(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 680 147**

(51) Int. CI.:

G06F 9/30 (2008.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **15.11.2012 PCT/IB2012/056434**

(87) Fecha y número de publicación internacional: **19.09.2013 WO13136143**

(96) Fecha de presentación y número de la solicitud europea: **15.11.2012 E 12871072 (0)**

(97) Fecha y número de publicación de la concesión europea: **13.06.2018 EP 2769305**

(54) Título: **Instrucción para cargar datos hasta una frontera de memoria especificada indicada por la instrucción**

(30) Prioridad:

15.03.2012 US 201213421456

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
04.09.2018

(73) Titular/es:

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

(72) Inventor/es:

BRADBURY, JONATHAN DAVID;
GSCHWIND, MICHAEL KARL;
SLEGEL, TIMOTHY;
SCHWARZ, ERIC MARK y
JACOBI, CHRISTIAN

(74) Agente/Representante:

ELZABURU, S.L.P

DESCRIPCIÓN

Instrucción para cargar datos hasta una frontera de memoria especificada indicada por la instrucción

Antecedentes

La invención se refiere, en general, al procesamiento de datos y, en particular, a la carga de datos en registros.

El procesamiento de datos incluye diversos tipos de procesamiento, incluyendo la carga de datos en registros. La carga de datos en un registro incluye, pero no se limita a, la carga de datos de caracteres, tales como cadenas de datos de caracteres, datos de número entero o cualquier otro tipo de datos. Entonces se pueden usar y / o manipular los datos que se cargan.

Las instrucciones actuales para realizar diversos tipos de procesamiento, incluyendo la carga de los datos en registros, tienden a ser ineficientes.

El documento US 6.513.107 describe una unidad de transferencia de vectores para manejar transferencias de datos de vector entre una memoria y un procesador de datos en un sistema informático. Las instrucciones de transferencia de datos de vector se publican en una cola de instrucciones en la unidad de transferencia de vectores. Las instrucciones de programa para realizar una transferencia en ráfaga incluyen determinar la dirección de partida de los datos de vector que se van a transferir, la dirección de fin de los datos de vector que se van a transferir, y si la dirección de fin de los datos de vector que se van a transferir se encuentra dentro de la misma página de memoria virtual que la dirección de partida. La dirección de fin de los datos de vector que se van a transferir se determina sobre la base del número de elementos de datos que se van a transferir, el intervalo de los datos de vector que se van a transferir y la anchura de los elementos de datos de vector que se van a transferir. Cuando la cantidad de datos que se van a transferir es divisible por un factor de dos, la multiplicación del intervalo y la anchura de los elementos de datos se lleva a cabo mediante desplazamiento. Una excepción de error de dirección tiene lugar cuando la dirección de fin de los datos de vector que se van a transferir no se encuentra dentro de la misma página de memoria virtual que la dirección de partida. La dirección de fin de los datos de vector que se van a transferir se determina en paralelo con la determinación de la dirección de partida de los datos de vector que se van a transferir.

El documento US 2005/0071583 describe una arquitectura que está estructura para manejar referencias de memoria no alineadas. Se divulga un método para cargar datos no alineados que están almacenados en varios sitios de memoria, incluyendo una etapa de carga de una primera parte de los datos no alineados en un primer sitio de almacenamiento y de rotación de la primera parte de una primera posición a una segunda posición en el primer sitio de memoria. A continuación, una segunda parte de los datos no alineados se carga en un segundo sitio de almacenamiento y se rota de una posición a otra posición. Entonces, el primer sitio de almacenamiento se combina con el segundo sitio de almacenamiento usando una operación lógica para dar un sitio de almacenamiento resultante.

Compendio

Se abordan los inconvenientes de la técnica anterior y se proporcionan ventajas por medio de la provisión de un producto de programa informático, sistema y método, tal como se define mediante las reivindicaciones adjuntas.

En la presente memoria también se describen y se reivindican métodos y sistemas en relación con uno o más aspectos de la presente invención. Además, en la presente memoria también se describen servicios en relación con uno o más aspectos de la presente invención.

Se logran elementos y ventajas adicionales por medio de las técnicas de la presente invención. Otras formas de realización y aspectos de la invención se describen en detalle en la presente memoria.

Breve descripción de los dibujos

Algunas formas de realización de la presente invención se describirán a continuación solo a modo de ejemplo con referencia a los dibujos adjuntos, en los que:

la figura 1 ilustra un ejemplo de un entorno informático para incorporar y usar una o más formas de realización de la presente invención;

la figura 2A ilustra otro ejemplo de un entorno informático para incorporar y usar una o más formas de realización de la presente invención;

la figura 2B ilustra detalles adicionales de la memoria de la figura 2A, de acuerdo con una forma de realización de la presente invención;

la figura 3 ilustra un formato de una instrucción de vector de carga a frontera de bloque de acuerdo con una forma de realización de la presente invención;

la figura 4 ilustra la lógica asociada con la instrucción de vector de carga a frontera de bloque, de acuerdo con una forma de realización de la presente invención;

la figura 5 ilustra datos que se van a cargar en un registro de vector, de acuerdo con una forma de realización de la presente invención;

la figura 6 ilustra un archivo de registro de acuerdo con una forma de realización de la presente invención;

la figura 7 ilustra una forma de realización de un producto de programa informático que incorpora una o más formas de realización de la presente invención;

la figura 8 ilustra una forma de realización de un sistema informático de anfitrión para incorporar y usar una o más formas de realización de la presente invención;

la figura 9 ilustra un ejemplo adicional de un sistema informático para incorporar y usar una o más formas de realización de la presente invención;

la figura 10 ilustra otro ejemplo de un sistema informático que comprende una red de ordenadores para incorporar y usar una o más formas de realización de la presente invención;

la figura 11 ilustra diversos elementos de un sistema informático para incorporar y usar una o más formas de realización de la presente invención;

la figura 12A ilustra la unidad de ejecución del sistema informático de la figura 11 para incorporar y usar una o más formas de realización de la presente invención;

la figura 12B ilustra la unidad de ramificación del sistema informático de la figura 11 para incorporar y usar una o más formas de realización de la presente invención;

la figura 12C ilustra la unidad de carga / almacenamiento del sistema informático de la figura 11 para incorporar y usar uno o más aspectos de la presente invención y

la figura 13 ilustra un sistema informático de anfitrión emulado para incorporar y usar una o más formas de realización de la presente invención.

