esto incluye obtener el especificador no contiguo de la instrucción a emularse, y usar una o más reglas para crear el especificador contiguo del especificador no contiguo. En una realización, el opcode de la instrucción que tiene el especificador no contiguo indica, al menos implícitamente por su formato, que la instrucción incluye un especificador no contiguo. Por ejemplo, el formato de la instrucción se indica por uno o más bits del opcode (por ejemplo, los primeros dos bits), y basándose en el formato, el procesador (por ejemplo, el compilador, traductor, emulador del procesador) entiende que esta instrucción incluye un especificador no contiguo, en el que parte del especificador de un recurso, tal como un registro, está incluido en un campo de la instrucción y una o más otras partes del especificador están ubicadas en uno o más otros campos de la instrucción.

El opcode, como un ejemplo, también proporciona una indicación al procesador de la una o más reglas usadas para generar el especificador contiguo del especificador no contiguo. Por ejemplo, el opcode puede indicar que una instrucción particular es una instrucción de registro de vector y, por lo tanto, tiene un campo RXB. Por lo tanto, el procesador accede a información (por ejemplo, reglas almacenadas en memoria o almacenamiento externo) que indica para una instrucción con un campo RXB, el campo de RXB proporciona el bit más significativo para su campo de registro correspondiente. Las reglas especifican que, por ejemplo, para generar el campo contiguo, los bits del campo de registro se combinan con el uno o más bits del campo de RXB asociado con el operando de registro particular

Posterior a la generación del especificador contiguo, se usa el especificador contiguo sin tener en cuenta el especificador no contiguo. Por ejemplo, en la ETAPA 808, el código se optimiza usando el especificador contiguo sin tener en cuenta el especificador no contiguo. De manera similar, se asignan uno o más registros usando el especificador contiguo y sin tener en cuenta el especificador no contiguo, ETAPA 810. Aún más, en la ETAPA 812, el código emulado se genera sin tener en cuenta el especificador no contiguo y usando la asignación realizada en la ETAPA 810. Es decir, en estas etapas, no hay indicación de que el especificador contiguo se generó de un especificador no contiguo. El especificador no contiguo se ignora.

Se describen detalles adicionales con respecto a la traducción de un especificador no contiguo a un especificador contiguo con referencia a los ejemplos en las Figuras 9A, 9B y 11. Haciendo referencia de manera inicial a la Figura 9A, se representa una instrucción 900 de carga de vector (VL). En un ejemplo, la instrucción de carga de vector incluye los campos opcode 902a (por ejemplo, los bits 0-7), 902b (por ejemplo, los bits 40-47) que indican una operación de carga de vector; un campo 904 de registro de vector (por ejemplo, los bits 8-11) usado para designar un registro de vector (V_1); un campo de índice (X_2) 906 (por ejemplo, los bits 12-15); un campo de base (B_2) 908 (por ejemplo, los bits 16-19); un campo de desplazamiento (D_2) 910 (por ejemplo, los bits 20-31); y un campo 912 RXB (por ejemplo, los bits 36-39). Cada uno de los campos 904-912 en un ejemplo está separado e independiente del campo o campos del opcode. Además, en una realización están separados e independientes uno del otro; sin embargo, en otras realizaciones, puede combinarse más de un campo. Se describe a continuación información adicional sobre el uso de estos campos.

En un ejemplo, los bits seleccionados (por ejemplo, los primeros dos bits del opcode designados por el campo de opcode 902a) especifican la longitud y formato de la instrucción. En este ejemplo particular, la longitud son tres semi-palabras y el formato es un registro de operación de almacenamiento de vector-e-índice con un campo de opcode extendido. El campo de vector (V_1), junto con su correspondiente bit de extensión especificado por RXB, designa un registro de vector (es decir, un especificador no contiguo). En particular, para los registros de vector, el registro que

contiene el operando se especifica usando, por ejemplo, un campo de cuatro bits del campo de registro con la adición de su bit de extensión de registro (RXB) como el bit más significativo. Por ejemplo, si el campo de bit cuatro en V_1 es 0010 binario y el bit de extensión para este operando es 1 binario, entonces el campo de 5 bits es 10010 binario, que indica el número de registro 18 (en decimal).

El número en subíndice asociado con el campo de la instrucción indica el operando al que se aplica el campo. Por ejemplo, el número de subíndice 1 asociado con V_1 indica el primer operando, y así sucesivamente. Esto se usa para determinar qué bit del campo de RXB se combina con el campo de registro. El operando de registro es un registro en longitud, que es, por ejemplo, 128 bytes. En un ejemplo, en un registro de la instrucción de operación de almacenamiento de vector-e-índice, los contenidos de registro generales designados por los campos X_2 y B_2 se añaden a los contenidos del campo D_2 para formar la segunda dirección de operando. El desplazamiento, D_2, para la instrucción de carga de vector se trata como un número entero sin signo de 12 bits, en un ejemplo.

En este ejemplo, puesto que V_1 es el primer operando, la ubicación más a la izquierda (por ejemplo, el bit 0) del RXB está asociada con este operando. Por lo tanto, el valor ubicado en la ubicación más a la izquierda se combina con el valor en el campo de registro V_1 para generar el especificador contiguo, como se describe en el presente documento

De acuerdo con un aspecto de la presente invención, la instrucción 900 de carga de vector, que se define, por ejemplo, en la z/Architecture, se emula en una instrucción 950 indexada de vector de carga definida, por ejemplo, en la arquitectura PowerPC. Aunque, en este ejemplo, la z/Architecture es la arquitectura de origen y PowerPC es la arquitectura objetivo, esto es únicamente un ejemplo. Pueden usarse muchas otras arquitecturas para una o ambas de las arquitecturas de origen y objetivo.

Cada arquitectura tiene asociada con ella registros particulares que puede usar. Por ejemplo, en la z/Architecture, hay 32 registros de vector y otros tipos de registros pueden mapearse a un cuadrante de los registros de vector. Como un ejemplo, como se muestra en la Figura 10, si hay un fichero 1000 de registro que incluye 32 registros 1002 de vector y cada registro es de 128 bits en longitud, entonces 16 registros de punto de coma flotante 1004, que son 64 bits en longitud, pueden solapar los registros de vector. Por lo tanto, como un ejemplo, cuando se modifica el registro de punto de coma flotante 2, entonces también se modifica el registro de vector 2. También son posibles otros mapeos para otros tipos de registros.

De manera similar, la arquitectura objetivo PowerPC u otra tiene un conjunto de registros asignados a ella. Este conjunto de registros puede ser diferente o el mismo que el conjunto de registros asignados a la arquitectura de origen. El registro objetivo puede tener más o menos registros disponibles para un tipo particular de instrucción. Por ejemplo, en el ejemplo representado en la Figura 9A, la instrucción de carga de vector y la instrucción indexada de carga de vector tienen 32 registros de vector disponibles para ellas. De nuevo, son posibles otros ejemplos.

Como se indica por el opcode, la instrucción de carga de vector incluye un especificador no contiguo que, en este ejemplo, se representa en los campos V_1 y RXB. Estos campos no contiguos se combinan para crear un índice contiguo en la instrucción 950 indexada de carga de vector. Este especificador contiguo se indica en el campo 954 VRT de la instrucción 950. En este ejemplo particular, como se muestra en el código VL v18, 0(0, gr5), el registro de vector que se especifica es el registro 18. Este registro se especifica en la instrucción por el especificador no contiguo proporcionado por el campo V_1 y el campo de RXB. En este ejemplo, el campo V1 incluye un valor de 2 (0010 binario) y el campo de RXB incluye un valor de 8 (1000 binario). Basándose en las reglas predefinidas, puesto que V_1 es el primer operando, el bit más a la izquierda (1) de 1000 se concatena con los bits en el campo V1 (0010) para producir un especificador contiguo de 10010, que es el valor 18 en decimal.

Como se muestra en el número de referencia 956, una representación de 18 se coloca en el campo VRT de la instrucción indexada de carga de vector, que corresponde al campo de registro (V_1) de la instrucción de carga de vector. Para ser más exhaustivos, los campos RA y RB de la instrucción 950 corresponden al X_2 y B_2, respectivamente, de la instrucción 900. El campo D_2 de la instrucción 900 no tiene campo correspondiente en la instrucción 950; y los campos opcode de la instrucción 900 corresponden a los campos opcode de la instrucción 950.

Se representa un ejemplo adicional en la Figura 9B. En este ejemplo, como con el ejemplo representado en la Figura 9A, el especificador no contiguo (V_1, RXB) de la instrucción 900 se transforma en un especificador contiguo (VRT) de la instrucción 950. Sin embargo, en este ejemplo, el registro asignado para la instrucción 950 no tiene el mismo número que el especificador contiguo transformado; en su lugar, el especificador contiguo se mapea a un registro diferente. Por ejemplo, en el ejemplo en la Figura 9A, el especificador no contiguo referencia al registro 18, como lo hace el especificador contiguo. Es decir, hay un uno para un mapeo. Sin embargo, en la Figura 9B, el especificador no contiguo de 18 se transforma en un especificador contiguo de 18, pero a continuación, el 18 del especificador contiguo se mapea a un registro diferente, tal como el registro 7 (véase el número de referencia 980). Es decir, el registro 18 en la arquitectura de origen se mapea al registro 7 en la arquitectura objetivo, en este ejemplo particular. Un mapeo de este tipo está predefinido y es accesible al procesador.

Un ejemplo adicional más se representa en la Figura 11. En este ejemplo, en lugar de asignarse a un registro durante la emulación, como en las Figuras 9A y 9B, la asignación es a la memoria. En este ejemplo, se usa una instrucción,

VLR, para mover los contenidos de un registro de vector, VR 18, a otro registro de vector, VR 24. Sin embargo, en este ejemplo, se supone que el fichero de registro no es lo suficientemente grande para incluir estos registros de vector, por lo que, en su lugar, se usa la memoria. Es decir, hay una porción contigua de memoria que almacena una pluralidad de vectores como una serie. La serie se inicia en una dirección, rvbase, en la que se almacena el primer registro, por ejemplo, el registro 0; y a continuación, el siguiente registro se almacena en un desplazamiento, por ejemplo, 16 bytes, desde rvbase; y el tercer registro se almacena en el desplazamiento del segundo registro, y así sucesivamente. Por lo tanto, en este ejemplo, el registro 18 está a un desplazamiento 288 de rvbase, y el registro 24 está en un desplazamiento 384 de rvbase.

En este ejemplo, hay dos especificadores no contiguos (V_1, RXB; y V_2, RXB). Por lo tanto, se generan dos especificadores contiguos. Por ejemplo, puesto que V_1 es el primer operando, el primer especificador contiguo se genera mediante una concatenación de los bits en V_1 con el bit 0 de RXB. Puesto que V_1 incluye 1000 en binario (8 decimal) y RXB incluye 1100 en binario (12 decimal), el primer especificador contiguo se forma concatenando 1 (desde el bit 0 de RXB) con 1000 (desde V_1) que proporciona 11000 (24 en decimal). De manera similar, se genera el segundo especificador contiguo por una concatenación de 0010 (2 en decimal para V_2) y 1 (desde el bit 1 de RXB) que proporciona 10010 (18 en decimal). Puesto que estos registros están en memoria, el registro de vector 24 está en un desplazamiento 384 de rvbase, y el registro de vector 18 está en un desplazamiento 288 de rvbase. Estos valores se muestran en la Figura 11 en 1102, 1104, respectivamente.

El pseudo-código a la derecha de la Figura 11 y las instrucciones a la izquierda describen el movimiento de un número contiguo de bytes que corresponde a un registro de vector en un desplazamiento de vector en 18 (que corresponde a un desplazamiento de byte en 288) a un desplazamiento de vector en 24 (que corresponde a un desplazamiento de byte en 384). En particular, una carga inmediata (LI) carga un valor de 288 en rtemp1, y a continuación se realiza un carga de vector en una dirección proporcionada por rvbase más el desplazamiento en rtemp1, y el valor se almacena en un registro de vector temporal, vtemp2. A continuación, la siguiente carga inmediata carga 384 en rtemp1, y se realiza un almacenamiento de nuevo en la memoria en una ubicación que corresponde a la dirección más el desplazamiento en el registro de vector 24 (por ejemplo, el desplazamiento 288).

Aunque se han descrito anteriormente diversos ejemplos, son posibles muchos otros ejemplos y variaciones. La información adicional con respecto a instrucciones de vector y el uso del campo de RXB se describe en una solicitud de patente presentada junto con la presente, titulada "Instruction to Load Data Up to A Specified Memory Boundary Indicated by the Instruction", N.º de Serie de Estados Unidos 13/421456, (N.º de Expediente de IBM POU920120030US1), Jonathan D. Bradbury et al.

Además, se mencionan en el presente documento diversas arquitecturas. Una realización de la z/Architecture se describe en una publicación de IBM® titulada, "z/Architecture Principles of Operation", Publicación de IBM® N.º SA22-7832-08, novena edición, agosto de 2010. IBM® y Z/ARCHITECTURE® son marcas comerciales registradas de International Business Machines Corporation, Armonk, Nueva York, Estados Unidos. Otros nombres usados en el presente documento pueden ser marcas comerciales registradas, marcas comerciales, o nombres de producto de International Business Machines Corporation u otras compañías. Adicionalmente, una realización de Power Architecture se describe en "Power ISA™ Version 2.06 Revision B", International Business Machines Corporation, 23 de julio de 2010. POWER ARCHITECTURE® es una marca comercial registrada de International Business Machines Corporation. Aún además, se describe una realización de una arquitectura de Intel en "Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 2B, Instructions Set Reference, A-L", número de pedido 253666-041US, diciembre de 2011, e "Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 2B, Instructions Set Reference, M-Z", número de pedido 253667-041US, diciembre de 2011. Intel® es una marca comercial registrada de Intel Corporation, Santa Clara, California.

Se describe en detalle en el presente documento una técnica para transmitir especificadores no contiguos de una instrucción definida para una arquitectura de sistema para especificadores contiguos para una instrucción definida para otra arquitectura de sistema. La emulación de arquitectura anterior no ha tratado satisfactoriamente la emulación de sistemas con especificadores no contiguos, y particularmente especificadores de registro no contiguos, ya sea en conjuntos de instrucciones fijas o variables. Sin embargo, de acuerdo con un aspecto de la presente invención, se proporciona una técnica para extender emuladores anteriores para manejar especificadores no contiguos. La técnica incluye, por ejemplo, leer especificadores no contiguos, generar un índice contiguo de un especificador no contiguo, y usar el índice contiguo para acceder a un recurso homogéneo o representar un recurso homogéneo.

En una realización adicional, de acuerdo con una implementación JIT, se usa un índice contiguo para realizar decisiones de asignación, que representan opcionalmente un recurso accedido por un especificador no contiguo por un recurso no contiguo/no homogéneo, pero que no refleja el particionamiento por límites de especificador no contiguo, sino por decisiones de optimización. Es decir, en una realización, una instrucción definida para una arquitectura tiene al menos un especificador no contiguo para al menos un recurso, y ese al menos un especificador no contiguo se transforma en al menos un especificador contiguo. Ese al menos un especificador contiguo se usa para seleccionar al menos un recurso para una instrucción de otra arquitectura para su uso. La instrucción de la otra arquitectura, sin embargo, usa especificadores no contiguos. Por lo tanto, el al menos un especificador contiguo para el al menos un recurso seleccionado se transforma a continuación en al menos un especificador no contiguo para su uso por la

instrucción de la segunda arquitectura. En una realización, esto se realiza por un emulador.

En una realización, se proporciona un emulador para emular la ejecución de instrucción de un primer conjunto de instrucciones de arquitectura informática en un procesador diseñado para una segunda arquitectura informática. El emulador incluye, por ejemplo, capturar instrucciones de una aplicación por el programa de emulación; interpretar el opcode de las instrucciones para seleccionar un módulo de emulación para emular las instrucciones; determinar del opcode que las instrucciones emplean campos de registro no contiguos; combinar campos de registro no contiguos de la instrucción para formar un campo de registro combinado; y usar el campo de registro combinado por instrucciones del módulo de emulación para emular las instrucciones.

Además, en una realización, el espacio de registro incluye una subsección, y el primer conjunto de instrucciones de arquitectura informática incluye primeras instrucciones que tienen campos de registro para acceder únicamente a la subsección, y segundas instrucciones que tienen campos de registro no contiguos para acceder a todo el espacio de registro.

En una realización, el campo de RXB está en la misma ubicación para todas las instrucciones usando el campo de RXB. Los bits RXB son bits significativos en que el bit 36 del campo de RXB se usa para los bits extendidos 8-11 de la instrucción; el bit 37 de RXB se usa para los bits extendidos 12-15; el bit 38 de RXB se usa para los bits extendidos 16-19; y el bit 39 de RXB se usa para los bits extendidos 32-35, como ejemplos. Además, la decisión para usar un bit del RXB como un bit de extensión es dependiente de opcode (por ejemplo, R_1 frente a V_1). Además, los especificadores no contiguos pueden usar campos distintos de los campos RXB.

En el presente documento, memoria, memoria principal, almacenamiento y almacenamiento principal se usan de manera intercambiable, a menos que se indique lo contrario explícitamente o por contexto.

Se proporcionan detalles adicionales con relación a la instalación de vector, que incluyen ejemplos de instrucciones, como parte de esta descripción detallada en mayor detalle a continuación.

Como se apreciará por un experto en la materia, uno o más aspectos de la presente invención pueden realizarse como un sistema, método o producto de programa informático. Por consiguiente, uno o más aspectos de la presente invención pueden tomar la forma de una realización completamente de hardware, una realización completamente de software (que incluye firmware, software residente, micro-código, etc.) o una realización que combina aspectos de software y hardware que pueden todos denominarse de manera general en el presente documento como un "circuito", "módulo" o "sistema". Adicionalmente, uno o más aspectos de la presente invención pueden tomar la forma de un producto de programa informático realizado en uno o más medio o medios legibles por ordenador que tienen código de programa legible por ordenador incorporado en los mismos.

Puede utilizarse cualquier combinación de uno o más medio o medios legibles por ordenador. El medio legible por ordenador puede ser un medio de almacenamiento legible por ordenador. Un medio de almacenamiento legible por ordenador puede ser, por ejemplo, pero sin limitación, uno electrónico, magnético, óptico, electromagnético, sistema de infrarrojos o de semiconductores, aparatos, o dispositivo, o cualquier combinación adecuada de lo anterior. Ejemplos más específicos (una lista no exhaustiva) del medio de almacenamiento legible por ordenador incluyen lo siguiente: una conexión eléctrica que tiene uno o más alambres, un disquete de ordenador portátil, un disco duro, una memoria de acceso aleatorio (RAM), una memoria de sólo lectura (ROM), una memoria de sólo lectura programable borrable (EPROM o memoria flash), una fibra óptica, una memoria solo de lectura de disco compacto portátil (CD-ROM), un dispositivo de almacenamiento óptico, un dispositivo de almacenamiento magnético, o cualquier combinación adecuada de lo anterior. En el contexto de este documento, un medio de almacenamiento legible por ordenador puede ser cualquier medio tangible que pueda contener o almacenar un programa para su uso por o en relación con un sistema, aparato, o dispositivo de ejecución de instrucciones.

Haciendo referencia ahora a la Figura 12, en un ejemplo, un producto 1200 de programa informático incluye, por ejemplo, uno o más medios 1202 de almacenamiento legibles por ordenador no transitorios para almacenar medios o lógica 1204 de código de programa legible por ordenador en los mismos para proporcionar y facilitar uno o más aspectos de la presente invención.

El código de programa incorporado en un medio legible por ordenador puede transmitirse usando un medio apropiado, incluyendo, pero sin limitación, inalámbrico, alámbrico, cable de fibra óptica, RF, etc., o cualquier combinación adecuada de lo anterior.

El código de programa informático para llevar a cabo operaciones para uno o más aspectos de la presente invención puede escribirse en cualquier combinación de uno o más lenguajes de programación, incluyendo un lenguaje de programación orientado a objetos, tal como Java, Smalltalk, C++ o similares, y lenguajes de programación procedurales convencionales, tales como el lenguaje de programación "C", ensamblador o lenguajes de programación similares. El código de programa puede ejecutarse completamente en el ordenador del usuario, parcialmente en el ordenador del usuario, como un paquete de software independiente, parcialmente en el ordenador del usuario y parcialmente en un ordenador remoto o completamente en el ordenador o servidor remoto. En el último escenario, el

ordenador remoto puede estar conectado al ordenador del usuario a través de un tipo de red, que incluye una red de área local (LAN) o una red de área extensa (WAN), o la conexión puede hacerse a un ordenador externo (por ejemplo, a través de Internet usando un Proveedor de Servicio de Internet).

Uno o más aspectos de la presente invención se describen en el presente documento con referencia unas ilustraciones de diagrama de flujo y/o diagramas de bloques de los métodos, aparato (sistemas) y productos de programa informático de acuerdo con las realizaciones de la invención. Se entenderá que cada bloque de las ilustraciones de diagrama de flujo y/o diagramas de bloques, y combinaciones de bloques en las ilustraciones de diagrama de flujo y/o diagramas de bloques, pueden implementarse por instrucciones de programa informático. Estas instrucciones de programa informático pueden proporcionarse a un procesador de un ordenador de fin general, ordenador de fin especial, u otro aparato de procesamiento de datos programable para producir una máquina, de manera que las instrucciones, que se ejecutan mediante el procesador del ordenador u otro aparato de procesamiento de datos programable, crean medios para implementar las funciones/actos especificados en el diagrama de flujo y/o bloque o bloques del diagrama de bloques.

Estas instrucciones de programa informático pueden almacenarse también en un medio legible por ordenador que puede dirigir un ordenador, otro aparato de procesamiento de datos programable, u otros dispositivos para funcionar de una manera particular, de manera que las instrucciones almacenadas en el medio legible por ordenador producen un artículo de fabricación que incluye instrucciones que implementan la función/acto especificado en el diagrama de flujo y/o bloque o bloques del diagrama de bloques.

Las instrucciones de programa informático pueden cargarse también en un ordenador, otro aparato de procesamiento de datos programable, u otros dispositivos para provocar que se realice una serie de etapas operacionales en el ordenador, otro aparato programable u otros dispositivos para producir un proceso implementado por ordenador de manera que las instrucciones que se ejecutan en el ordenador u otro aparato programable proporcionan procesos para implementar las funciones/actos especificados en el diagrama de flujo y/o bloque o bloques del diagrama de bloques.

El diagrama de flujo y diagramas de bloques en las figuras ilustran la arquitectura, funcionalidad, y operación de posibles implementaciones de sistemas, métodos y productos de programa informático de acuerdo con diversas realizaciones de uno o más aspectos de la presente invención. En este sentido, cada bloque en el diagrama de flujo o diagramas de bloques puede representar un módulo, segmento, o porción de código, que comprende una o más instrucciones ejecutables para implementar la función o funciones lógicas especificadas. Debería observarse también que, en algunas implementaciones alternativas, las funciones indicadas en el bloque pueden tener lugar fuera del orden indicado en las figuras. Por ejemplo, dos bloques mostrados en serie, de hecho, pueden ejecutarse sustancialmente de manera concurrente, o los bloques pueden ejecutarse en ocasiones en el orden inverso, dependiendo de la funcionalidad implicada. Se entenderá también que cada bloque de los diagramas de bloques y/o ilustración de diagrama de flujo, y combinaciones de bloques en los diagramas de bloques y/o ilustración de diagrama de flujo, puede implementarse por sistemas basados en hardware de fin especial que realizan las funciones o actos especificados, o combinaciones de hardware de fin especial e instrucciones informáticas.