Descripción detallada

De acuerdo con un aspecto de la presente invención, se proporciona una capacidad para facilitar la carga de datos en un registro. Como ejemplos, los datos incluyen datos de caracteres, datos de números enteros y / u otros tipos de datos. Además, el registro es un registro de vector u otro tipo de registro.

Los datos de caracteres incluyen, pero no están limitados a, caracteres alfabéticos en cualquier lenguaje, dígitos numéricos, puntuación y / u otros símbolos. Los datos de caracteres pueden ser, o no, cadenas de datos. Hay normas que se encuentran asociadas con los datos de caracteres, ejemplos de los cuales incluyen, pero no están limitados a, ASCII (*American Standard Code for Information Interchange*, código normalizado americano para intercambio de información); Unicode, incluyendo, pero sin limitarse a, UTF (*Unicode Transformation Format*, formato de transformación de Unicode) 8; UTF16, etc.

Un registro de vector (al que también se hace referencia como vector) incluye uno o más elementos y cada elemento es de uno, dos o cuatro bytes de longitud, como ejemplos. Además, un operando de vector es, por ejemplo, un operando de SIMD (*Single Instruction, Multiple Data*, una sola instrucción, múltiples datos) que tiene una pluralidad de elementos. En otras formas de realización, los elementos pueden ser de otros tamaños y un operando de vector no necesita ser SIMD y / o puede incluir un elemento.

En un ejemplo, se proporciona una instrucción de vector de carga a frontera de bloque que carga un número variable de bytes de datos de la memoria a un registro de vector al tiempo que se asegura que no se cruce una frontera especificada de la memoria de la cual se están cargando los datos. La frontera puede ser especificada de forma explícita por la instrucción (por ejemplo, un valor variable en el texto de instrucción, un valor de texto de instrucción fijo codificado en el código de operación, una frontera basada en registro especificada en la instrucción, etc.) o la frontera puede ser determinada de forma dinámica por la máquina. Por ejemplo, la instrucción especifica que los datos se van a cargar en una página o frontera de memoria caché y la máquina determina la línea de memoria caché o el tamaño de página (por ejemplo, consulta, por ejemplo, en una memoria de almacenamiento intermedio de traducción anticipada para determinar el tamaño de página) y carga a ese punto.

Como un ejemplo adicional, esta instrucción también se usa para alinear accesos de datos a una frontera seleccionada.

En una forma de realización, la instrucción solo carga bytes de registro de vector (un primer operando) con bytes correspondientes de un segundo operando que están dentro de un bloque de memoria principal especificado por la

instrucción. Tal como se usa en la presente memoria, un bloque de memoria principal (que también se conoce como almacenamiento principal) es cualquier bloque de memoria de un tamaño especificado. También se hace referencia al tamaño especificado como frontera del bloque, siendo la frontera el extremo del bloque.

En una forma de realización adicional, se cargan otros tipos de registro. Es decir, el registro que se carga no es un registro de vector sino otro tipo de registro. En este contexto, se hace referencia a la instrucción como instrucción de carga a frontera de bloque, que se usa para cargar datos en un registro.

Una forma de realización de un entorno informático para incorporar y usar uno o más aspectos de la presente invención se describe con referencia a la figura 1. Un entorno informático 100 incluye, por ejemplo un procesador 102 (por ejemplo, una unidad de procesamiento central), una memoria 104 (por ejemplo, una memoria principal) y uno o más dispositivos de entrada / salida (E / S) y / o interfaz 106 que están acoplados entre sí a través de, por ejemplo, uno o más buses 108 y / u otras conexiones.

En un ejemplo, el procesador 102 se basa en la arquitectura z/Architecture ofrecida por International Business Machines Corporation y es parte de un servidor, tal como el servidor System Z, que también es ofrecido por International Business Machines Corporation e implementa la arquitectura z/Architecture. Una forma de realización de la arquitectura z/Architecture se describe en una publicación de IBM®, que lleva por título "*z/Architecture Principles of Operation*" publicación de IBM® n.º SA22-7832-08, novena edición, agosto de 2010. En un ejemplo, el procesador ejecuta un sistema operativo, tal como Z/OS, también ofrecido por International Business Machines Corporation. IBM®, Z/ARCHITECTURE® y Z/OS® son marcas registradas de International Business Machines Corporation, Armonk, Nueva York, Estados Unidos de América. Otros nombres usados en la presente memoria pueden ser de marcas registradas, marcas o nombre de producto de International Business Machines Corporation u otras compañías.

En un ejemplo adicional, el procesador 102 se basa en la Power Architecture de International Business Machines Corporation. Una forma de realización de la 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 registrada de International Business Machines Corporation.