Además de lo anterior, pueden proporcionarse, ofrecerse, desplegarse, gestionarse, servirse, etc., uno o más aspectos de la presente invención por un proveedor de servicio que ofrece gestión de entornos de cliente. Por ejemplo, el proveedor de servicio puede crear, mantener, soportar, etc., código informático y/o una infraestructura informática que realiza uno o más aspectos de la presente invención para uno o más clientes. A cambio, el proveedor de servicio puede recibir pago del cliente bajo una suscripción y/o acuerdo de cuota, como ejemplos. Adicionalmente o como alternativa, el proveedor de servicio puede recibir pago de la venta de contenido de anuncio a una o más terceras partes.

En un aspecto de la presente invención, puede desplegarse una aplicación para realizar uno o más aspectos de la presente invención. Como un ejemplo, el despliegue de una aplicación comprende proporcionar infraestructura informática operable para realizar uno o más aspectos de la presente invención.

Como un aspecto adicional de la presente invención, una infraestructura informática puede desplegarse que comprende integrar código legible por ordenador en un sistema informático, en el que el código en combinación con el sistema informático puede realizar uno o más aspectos de la presente invención.

Como un aspecto adicional más de la presente invención, puede proporcionarse un proceso para integrar infraestructura informática que comprende integrar código legible por ordenador en un sistema informático. El sistema informático comprende un medio legible por ordenador, en el que el medio informático comprende uno o más aspectos de la presente invención. El código en combinación con el sistema informático puede realizar uno o más aspectos de la presente invención.

Aunque se han descrito anteriormente diversas realizaciones, estas son únicamente ejemplos. Por ejemplo, los entornos informáticos de otras arquitecturas pueden incorporar y usar uno o más aspectos de la presente invención. Además, pueden usarse vectores de otros tamaños u otros registros, y pueden realizarse cambios a las instrucciones sin alejarse del alcance de la presente invención. Adicionalmente, pueden usarse otras instrucciones en el

procesamiento. Además, puede usarse uno o más aspectos de la invención relacionados con la transformación de especificadores no contiguos a especificadores contiguos en otros contextos. Además, los especificadores pueden ser para otros que no sean registros. Son también posibles otros cambios.

Además, otros tipos de entornos informáticos pueden beneficiarse de uno o más aspectos de la presente invención. Como un ejemplo, puede usarse un sistema de procesamiento de datos adecuado para almacenar y/o ejecutar código de programa que incluye al menos dos procesadores acoplados directa o indirectamente a elementos de memoria a través de un bus de sistema. Los elementos de memoria incluyen, por ejemplo, memoria local empleada durante ejecución real del código de programa, almacenamiento en bruto, y memoria caché que proporciona almacenamiento temporal de al menos algún código de programa para reducir el número de veces que debe recuperarse el código del almacenamiento en bruto durante la ejecución.

Los dispositivos de entrada/salida o E/S (que incluyen, pero sin limitación, teclados, pantallas, dispositivos apuntadores, DASD, cinta, CD, DVD, unidades de memoria y otros medios de memoria, etc.) pueden acoplarse al sistema ya sea directamente o a través de controladores de E/S intermedios. Los adaptadores de red pueden acoplarse también al sistema para posibilitar que el sistema de procesamiento de datos se acople a otros sistemas de procesamiento de datos o impresoras remotas o dispositivos de almacenamiento a través de redes privadas o públicas intermedias. Los módems, módems de cable, y tarjetas de Ethernet son solamente unos pocos de los tipos disponibles de adaptadores de red.

Haciendo referencia a la Figura 13, se representan componentes representativos de un sistema 5000 de ordenador de anfitrión para implementar uno o más aspectos de la presente invención. El ordenador 5000 de anfitrión representativo comprende una o más CPU 5001 en comunicación con memoria informática (es decir, almacenamiento central) 5002, así como interfaces de E/S a dispositivos 5011 de medios de almacenamiento y redes 5010 para comunicar con otros ordenadores o SAN y similares. La CPU 5001 es compatible con una arquitectura que tiene un conjunto de instrucciones de arquitectura y funcionalidad de arquitectura. La CPU 5001 puede tener traducción de dirección dinámica (DAT) 5003 para transformar direcciones de programa (direcciones virtuales) en direcciones de memoria reales. Una DAT típicamente incluye una memoria intermedia de traducción adelantada (TLB) 5007 para almacenar en caché traducciones de modo que los últimos accesos al bloque de memoria 5002 informática no requieren el retardo de traducción de dirección. Típicamente, se emplea una caché 5009 entre la memoria 5002 informática y el procesador 5001. La caché 5009 puede tener jerárquicamente una gran caché disponible para más de una CPU y cachés más pequeñas, más rápidas (nivel inferior) entre la caché grande y cada CPU. En algunas implementaciones, las cachés de nivel inferior se dividen para proporcionar cachés de nivel bajo separadas para recuperación de instrucciones y accesos de datos. En una realización, se recupera una instrucción de la memoria 5002 por una unidad 5004 de recuperación de instrucción mediante una caché 5009. La instrucción se decodifica en una unidad 5006 de decodificación de instrucción y se despacha (con otras instrucciones en algunas realizaciones) a la unidad o unidades 5008 de ejecución de instrucciones. Típicamente se emplean varias unidades 5008 de ejecución, por ejemplo, una unidad de ejecución aritmética, una unidad de ejecución de punto flotante y una unidad de ejecución de instrucción de ramal. La instrucción se ejecuta por la unidad de ejecución, que accede a los operandos de los registros de instrucción especificados o memoria según sea necesario. Si ha de accederse a un operando (cargarse o almacenarse) de la memoria 5002, una unidad de carga/almacén 5005 típicamente maneja el acceso bajo el control de la instrucción que se está ejecutando. Las instrucciones pueden ejecutarse en circuitos de hardware o en microcódigo interno (firmware) o por una combinación de ambos.

Como se indica, un sistema informático incluye información en almacenamiento local (o principal), así como direccionamiento, protección y referencia y grabación de cambio. Algunos aspectos del direccionamiento incluyen el formato de direcciones, el concepto de espacios de dirección, los diversos tipos de direcciones, y la manera en la que un tipo de dirección se traduce a otro tipo de dirección. Alguno del almacenamiento principal incluye ubicaciones de almacenamiento asignadas permanentemente. El almacenamiento principal proporciona el sistema con almacenamiento de datos de acceso rápido directamente direccionable. Han de cargarse tanto datos como programas en almacenamiento principal (de dispositivos de entrada) antes de que se procesen.

El almacenamiento principal puede incluir uno o más almacenamientos de memoria intermedia de acceso más rápido más pequeños, en ocasiones denominados cachés. Una caché está asociada físicamente de manera típica con una CPU o un procesador de E/S. Los efectos, excepto en rendimiento, de la construcción física y uso de medios de almacenamiento distintos en general no son observables por el programa.

Las cachés separadas pueden mantenerse para instrucciones y para operandos de datos. La información dentro de una caché se mantiene en bytes contiguos en un límite integral denominado un bloque de caché o línea de caché (o línea, por resumir). Un modelo puede proporcionar una instrucción EXTRAER ATRIBUTO DE CACHÉ que devuelve el tamaño de una línea de caché en bytes. Un modelo puede proporcionar también instrucciones PRE-RECUPERAR DATOS y PRE-RECUPERAR DATOS RELATIVOS LARGO que efectúan la pre-recuperación de almacenamiento en la caché de datos o de instrucciones o la liberación de datos de la caché.

El almacenamiento se observa como una cadena de bits horizontal larga. Para la mayoría de las operaciones, los accesos a almacenamiento continúan en una secuencia de izquierda a derecha. La cadena de bits se subdivide en

unidades de ocho bits. Una unidad de ocho bits se denomina un byte, que es el bloque de construcción básica de todos los formatos de información. Cada ubicación de byte en el almacenamiento se identifica por un número entero no negativo único, que es la dirección de esa ubicación de byte o, simplemente, la dirección de byte. Las ubicaciones de bytes adyacentes tienen direcciones consecutivas, empezando con 0 en la izquierda y continuando en una secuencia de izquierda a derecha. Las direcciones son números enteros binarios sin signo y son 24, 31, o 64 bits.

La información se transmite entre almacenamiento y una CPU o un subsistema de canal un byte, o un grupo de bytes, a la vez. A menos que se especifique lo contrario, en, por ejemplo, la z/Architecture, un grupo de bytes en almacenamiento se direcciona por el byte más a la izquierda del grupo. El número de bytes en el grupo está implicado o se especifica explícitamente por la operación que va a realizarse. Cuando se usa en una operación de CPU, un grupo de bytes se denomina un campo. Dentro de cada grupo de bytes, en, por ejemplo, la z/Architecture, se numeran los bits en una secuencia de izquierda a derecha. En la z/Architecture, los bits más a la izquierda en ocasiones se denominan como los bits "de orden alto" y los bits más a la derecha como los bits "de orden bajo". Sin embargo, los números de bits no son direcciones de almacenamiento. Únicamente pueden direccionarse bytes. Para operar en bits individuales de un byte en almacenamiento, se accede al byte entero. Los bits en un byte se numeran 0 a 7, de izquierda a derecha (en, por ejemplo, la z/Architecture). Los bits en una dirección pueden numerarse 8-31 o 40-63 para direcciones de 24 bits, o 1-31 o 33-63 para direcciones de 31 bits; se numeran 0-63 para direcciones de 64 bits. Dentro de cualquier otro formato de longitud fija de múltiples bytes, los bits que componen el formato se numeran de manera consecutiva empezando desde 0. Para los fines de detección de error, y preferentemente para su corrección, puede transmitirse uno o más bits de comprobación con cada byte o con un grupo de bytes. Tales bits de comprobación se generan automáticamente por la máquina y no pueden controlarse directamente por el programa. Las capacidades de almacenamiento se expresan en número de bytes. Cuando la longitud de un campo de almacenamiento-operando se ve implicada por el código de operación de una instrucción, se dice que el campo tiene una longitud fija, que puede ser uno, dos, cuatro, ocho o dieciséis bytes. Pueden verse implicados campos más grandes para algunas instrucciones. Cuando la longitud de un campo de almacenamiento-operando no se ve implicado sino que se establece explícitamente, el campo se dice que tiene una longitud variable. Los operandos de longitud variable pueden variar en longitud en incrementos de un byte (o con algunas instrucciones, en múltiplos de dos bytes u otros múltiplos). Cuando la información se coloca en almacenamiento, los contenidos de únicamente aquellas ubicaciones de bytes que se sustituyen están incluidos en el campo designado, incluso aunque la anchura de la ruta física al almacenamiento pueda ser mayor que la longitud del campo que se está almacenando.

Ciertas unidades de información han de estar en un límite integral en el almacenamiento. Un límite se denomina integral para una unidad de información cuando su dirección de almacenamiento es un múltiplo de la longitud de la unidad en bytes. Se proporcionan nombres especiales a los campos de 2, 4, 8, y 16 bytes en un límite integral. Una semi-palabra es un grupo dos bytes consecutivos en un límite de dos bytes y es el bloque de construcción básica de instrucciones. Una palabra es un grupo de cuatro bytes consecutivos en un límite de cuatro bytes. Una palabra doble es un grupo de ocho bytes consecutivos en un límite de ocho bytes. Una palabra cuádruple es un grupo de 16 bytes consecutivos en un límite de 16 bytes. Cuando las direcciones de almacenamiento designan semi-palabras, palabras, dobles palabras y palabras cuádruples, la representación binaria de la dirección contiene uno, dos, tres o cuatro bits cero más a la derecha, respectivamente. Las instrucciones tienen que estar en límites integrales de dos bytes. Los operandos de almacenamiento de la mayoría de las instrucciones no tienen requisitos de alineación de límite.

En dispositivos que implementan cachés separadas para instrucciones y operandos de datos, puede experimentarse un retardo significativo si el programa almacena en una línea de caché a partir de la cual se recuperan posteriormente instrucciones, independientemente de si el almacenamiento modifica las instrucciones que se recuperan posteriormente.

En una realización, la invención puede ponerse en práctica por software (en ocasiones denominado código interno con licencia, firmware, micro-código, milicode, pico-código y similares, cualquiera de los cuales puede ser consistente con uno o más aspectos la presente invención). Haciendo referencia a la Figura 13, el código de programa de software que realiza uno o más aspectos de la presente invención puede accederse por el procesador 5001 del sistema anfitrión 5000 de los dispositivos 5011 de medios de almacenamiento a largo plazo, tales como una unidad de CD-ROM, unidad de cinta o unidad de disco. El código de programa de software puede realizarse en cualquiera de una diversidad de medios conocidos para su uso con un sistema de procesamiento de datos, tal como un disquete, disco duro, o CD ROM. El código puede distribuirse en tales medios, o distribuirse a usuarios desde la memoria 5002 informática o almacenamiento de un sistema informático a través de una red 5010 a otros sistemas informáticos para su uso por usuarios de otros sistemas de este tipo.

El código de programa de software incluye un sistema operativo que controla la función e interacción de los diversos componentes informáticos y uno o más programas de aplicación. El código de programa normalmente se pagina desde el dispositivo 5011 de medios de almacenamiento al almacenamiento 5002 informático de velocidad relativamente superior donde está disponible para su procesamiento por el procesador 5001. Las técnicas y métodos para realizar código de programa de software en memoria, en medios físicos y/o distribuir código de software mediante redes son bien conocidos y no se analizarán adicionalmente en el presente documento. El código de programa, cuando se crea y almacena en un medio de almacenamiento (que incluye, pero sin limitación, módulos de memoria electrónica (RAM), memoria flash, Discos Compactos (CD), DVD, Cinta Magnética y similares a menudo se denomina como un "producto

de programa informático". El medio de producto de programa informático es típicamente legible por un circuito de procesamiento preferentemente en un sistema informático para su ejecución por el circuito de procesamiento.

La Figura 14 ilustra una estación de trabajo o servidor representativo en el que pueden ponerse en práctica uno o más aspectos de la presente invención. El sistema 5020 de la Figura 14 comprende un sistema 5021 informático base representativo, tal como un ordenador personal, una estación de trabajo o un servidor, que incluye dispositivos periféricos opcionales. El sistema 5021 informático base incluye uno o más procesadores 5026 y un bus empleado para conectar y posibilitar la comunicación entre el procesador o procesadores 5026 y los otros componentes del sistema 5021 de acuerdo con técnicas conocidas. El bus conecta el procesador 5026 a la memoria 5025 y al almacenamiento a largo plazo 5027 que puede incluir un disco duro (que incluye cualquiera de medios magnéticos, CD, DVD y Memoria Flash, por ejemplo) o una unidad de cinta por ejemplo. El sistema 5021 puede incluir también un adaptador de interfaz de usuario, que conecta el microprocesador 5026 mediante el bus a uno o más dispositivos de interfaz, tales como un teclado 5024, un ratón 5023, una impresora/escáner 5030 y/u otros dispositivos de interfaz, que puede ser cualquier dispositivo de interfaz de usuario, tal como una pantalla táctil, almohadilla de entrada digitalizada. El bus también conecta un dispositivo 5022 de visualización, tal como una pantalla de LCD o monitor, al microprocesador 5026 mediante un adaptador de pantalla.

El sistema 5021 puede comunicar con otros ordenadores o redes informáticas por medio de un adaptador de red que puede comunicar 5028 con una red 5029. Adaptadores de red ejemplares son canales de comunicación, anillo con paso de testigo, Ethernet o módems. Como alternativa, el sistema 5021 puede comunicar usando una interfaz inalámbrica, tal como una tarjeta de CDPD (datos de paquetes digitales celulares). El sistema 5021 puede asociarse con tales otros ordenadores en una red de área local (LAN) o una red de área extensa (WAN), o el sistema 5021 puede ser un cliente en una disposición de cliente/servidor con otro ordenador, etc. Todas estas configuraciones, así como el hardware y software de comunicaciones apropiadas, son conocidos en la técnica.

La Figura 15 ilustra una red 5040 de procesamiento de datos en la que puede ponerse en práctica uno o más aspectos de la presente invención. La red 5040 de procesamiento de datos puede incluir una pluralidad de redes individuales, tales como una red inalámbrica y una red alámbrica, cada una de las cuales puede incluir una pluralidad de estaciones de trabajo 5041, 5042, 5043, 5044 individuales. Adicionalmente, como apreciarán los expertos en la materia, puede incluirse una o más LAN, donde una LAN puede comprender una pluralidad de estaciones de trabajo inteligentes acopladas a un procesador de anfitrión.

Haciendo referencia aún a la Figura 15, las redes pueden incluir también ordenadores centrales o servidores, tales como un ordenador de pasarela (servidor cliente 5046) o servidor de aplicación (servidor remoto 5048 que puede acceder a un repositorio de datos y puede accederse también directamente desde una estación de trabajo 5045). Un ordenador 5046 de pasarela sirve como un punto de entrada en cada red individual. Es necesaria una pasarela cuando se conecta un protocolo de interconexión de red a otro. La pasarela 5046 puede acoplarse preferentemente a otra red (la Internet 5047 por ejemplo) por medio de un enlace de comunicaciones. La pasarela 5046 puede estar directamente acoplada a una o más estaciones de trabajo 5041, 5042, 5043, 5044 usando un enlace de comunicaciones. El ordenador de pasarela puede implementarse usando un servidor IBM eServer™ System z disponible a partir de International Business Machines Corporation.

Haciendo referencia de manera concurrente a la Figura 14 y a la Figura 15, el código de programación de software que puede incorporar uno o más aspectos de la presente invención puede accederse por el procesador 5026 del sistema 5020 de los medios 5027 de almacenamiento a largo plazo, tales como una unidad de CD-ROM o unidad de disco. El código de programación de software puede realizarse en cualquiera de una diversidad de medios conocidos para su uso con un sistema de procesamiento de datos, tal como un disquete, disco duro, o CD ROM. El código puede distribuirse en tales medios, o puede distribuirse a los usuarios 5050, 5051 de la memoria o almacenamiento de un sistema informático a través de una red a otros sistemas informáticos para su uso por usuarios de tales otros sistemas.

Como alternativa, el código de programación puede realizarse en la memoria 5025, y accederse por el procesador 5026 usando el bus de procesador. Tal código de programación incluye un sistema operativo que controla la función e interacción de los diversos componentes informáticos y uno o más programas 5032 de aplicación. El código de programa normalmente está paginado desde los medios 5027 de almacenamiento a memoria 5025 de alta velocidad donde está disponible para su procesamiento por el procesador 5026. Las técnicas y métodos para realizar código de programación de software en memoria, en medios físicos y/o distribuir código de software mediante redes son bien conocidos y no se analizarán adicionalmente en el presente documento. El código de programa, cuando se crea y almacena en un medio de almacenamiento (que incluye, pero sin limitación, módulos de memoria electrónica (RAM), memoria flash, Discos Compactos (CD), DVD, Cinta Magnética y similares a menudo se denomina como un "producto de programa informático". El medio de producto de programa informático es típicamente legible por un circuito de procesamiento preferentemente en un sistema informático para su ejecución por el circuito de procesamiento.

La caché está en su mayoría fácilmente disponible para el procesador (normalmente más rápidas y más pequeñas que otras cachés del procesador) es la caché más inferior (L1 o nivel uno) y el almacén principal (memoria principal) es la caché de nivel más alta (L3 si hay 3 niveles). La caché de nivel más baja a menudo se divide en una caché de instrucciones (I-caché) que mantiene instrucciones de máquina a ejecutarse y una caché de datos (D-caché) que

maneja operandos de datos.

Haciendo referencia a la Figura 16, se representa una realización de procesador ejemplar para el procesador 5026. Típicamente se emplean uno o más niveles de caché 5053 para almacenar en memoria intermedia bloques para mejorar el rendimiento de procesador. La caché 5053 es una memoria intermedia de alta velocidad que mantiene líneas de caché de datos de memoria que es probable que se usen. Las líneas de caché típicas son 64, 128 o 256 bytes de datos de memoria. Las cachés de separadas a menudo se emplean para almacenar en caché instrucciones que para almacenar en caché datos. La coherencia de caché (sincronización de copias de líneas en memoria y las cachés) a menudo se proporciona por diversos algoritmos "sondeo" bien conocidos en la técnica. El almacén 5025 de memoria principal de un sistema de procesador a menudo se denomina como una caché. En un sistema de procesador que tiene 4 niveles de caché 5053, un almacén 5025 principal a menudo se denomina como la caché de nivel 5 (L5) puesto que típicamente es más rápido y únicamente mantiene una porción del almacenamiento no volátil (DASD, cinta, etc.) que está disponible para un sistema informático. El almacén 5025 principal "almacena en caché" páginas de datos paginados en y fuera del almacén 5025 principal por el sistema operativo.

Un contador de programa (contador de instrucción) 5061 mantiene el seguimiento de la dirección de la instrucción actual que va a ejecutarse. Un contador de programa en un procesador z/Architecture es de 64 bits y puede truncarse a 31 o 24 bits para soportar límites de direccionamiento anterior. Un contador de programa se realiza típicamente en una PSW (palabra de estado de programa) de un ordenador de manera que persiste durante la conmutación de contexto. Por lo tanto, un programa en progreso, que tiene un valor de contador de programa, puede interrumpirse, por ejemplo, mediante el sistema operativo (conmutación de contexto del entorno de programa al entorno del sistema operativo). La PSW del programa mantiene el valor de contador de programa mientras el programa no está activo, y se usa el contador de programa (en la PSW) del sistema operativo mientras se está ejecutando el sistema operativo. Típicamente, el contador de programa se incrementa por una cantidad igual al número de bytes de la instrucción actual. Las instrucciones RISC (Informática de Conjunto Reducido de Instrucciones) típicamente están fijadas en longitud mientras que las instrucciones CISC (Informática de Conjunto Complejo de Instrucciones) típicamente tienen longitud variable. Las instrucciones de IBM z/Architecture son instrucciones CISC que tienen una longitud de 2, 4 o 6 bytes. El contador de programa 5061 se modifica por cualquiera de una operación de conmutación de contexto o una operación de ramal tomada de una instrucción de ramal, por ejemplo. En una operación de conmutación de contexto, se graba el valor de contador de programa actual en la palabra de estado de programa junto con otra información de estado acerca del programa que se está ejecutando (tal como códigos de condición), y se carga un nuevo valor de contador de programa que apunta a una instrucción de un nuevo módulo de programa para que se ejecute. Se realiza una operación de ramal tomado para permitir que el programa haga decisiones o realice bucles dentro del programa cargando el resultado de la instrucción de ramal en el contador de programa 5061.

Típicamente se emplea una unidad 5055 de recuperación de instrucción para recuperar instrucciones en nombre del procesador 5026. La unidad de recuperación recupera "siguientes instrucciones secuenciales", instrucciones objetivo de instrucciones de ramal tomado, o primeras instrucciones de un programa que sigue una conmutación de contexto. Las unidades de recuperación de instrucción modernas a menudo emplean técnicas de pre-recuperación para pre recuperar de manera especulativa instrucciones basándose en la probabilidad de que las instrucciones pre recuperadas puedan usarse. Por ejemplo, una unidad de recuperación puede recuperar 16 bytes de instrucciones que incluyen la siguiente instrucción secuencial y bytes adicionales de instrucciones secuenciales adicionales.