En todavía un ejemplo adicional, el procesador 102 se basa en la arquitectura Intel ofrecida por Intel Corporation. Un ejemplo de la arquitectura Intel se describe en "*Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 2B, Instructions Set Reference, A-L*" (Manual del desarrollador de las arquitecturas Intel® 64 e IA-32: Vol. 2B, referencia de conjunto de instrucciones, A-L), número de pedido 253666-041US, diciembre de 2011 y "*Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 2B, Instructions Set Reference, M-Z*" (Manual del desarrollador de las arquitecturas Intel® "Intel® 64 e IA-32 Vol. 2B, referencia de conjunto de instrucciones, M-Z) número de pedido 253667-041US, diciembre 2011. Intel® es una marca registrada de Intel Corporation, Santa Clara, California.

Otra forma de realización de un entorno informático para incorporar y usar uno o más aspectos de la presente invención se describe con referencia a la figura 2A. En este ejemplo, un entorno informático 200 incluye, por ejemplo, una unidad de procesamiento central nativa 202, una memoria 204 y uno o más dispositivos de entrada / salida y / o interfaz 206 acoplados entre sí a través de, por ejemplo, uno o más buses 208 y / u otras conexiones. Como ejemplos, el entorno informático 200 puede incluir un procesador PowerPC, un servidor pSeries o un servidor xSeries ofrecidos por International Business Machines Corporation, Armonk, Nueva York; un HP SuperDome con procesadores Intel Itanium II ofrecidos por Hewlett Packard Co., Palo Alto, California y / o u otras máquinas basadas en arquitecturas ofrecidas por International Business Machines Corporation, Hewlett Packard, Intel, Oracle u otros.

La unidad de procesamiento central nativa 202 incluye uno o más registros nativos 210, tales como uno o más registros de propósito general y / o uno o más registros de propósito especial usados durante el procesamiento dentro del entorno. Estos registros incluyen una información que representa el estado del entorno en cualquier instante particular en el tiempo.

Además, la unidad de procesamiento central nativa 202 ejecuta instrucciones y código que están almacenados en la memoria 204. En un ejemplo particular, la unidad de procesamiento central ejecuta el código de emulador 212 almacenados en la memoria 204. Este código permite que el entorno de procesamiento configurado en una arquitectura emule otra arquitectura. Por ejemplo, el código de emulador 212 permite que máquinas basadas en arquitecturas diferentes de la arquitectura z/Architecture, tales como procesadores PowerPC, servidores pSeries, servidores xSeries, servidores HP SuperDome u otros emulen la arquitectura z/Architecture y ejecuten soporte lógico e instrucciones que se desarrollan basándose en la arquitectura z/Architecture.

Detalles adicionales en relación con el código de emulador 212 se describen con referencia a la figura 2B. Las instrucciones de invitado 250 comprenden unas instrucciones de soporte lógico (por ejemplo, instrucciones de máquina) que se desarrollaron para ejecutarse en una arquitectura diferente de la de la CPU nativa 202. Por ejemplo, las instrucciones de invitado 250 pueden haber sido diseñadas para ejecutarse en un procesador de arquitectura z/Architecture 102 pero, en su lugar, las mismas se emulan en la CPU nativa 202, que puede ser, por ejemplo, un procesador Itanium II de Intel. En un ejemplo, el código de emulador 212 incluye una unidad de

extracción de instrucciones 252 para obtener una o más instrucciones de invitado 250 de la memoria 204 y para proporcionar opcionalmente un almacenamiento en memoria caché local para las instrucciones obtenidas. Este incluye también una rutina de traducción de instrucciones 254 para determinar el tipo de instrucción de invitado que se ha obtenido y para traducir la instrucción de invitado a una o más instrucciones nativas 256 correspondientes. Esta traducción incluye, por ejemplo, identificar la función que va a ser realizada por la instrucción de invitado y escoger la instrucción o instrucciones nativas para realizar esa función.

Además, el emulador 212 incluye una rutina de control de emulación 260 para dar lugar a que se ejecuten las instrucciones nativas. La rutina de control de emulación 260 puede dar lugar a que la CPU nativa 202 ejecute una rutina de instrucciones nativas que emulan una o más instrucciones de invitado obtenidas previamente y, a la conclusión de tal ejecución, devuelvan el control a la rutina de extracción de instrucciones para emular la obtención de la instrucción de invitado siguiente o un grupo de instrucciones de invitado. La ejecución de las instrucciones nativas 256 puede incluir cargar datos en un registro de la memoria 204; almacenar los datos de vuelta a la memoria de un registro o realizar un algún tipo de operación aritmética o lógica, según sea determinado por la rutina de traducción.

Cada rutina se implementa, por ejemplo, en un soporte lógico, que se almacena en memoria y es ejecutado por la unidad de procesamiento central nativa 202. En otros ejemplos, una o más de las rutinas u operaciones se implementan en soporte lógico inalterable, soporte físico, soporte lógico o alguna combinación de los mismos. Los registros del procesador emulado se pueden emular usando los registros 210 de la CPU nativa o al usar sitios en la memoria 204. En algunas formas de realización, las instrucciones de invitado 250, las instrucciones nativas 256 y el código de emulador 212 pueden residir en la misma memoria o pueden estar dispersados entre diferentes dispositivos de memoria.

Tal como se usa en la presente memoria, el soporte lógico inalterable incluye, por ejemplo, microcódigo, minicódigo y / o macrocódigo del procesador. Este incluye, por ejemplo, las instrucciones a nivel de soporte físico y / o estructuras de datos que se usan en la implementación de código máquina de nivel superior. En un ejemplo, este incluye, por ejemplo, un código patentado que se proporciona, por lo general, como microcódigo que incluye soporte lógico de confianza o microcódigo específico del soporte físico subyacente y controla el acceso del sistema operativo al soporte físico de sistema.