Las instrucciones recuperadas se ejecutan a continuación por el procesador 5026. En una realización, la instrucción o instrucciones recuperadas se pasan a una unidad 5056 de despacho de la unidad de recuperación. La unidad de despacho decodifica la instrucción o instrucciones y reenvía información acerca de la instrucción o instrucciones decodificadas a las unidades 5057, 5058, 5060 apropiadas. Una unidad 5057 de ejecución recibirá típicamente información acerca de instrucciones aritméticas decodificadas de la unidad 5055 de recuperación de instrucción y realizará operaciones aritméticas en operandos de acuerdo con el opcode de la instrucción. Los operandos se proporcionan a la unidad 5057 de ejecución preferentemente desde la memoria 5025, los registros 5059 de arquitectura desde un campo inmediato de la instrucción que se está ejecutando. Los resultados de la ejecución, cuando se almacenan, se almacenan en la memoria 5025, registros 5059 o en otro hardware de máquina (tal como registros de control, registros de PSW y similares).

Un procesador 5026 típicamente tiene una o más unidades 5057, 5058, 5060 para ejecutar la función de la instrucción. Haciendo referencia a la Figura 17A, una unidad 5057 de ejecución puede comunicar con registros 5059 generales de arquitectura, una unidad 5056 de decodificación/despacho, una unidad 5060 de almacén de carga, y otras 5065 unidades de procesador por medio de la lógica 5071 de interconexión. Una unidad 5057 de ejecución puede emplear varios circuitos 5067, 5068, 5069 de registro para mantener la información en la que operará la unidad aritmético-lógica (ALU) 5066. La ALU realiza operaciones aritméticas tales como añadir, restar, multiplicar y dividir así como la función lógica tal como y, o y o exclusiva (XOR), rotar y desplazar. Preferentemente la ALU soporta operaciones especializadas que son dependientes del diseño. Otros circuitos pueden proporcionar otras funciones 5072 de arquitectura que incluyen códigos de condición y recuperar la lógica de soporte por ejemplo. Típicamente el resultado de una operación de ALU se mantiene en un circuito 5070 de registro de salida que puede reenviar el resultado a una diversidad de otras funciones de procesamiento. Hay muchas disposiciones de unidades de procesador, la presente descripción se pretende únicamente para proporcionar un entendimiento representativo de una realización.

Una instrucción ADD, por ejemplo, se ejecutaría en una unidad 5057 de ejecución que tiene funcionalidad aritmética y lógica mientras que una instrucción de punto flotante, por ejemplo, se ejecutaría en una ejecución de punto flotante que tiene capacidad de punto flotante especializada. Preferentemente, una unidad de ejecución opera en operandos identificados por una instrucción realizando una función definida opcode en los operandos. Por ejemplo, una instrucción ADD puede ejecutarse por una unidad 5057 de ejecución en operandos hallando en dos registros 5059 identificados por campos de registro de la instrucción.

La unidad 5057 de ejecución realiza la adición aritmética en dos operandos y almacena el resultado en un tercer operando donde el tercer operando puede ser un tercer registro o uno de los dos registros de origen. La unidad de ejecución utiliza preferentemente una Unidad Aritmético Lógica (ALU) 5066 que puede realizar una diversidad de funciones lógicas tales como Desplazar, Rotar, y, o y XOR así como una diversidad de funciones algebraicas que incluyen cualquiera de añadir, restar, multiplicar y dividir. Algunas ALU 5066 están diseñadas para operaciones escalares y algunas para punto flotante. Los datos pueden ser Big Endian (donde el byte menos significativo es en la dirección de byte más alta) o Little Endian (donde el byte menos significativo es en la dirección de byte más baja) dependiendo de la arquitectura. IBM z/Architecture es Big Endian. Los campos con signo pueden ser de signo y magnitud, en complemento de 1 o en complemento de 2 dependiendo de la arquitectura. Un número de complemento de 2 es ventajoso en que la ALU no necesita diseñar una capacidad de resta puesto que un valor negativo o un valor positivo en complemento de 2 requiere únicamente una adición dentro de la ALU. Los números se describen comúnmente en forma abreviada, donde un campo de 12 bits define una dirección de un bloque de 4096 bytes y se describe comúnmente como un bloque de 4 Kbytes (Kilobytes), por ejemplo.

Haciendo referencia a la Figura 17B, la información de instrucción de ramal para ejecutar una instrucción de ramal se envía típicamente a una unidad 5058 de ramal que a menudo emplea un algoritmo de predicción de ramal tal como una tabla 5082 de historial de ramal para predecir el resultado del ramal antes de que se completen las otras operaciones condicionales. El objetivo de la instrucción de ramal actual se recuperará y se ejecutará de manera especulativa antes de que se completen las operaciones condicionales. Cuando se completan las operaciones condicionales, las instrucciones de ramal ejecutadas de manera especulativa se completan o descartan basándose en las condiciones de la operación condicional y el resultado especulado. Una instrucción de ramal típica puede probar los códigos de condición y ramal a una dirección objetivo si los códigos de condición cumplen el requisito de ramal de la instrucción de ramal, puede calcularse una dirección objetivo basándose en varios números que incluyen unos hallados en campos de registro o un campo inmediato de la instrucción, por ejemplo. La unidad 5058 de ramal puede emplear una ALU 5074 que tiene una pluralidad de circuitos 5075, 5076, 5077 de registro de entrada y un circuito 5080 de registro de salida. La unidad 5058 de ramal puede comunicar con registros 5059 generales, unidad 5056 de despacho de decodificación u otros circuitos 5073, por ejemplo.

La ejecución de un grupo de instrucciones puede interrumpirse por una diversidad de razones que incluyen una conmutación de contexto iniciada por un sistema operativo, una excepción de programa o error que provoca una conmutación de contexto, una señal de interrupción de E/S que provoca una conmutación de contexto o actividad de múltiples hilos de una pluralidad de programas (en un entorno de múltiples hilos), por ejemplo. Preferentemente una acción de conmutación de contexto graba información de estado acerca de un programa actualmente en ejecución y a continuación carga información de estado acerca de otro programa que se está invocando. La información de estado puede grabarse en registros de hardware o en memoria, por ejemplo. La información de estado preferentemente comprende un valor de contador de programa que apunta a una siguiente instrucción a ejecutarse, códigos de condición, información de traducción de memoria y contenido de registro de arquitectura. Una actividad de conmutación de contexto puede ejercerse por circuitos de hardware, programas de aplicación, programas de sistema operativo o código de firmware (microcódigo, pico-código o código interno con licencia (LIC)) en solitario o en combinación.

Un procesador accede a operandos de acuerdo con métodos de instrucción definida. La instrucción puede proporcionar un operando inmediato que usa el valor de una porción de la instrucción, puede proporcionar uno o más campos de registro que apuntan explícitamente a cualesquiera registros de fin general o registros de fin especial (registros de punto flotante, por ejemplo). La instrucción puede utilizar registros implicados identificados por un campo opcode como operandos. La instrucción puede utilizar ubicaciones de memoria para operandos. Una ubicación de memoria de un operando puede proporcionarse por un registro, un campo inmediato, o una combinación de registros y campo inmediato como se ejemplifica por la función de desplazamiento largo de z/Architecture en donde la instrucción define un registro de base, un registro de índice y un campo inmediato (campo de desplazamiento) que se añaden juntos para proporcionar la dirección del operando en memoria, por ejemplo. La ubicación en el presente documento típicamente implica una ubicación en memoria principal (almacenamiento principal) a menos que se indique lo contrario.

Haciendo referencia a la Figura 17C, un procesador accede a almacenamiento usando una unidad 5060 de carga/almacén. La unidad 5060 de carga/almacén puede realizar una operación de carga obteniendo la dirección del operando objetivo en memoria 5053 y cargar el operando en un registro 5059 u otra ubicación de memoria 5053, o puede realizar una operación de almacén obteniendo la dirección del operando objetivo en memoria 5053 y almacenar datos obtenidos de un registro 5059 u otra ubicación de memoria 5053 en la ubicación de operando objetivo en

memoria 5053. La unidad 5060 de carga/almacén puede ser especulativa y puede acceder a memoria en una secuencia que está desordenada con relación a la secuencia de instrucción sin embargo, la unidad 5060 de carga/almacén es para mantener la apariencia a programas en los que se ejecutaron instrucciones en orden. Una unidad 5060 de carga/almacén puede comunicar con registros 5059 generales, unidad 5056 de decodificación/despacho, interfaz 5053 de caché/memoria u otros elementos 5083 y comprende diversos circuitos de registro, ALU 5085 y lógica 5090 de control para calcular direcciones de almacenamiento y para proporcionar la generación de secuencias en tuberías para mantener las operaciones en orden. Algunas operaciones pueden estar desordenadas pero la unidad de carga/almacén proporciona funcionalidad para hacer que las operaciones desordenadas aparezcan para el programa como que se han realizado en orden, como es bien conocido en la técnica.

Preferentemente las direcciones que un programa de aplicación "observa" a menudo se denominan como direcciones virtuales. Las direcciones virtuales se denominan en ocasiones como "direcciones lógicas" y "direcciones efectivas". Estas direcciones virtuales son virtuales en que se redirigen a la ubicación de memoria física por una de una diversidad de tecnologías de traducción de dirección dinámica (DAT) que incluyen, pero sin limitación, prefijar simplemente una dirección virtual con un valor de desplazamiento, traducir la dirección virtual mediante una o más tablas de traducción, las tablas de traducción preferentemente que comprenden al menos una tabla de segmentos y una tabla de página en solitario o en combinación, preferentemente, la tabla de segmentos que tiene una entrada que apunta a la tabla de páginas. En z/Architecture, se proporciona una jerarquía de traducción que incluye una primera tabla de región, una segunda tabla de región, una tercera tabla de región, una tabla de segmento y una tabla de página opcional. El rendimiento de la traducción de dirección a menudo se mejora utilizando una memoria intermedia de traducción adelantada (TLB) que comprende entradas que mapean una dirección virtual a una ubicación de memoria física asociada. Las entradas se crean cuando la DAT traduce una dirección virtual usando las tablas de traducción. Posterior al uso de la dirección virtual pueden a continuación utilizar la entrada de la TLB rápida en lugar de los accesos de tabla de traducción secuencial lenta. El contenido de la TLB puede gestionarse por una diversidad de algoritmos de sustitución que incluyen LRU (Menos Recientemente usado).

En el caso donde el procesador es un procesador de un sistema multi-procesador, cada procesador tiene la responsabilidad de mantener recursos compartidos, tal como E/S, cachés, TLB y memoria, interbloqueado por coherencia. Típicamente, las tecnologías de "sondeo" se utilizarán al mantener la coherencia de caché. En un entorno de sondeo, cada línea de caché puede marcarse como que está en uno cualquiera de un estado compartido, un estado exclusivo, un estado cambiado, un estado inválido y similares para facilitar la compartición.

Las unidades 5054 de E/S (Figura 16) proporcionan al procesador con medios para fijarse a dispositivos periféricos que incluyen cinta, disco, impresoras, pantallas y redes, por ejemplo. Las unidades de E/S a menudo se presentan al programa informático por controladores de software. En ordenadores principales, tales como System z de IBM®, los adaptadores de canal y adaptadores de sistema abiertos son unidades de E/S del ordenador principal que proporcionan las comunicaciones entre el sistema operativo y dispositivos periféricos.

Además, otros tipos de entornos informáticos pueden beneficiarse de uno o más aspectos de la presente invención. Como un ejemplo, un entorno puede incluir un emulador (por ejemplo, software u otros mecanismos de emulación), en el que una arquitectura particular (que incluye, por ejemplo, ejecución de instrucción, funciones de arquitectura, tal como traducción de dirección, y registros de arquitectura) o un subconjunto de las mismas se emula (por ejemplo, en un sistema informático nativo que tiene un procesador y memoria). En un entorno de este tipo, una o más funciones de emulación del emulador pueden implementar uno o más aspectos de la presente invención, incluso aunque un ordenador que ejecuta el emulador puede tener una arquitectura diferente de las capacidades que se están emulando. Como un ejemplo, en modo de emulación, se decodifica la instrucción específica u operación que se está emulando, y se crea una función de emulación apropiada para implementar la instrucción u operación individual.

En un entorno de emulación, un ordenador de anfitrión incluye, por ejemplo, una memoria para almacenar instrucciones y datos; una unidad de recuperación de instrucción para recuperar instrucciones de memoria y para proporcionar, opcionalmente, almacenamiento en memoria intermedia local para la instrucción recuperada; una unidad de decodificación de instrucción para recibir instrucciones recuperadas y para determinar el tipo de instrucciones que se han recuperado; y una unidad de ejecución de instrucción para ejecutar las instrucciones. La ejecución puede incluir datos de carga en un registro de la memoria; almacenar datos de vuelta a la memoria de un registro; o realizar algún tipo de operación aritmética o lógica, como se determina por la unidad de decodificación. En un ejemplo, cada unidad se implementa en software. Por ejemplo, las operaciones que se realizan por las unidades se implementan como una o más subrutinas dentro del software de emulador.

Más particularmente, en un ordenador central, las instrucciones de máquina de arquitectura se usan por programadores, normalmente hoy en día programadores de "C", a menudo por medio de una aplicación de compilador. Estas instrucciones almacenadas en el medio de almacenamiento pueden ejecutarse de manera nativa en un Servidor de z/Architecture IBM®, o como alternativa en máquinas al ejecutar otras arquitecturas. Pueden emularse en los servidores de ordenadores centrales de IBM® existentes y futuros y en otras máquinas de IBM® (por ejemplo, servidores de Power Systems y servidores System x®). Pueden ejecutarse en máquinas que ejecutan Linux en una amplia diversidad de máquinas usando hardware fabricado por IBM®, Intel®, AMD™, y otras. Además de la ejecución en ese hardware bajo una z/Architecture, puede usarse Linux así como máquinas que usan emulación por Hercules,

UMX, o FSI (Fundamental Software, Inc), donde la ejecución en general es en un modo de emulación. En modo de emulación, se ejecuta software de emulación por un procesador nativo para emular la arquitectura de un procesador emulado.

El procesador nativo típicamente ejecuta software de emulación que comprende cualquiera de firmware o un sistema operativo nativo para realizar la emulación del procesador emulado. El software de emulación es responsable de recuperar y ejecutar instrucciones de la arquitectura de procesador emulada. El software de emulación mantiene un contador de programa emulado para mantener el seguimiento de límites de instrucción. El software de emulación puede recuperar una o más instrucciones de máquina emuladas a la vez y convertir la una o más instrucciones de máquina emuladas a un correspondiente grupo de instrucciones de máquina nativas para su ejecución por el procesador nativo. Estas instrucciones convertidas pueden almacenarse en caché de manera que puede conseguirse una conversión más rápida. No obstante, el software de emulación es para mantener las reglas de arquitectura de la arquitectura de procesador emulada para asegurar que los sistemas operativos y aplicaciones escritos para el procesador emulado operan correctamente. Adicionalmente, el software de emulación es para proporcionar recursos identificados por la arquitectura de procesador emulada que incluye, pero sin limitación, registros de control, registros de fin general, registros de punto flotante, función de traducción de dirección dinámica que incluye tablas de segmento y tablas de página por ejemplo, mecanismos de interrupción, mecanismos de conmutación de contexto, relojes de hora del día (TOD) e interfaces de arquitectura a subsistemas de E/S de manera que un sistema operativo o un programa de aplicación designado para ejecutarse en el procesador emulado, puede ejecutarse en el procesador nativo que tiene el software de emulación.

Se decodifica una instrucción específica que se está emulando, y se solicita una subrutina para realizar la función de la instrucción individual. Se implementa una función de software de emulación que emula una función de un procesador emulado, por ejemplo, en una subrutina o controlador de "C", o algún otro método de suministro de un controlador para el hardware específico como estará dentro de la habilidad de los expertos en la materia después de entender la descripción de la realización preferida. Diversas patentes de emulación de software y hardware que incluyen, pero sin limitación la Patente de Cartas de Estados Unidos N.º 5.551.013, titulada "Multiprocessor for Hardware Emulation", por Beausoleil et al.; y Patente de Cartas de Estados Unidos N.º 6.009.261, titulada "Preprocessing of Stored Target Routines for Emulating Incompatible Instructions on a Target Processor", por Scalzi et al; y Patente de Cartas de Estados Unidos N.º 5.574.873, titulada "Decoding Guest Instruction to Directly Access Emulation Routines that Emulate the Guest Instructions", por Davidian et al; y Patente de Cartas de Estados Unidos N.º 6.308.255, titulada "Symmetrical Multiprocessing Bus and Chipset Used for Coprocessor Support Allowing Non-Native Code to Run in a System", por Gorishek et al; y Patente de Cartas de Estados Unidos N.º 6.463.582, titulada "Dynamic Optimizing Object Code Translator for Architecture Emulation and Dynamic Optimizing Object Code Translation Method", por Lethin et al; y la Carta de Patente de Estados Unidos 5.790.825, titulada "Method for Emulating Guest Instructions on a Host Computer Through Dynamic Recompilation of Host Instructions", por Eric Traut; y muchas otras, ilustran una diversidad de maneras conocidas para conseguir la emulación de un formato de instrucción diseñado para una máquina diferente para una máquina objetivo disponible para los expertos en la materia.

En la Figura 18, se proporciona un ejemplo de un sistema 5092 informático de anfitrión emulado que emula un sistema 5000' ordenador de anfitrión de una arquitectura de anfitrión. En el sistema 5092 informático de anfitrión emulado, el procesador de anfitrión (CPU) 5091 es un procesador de anfitrión emulado (o procesador de anfitrión virtual) y comprende un procesador 5093 de emulación que tiene una arquitectura de conjunto de instrucciones nativas diferentes que las del procesador 5091 del ordenador 5000' anfitrión. El sistema 5092 informático de anfitrión emulado tiene la memoria 5094 accesible al procesador 5093 de emulación. En la realización de ejemplo, la memoria 5094 se particiona en una porción de memoria 5096 de ordenador de anfitrión y una porción de rutinas 5097 de emulación. La memoria 5096 ordenador anfitrión está disponible para programas del ordenador 5092 de anfitrión emulado de acuerdo con la arquitectura del ordenador de anfitrión. El procesador 5093 de emulación ejecuta instrucciones nativas de un conjunto de instrucciones de arquitectura de una arquitectura distinta de la del procesador 5091 emulado, las instrucciones nativas obtenidas de la memoria 5097 de rutinas de emulación, y puede acceder a una instrucción de anfitrión para la ejecución de un programa en memoria 5096 informática de anfitrión empleando una o más instrucción o instrucciones obtenidas en una secuencia y rutina de acceso/decodificación que puede decodificar la instrucción o instrucciones de anfitrión accedidas para determinar una rutina de ejecución de instrucción nativa para emular la función de la instrucción de anfitrión accedida. Otras funciones que se definen para la arquitectura del sistema 5000' informático de anfitrión pueden emularse mediante rutinas de funciones de arquitectura, que incluyen tales funciones como registros de fin general, registros de control, traducción de dirección dinámica y caché de soporte y procesador de subsistema de E/S, por ejemplo. Las rutinas de emulación pueden aprovecharse también de funciones disponibles en el procesador 5093 de emulación (tal como registros generales y traducción dinámica de direcciones virtuales) para mejorar el rendimiento de las rutinas de emulación. Los motores de hardware especial y de descarga pueden proporcionarse también para ayudar al procesador 5093 a emular la función del ordenador 5000' de anfitrión.

La terminología usada en el presente documento es para el fin de describir realizaciones particulares únicamente y no se pretende que sea para limitar la invención. Como se usa en el presente documento, las formas singulares "un", "una" y "el", "la" se pretende que incluyan las formas plurales también, a menos que el contexto lo indique claramente lo contrario. Se entenderá adicionalmente que los términos "comprende" y/o "comprendiendo/que comprende", cuando se usan en esta memoria descriptiva, especifican la presencia de características indicadas, elementos integrantes,

etapas, operaciones, elementos y/o componentes, pero no excluyen la presencia o adición de una o más otras características, elementos integrantes, etapas, operaciones, elementos, componentes y/o grupos de los mismos.

Las correspondientes estructuras, materiales, actos y equivalentes de todos los significados o elementos de etapa más función en las reivindicaciones a continuación, si las hubiera, se pretende que incluyan cualquier estructura, material o acto para realizar la función en combinación con otros elementos reivindicados como se revindica específicamente. La descripción de uno o más aspectos de la presente invención se ha presentado para fines de ilustración y descripción, pero no se pretende que sea exhaustiva o limite a la invención en la forma desvelada. Serán evidentes muchas modificaciones y variaciones para los expertos en la materia sin alejarse del alcance de la invención. La realización se eligió y describió para explicar mejor los principios de la invención y la aplicación práctica, y para posibilitar a otros expertos en la materia entender la invención para diversas realizaciones con diversas modificaciones como que son adecuadas al uso particular contemplado.

Capítulo 23. Instrucciones de cadena de vector

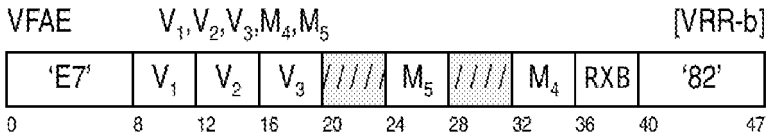
Instalación de cadena de vector

Instrucciones

A menos que se especifique lo contrario, todos los operandos son operandos de registro de vector. Una "V" en la sintaxis de ensamblador designa un operando de vector.

Nombre	Mnemotécnica	Características					Opcode	Página
VECTOR HALLAR CUALQUIER IGUAL	VFAE	VRR-b C* VF	o[9] SP	Dv			E782	23-1
VECTOR HALLAR ELEMENTO IGUAL	VFEE	VRR-b C* VF	o[9] SP	Dv			E780	23-2
VECTOR HALLAR ELEMENTO NO IGUAL	VFENE	VRR-b C* VF	o[9] SP	Dv			E781	23-3
VECTOR COMPARAR RANGO DE CADENA	VSTRC	VRR-d C* VF	o[9] SP	Dv			E78A	23-4

VECTOR HALLAR CUALQUIER IGUAL



El procedimiento de izquierda a derecha, cada elemento de número entero binario sin signo del segundo operando se compara para su igualdad con cada elemento de número entero binario sin signo del tercer operando y opcionalmente cero si se establece la bandera de búsqueda cero en el campo M_5.

Si la bandera de tipo de resultado (RT) en el campo M_5 es cero, entonces para cada elemento en el segundo operando que coincide con cualquier elemento en el tercer operando, u opcionalmente cero, las posiciones de bits del correspondiente elemento en el primer operando se establecen a unos, de lo contrario se establecen a cero. Si la bandera de tipo de resultado (RT) en el campo M_5 es uno, entonces el índice de byte del elemento más a la izquierda en el segundo operando que coincide con un elemento en el tercer operando o cero se almacena en el byte siete del primer operando.

Cada instrucción tiene una sección de mnemotécnica extendida que describe mnemotécnicas extendidas recomendadas y su correspondiente sintaxis de ensamblador máquina.

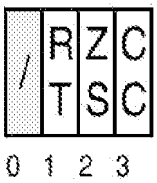
Nota de programación: para todas las instrucciones que establecen opcionalmente el código de condición, puede degradarse el rendimiento si se establece el código de condición.