En un ejemplo, una instrucción de invitado 250 que se obtiene, se traduce y se ejecuta es la instrucción que se describe en la presente memoria. La instrucción, que es de una arquitectura (por ejemplo, la arquitectura z/Architecture) se extrae de la memoria, se traduce y se representa como una secuencia de instrucciones nativas 256 de otra arquitectura (por ejemplo, PowerPC, pSeries, xSeries, Intel, etc.). Entonces se ejecutan estas instrucciones nativas.

En una forma de realización, la instrucción que se describe en la presente memoria es una instrucción de vector, que es parte de un recurso de vectores, que se proporciona de acuerdo con un aspecto de la presente invención. El recurso de vectores proporciona, por ejemplo vectores de tamaño fijo que varían de uno a 16 elementos. Cada vector incluye unos datos sobre los que se opera por medio de unas instrucciones de vector definidas en el recurso. En una forma de realización, si un vector está compuesto por múltiples elementos, entonces cada elemento se procesa en paralelo con los otros elementos. La compleción de la instrucción no tiene lugar hasta que se ha completado el procesamiento de todos los elementos.

Tal como se describe en la presente memoria, las instrucciones de vector se pueden implementar como parte de diversas arquitecturas, incluyendo, pero sin limitarse a, la arquitectura z/Architecture, Power, Intel, etc. A pesar de que se describe una forma de realización en la presente memoria para la arquitectura z/Architecture, las instrucciones de vector y uno o más aspectos de la presente invención se pueden basar en muchas otras arquitecturas. La arquitectura z/Architecture es solo un ejemplo.

En una forma de realización en la que el recurso de vectores se implementa como parte de la arquitectura z/Architecture, para usar los registros de vector e instrucciones, un control de habilitación de vector y un control de registro en un registro de control especificado (por ejemplo, un registro de control 0) se establecen a, por ejemplo, uno. Si se instala el recurso de vectores y se ejecuta una instrucción de vector sin los controles de habilitación establecidos, se reconoce una excepción de datos. Si no se instala el recurso de vectores y se ejecuta una instrucción de vector, se reconoce una excepción de operación.

Los datos de vector aparecen en el almacenamiento, por ejemplo, en la misma secuencia de izquierda a derecha que otros formatos de datos. Los bits de un formato de datos que se numeran 0 - 7 constituyen el byte en el sitio de byte más a la izquierda (de numeración más baja) en el almacenamiento, los bits 8 - 15 forman el byte en el siguiente sitio secuencial y así sucesivamente. En un ejemplo adicional, los datos de vector pueden aparecer en el almacenamiento en otra secuencia, tal como de derecha a izquierda.

Muchas de las instrucciones de vector dotadas del recurso de vectores tienen un campo de bits especificados. Este campo, al que se hace referencia como bit de ampliación de registro o RXB, incluye el bit más significativo para cada

uno de los operandos designados de registro de vector. Los bits para las designaciones de registro no especificados por la instrucción se han de reservar y de establecer a cero.

En un ejemplo, el campo de RXB incluye cuatro bits (por ejemplo, los bits 0 - 3) y los bits se definen tal 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 del registro de vector de la instrucción, si la hay.

2 - bit más significativo para la tercera designación de registro de vector de la instrucción, si la hay.

3 - bit más significativo para la cuarta designación de registro de vector de la instrucción, si la hay.

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

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

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

Una instrucción, que se proporciona de acuerdo con un aspecto de la presente invención que incluye el campo de RXB, es una instrucción de vector de carga a frontera de bloque, un ejemplo de la cual se ilustra en la figura 3. En un ejemplo, la instrucción de vector de carga a frontera de bloque 300 incluye los campos de código de operación 302a (por ejemplo, los bits 0 - 7), 302b (por ejemplo, los bits 40 - 47) que indican una operación de vector de carga a frontera de bloque; un campo de registro de vector 304 (por ejemplo, 8 - 11) que se usa para designar un registro de vector (V_1); un campo de índice (X_2) 306 (por ejemplo, los bits 12 - 15); un campo de base (B_2) 308 (por ejemplo, los bits 16 - 19); un campo de desplazamiento (D_2) 310 (por ejemplo, los bits 20 - 31); un campo de máscara (M_3) 312 (por ejemplo, los bits 32 - 35) y un campo de RXB 316 (por ejemplo, los bits 36 - 39). Cada uno de los campos 304 - 314, en un ejemplo, es separado e independiente del campo o campos de código de operación. Además, en una forma de realización, son separados e independientes entre sí; no obstante, en otras formas de realización, se puede combinar más de un campo. Información adicional acerca del uso de estos campos se describe en lo sucesivo en la presente memoria.

En un ejemplo, los bits seleccionados (por ejemplo, los primeros dos bits) del código de operación designado por el campo de código de operación 302a especifican la longitud y el formato de la instrucción. En este ejemplo particular, la longitud es de tres medias palabras y el formato es una operación de registro de vector y almacenamiento con índice con un campo de código de operación ampliado. El campo de vector (V) junto con su bit de ampliación correspondiente especificado por RXB, designa un registro de vector. 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 del bit de ampliación de registro (RXB, *register extension bit*) como el bit más significativo. Por ejemplo, si el campo de cuatro bits es 0110 y el bit de ampliación es 0, entonces el campo de cinco bits 00110 indica el número de registro 6.

El número de subíndice asociado con un campo de la instrucción denota el operando al cual se aplica el campo. Por ejemplo, el número de subíndice 1 asociado con V_1 denota el primer operando y así sucesivamente. El operando de registro es de una longitud de un registro, que es, por ejemplo, de 128 bits.