Si la bandera de tipo de resultado (RT) en el campo M_5 es uno y no se hallan bytes para que sean iguales, o cero si se establece la bandera de búsqueda cero, se almacena un índice igual al número de bytes en el vector en el byte siete del primer operando.

El campo M_4 especifica el control de tamaño de elemento (ES). El control de ES especifica el tamaño de los elementos en los operandos de registro de vector. Si se especifica un valor reservado, se reconoce una excepción de especificación.

0 - Byte
1 - Semi-palabra
2 - Palabra
3-15 - Reservado

El campo M_5 tiene el siguiente formato:



Los bits del campo M_5 se definen como sigue:

- *Tipo de resultado (RT):* si es cero, cada elemento resultante es una máscara de todas las comparaciones de rango en ese elemento. Si es uno, se almacena un índice de byte en el byte siete del primer operando y se almacenan ceros en todos los otros elementos.
- *Búsqueda de cero (ZS):* si es uno, cada elemento del segundo operando se compara también a cero.
- *Establecer código de condición (CC):* si es cero, el código de condición no se establece y permanece sin variar. Si es uno, el código de condición se establece como se especifica en la siguiente sección.

Condiciones especiales

Se reconoce una excepción de especificación y no se toma otra acción si tiene lugar cualquiera de lo siguiente:

1. El campo M4 contiene un valor de 3-15.
2. El bit 0 del campo M5 no es cero.

Código de condición resultante:

Si la bandera CC es cero, el código permanece sin variar.

Si la bandera CC es uno, el código se establece como sigue:

0 Si se establece el bit ZS, no hubo coincidencias en un elemento indexado menor que cero en el segundo operando.

1 Algunos elementos del segundo operando coinciden con al menos un elemento en el tercer operando

2 Todos los elementos del segundo operando coinciden con al menos un elemento en el tercer operando

3 Ningún elemento en el segundo operando coincide con algún elemento en el tercer operando

Excepciones de programa:

1 Datos con DXC FE, Registro de Vector

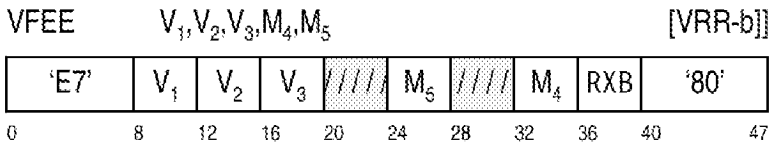
- Operación si la instalación de extensión de vector no está instalada
- Especificación (valor ES reservado)
- Restricción de transacción

Mnemotécnicas extendidas:

VFAEB	V_1,V_2,V_3,M_5	VFAE $V_1,V_2,V_3,0,M_5$
VFAEH	V_1,V_2,V_3,M_5	VFAE $V_1,V_2,V_3,1,M_5$
VFAEF	V_1,V_2,V_3,M_5	VFAE $V_1,V_2,V_3,2,M_5$
VFAEBS	V_1,V_2,V_3,M_5	VFAE $V_1,V_2,V_3,0,(M_5 \mid X'1')$
VFAEHS	V_1,V_2,V_3,M_5	VFAE $V_1,V_2,V_3,1,(M_5 \mid X'1')$
VFAEFS	V_1,V_2,V_3,M_5	VFAE $V_1,V_2,V_3,2,(M_5 \mid X'1')$

VFAEZB	V_1,V_2,V_3,M_5	VFAE $V_1,V_2,V_3,0,(M_5 \mid X'2')$
VFAEZH	V_1,V_2,V_3,M_5	VFAE $V_1,V_2,V_3,1,(M_5 \mid X'2')$
VFAEZF	V_1,V_2,V_3,M_5	VFAE $V_1,V_2,V_3,2,(M_5 \mid X'2')$
VFAEZBS	V_1,V_2,V_3,M_5	VFAE $V_1,V_2,V_3,0,(M_5 \mid X'3')$
VFAEZHS	V_1,V_2,V_3,M_5	VFAE V_1,V2,V3,1,(M_5 | X'3')
VFAEZFS	V_1,V_2,V_3,M_5	VFAE $V_1,V_2,V_3,2,(m_5 \mid X'3')$

VECTOR HALLAR ELEMENTO IGUAL



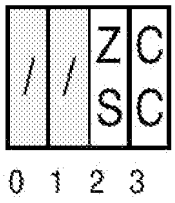
Continuando de izquierda a derecha, los elementos de número entero binario sin signo del segundo operando se comparan con los correspondientes elementos de número entero binario sin signo del tercer operando. Si dos elementos son iguales, el índice de byte del primer byte del elemento igual más a la izquierda se coloca en el byte siete del primer operando. Se almacenan ceros en los restantes bytes del primer operando. Si no se hallan bytes que sean iguales, o cero, si se establece la comparación de cero, a continuación se almacena un índice igual al número de bytes en el vector en el byte siete del primer operando. Se almacenan ceros en los bytes restantes.

Si se establece el bit de Búsqueda de Cero (ZS) en el campo M_5, entonces cada elemento en el segundo operando se compara también para su igualdad con cero. Si se halla un elemento cero en el segundo operando antes de que cualesquiera otros elementos del segundo y terceros operandos sean iguales, el índice de byte del primer byte del elemento hallado que es cero se almacena en el byte siete el primer operando y se almacenan ceros en todas las otras ubicaciones de bytes. Si la bandera Establecer Código de Condición (CC) es uno, entonces el código de condición se establece a cero.

El campo M_4 especifica el control de tamaño de elemento (ES). El control de ES especifica el tamaño de los elementos en los operandos de registro de vector. Si se especifica un valor reservado, se reconoce una excepción de especificación.

0 - Byte
1 - Semi-palabra
2 - Palabra
3-15 - Reservado

El campo M_5 tiene el siguiente formato:



Los bits del campo M_5 se definen como sigue:

- *Reservado:* los bits 0-1 están reservados y deben ser cero. De lo contrario, se reconoce una excepción de especificación.
- *Búsqueda de cero (ZS):* si es uno, cada elemento del segundo operando se compara también a cero.
- *Establecer código de condición (CC):* si es cero, el código de condición permanece sin variar. Si es uno, el código de condición se establece como se especifica en la siguiente sección.

Condiciones especiales

Se reconoce una excepción de especificación y no se toma otra acción si tiene lugar cualquiera de lo siguiente:

1. El campo M_4 contiene un valor de 3-15.

2. Los bits 0-1 del campo M5 no son cero.

Código de condición resultante:

Si el bit 3 del campo M_5 se establece a uno, el código se establece como sigue:

0 Si se establece el bit de comparación de cero, la comparación detecta un elemento cero en el segundo operando en un elemento con un índice menor que cualesquiera comparaciones iguales.
1 La comparación detecta una coincidencia entre el segundo y terceros operandos en algún elemento. Si se establece el bit de comparación de cero, esta coincidencia tiene lugar en un elemento con un índice menor o igual que el elemento de comparación de cero.
2 –
3 Ningún elemento se compara igual.

Si el bit 3 del campo M_5 es cero, el código permanece sin variar.

Excepciones de programa:

- Datos con DXC FE, registro de vector
- Operación si la instalación de extensión de vector no está instalada
- Especificación (valor ES reservado)
- Restricción de transacción

Mnemotécnicas extendidas:

VFEEB	V_1,V_2,V_3,M_5	VFEE $V_1,V_2,V_3,0,M_5$
VFEEH	V_1,V_2,V_3,M_5	VFEE $V_1,V_2,V_3,1,M_5$
VFEEF	V_1,V_2,V_3,M_5	VFEE $V_1,V_2,V_3,0,(M_5 \mid X'1')$
VFEEHS	V_1,V_2,V_3,M_5	VFEE $V_1,V_2,V_3,1,(M_5 \mid X'1')$
VFEEFS	V_1,V_2,V_3,M_5	VFEE $V_1,V_2,V_3,2,(M_5 \mid X'1')$
VFEEZB	V_1,V_2,V_3,M_5	VFEE $V_1,V_2,V_3,0,(M_5 \mid X'2')$
VFEEZH	V_1,V_2,V_3,M_5	VFEE $V_1,V_2,V_3,1,(M_5 \mid X'2')$
VFEEZF	V_1,V_2,V_3,M_5	VFEE $V_1,V_2,V_3,2,(M_5 \mid X'2')$
VFEEZBS	V_1,V_2,V_3,M_5	VFEE $V_1,V_2,V_3,0,(M_5 \mid X'3')$
VFEEZHS	V_1,V_2,V_3,M_5	VFEE $V_1,V_2,V_3,1,(M_5 \mid X'3')$
VFEEZFS	V_1,V_2,V_3,M_5	VFEE $V_1,V_2,V_3,2,(M_5 \mid X'3')$

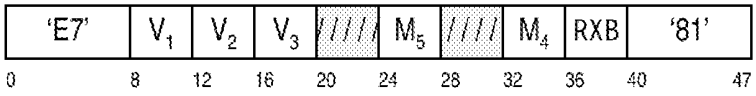
Notas de programación:

1. Un índice de byte siempre se almacena en el primer operando para cualquier tamaño de elemento. Por ejemplo, si el tamaño de elemento se estableció a semi-palabra y la 2ª semi-palabra indexada se comparó igual, entonces se almacenaría un índice de byte de 4.

2. El tercer operando no debe contener elementos con un valor de cero. Si el tercer operando no contiene un cero y coincide con un elemento cero en el segundo operando antes de cualesquiera otras comparaciones, se establece el código de condición uno independientemente del ajuste de bit de comparación de cero.

VECTOR HALLAR ELEMENTO NO IGUAL

VFENE V_1,V_2,V_3,M_4,M_5 [VRR-b]



Continuando de izquierda a derecha, los elementos de número entero binario sin signo del segundo operando se comparan con los correspondientes elementos de número entero binario sin signo del tercer operando. Si dos elementos no son iguales, el índice de byte del elemento no igual más a la izquierda se coloca en el byte siete del primer operando y se almacenan ceros en todos los otros bytes. Si el bit de Establecimiento de Código de Condición (CC) en el campo M_5 se establece a uno, el código de condición se establece para indicar qué operando era mayor. Si todos los elementos eran iguales, a continuación un índice de byte igual al tamaño del vector se coloca en el byte siete del primer operando y se colocan ceros en todas las otras ubicaciones de bytes. Si el bit CC es uno, se establece

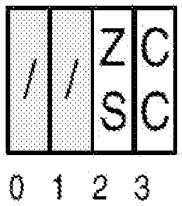
el código de condición tres.

Si se establece el bit de búsqueda de cero (ZS) en el campo M_5, cada elemento en el segundo operando también se compara para su igualdad con cero. Si se halla un elemento cero en el segundo operando antes de que se halle que cualquier otro elemento del segundo operando que sea desigual, el índice de byte del primer byte del elemento hallado que es cero se almacena en el byte siete del primer operando. Se almacenan ceros en todos los otros bytes y se establece el código de condición 0.

El campo M_4 especifica el control de tamaño de elemento (ES). El control de ES especifica el tamaño de los elementos en los operandos de registro de vector. Si se especifica un valor reservado, se reconoce una excepción de especificación.

0 - Byte
1 - Semi-palabra
2 - Palabra
3-15 - Reservado

El campo M_5 tiene el siguiente formato:



Los bits del campo M_5 se definen como sigue:

- *Búsqueda de cero (ZS):* si es uno, cada elemento del segundo operando se compara también a cero.
- *Establecer código de condición (CC):* si es cero, el código de condición no se establece y permanece sin variar. Si es uno, el código de condición se establece como se especifica en la siguiente sección.

Condiciones especiales

Se reconoce una excepción de especificación y no se toma otra acción si tiene lugar cualquiera de lo siguiente:

1. El campo M_4 contiene un valor de 3-15.
2. Los bits 0-1 del campo M_5 no son cero.

Código de condición resultante:

Si el bit 3 del campo M_5 se establece a uno, el código se establece como sigue:

0 si el cero, se establece el bit de comparación, la comparación detecta un elemento cero en ambos operandos en un elemento indexado menor que cualquier comparación de desigualdad
1 Se detecta un desajuste de elemento y el elemento en VR2 es menor que el elemento en VR3
2 Se detecta un desajuste de elemento y el elemento en VR2 es mayor que el elemento en VR3
3 Todos los elementos se comparan igual, y si se establece el bit de comparación de cero, no se hallaron elementos cero en el segundo operando.

Si el bit 3 del campo M_5 es cero, el código permanece sin variar.

Excepciones de programa:

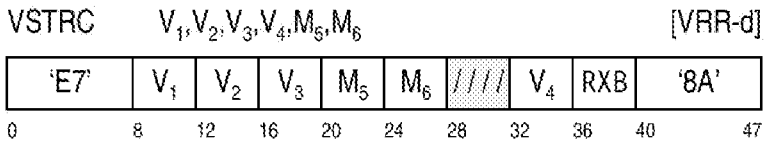
- Datos con DXC FE, registro de vector
- Operación si la instalación de extensión de vector no está instalada
- Especificación (valor ES reservado)
- Restricción de transacción

Mnemotécnicas extendidas:

VFENEB	V1,V_2,V3,M5	VFENE V_1,V_2,V_3,0,M_5
VFENEH	V_1,V_2,V_3,M_5	VFENE V_1,V_2,V_3,1,M_5

VFENEF	V_1,V_2,V_3,M_5	VFENE $V_1,V_2,V_3,2,M_5$
VFENEBS	V_1,V_2,V_3,M_5	VFENE $V_1,V_2,V_3,0,(M_5 \mid X'1')$
VFENEHS	V_1,V_2,V_3,M_5	VFENE $V_1,V_2,V_3,1,(M_5 \mid X'1')$
VFENEFS	V_1,V_2,V_3,M_5	VFENE $V_1,V_2,V_3,2,(M_5 \mid X'1')$
VFENEZB	V_1,V_2,V_3,M_5	VFENE $V_1,V_2,V_3,0,(M_5 \mid X'2')$
VFENEZH	V_1,V_2,V_3,M_5	VFENE $V_1,V_2,V_3,1,(M_5 \mid X'2')$
VFENEZF	V_1,V_2,V_3,M_5	VFENE V1,V2,V3,2,(M_5 | X'2')
VFENEZBS	V_1,V_2,V_3,M_5	VFENE $V_1,V_2,V_3,0,(M_5 \mid X'3')$
VFENEZHS	V_1,V_2,V_3,M_5	VFENE $V_1,V_2,V_3,1,(M_5 \mid X'3')$
VFENEZFS	V_1,V_2,V_3,M_5	VFENE $V_1,V_2,V_3,2,(M_5 \mid X'3')$

VECTOR COMPARAR RANGO DE CADENA



Continuando de izquierda a derecha, los elementos de número entero binario sin signo en el segundo operando se comparan a rangos de valores definidos por los pares de elementos par-impar en el tercero y cuarto operandos. Los combinados con los valores de control del cuarto operando definen el rango de comparaciones a realizarse. Si un elemento coincide con cualquiera de los rangos especificados por el tercero y cuarto operandos, se considera que es una coincidencia.

Si la bandera de tipo de resultado (RT) en el campo M_6 es cero, las posiciones de bits del elemento en el primer operando que corresponde al elemento que se está comparando en el segundo operando se establecen a uno si el elemento coincide con cualquiera de los rangos, de lo contrario se establecen a cero.

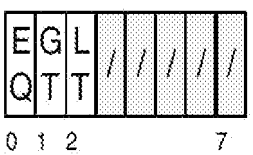
Si la bandera de tipo de resultado (RT) en el campo M6 se establece a uno, el índice de byte del primer elemento en el segundo operando que coincide como cualquiera de los rangos especificados por el tercer y cuarto operandos o una comparación de cero, si la bandera ZS se establece a uno, se coloca en el byte siete del primer operando y se almacenan ceros en los bytes restantes. Si no coinciden elementos, a continuación se coloca un índice igual al número de bytes en un vector en el byte siete del primer operando y se almacenan ceros en los bytes restantes.

La bandera de búsqueda de cero (ZS) en el campo M_6, si se establece a uno, añadirá una comparación a cero de los segundos elementos de operando a los rangos proporcionados por el tercer y cuarto operandos. Si una comparación de cero en un elemento indexado es más baja que cualquier otra comparación verdadera, a continuación, el código de condición se establece a cero.

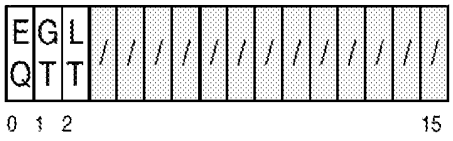
Los operandos contienen elementos del tamaño especificado por el control de tamaño de elemento en el campo M_5.

Los elementos del cuarto operando tienen el siguiente formato:

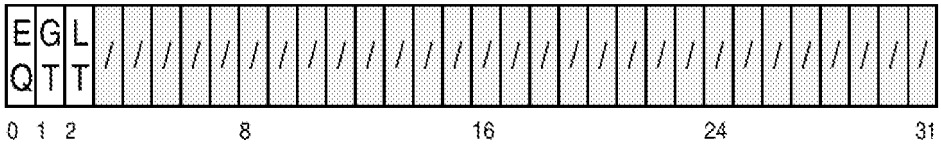
Si ES equivale a 0:



Si ES equivale a 1:



Si ES equivale a 2:



Los bits en los elementos del cuarto operando se definen como sigue:

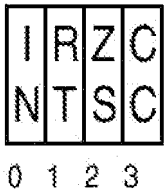
- *Igual (EQ):* cuando se realiza una comparación para igualdad.
- *Mayor que (GT):* cuando se realiza una comparación mayor que.
- *Menor que (LT):* cuando se realiza una comparación menor que.
- Todos los otros bits están reservados y deben ser cero para asegurar compatibilidad futura.

Los bits de control pueden usarse en cualquier combinación. Si no se establece ninguno de los bits, la comparación siempre producirá un resultado falso. Si se establecen todos los bits, la comparación siempre producirá un resultado verdadero.

El campo M_5 especifica el control de tamaño de elemento (ES). El control de ES especifica el tamaño de los elementos en los operandos de registro de vector. Si se especifica un valor reservado, se reconoce una excepción de especificación.

0 - Byte
1 - Semi-palabra
2 - Palabra
3-15 - Reservado

El campo M_6 tiene el siguiente formato:



Los bits del campo M_6 se definen como sigue:

- *Invertir resultado (IN):* si es cero, la comparación continúa con el par de valores en el vector de control. Si es uno, se invierte el resultado de los pares de las comparaciones en los rangos.
- *Tipo de resultado (RT):* si es cero, cada elemento resultante es una máscara de todas las comparaciones de rango en ese elemento. Si es uno, se almacena un índice en el byte siete del primer operando. Se almacenan cero en los bytes restantes.
- *Búsqueda de cero (ZS):* si es uno, cada elemento del segundo operando se compara también a cero.
- *Establecer código de condición (CC):* si es cero, el código de condición no se establece y permanece sin variar. Si es uno, el código de condición se establece como se especifica en la siguiente sección.

Condiciones especiales

Se reconoce una excepción de especificación y no se toma otra acción si tiene lugar cualquiera de lo siguiente:

1. El campo M_4 contiene un valor de 3-15.

Código de condición resultante:

0	Si ZS=1 y se halla un cero en un elemento indexado menor que cualquier comparación
1	Comparación hallada
2	—
3	Ninguna comparación hallada

Excepciones de programa:

- Datos con DXC FE, registro de vector
- Operación si la instalación de extensión de vector no está instalada
- Especificación (valor ES reservado)
- Restricción de transacción

Mnemotécnicas extendidas:

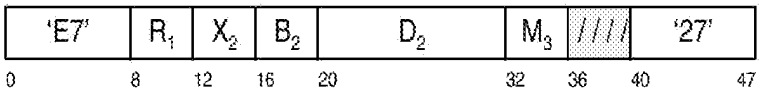
VSTRCB	V_1,V_2,V_3,V_4,M_6	VSTRC $V_1,V_2,V_3,V_4,0,M_6$
VSTRCH	V_1,V_2,V_3,V_4,M_6	VSTRC $V_1,V_2,V_3,V_4,1,M_6$
VSTRCF	V_1,V_2,V_3,V_4,M_6	VSTRC $V_1,V_2,V_3,V_4,2,M_6$
VSTRCBS	V_1,V_2,V_3,V_4,M_6	VSTRC $V_1,V_2,V_3,V_4,0,(M_6 \mid X'1')$
VSTRCHS	V_1,V_2,V_3,V_4,M_6	VSTRC $V_1,V_2,V_3,V_4,1,(M_6 \mid X'1')$
VSTRCFS	V_1,V_2,V_3,V_4,M_6	VSTRC $V_1,V_2,V_3,V_4,2,(M_6 \mid X'1')$
VSTRCZB	V_1,V_2,V_3,V_4,M_6	VSTRC $V_1,V_2,V_3,V_4,0,(M_6 \mid X'2')$
VSTRCZH	V_1,V_2,V_3,V_4,M_6	VSTRC $V_1,V_2,V_3,V_4,1,(M_6 \mid X'2')$
VSTRCZF	V_1,V_2,V_3,V_4,M_6	VSTRC $V_1,V_2,V_3,V_4,2,(M_6 \mid X'2')$
VSTRCZBS	V_1,V_2,V_3,V_4,M_6	VSTRC $V_1,V_2,V_3,V_4,0,(M_6 \mid X'3')$
VSTRCZHS	V_1,V_2,V_3,V_4,M_6	VSTRC $V_1,V_2,V_3,V_4,1,(M_6 \mid X'3')$
VSTRCZFS	V_1,V_2,V_3,V_4,M_6	VSTRC $V_1,V_2,V_3,V_4,2,(M_6 \mid X'3')$

	VR2 →	A	b	c	d	e	p	1	2
GE	A	T	T	T	T	T	T	F	F
LE	Z	T	F	T	F	F	T	F	F
GE	a	p	T	F	T	T	F	F	F
LE	c	T	T	T	F	F	T	T	T
LE	4	F	F	F	F	F	F	T	T
GE	0	T	T	T	T	T	T	T	T
EQ	d	F	F	F	т	F	F	F	F
EQ	d	F	F	F	T	F	F	F	F
VR4↑	VR3 ↑								
IN=0	VR1 (a)→	FFFF	FFFF	FFFF	FFFF	0000	FFFF	FFFF	FFFF
IN=1	VR1 (a)→	0000	0000	0000	0000	FFFF	0000	0000	0000
IN=0	VR1(b) →	0000	0000	0000	0000				
IN=1	VR1(b) →	0000	0000	0000	0008				
					índice				

Figura 23-1. ES=1,ZS=0 VR1(a) Resultados con RT=0 VR1(b) Resultados con RT=1

CONTADOR DE CARGA PARA LÍMITE DE BLOQUE

LCBB $R_1,D_2(X_2,B_2),M_3$ [RXE]



Se coloca un número entero binario sin singo de 32 bits que contiene el número de bytes posibles para cargar de la segunda ubicación de operando sin cruzar un límite de bloque especificado, limitado a dieciséis en el primer operando.

El desplazamiento se trata como un número entero sin signo de 12 bits.

La segunda dirección de operando no se usa para tratar datos.

El campo M_3 especifica un código que se usa para señalizar la CPU como el tamaño de límite de bloque para calcular el número de posibles bytes cargados. Si se especifica un valor reservado, se reconoce entonces una excepción de especificación.

Límite de código

0 64 Bytes
1 128 Bytes
2 256 Bytes
3 512 Bytes
4 1 KBytes
5 2 KBytes
6 4 KBytes
7-15 Reservados

Código de condición resultante:

0	Operando uno es dieciséis
1	--
2	--
3	Operando uno es menos de dieciséis

Código de condición resultante:

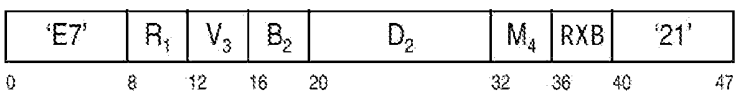
Excepciones de programa:

- Operación si la instalación de extensión de vector no está instalada
- Especificación

Nota de programación: se espera que se usará CONTADOR DE CARGA PARA LÍMITE DE BLOQUE en conjunto con CARGA DE VECTOR PARA LÍMITE DE BLOQUE para determinar el número de bytes que se cargaron.

CARGA DE VECTOR GR DE ELEMENTO VR

VLGV $R_1,V_3,D_2(B_2),M_4$ [VRS-c]



El elemento del tercer operando de tamaño especificado por el valor ES en el campo M4 e indexado por la segunda dirección de operando se coloca en la primera ubicación de operando. El tercer operando es un registro de vector. El primer operando es un registro general. Si el índice especificado por la segunda dirección de operando es mayor que el elemento con número más alto en el tercer operando, del tamaño de elemento especificado, los datos en el primer operando son impredecibles.

Si el elemento de registro de vector es menor que una palabra doble, el elemento está alineado a la derecha en el registro general de 64 bits y los ceros rellenan los bits restantes.

La segunda dirección de operando no se usa para tratar datos; en su lugar se usan los 12 bits más a la derecha de la dirección para especificar el índice de un elemento dentro del segundo operando.

El campo M_4 especifica el control de tamaño de elemento (ES). El control de ES especifica el tamaño de los elementos en los operandos de registro de vector. Si se especifica un valor reservado, se reconoce una excepción de especificación.

0 - Byte
1 - Semi-palabra
2 - Palabra
3 - Palabra doble
4-15 - Reservado sin variar.

Código de condición resultante: el código está sin variar.

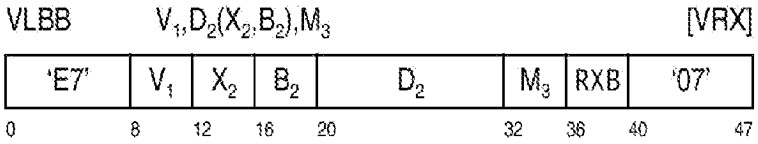
Excepciones de programa:

- Datos con DXC FE, registro de vector
- Operación si la instalación de extensión de vector no está instalada
- Especificación (valor ES reservado)
- Restricción de transacción

Mnemotécnicas extendidas:

VLGVB	R1,V3,D_2(B2)	VLGV R1,V3,D2(B2),0
VLGVH	R1,V3,D2(B2)	VLGV R1,V3,D2(B2),1
VLGVF	R1,V3,D2(B2)	VLGV R1,V3,D2(B2),2
VLGVG	R1,V3,D2(B2)	VLGV R1,V3,D2(B2),3

CARGA DE VECTOR PARA LÍMITE DE BLOQUE



El primer operando se carga empezando en el elemento de byte indexado cero con bytes del segundo operando. Si se encuentra una condición de límite, el resto del primer operando es impredecible. No se reconocen excepciones de acceso en bytes no cargados.

El desplazamiento para VLBB se trata como un número entero sin singo de 12 bits.

El campo M_3 especifica un código que se usa para señalizar la CPU en cuanto al tamaño de límite de bloque para cargar. Si se especifica un valor reservado, se reconoce una excepción de especificación.

Límite de código

0 64 Bytes
1 128 Bytes
2 256 Bytes
3 512 Bytes
4 1 KBytes
5 2 KBytes
6 4 KBytes
7-15 Reservados

Código de condición resultante: el código permanece sin variar.

Excepciones de programa:

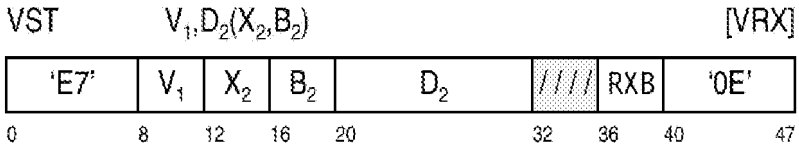
- Acceso (cargar, operando 2)
- Datos con DXC FE, registro de vector
- Operación si la instalación de extensión de vector no está instalada
- Especificación (Código de límite de bloque reservado)
- Restricción de transacción

Notas de programación:

1. En ciertas circunstancias, pueden cargarse datos pasado el límite de bloque. Sin embargo, esto tendrá lugar

únicamente si no hay excepciones de acceso en esos datos.

ALMACÉN DE VECTOR



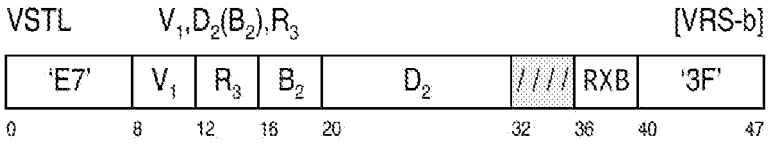
El valor de 128 bits en el primer operando se almacena en la ubicación de almacenamiento especificada por el segundo operando. El desplazamiento para VST se trata como un número entero sin singo de 12 bits.

Código de condición resultante: el código permanece sin variar.

Excepciones de programa:

- Acceso (almacenar, operando 2)
- Datos con DXC FE, registro de vector
- Operación si la instalación de extensión de vector no está instalada
- Restricción de transacción

ALMACÉN DE VECTOR CON LONGITUD



Continuando de izquierda a derecha, los bytes del primer operando se almacenan en la segunda ubicación de operando. El tercer operando de registro general especificado contiene un número entero sin signo de 32 bits que contiene un valor que representa el byte indexado más alto para almacenar. Si el tercer operando contiene un valor mayor o igual que el índice de byte más alto del vector, se almacenan todos los bytes del primer operando.

Las excepciones de acceso se reconocen únicamente en bytes almacenados.

El desplazamiento para ALMACÉN DE VECTOR con LONGITUD se trata como un número entero sin signo de 12 bits.

Código de condición resultante: el código de condición permanece sin variar.

Excepciones de programa:

- Acceso (almacenar, operando 2)
- Datos con DXC FE, registro de vector
- Operación si la instalación de extensión de vector no está instalada
- Restricción de transacción

Descripción RXB

Todas las instrucciones de vector tienen un campo en los bits 36-40 de la instrucción etiquetada como RXB. Este campo contiene los bits más significativos para todos los operandos de registro de vector designados. Los bits para las designaciones de registro no especificadas por la instrucción se reservan y deben establecerse a cero; de lo contrario, el programa puede no operar de manera compatible en el futuro. El bit más significativo se concatena a la izquierda de la designación de registro de cuatro bits para crear la designación de registro de vector de cinco bits.

Los bits se definen como sigue:

0. El bit más significativo para la designación de registro de vector en los bits 8-11 de la instrucción.
1. El bit más significativo para la designación de registro de vector en los bits 12-15 de la instrucción.
2. El bit más significativo para la designación de registro de vector en los bits 16-19 de la instrucción.
3. El bit más significativo para la designación de registro de vector en los bits 32-35 de la instrucción.

Control de activación de vector

Los registros de vector y las instrucciones pueden únicamente usarse si tanto el control de activación de vector (bit 46) como el control de registro de AFP (bit 45) en el registro de control cero se establecen a uno. Si se instala la instalación de vector y se ejecuta una instrucción de vector sin los bits de activación establecidos, se reconoce una excepción de datos con DXC FE hex. Si la instalación de vector no está instalada, se reconoce una excepción de operación.

REIVINDICACIONES

1. Un producto de programa informático para emular instrucciones en un entorno informático, comprendiendo el producto de programa informático:
un medio de almacenamiento legible por ordenador legible por un circuito de procesamiento y que almacena instrucciones para su ejecución por el circuito de procesamiento para realizar un método que comprende:

determinar a partir de una primera instrucción definida para una primera arquitectura informática que la primera instrucción incluye un especificador de operando de registro no contiguo que tiene una primera porción y una segunda porción, no contigua con la primera porción;
obtener (750) el especificador de operando de registro no contiguo a partir de la primera instrucción que comprende obtener la primera porción de un primer campo (V1) de la primera instrucción y la segunda porción de un segundo campo (RXB) de la primera instrucción, estando el primer campo separado del segundo campo por al menos un campo intermedio, en donde una porción del opcode de la primera instrucción especifica el primer campo y el segundo campo usados para designar el especificador no contiguo;
generar (752) un especificador de operando de registro contiguo usando la primera porción y la segunda porción obtenidas de la primera instrucción, usando la generación una o más reglas almacenadas en memoria o almacenamiento externo dependiendo del formato de la instrucción especificado por el opcode de la primera instrucción;
usar (754) el especificador de operando de registro contiguo en lugar del especificador de operando de registro no contiguo para indicar un recurso a usar en la ejecución de una segunda instrucción, estando la segunda instrucción definida para una segunda arquitectura informática diferente de la primera arquitectura informática y emular una función de la primera instrucción; y
ejecutar la segunda instrucción para emular la función de la primera instrucción, usando la ejecución el recurso indicado por el especificador de operando de registro contiguo sin tener más en cuenta el especificador de operando de registro no contiguo.

2. El producto de programa informático de la reivindicación 1, en donde la primera porción incluye un primero o más bits, y la segunda porción incluye un segundo o más bits, y la generación comprende concatenar el segundo o más bits con el primero o más bits para formar el especificador de operando de registro contiguo, en donde el segundo o más bits son los bits más significativos del especificador de operando de registro contiguo.

3. El producto de programa informático de la reivindicación 2, en donde el primer campo tiene una posición de operando asociada con el mismo, y el un segundo o más bits son un subconjunto de una pluralidad de bits del segundo campo, y en donde la obtención comprende seleccionar el un segundo o más bits de la pluralidad de bits del segundo campo basándose en la posición de operando del primer campo.

4. El producto de programa informático de la reivindicación 3, en donde la posición de operando del primer campo es como un primer operando, y en donde el un segundo o más bits se seleccionan de la ubicación más a la izquierda del segundo campo.

5. El producto de programa informático de cualquier reivindicación anterior, en donde el primer campo consiste en un campo de registro, el segundo campo consiste en un campo de extensión, la primera porción consiste en una pluralidad de bits del campo de registro, la segunda porción consiste en un bit del campo de extensión en una ubicación de la instrucción que corresponde al campo de registro, y la generación comprende concatenar el bit del campo de extensión con los bits del campo de registro para proporcionar el especificador de operando de registro contiguo.

6. El producto de programa informático de cualquier reivindicación anterior, en donde el uso del especificador de operando de registro contiguo para indicar un recurso incluye usar el especificador de operando de registro contiguo para mapear un registro a usar por la segunda instrucción.

7. El producto de programa informático de la reivindicación 6, en donde el registro mapeado por el especificador de operando de registro contiguo tiene el mismo valor que el especificador de operando de registro contiguo.

8. El producto de programa informático de la reivindicación 6, en donde el registro mapeado por el especificador de operando de registro contiguo tiene un valor diferente del especificador de operando de registro contiguo.

9. El producto de programa informático de cualquier reivindicación anterior, en donde la primera arquitectura informática incluye un conjunto de instrucciones que comprende primeras instrucciones que tienen campos de registro para acceder a una subsección de un espacio de registro de la primera arquitectura informática, y que tiene segundas instrucciones que tienen campos de registro no contiguos para acceder a la subsección y subsecciones restantes del espacio de registro, las primeras instrucciones tienen impedido el acceso a las subsecciones restantes.

10. El producto de programa informático de la reivindicación 1, en donde el primer campo consiste en un campo de registro, el segundo campo consiste en un campo de extensión, la primera porción consiste en una pluralidad de bits del campo de registro, la segunda porción consiste en un bit del campo de extensión en una ubicación de la instrucción

que corresponde al campo de registro, y la generación comprende concatenar el bit del campo de extensión con los bits del campo de registro para proporcionar el especificador de operando de registro contiguo, y que comprende adicionalmente:

obtener, mediante el procesador, a partir de la primera instrucción, otro especificador de operando de registro no contiguo, teniendo el otro especificador de operando de registro no contiguo otra primera porción y otra segunda porción, en donde la obtención comprende obtener la otra primera porción de otro primer campo de la instrucción y la otra segunda porción del otro bit del campo de extensión, estando el otro primer campo separado del primer campo y del campo de extensión;
generar otro especificador de operando de registro contiguo usando la otra primera porción y el otro bit, usando la generación una o más reglas basándose en el opcode de la primera instrucción; y
usar el otro especificador de operando de registro contiguo para indicar un recurso a usar en la ejecución de la segunda instrucción.

11. Un sistema informático para emular instrucciones en un entorno informático, comprendiendo el sistema informático:

una memoria; y
un procesador en comunicaciones con la memoria, en donde el sistema informático está configurado para realizar un método, comprendiendo dicho método:

determinar a partir de una primera instrucción definida para una primera arquitectura informática que la primera instrucción incluye un especificador de operando de registro no contiguo que tiene una primera porción y una segunda porción, no contigua con la primera porción; obtener el especificador de operando de registro no contiguo a partir de la primera instrucción que comprende obtener (750) la primera porción de un primer campo (V1) de la primera instrucción y la segunda porción de un segundo campo (RXB) de la primera instrucción, estando el primer campo separado del segundo campo por al menos un campo intermedio, en donde una porción del opcode de la primera instrucción especifica el primer campo y el segundo campo usados para designar el especificador no contiguo;
generar (752) un especificador de operando de registro contiguo usando la primera porción y la segunda porción obtenidas de la primera instrucción, usando la generación una o más reglas almacenadas en memoria o almacenamiento externo dependiendo del formato de la instrucción especificado por el opcode de la primera instrucción;
usar (754) el especificador de operando de registro contiguo en lugar del especificador de operando de registro no contiguo para indicar un recurso a usar en la ejecución de una segunda instrucción, estando la segunda instrucción definida para una segunda arquitectura informática diferente de la primera arquitectura informática y emular una función de la primera instrucción; y
ejecutar la segunda instrucción para emular la función de la primera instrucción, usando la ejecución el recurso indicado por el especificador de operando de registro contiguo sin tener más en cuenta el especificador de operando de registro no contiguo.

12. El sistema informático de la reivindicación 11, en donde la primera porción incluye un primero o más bits, y la segunda porción incluye un segundo o más bits, y la generación comprende concatenar el segundo o más bits con el primero o más bits para formar el especificador de operando de registro contiguo, en donde el segundo o más bits son los bits más significativos del especificador de operando de registro contiguo.

13. El sistema informático de la reivindicación 12, en donde el primer campo tiene una posición de operando asociada con el mismo, y el segundo o más bits son un subconjunto de una pluralidad de bits del segundo campo, y en donde la obtención comprende seleccionar el segundo o más bits de la pluralidad de bits del segundo campo basándose en la posición de operando del primer campo.

14. El sistema informático de la reivindicación 13, en donde la posición de operando del primer campo es como un primer operando, y en donde el segundo o más bits se seleccionan a partir de una ubicación más a la izquierda del segundo campo.

15. El sistema informático de cualquiera de las reivindicaciones 11 a 14, en donde el primer campo comprende un campo de registro, el segundo campo comprende un campo de extensión, la primera porción comprende una pluralidad de bits del campo de registro, la segunda porción comprende un bit del campo de extensión en una ubicación que corresponde al campo de registro, y la generación comprende concatenar el bit del campo de extensión con los bits del campo de registro para proporcionar el especificador de operando de registro contiguo.

16. El sistema informático de cualquiera de las reivindicaciones 11 a 15, en donde el uso del especificador de operando de registro contiguo para indicar un recurso incluye usar el especificador de operando de registro contiguo para mapear un registro a usar por la segunda instrucción.

17. El sistema informático de la reivindicación 16, en donde el registro mapeado por el especificador de operando de registro contiguo tiene uno de: el mismo valor que el especificador de operando de registro contiguo o tiene un valor

diferente del especificador de operando de registro contiguo.

18. Un método de emulación de instrucciones en un entorno informático, comprendiendo el método:

determinar a partir de una primera instrucción definida para una primera arquitectura informática que la primera instrucción incluye un especificador de operando de registro no contiguo que tiene una primera porción y una segunda porción, no contigua con la primera porción;
obtener (750) el especificador de operando de registro no contiguo a partir de la primera instrucción que comprende obtener la primera porción de un primer campo (V1) de la primera instrucción y la segunda porción de un segundo campo (RXB) de la primera instrucción, estando el primer campo separado del segundo campo por al menos un campo intermedio, en donde una porción del opcode de la primera instrucción especifica el primer campo y el segundo campo usados para designar el especificador no contiguo;
generar (752) un especificador de operando de registro contiguo usando la primera porción y la segunda porción obtenidas de la primera instrucción, usando la generación una o más reglas almacenadas en memoria o almacenamiento externo dependiendo del formato de la instrucción especificado por el opcode de la primera instrucción;
usar (754) el especificador de operando de registro contiguo en lugar del especificador de operando de registro no contiguo para indicar un recurso a usar en la ejecución de una segunda instrucción, estando la segunda instrucción definida para una segunda arquitectura informática diferente de la primera arquitectura informática y emular una función de la primera instrucción; y
ejecutar la segunda instrucción para emular la función de la primera instrucción, usando la ejecución el recurso indicado por el especificador de operando de registro contiguo sin tener más en cuenta el especificador de operando de registro no contiguo.

19. El método de la reivindicación 18, en donde la primera porción incluye un primero o más bits, y la segunda porción incluye un segundo o más bits, y la generación comprende concatenar el segundo o más bits con el primero o más bits para formar el especificador de operando de registro contiguo, en donde el segundo o más bits son los bits más significativos del especificador de operando de registro contiguo.

20. El método de la reivindicación 18 o la reivindicación 19, en donde el primer campo comprende un campo de registro, el segundo campo comprende un campo de extensión, la primera porción comprende una pluralidad de bits del campo de registro, la segunda porción comprende un bit del campo de extensión en una ubicación que corresponde al campo de registro, y la generación comprende concatenar el bit del campo de extensión con los bits del campo de registro para proporcionar el especificador de operando de registro contiguo.

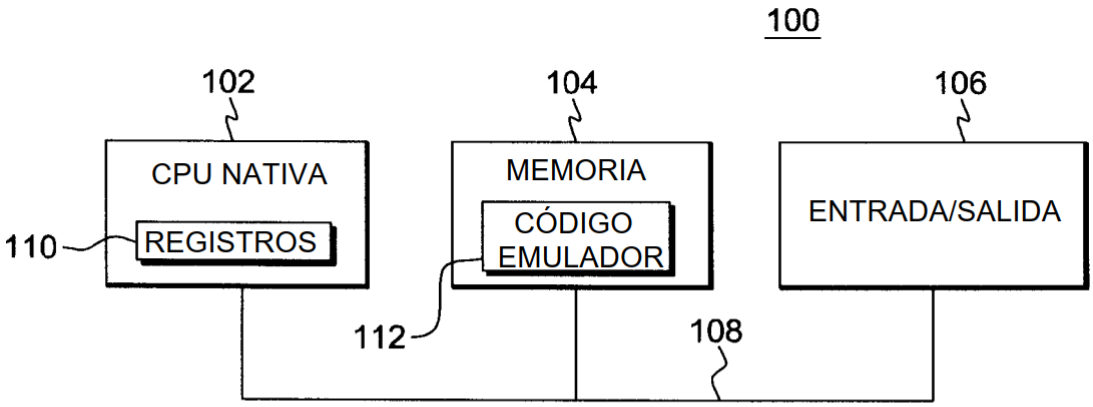

100
102
104
106
CPU NATIVA
REGISTROS
110
MEMORIA
CÓDIGO EMULADOR
112
ENTRADA/SALIDA
108

FIG. 1

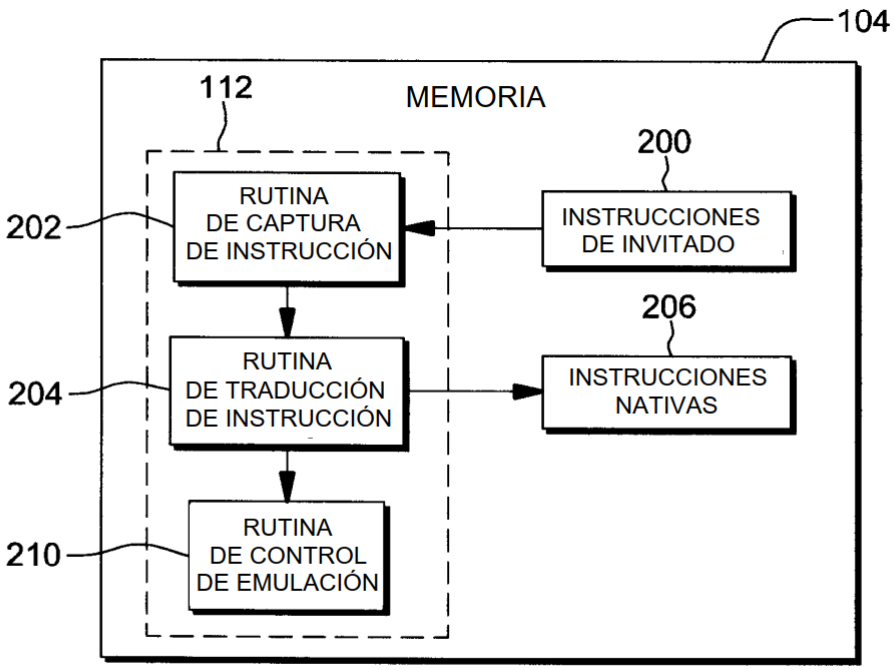

104
MEMORIA
112
200
RUTINA DE CAPTURA DE INSTRUCCIÓN
202
INSTRUCCIONES DE INVITADO
206
RUTINA DE TRADUCCIÓN DE INSTRUCCIÓN
204
INSTRUCCIONES NATIVAS
RUTINA DE CONTROL DE EMULACIÓN
210