En un ejemplo, en una instrucción de operación de registro de vector y almacenamiento con índice, el contenido de los registros generales designados por los campos X_2 y B_2 se añade al contenido del campo D_2 para formar la dirección del segundo operando. El desplazamiento D_2 para la instrucción de vector de carga a frontera de bloque se trata como un número entero sin signo de 12 bits, en un ejemplo.

El campo M_3, en una forma de realización, especifica un código que se usa para señalizar la CPU en lo que respecta a la frontera de bloque a la que cargar. Si se especifica un valor reservado, se reconoce una excepción de especificación. Códigos a modo de ejemplo y valores correspondientes son tal como sigue.

Código	**Frontera**
0	64 bytes
1	128 bytes
2	256 bytes

3	512 bytes
4	1 kbyte
5	2 kbytes
6	4 kbytes

En la ejecución de una forma de realización de la instrucción de vector de carga a frontera de bloque (VLBB, *Vector Load To Block Boundary*), procediendo en una forma de realización de izquierda a derecha, el primer operando (que se especifica en el registro designado por el campo V_1 más el bit de ampliación) se carga empezando en el elemento de byte indexado cero con bytes del segundo operando. El segundo operando se encuentra en un sitio de memoria designado por la dirección del segundo operando (a la que también se hace referencia como una dirección de partida). La carga empieza a partir del sitio de memoria y continúa a una dirección de fin calculada por la instrucción (o procesador), tal como se describe en lo sucesivo. Si se encuentra una condición de frontera, depende del modelo cómo se trata el resto del primer operando. No se reconocen excepciones de acceso en bytes no cargados. En un ejemplo, los bytes que no se cargan no se pueden predecir.

En la instrucción a modo de ejemplo anterior, la dirección de partida es determinada por el valor de registro de índice (X_2) + un valor de registro de base (B_2) + un desplazamiento (D_2); no obstante, en otras formas de realización, se dota de un valor de registro; una dirección de instrucción + desplazamiento especificado por texto de instrucción; un valor de registro + desplazamiento o un valor de registro + valor de registro de índice; como solo algunos ejemplos. Además, en una forma de realización, la instrucción no incluye el campo de RXB. En su lugar, no se usa ninguna ampliación o la ampliación se proporciona de otra forma, tal como a partir de un control fuera de la instrucción o se proporciona como parte de otro campo de la instrucción.

Detalles adicionales de una forma de realización del procesamiento de la instrucción de vector de carga a frontera de bloque se describen con referencia a la figura 4. En un ejemplo, un procesador del entorno informático está realizando esta lógica.

En una forma de realización, inicialmente se crea una máscara de frontera (BdyMask, *boundary mask*) que se usa para determinar la cercanía a una frontera especificada, en la etapa 400. Para crear la máscara, en un ejemplo, se toma una negación de un complemento a 2 de un tamaño de frontera (BdySize, *boundary size*) 402 creando la máscara de frontera 404 (por ejemplo, BdyMask = 0 - BdySize). El tamaño de frontera es proporcionado, en un ejemplo, por la instrucción (por ejemplo, el campo M_3) o, en otro ejemplo, es determinado por la máquina, tal como se describe en la presente memoria.

A continuación de lo anterior, se calcula una dirección de partida, que indica un sitio en la memoria a partir del cual va a comenzar la carga, en la etapa 410. Como ejemplos, la dirección de partida 412 puede ser proporcionada por un valor de registro; una instrucción de dirección más desplazamiento especificado por texto de instrucción; un valor de registro más desplazamiento; un valor de registro más valor de registro de índice o un valor de registro más valor de registro de índice más desplazamiento. En la instrucción que se proporciona en la presente memoria, la dirección de partida es proporcionada por el campo X_2, el campo B_2 y el campo D_2. Es decir, el contenido de los registros designados por X_2 y B_2 se añade al desplazamiento indicado por D_2 para proporcionar la dirección de partida. Las formas indicadas en lo que antecede para calcular una dirección de partida son solo ejemplos; también son posibles otros ejemplos.

A continuación, se calcula una dirección de fin que indica de dónde detener la carga, en la etapa 420. La entrada a este cálculo es, por ejemplo, el tamaño de frontera 402, la dirección de partida 412, el tamaño de vector 414 (por ejemplo, en bytes; por ejemplo, 16) y la máscara de frontera 404. En un ejemplo, la dirección de fin 422 se calcula tal como sigue:

Dirección de fin = mín (dirección de partida + (Bdysize) - (dirección de partida & ¬BdyMask)), dirección de partida + tamaño de vector).

A continuación de lo anterior, se carga el primer operando (es decir, el registro de vector designado), empezando en el byte indexado cero, desde la memoria comenzando en la dirección de partida y terminando en la dirección de fin, en la etapa 430. Esto permite que un número variable de bytes se carguen desde la memoria a un vector sin cruzar una frontera de memoria designada. Por ejemplo, si la frontera de memoria se encuentra a 64 bytes y la dirección de partida se encuentra a 58 bytes, entonces los bytes 58 - 64 se cargan en el registro de vector.

Un ejemplo de datos que se van a cargar en un registro de vector, de acuerdo con un aspecto de la presente invención, se ilustra en la figura 5. Tal como se indica, ningún dato se carga más allá de la frontera designada por la línea vertical de trazo discontinuo. Los sitios más allá de la frontera no son accesibles y no se toma ninguna excepción. En una forma de realización particular, el vector se carga de izquierda a derecha. No obstante, en otra

forma de realización, este se puede cargar de derecha a izquierda. En una forma de realización, la dirección de los vectores, de izquierda a derecha o de derecha a izquierda, se proporciona en tiempo de ejecución. Por ejemplo, la instrucción accede a un registro, un control de estatus u otra entidad que indica que la dirección de procesamiento es o bien de izquierda a derecha o bien de derecha a izquierda, como ejemplos. En una forma de realización, este control de dirección no se codifica como parte de la instrucción, sino que se proporciona a la instrucción en tiempo de ejecución.