FIG. 2

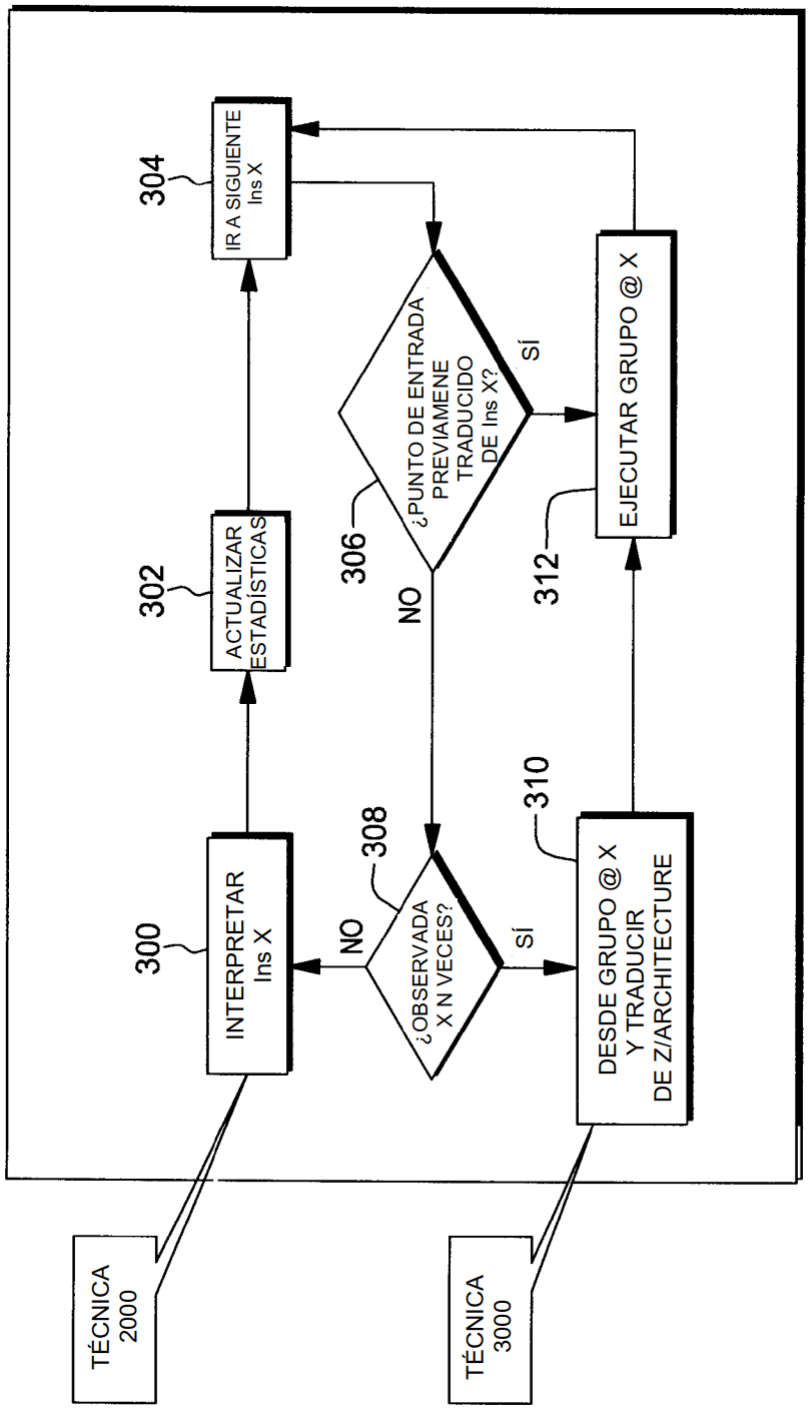

300
INTERPRETAR Ins X
302
ACTUALIZAR ESTADÍSTICAS
304
IR A SIGUIENTE Ins X
306
¿PUNTO DE ENTRADA PREVIAMENE TRADUCIDO DE Ins X?
SÍ
NO
308
¿OBSERVADA X N VECES?
NO
SÍ
310
DESDE GRUPO @ X Y TRADUCIR DE Z/ARCHITECTURE
312
EJECUTAR GRUPO @ X
TÉCNICA 2000
TÉCNICA 3000

FIG. 3

TÉCNICA 2000

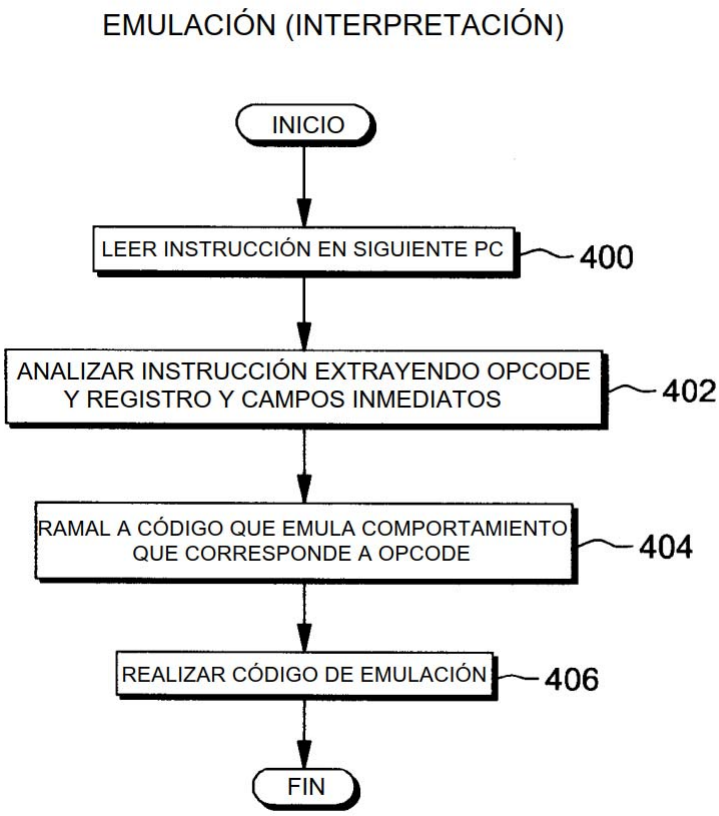


FIG. 4

TÉCNICA 3000

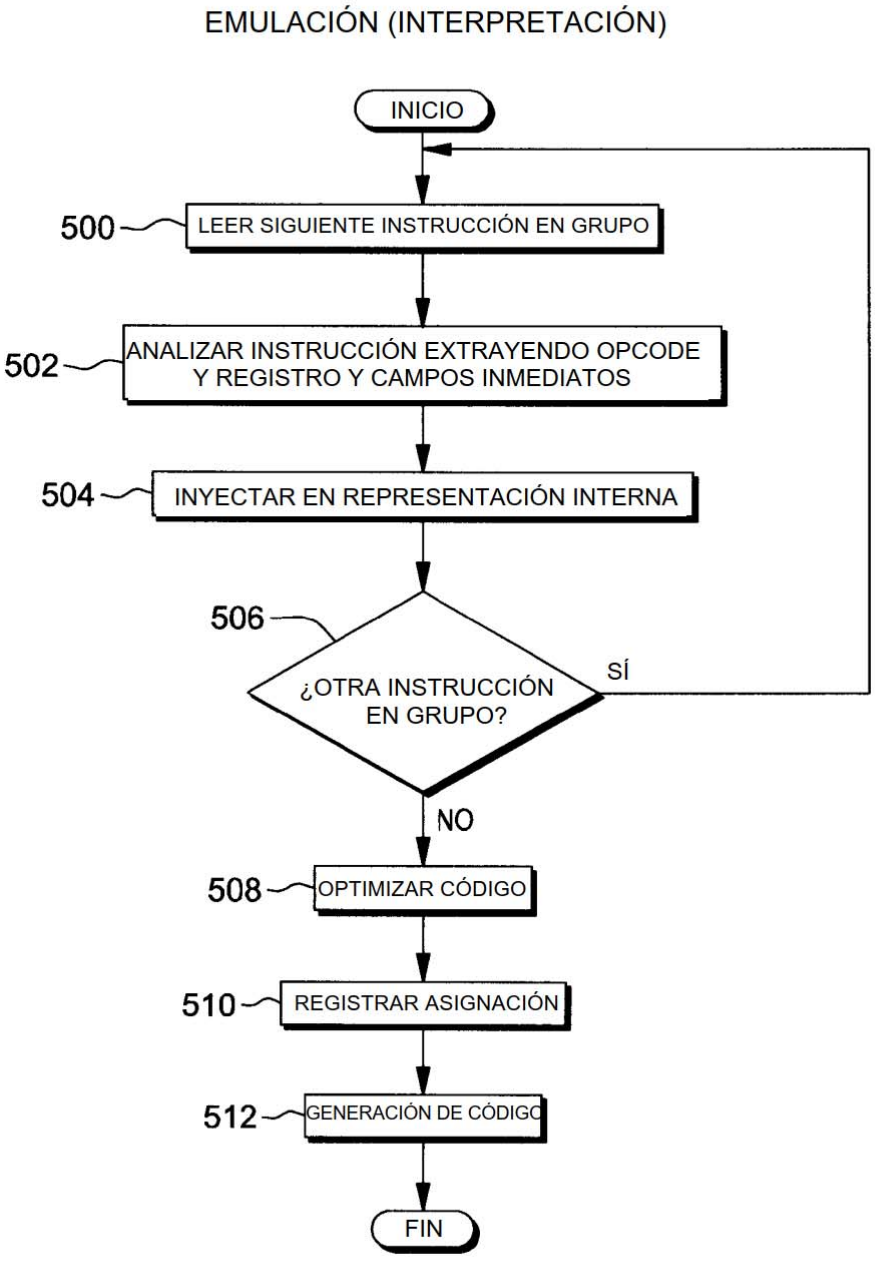


FIG. 5

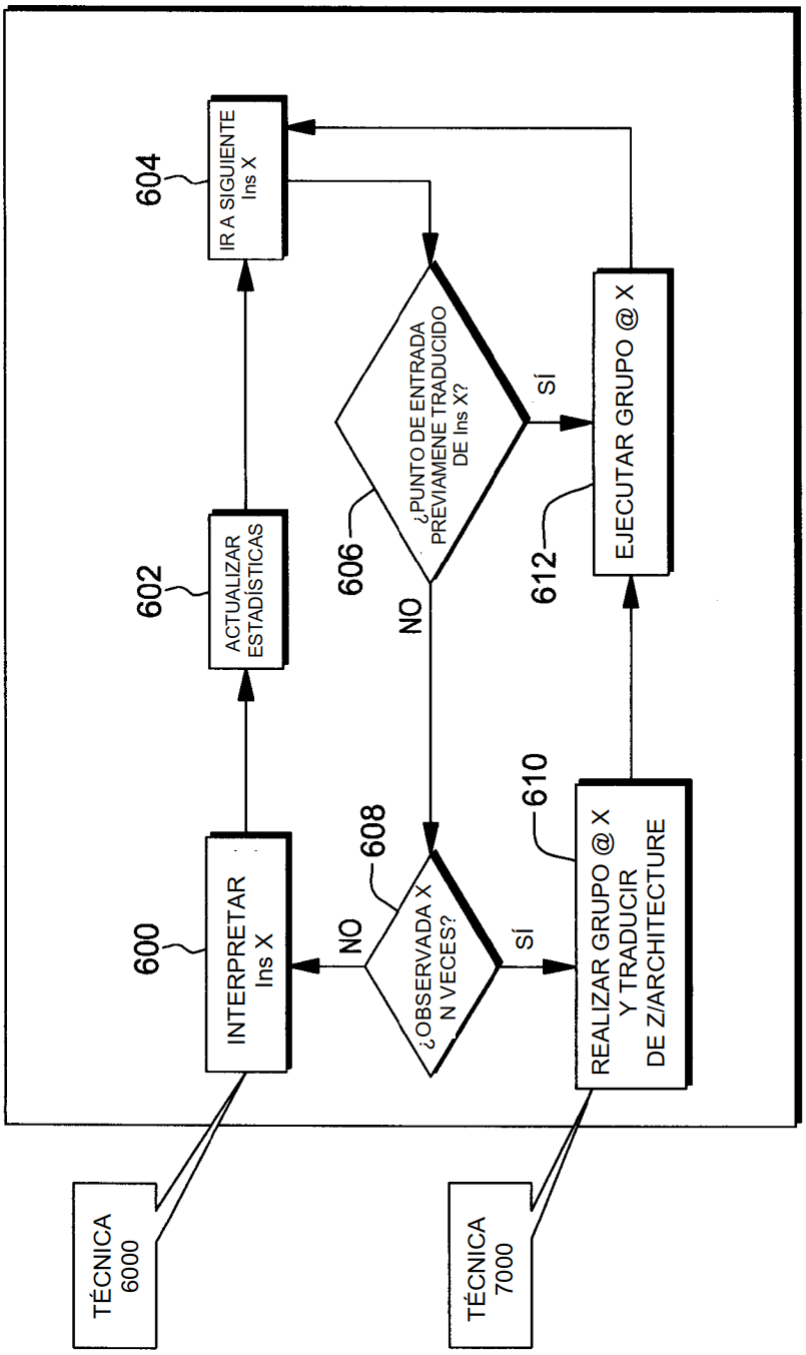

INTERPRETAR Ins X
600
ACTUALIZAR ESTADÍSTICAS
602
IR A SIGUIENTE Ins X
604
¿PUNTO DE ENTRADA PREVIAMENE TRADUCIDO DE Ins X?
606
SÍ
NO
EJECUTAR GRUPO @ X
612
¿OBSERVADA X N VECES?
608
NO
SÍ
REALIZAR GRUPO @ X Y TRADUCIR DE Z/ARCHITECTURE
610
TÉCNICA 6000
TÉCNICA 7000

FIG. 6

TÉCNICA 6000

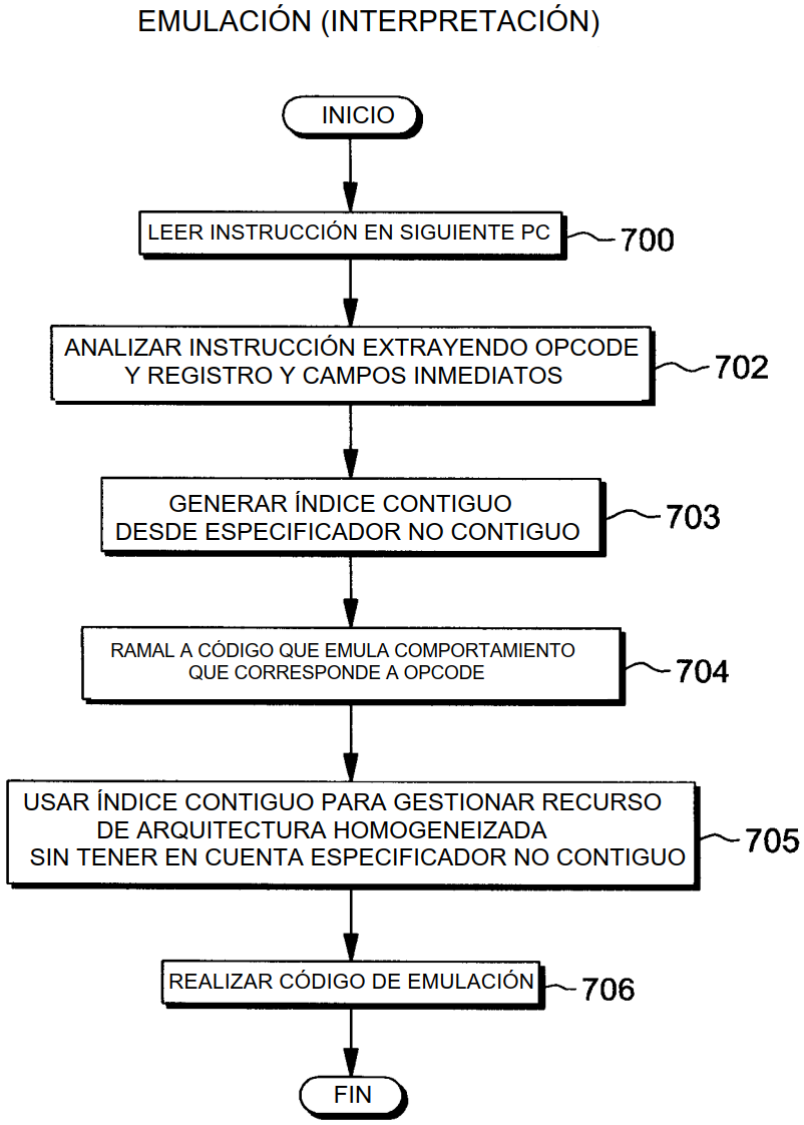


FIG. 7A

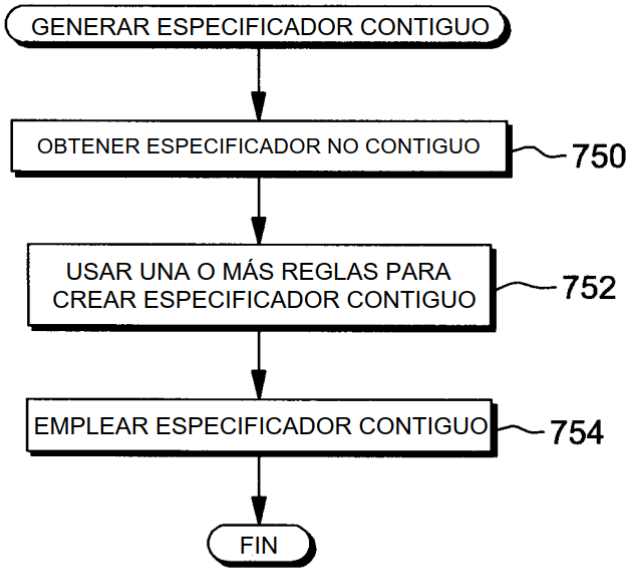

GENERAR ESPECIFICADOR CONTIGUO
OBTENER ESPECIFICADOR NO CONTIGUO
750
USAR UNA O MÁS REGLAS PARA CREAR ESPECIFICADOR CONTIGUO
752
EMPLEAR ESPECIFICADOR CONTIGUO
754
FIN

FIG. 7B

TÉCNICA 7000

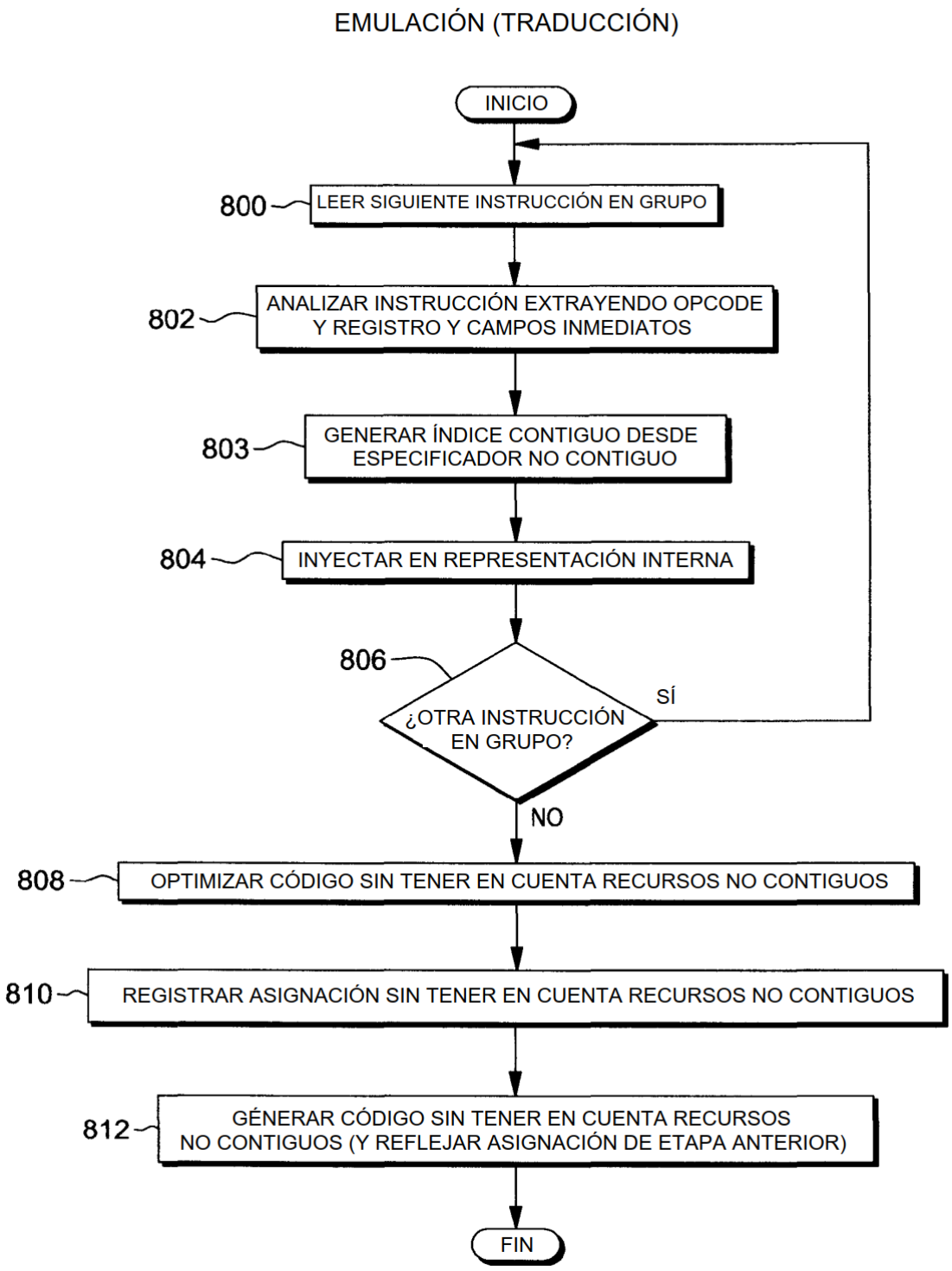


FIG. 8

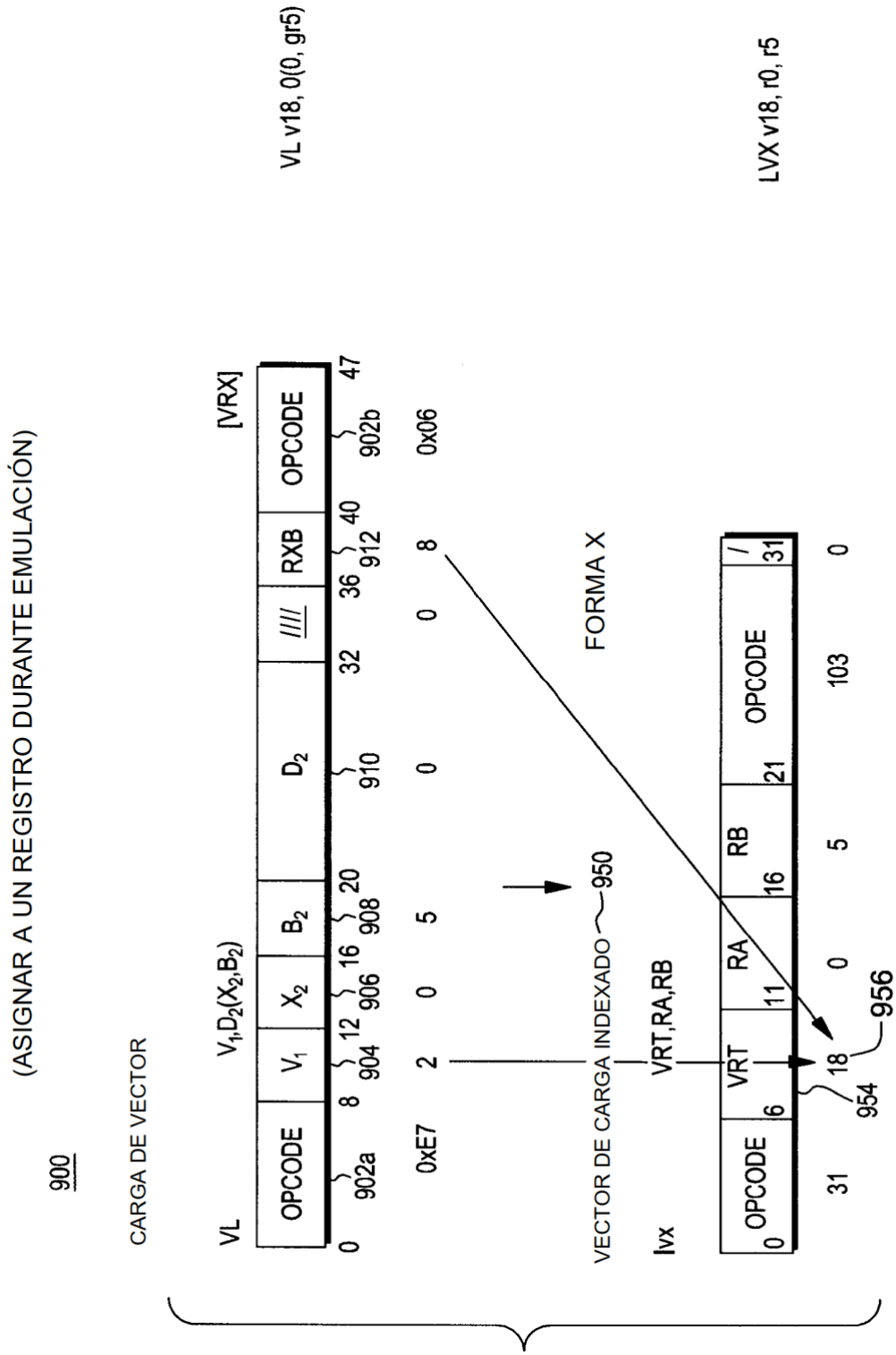

900
CARGA DE VECTOR
(ASIGNAR A UN REGISTRO DURANTE EMULACIÓN)
VL
V1,D2(X2,B2)
[VRX]
OPCODE
V1
X2
B2
D2
////
RXB
OPCODE
0
8
12
16
20
32
36
40
47
902a
904
906
908
910
912
902b
0xE7
0x06
VL v18, 0(0, gr5)
VECTOR DE CARGA INDEXADO
950
FORMA X
LVX v18, r0, r5
Ivx
VRT,RA,RB
OPCODE
VRT
RA
RB
OPCODE
/
0
6
11
16
21
31
31
18
0
5
103
0
954
956