Se ha descrito en lo que antecede un ejemplo de una instrucción de carga. Cuando se cargan datos, tales como datos de cadena, a menudo no se sabe si la cadena terminará antes de una frontera de página. La capacidad de cargar hasta esa frontera sin cruce requiere en primer lugar, por lo general, verificar el fin de la cadena. Algunas implementaciones también pueden tener una penalización por cruzar fronteras y el soporte lógico podría desear evitar las mismas. De este modo, es útil la capacidad de cargar hasta diversas fronteras. Se proporciona una instrucción que carga un número variable de bytes a un registro de vector al tiempo que se asegura que no se carguen los datos al otro lado de una frontera especificada.

En una forma de realización, hay 32 registros de vector y se pueden poner en correspondencia otros tipos de registros con un cuadrante de los registros de vector. Por ejemplo, tal como se muestra en la figura 6, si hay un archivo de registro 600 que incluye 32 registros de vector 602 y cada registro es de 128 bytes de longitud, entonces 16 registros de coma flotante 604, que son de 64 bytes de longitud, se pueden superponer a los registros de vector. De este modo, como un ejemplo, cuando se modifica el registro de coma flotante 2, entonces también se modifica el registro de vector 2. También son posibles otras puestas en correspondencia para otros tipos de registro.

En la presente memoria, memoria, memoria principal, almacenamiento y almacenamiento principal se usan de forma intercambiable a menos que se indique lo contrario, de forma explícita o por el contexto.

En el Anexo se proporcionan detalles adicionales en relación con el recurso de vectores, incluyendo ejemplos de otras instrucciones.

Tal como será apreciado por el experto en la técnica, uno o más aspectos de la presente invención se pueden implementar como un sistema, método o producto de programa informático. De este modo, uno o más aspectos de la presente invención pueden adoptar la forma de una forma de realización completamente de soporte físico, una forma de realización completamente de soporte lógico (incluyendo soporte lógico inalterable, soporte lógico residente, microcódigo, etc.) o una forma de realización que combine aspectos de soporte lógico y de soporte físico, a la totalidad de los cuales se puede hacer referencia en la presente memoria como “circuito”, “módulo” o “sistema”. Además, uno o más aspectos de la presente invención pueden adoptar la forma de un producto de programa informático implementado en uno o más medio o medios legibles por ordenador que tienen código de programa legible por ordenador implementado en el mismo.

Se puede usar 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 limitarse a, un sistema, un aparato o un dispositivo electrónico, magnético, óptico, electromagnético, de infrarrojos o de semiconductores o cualquier configuración apropiada de lo anterior. Ejemplos más específicos (una lista no exhaustiva) de los medios de almacenamiento legible por ordenador incluyen los siguientes: una conexión eléctrica que tiene uno o más hilos, un disco flexible informático portátil, un disco duro, una memoria de acceso aleatorio (RAM, *random access memory*), una memoria de solo lectura (ROM, *read-only memory*), una memoria de solo lectura programable borrable (EPROM, *erasable programmable read-only memory*, o memoria flash), una fibra óptica, un disco compacto - memoria de solo lectura (CD-ROM, *compact disc read-only memory*) portátil, un dispositivo de almacenamiento óptico, un dispositivo de almacenamiento magnético o cualquier combinación apropiada de lo anterior. En el contexto del presente 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, un aparato o un dispositivo de ejecución de instrucciones.

Haciendo referencia a continuación a la figura 7, en un ejemplo, un producto de programa informático 700 incluye, por ejemplo, uno o más medios de almacenamiento legible por ordenador no transitorio 702 para almacenar unos medios de código de programa legible por ordenador o una lógica 704 en los mismos para proporcionar y facilitar uno o más aspectos de la presente invención.

El código de programa implementado en un medio legible por ordenador se puede transmitir usando un medio apropiado, incluyendo, pero sin limitarse a, medios inalámbricos, cableados, cable de fibra óptica, RF, etc., o cualquier combinación apropiada de lo anterior.

El código de programa informático para llevar a cabo operaciones para uno o más aspectos de la presente invención se puede escribir 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 procedimentales convencionales, tales como el lenguaje de programación “C”, ensamblador o lenguajes de programación similares. El código de programa se puede ejecutar completamente en el ordenador del usuario, parcialmente en el ordenador del usuario, como un paquete de soporte lógico autónomo, parcialmente en el

ordenador del usuario y parcialmente en un ordenador remoto o completamente en el ordenador remoto o servidor. En el último escenario, el ordenador remoto se puede conectar al ordenador del usuario por medio de cualquier tipo de red, incluyendo una red de área local (LAN, *local area network*) o una red de área extensa (WAN, *wide area network*) o la conexión se puede hacer a un ordenador externo (por ejemplo, por medio de Internet usando un proveedor de servicios de Internet).

Uno o más aspectos de la presente invención se describen en la presente memoria con referencia a las instrucciones de diagrama de flujo y / o los diagramas de bloques de métodos, aparatos (sistemas) y productos de programa informático de acuerdo con formas de realización de la invención. Se entenderá que cada bloque de las instrucciones de diagrama de flujo y / o los diagramas de bloques y combinaciones de los bloques en las ilustraciones de diagrama de flujo y / o los diagramas de bloques, pueden ser implementados por instrucciones de programa informático. Estas instrucciones de programa informático se pueden proporcionar a un procesador de un ordenador de propósito especial, un ordenador de propósito especial u otro aparato de procesamiento de datos programable para producir una máquina, de tal forma que las instrucciones que se ejecutan a través del procesador del ordenador u otro aparato de procesamiento de datos programable, creen unos medios para implementar las funciones / acciones que se especifican en el diagrama de flujo y / o el bloque o bloques del diagrama de bloques.

Estas instrucciones de programa informático también se pueden almacenar en un medio legible por ordenador que puede indicar a un ordenador, otro aparato de procesamiento de datos programable u otros dispositivos que funcionen de una forma particular, de tal forma que las instrucciones almacenadas en el medio legible por ordenador produzcan un artículo de fabricación, incluyendo instrucciones que implementan la función / acción que se especifica en el diagrama de flujo y / o el bloque o bloques del diagrama de bloques.

Las instrucciones de programa informático también se pueden cargar en un ordenador, otro aparato de procesamiento de datos programable u otros dispositivos para dar lugar a que se realice una serie de etapas operativas en el ordenador, otros aparatos programables u otros dispositivos para producir un procesado implementado por ordenador de tal forma que las instrucciones que se ejecutan en el ordenador u otro aparato programable proporcionen procesos para implementar las funciones / acciones que se especifican en el diagrama de flujo y / o el bloque o bloques del diagrama de bloques.

El diagrama de flujo y los diagramas de bloques en las figuras ilustran la arquitectura, la funcionalidad y el funcionamiento de implementaciones de sistemas, métodos y productos de programa informático de acuerdo con diversas formas de realización de uno o más aspectos de la presente invención. A este respecto, cada bloque en el diagrama de flujo o diagrama de bloques puede representar un módulo, un segmento o una porción de código, que comprende una o más instrucciones ejecutables para implementar la función o funciones lógicas que se especifican. También se debe hacer notar 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 que se muestran en sucesión se pueden ejecutar, de hecho, de forma sustancialmente concurrente o, algunas veces, los bloques se pueden ejecutar en el orden inverso, dependiendo de la funcionalidad involucrada. También se hará notar que cada bloque de los diagramas de bloque y / o ilustración del diagrama de flujo y combinaciones de bloques en los diagramas de bloques y / o ilustración del diagrama de flujo, puede ser implementado por sistemas basados en soporte físico de propósito especial que realizan las funciones o acciones que se especifican, o combinaciones de soporte físico de propósito especial e instrucciones informáticas.

Además de lo anterior, uno o más aspectos de la presente invención pueden ser proporcionados, ofrecidos, implementados, gestionados, atendidos, etc., por un proveedor de servicios que ofrece una gestión de entornos de cliente. Por ejemplo, el proveedor de servicios puede crear, mantener, soportar, etc., un 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 servicios puede recibir un pago del cliente según un abono y / o acuerdo de cuota, como ejemplos. Adicionalmente, o como alternativa, el proveedor de servicios puede recibir un pago de la venta de contenido publicitario a una o más terceras partes.

En un aspecto de la presente invención, una aplicación se puede implementar para realizar uno o más aspectos de la presente invención. Como un ejemplo, la implementación de una aplicación comprende proporcionar una infraestructura informática que se puede hacer funcionar para realizar uno o más aspectos de la presente invención.

Como un aspecto adicional de la presente invención, se puede implementar una infraestructura informática que comprende integrar un 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 todavía un aspecto adicional de la presente invención, se puede proporcionar un proceso para integrar una infraestructura informática que comprende integrar un 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.

A pesar de que diversas formas de realización se han descrito en lo que antecede, estas son solo ejemplos. Por ejemplo, los entornos informáticos de otras arquitecturas pueden incorporar y usar uno o más aspectos de la presente invención. Además, se pueden usar registros de otros tamaños y se pueden hacer cambios a las instrucciones sin apartarse del alcance de la presente invención.

Además, otros tipos de entornos informáticos se pueden beneficiar de uno o más aspectos de la presente invención. Como un ejemplo, se puede usar un sistema de procesamiento de datos apropiado para almacenar y / o ejecutar código de programa, que incluye al menos dos procesadores acoplados directa o indirectamente a elementos de memoria por medio de un bus de sistema. Los elementos de memoria incluyen, por ejemplo, memoria local empleada durante la ejecución real del código de programa, almacenamiento global y memoria caché que proporcionan un almacenamiento temporal de al menos un cierto código de programa con el fin de reducir el número de veces que el código se ha de recuperar del almacenamiento global durante la ejecución.

Los dispositivos de entrada / salida o E / S (incluyendo, pero sin limitarse a, teclados, pantallas, dispositivos apuntadores, DASD, cinta, CD, DVD, unidad de memoria USB y otros medios de memoria, etc.) se pueden acoplar al sistema o bien directamente o bien por medio de controladores de E / S intermedios. También se pueden acoplar adaptadores de red al sistema para permitir que el sistema de procesamiento de datos se acople a otro sistema de procesamiento de datos o impresoras remotas o dispositivos de almacenamiento a través de redes privadas o públicas intermedias. Los módems, los módems de cable y las tarjetas de Ethernet son solo unos pocos de los tipos disponibles de adaptadores de red.