FIG. 9A

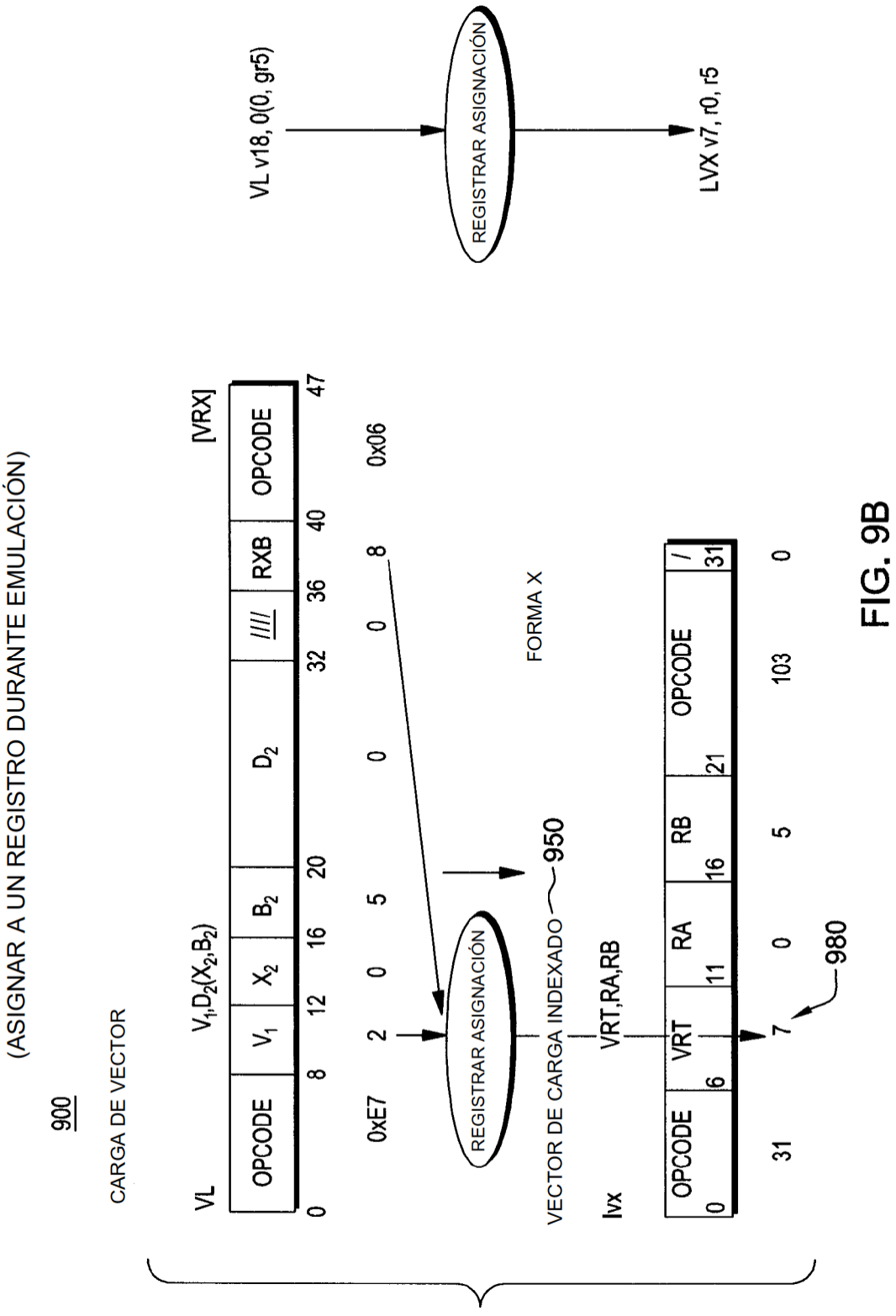

(ASIGNAR A UN REGISTRO DURANTE EMULACIÓN)
900
CARGA DE VECTOR
VL
$V_1,D_2(X_2,B_2)$
[VRX]
OPCODE
V_1
X_2
B_2
D_2
////
RXB
OPCODE
0
8
12
16
20
32
36
40
47
0xE7
2
0
5
0
0
8
0x06
REGISTRAR ASIGNACIÓN
950
VECTOR DE CARGA INDEXADO
FORMA X
lvx
VRT,RA,RB
OPCODE
VRT
RA
RB
OPCODE
/
0
6
11
16
21
31
31
7
0
5
103
0
980
VL v18, 0(0, gr5)
REGISTRAR ASIGNACIÓN
LVX v7, r0, r5

FIG. 9B

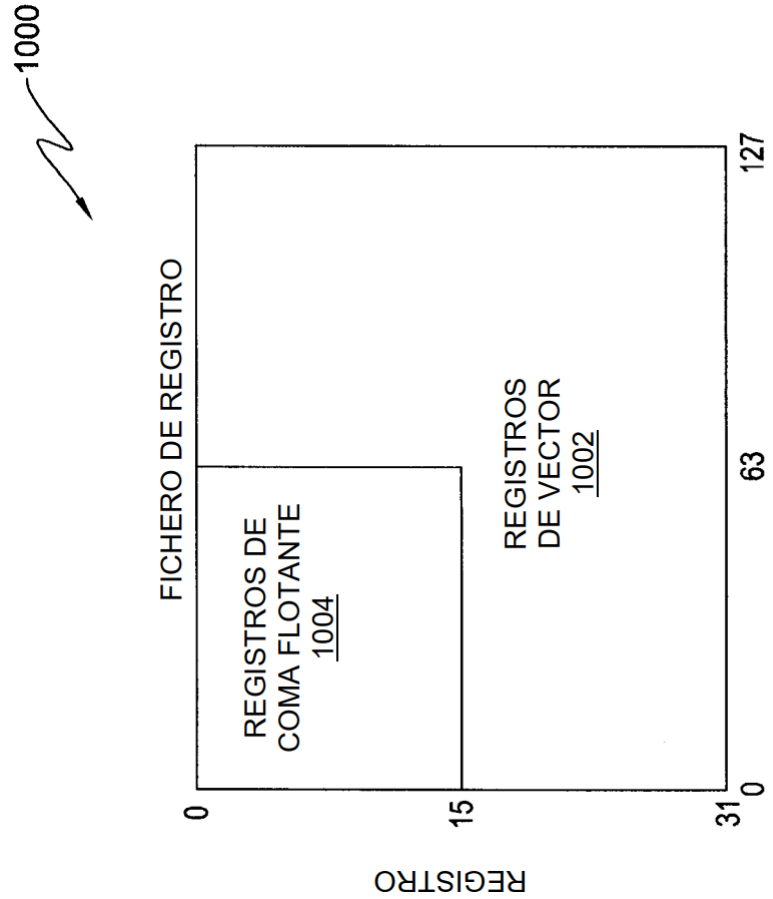

1000
FICHERO DE REGISTRO
REGISTROS DE COMA FLOTANTE 1004
REGISTROS DE VECTOR 1002
0
15
31
0
63
127
BITS
REGISTRO

FIG. 10

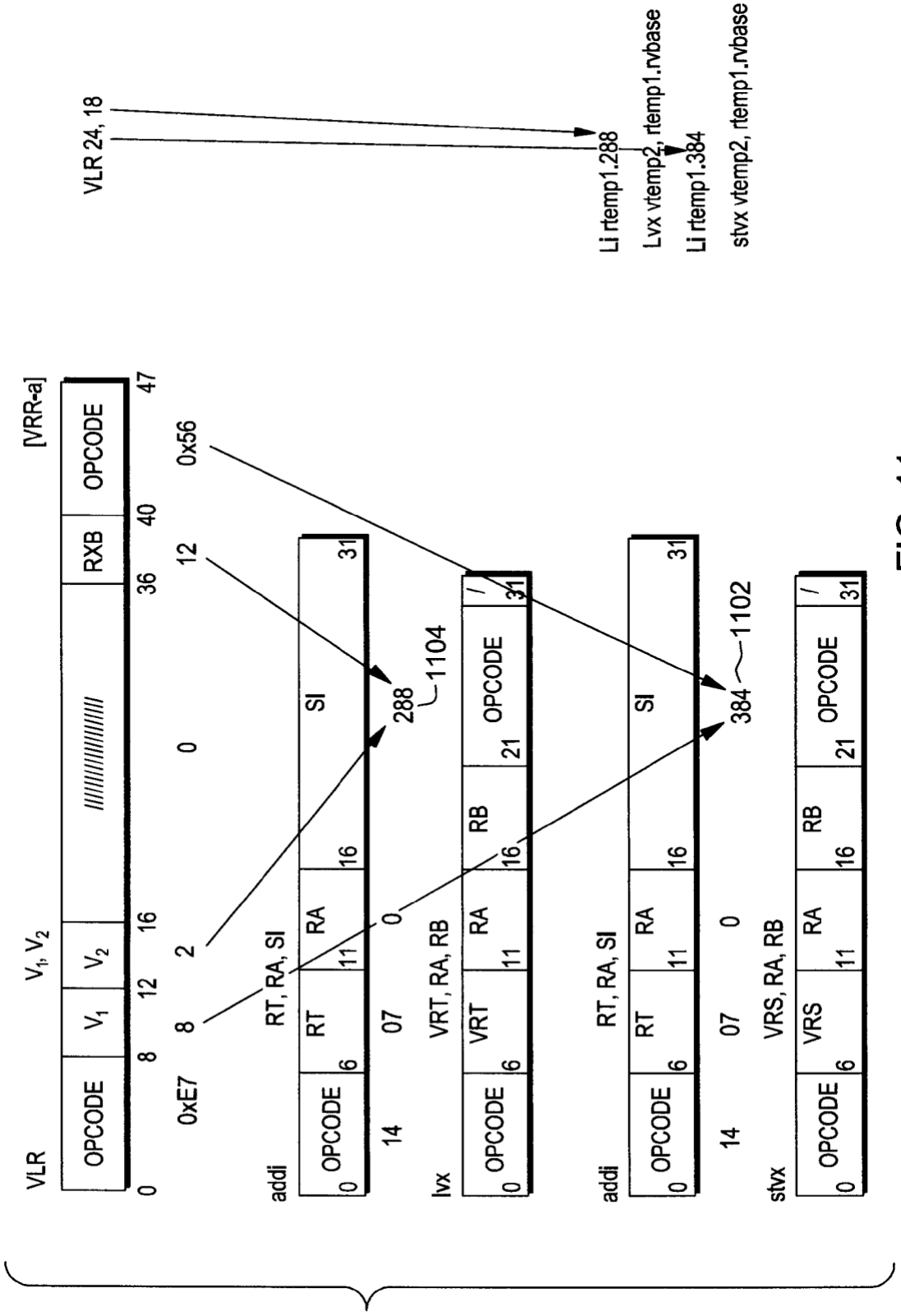

1100
(ASIGNAR A MEMORIA DURANTE EMULACIÓN)
VLR
[VRR-a]
OPCODE
V1
V2
RXB
OPCODE
V1, V2
0xE7
0x56
addi
RT, RA, SI
lvx
VRT, RA, RB
stvx
VRS, RA, RB
288
1104
384
1102
VLR 24, 18
Li rtemp1,288
Lvx vtemp2, rtemp1,rvbase
Li rtemp1,384
stvx vtemp2, rtemp1,rvbase

FIG. 11


PRODUCTO
DE PROGRAMA
INFORMÁTICO
1200
1204
LÓGICA
DE CÓDIGO
DE PROGRAMA
MEDIO DE
ALMACENAMIENTO
LEGIBLE POR
ORDENADOR
1202

FIG. 12

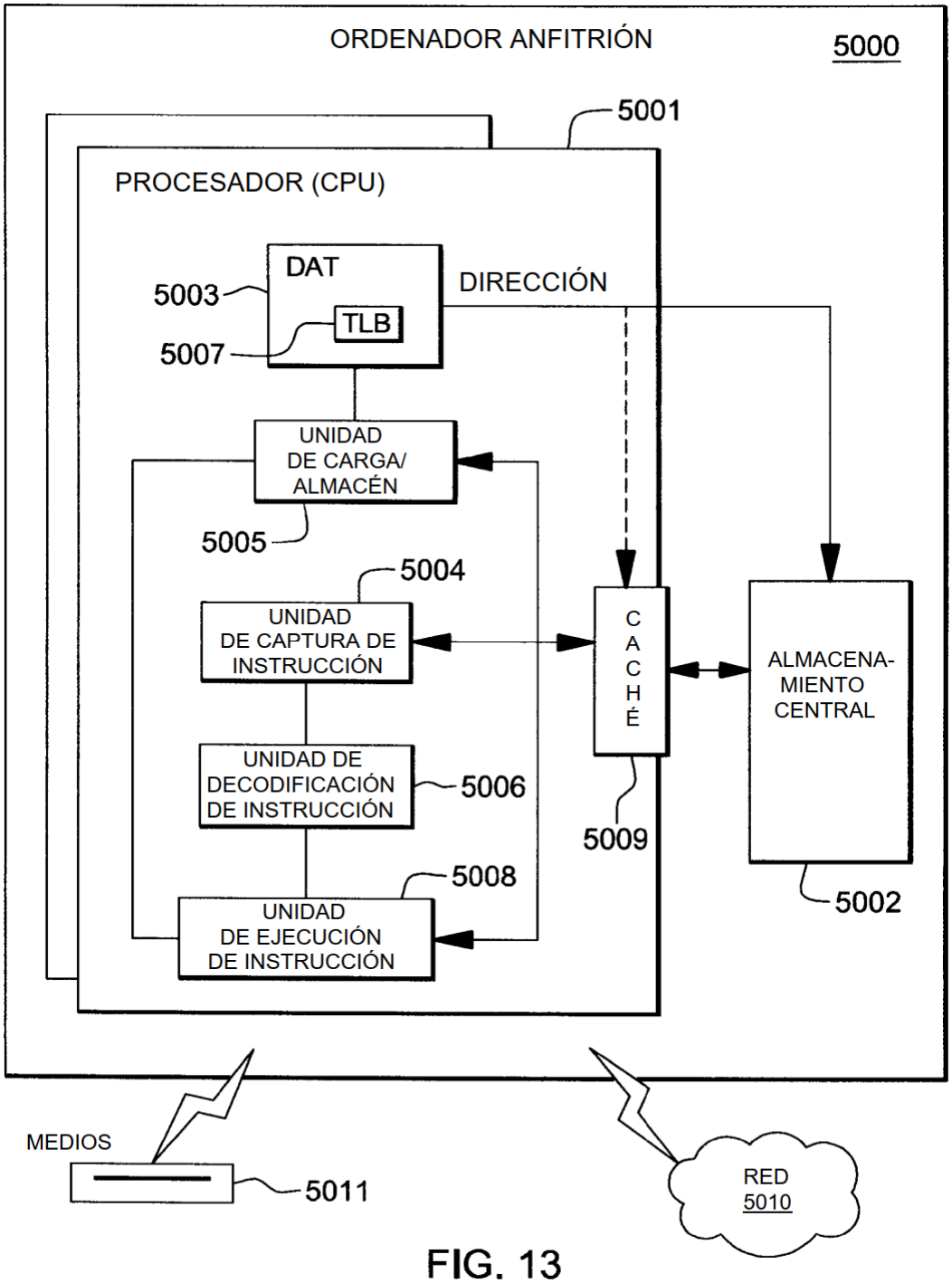

ORDENADOR ANFITRIÓN
5000
5001
PROCESADOR (CPU)
DAT
5003
TLB
5007
DIRECCIÓN
UNIDAD DE CARGA/ ALMACÉN
5005
5004
UNIDAD DE CAPTURA DE INSTRUCCIÓN
UNIDAD DE DECODIFICACIÓN DE INSTRUCCIÓN
5006
5008
UNIDAD DE EJECUCIÓN DE INSTRUCCIÓN
CACHÉ
5009
ALMACENA-MIENTO CENTRAL
5002
MEDIOS
5011
RED
5010

FIG. 13

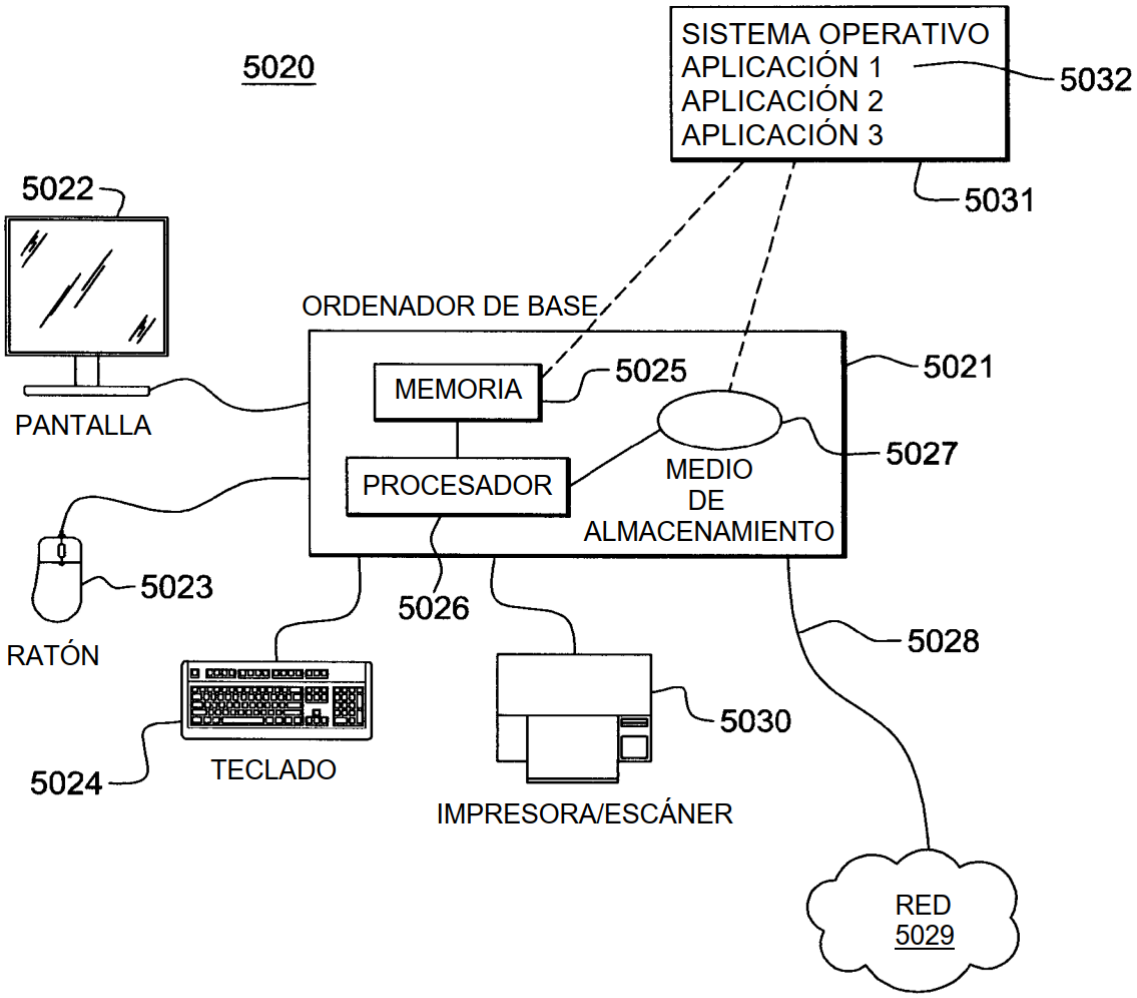

5020
SISTEMA OPERATIVO
APLICACIÓN 1
APLICACIÓN 2
APLICACIÓN 3
5032
5031
5022
ORDENADOR DE BASE
MEMORIA
5025
5021
PANTALLA
PROCESADOR
MEDIO
DE
ALMACENAMIENTO
5027
5023
RATÓN
5026
5028
5030
5024
TECLADO
IMPRESORA/ESCÁNER
RED
5029

FIG. 14

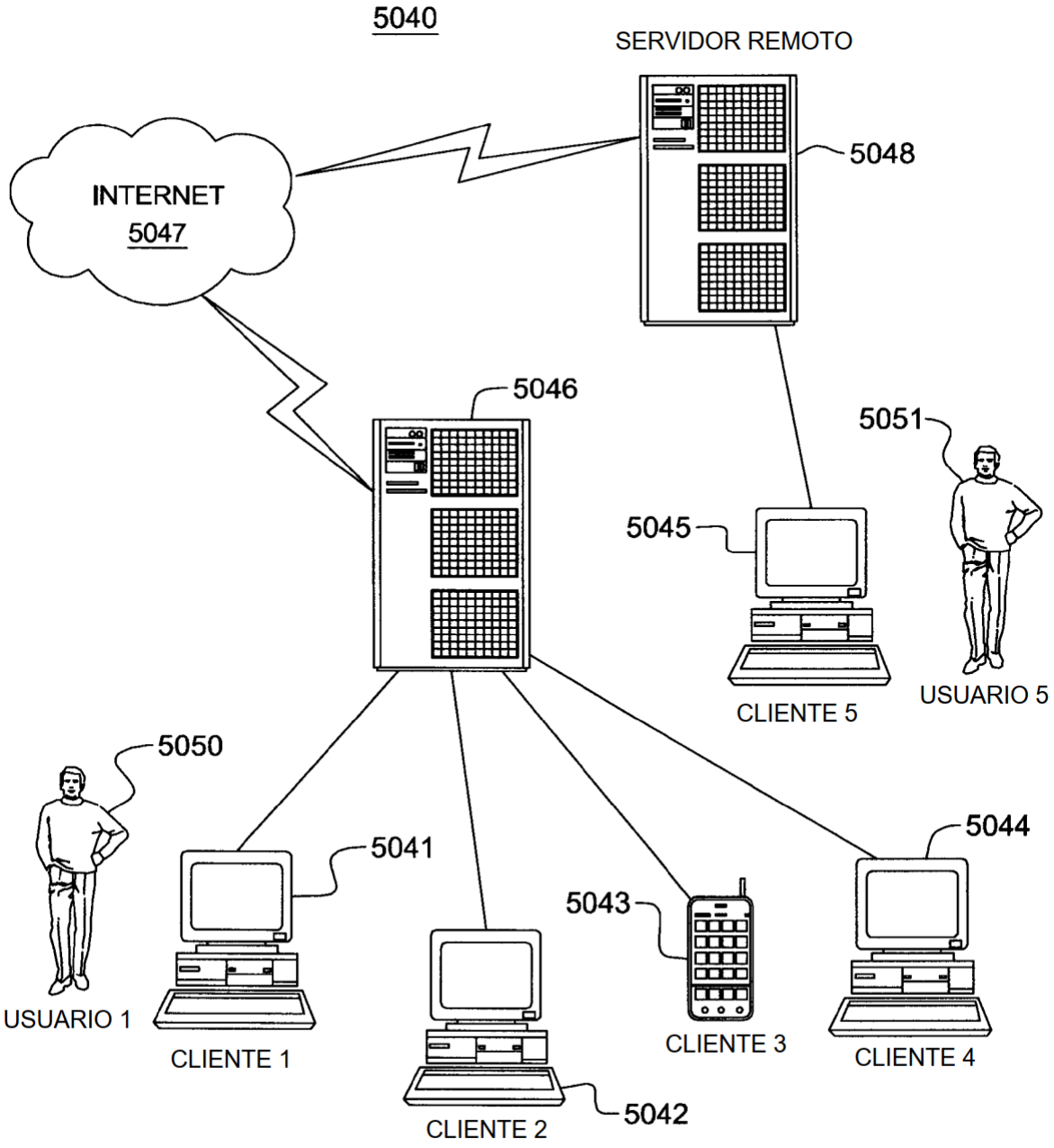

5040
SERVIDOR REMOTO
5048
INTERNET
5047
5046
5051
5045
USUARIO 5
CLIENTE 5
5050
5041
5044
5043
USUARIO 1
CLIENTE 1
CLIENTE 3
CLIENTE 4
CLIENTE 2
5042

FIG. 15

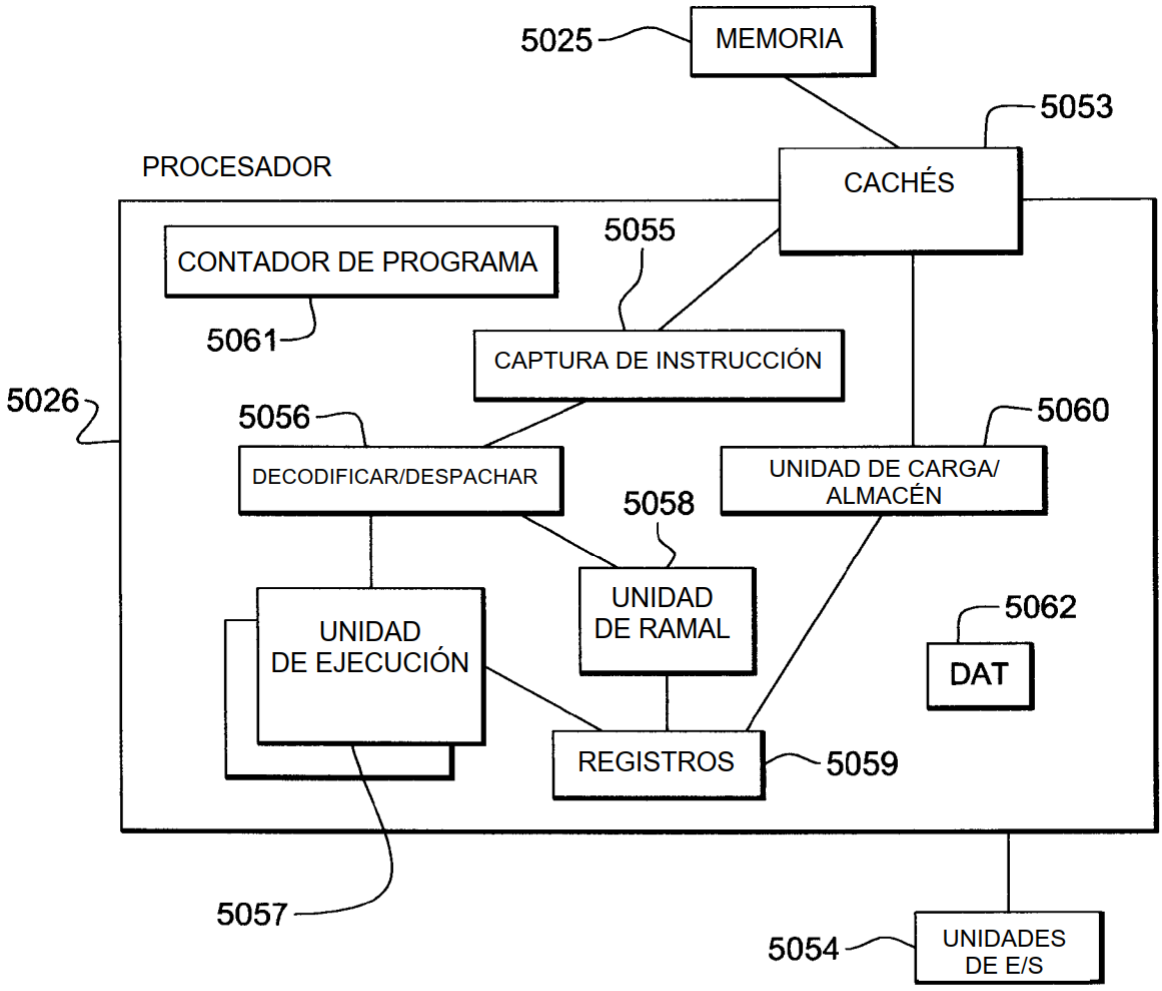

5025
MEMORIA
5053
PROCESADOR
CACHÉS
CONTADOR DE PROGRAMA
5055
5061
CAPTURA DE INSTRUCCIÓN
5026
5056
5060
DECODIFICAR/DESPACHAR
UNIDAD DE CARGA/
ALMACÉN
5058
UNIDAD
DE RAMAL
5062
UNIDAD
DE EJECUCIÓN
DAT
REGISTROS
5059
5057
5054
UNIDADES
DE E/S

FIG. 16

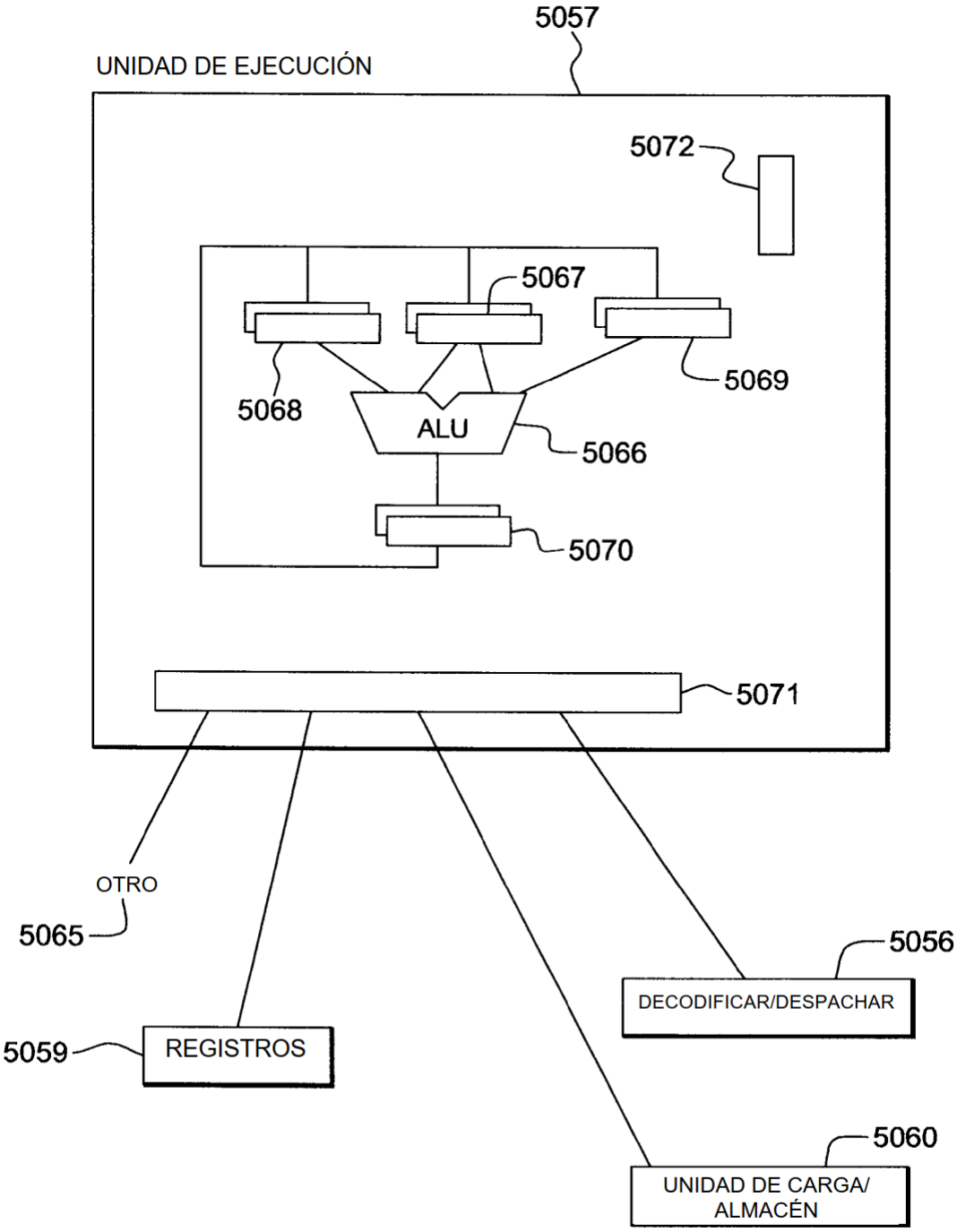

5057
UNIDAD DE EJECUCIÓN
5072
5067
5068
5069
ALU
5066
5070
5071
OTRO
5065
5056
DECODIFICAR/DESPACHAR
5059
REGISTROS
5060
UNIDAD DE CARGA/
ALMACÉN

FIG. 17A

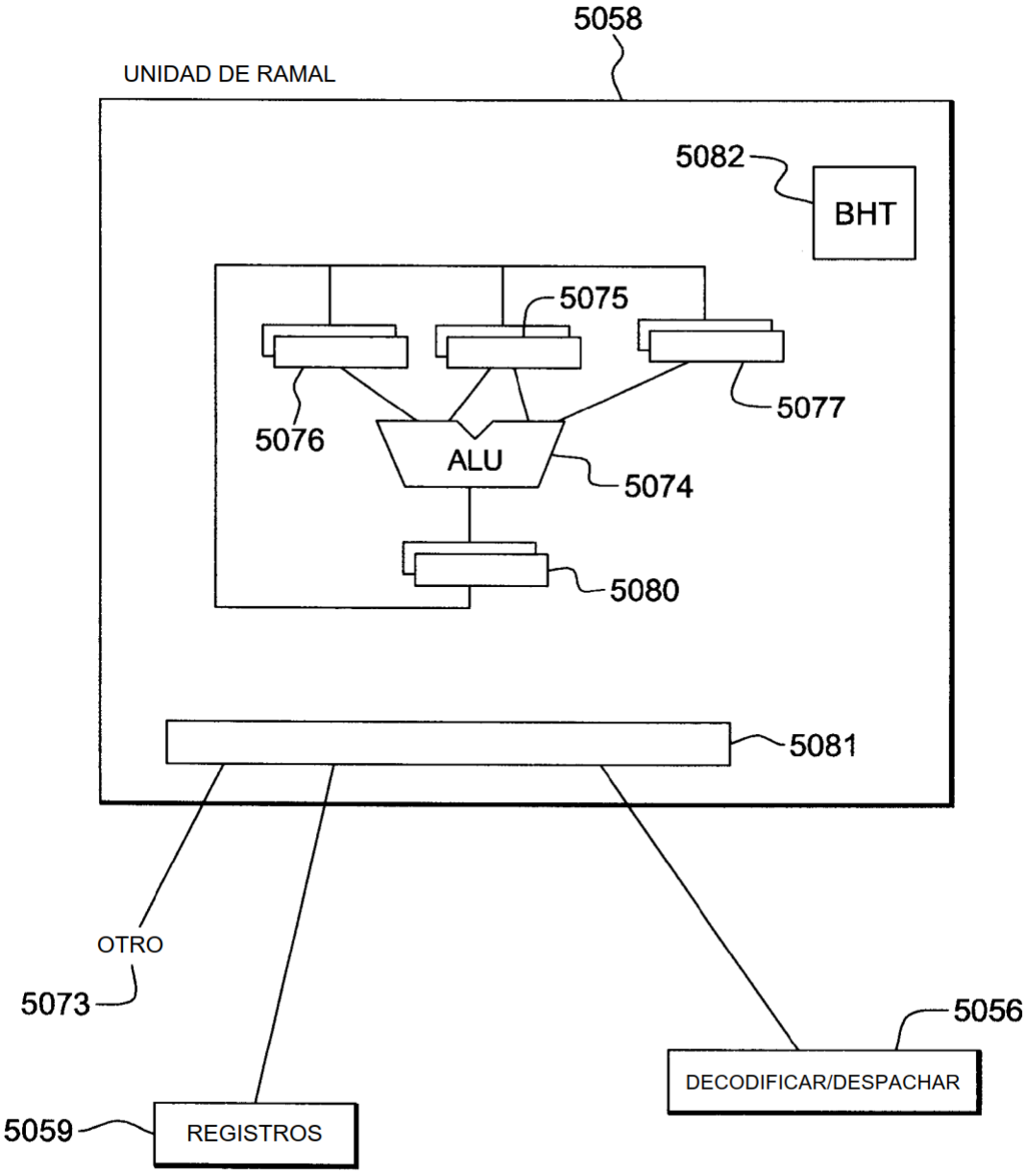

5058
UNIDAD DE RAMAL
5082
BHT
5075
5076
5077
ALU
5074
5080
5081
OTRO
5073
5056
DECODIFICAR/DESPACHAR
5059
REGISTROS

FIG. 17B

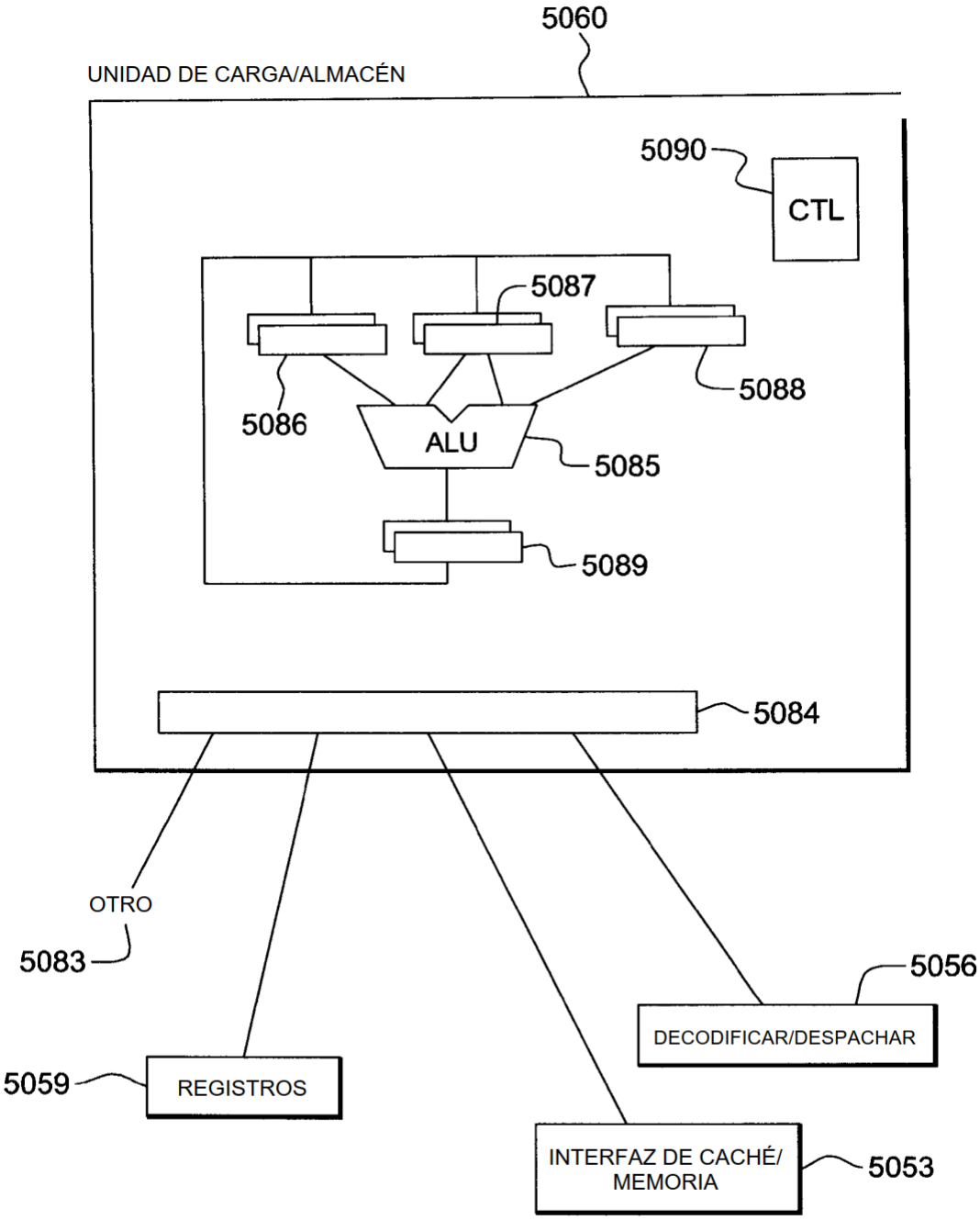

5060
UNIDAD DE CARGA/ALMACÉN
5090
CTL
5087
5088
5086
ALU
5085
5089
5084
OTRO
5083
5056
DECODIFICAR/DESPACHAR
5059
REGISTROS
INTERFAZ DE CACHÉ/
MEMORIA
5053

FIG. 17C

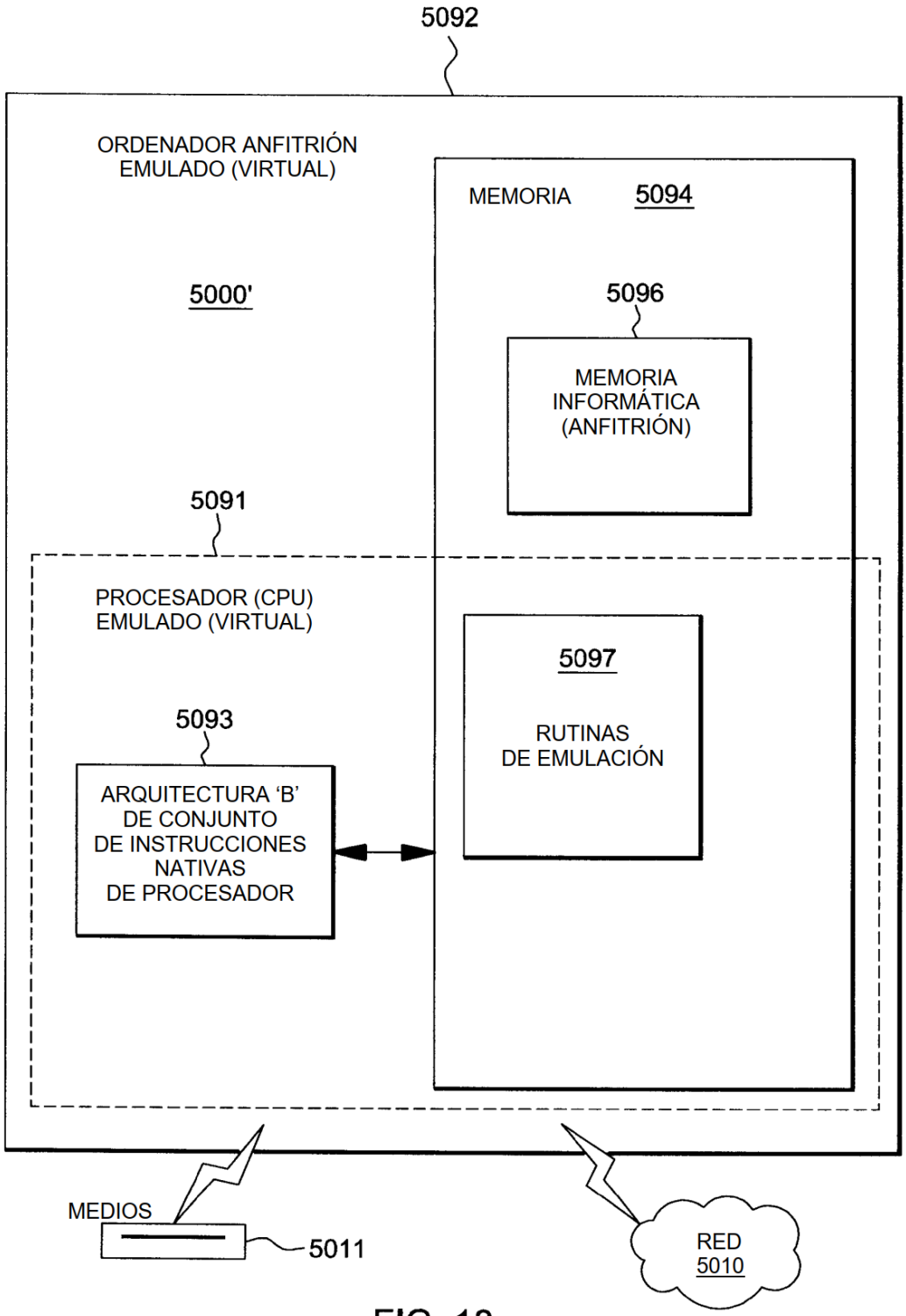

5092
ORDENADOR ANFITRIÓN
EMULADO (VIRTUAL)
MEMORIA
5094
5000'
5096
MEMORIA
INFORMÁTICA
(ANFITRIÓN)
5091
PROCESADOR (CPU)
EMULADO (VIRTUAL)
5097
RUTINAS
DE EMULACIÓN
5093
ARQUITECTURA 'B'
DE CONJUNTO
DE INSTRUCCIONES
NATIVAS
DE PROCESADOR
MEDIOS
5011
RED
5010

FIG. 18