Haciendo referencia a la figura 8, se ilustran componentes representativos de un sistema informático de anfitrión 5000 para implementar uno o más aspectos de la presente invención. El sistema informático de anfitrión 5000 representativo incluye una o más CPU 5001 en comunicación con la memoria informática (es decir, almacenamiento central) 5002, así como unas interfaces de E / S a dispositivos de medios de almacenamiento 5011 y redes 5010 para comunicarse con otros ordenadores o SAN y similares. La CPU 5001 es conforme con una arquitectura que tiene un conjunto de secciones diseñadas y una funcionalidad diseñada. La CPU 5001 puede tener una traducción dinámica de direcciones (DAT, *dynamic address translation*) 5003 para transformar direcciones de programa (direcciones virtuales) en direcciones reales de memoria. Una DAT incluye, por lo general, una memoria de almacenamiento intermedio de traducción anticipada (TLB, *translation lookaside buffer*) 5007 para guardar en memoria caché traducciones de tal forma que los accesos posteriores al bloque de memoria informática 5002 no requieren el retardo de la traducción de direcciones. Por lo general, se emplea una memoria caché 5009 entre la memoria informática 5002 y el procesador 5001. La memoria caché 5009 puede ser jerárquica, teniendo una memoria caché grande disponible para más de una CPU y unas memorias caché más pequeñas y más rápidas (de nivel inferior) entre la memoria caché grande y cada CPU. En algunas implementaciones, las memorias caché de nivel inferior se dividen para proporcionar memorias caché de nivel bajo separadas para la extracción de instrucciones y los accesos de datos. En una forma de realización, una instrucción es extraída de la memoria 5002 por una unidad de extracción de instrucciones 5004 a través de una memoria caché 5009. La instrucción se descodifica en una unidad de descodificación de instrucciones 5006 y se despacha (con otras instrucciones en algunas formas de realización) a la unidad o unidades de ejecución de instrucciones 5008. Por lo general, se emplean diversas unidades de ejecución 5008, por ejemplo una unidad de ejecución aritmética, una unidad de ejecución de coma flotante y una unidad de ejecución de instrucciones de ramificación. La instrucción es ejecutada por la unidad de ejecución, accediendo a operandos de registros especificados por la instrucción o memoria según sea necesario. Si se va a acceder a un operando (cargado o almacenado) desde la memoria 5002, una unidad de carga / almacenamiento 5005 gestiona, por lo general, el acceso bajo el control de la instrucción que se está ejecutando. Las instrucciones pueden ser ejecutadas en circuitos de soporte físico o microcódigo interno (soporte lógico inalterable) o por una combinación de ambos.

Tal como se indica, un sistema informático incluye una información en el almacenamiento local (o principal), así como direccionamiento, protección y referencia y registro de cambios. Algunos aspectos del direccionamiento incluyen el formato de las direcciones, el concepto de los espacios de dirección, los diversos tipos de direcciones y la forma en la que se traduce un tipo de dirección a otro tipo de dirección. Parte del almacenamiento principal incluye sitios de almacenamiento asignados de forma permanente. El almacenamiento principal dota al sistema de un almacenamiento de datos de acceso rápido directamente direccionable. Tanto los datos como los programas se han de cargar en el almacenamiento principal (desde dispositivos de entrada) antes de que se puedan procesar los mismos.

El almacenamiento principal puede incluir uno o más almacenamientos de memoria de almacenamiento intermedio más pequeños y de acceso más rápido, que a veces se denominan memoria caché. La memoria caché esta, por lo general, asociada físicamente con una CPU y / o un procesador de E / S. Los efectos, excepto en el desempeño, de la construcción física y el uso de medios de almacenamiento distintos no son, en general, observables por el programa.

Se pueden mantener memorias caché separadas para instrucciones y para operandos de datos. La información dentro de una memoria caché se mantiene en bytes contiguos en una frontera integral que se denomina bloque de memoria caché o línea de memoria caché (o línea, por brevedad). Un modelo puede proporcionar una instrucción de EXTRACT CACHE ATTRIBUTE (extraer atributos de memoria caché) que devuelve el tamaño de una línea de

memoria caché en bytes. Un modelo también puede proporcionar instrucciones de PREFETCH DATA (pre-extraer datos) y de PREFETCH DATA RELATIVE LONG (pre-extraer datos relativamente largos) que realizan la pre-extracción de almacenamiento a la memoria caché de instrucciones o datos o la liberación de datos de la memoria caché.

El almacenamiento se ve como una cadena horizontal larga de bits. Para la mayor parte de las operaciones, los accesos al almacenamiento proceden 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 byte, que es el bloque de creación básico de todos los formatos de información. Cada sitio de byte en el almacenamiento se identifica por medio de un número entero no negativo único, que es la dirección de ese sitio de byte o, simplemente, la dirección de byte. Los sitios de byte adyacentes tienen direcciones consecutivas empezando por cero a la izquierda y procediendo en una secuencia de izquierda a derecha. Las direcciones son números enteros binarios sin signo y son de 24, de 31 o de 64 bits.

La información se transmite entre el almacenamiento y una CPU o un subsistema de canal, un byte o un grupo de bytes de cada vez. A menos que se especifique lo contrario, por ejemplo en la arquitectura z/Architecture, un grupo de bytes en el almacenamiento es direccionado por el byte más a la izquierda del grupo. El número de bytes en el grupo o bien es implícito o bien es especificado de forma explícita por la operación que se va a realizar. Cuando se usa en una operación de CPU, un grupo de bytes se denomina campo. Dentro de cada grupo de bytes, por ejemplo, en la arquitectura z/Architecture, los bits se numeran en una secuencia de izquierda a derecha. En la arquitectura z/Architecture, se hace referencia algunas veces a los bits más a la izquierda como “bits de orden alto” y a los bits más a la derecha como bits de “orden bajo”. Los números de bit no son direcciones de almacenamiento, no obstante. Solo se pueden direccionar los bytes. Para operar sobre los bits individuales de un byte en el almacenamiento, se accede a todo el byte. Los bits en un byte se numeran de 0 a 7, de izquierda a derecha (por ejemplo, en la arquitectura z/Architecture). Los bits en una dirección se pueden numerar 8 - 31 o 40 - 63 para las direcciones de 24 bits o 1 - 31 o 33 - 63 para las direcciones de 31 bits; estos se numeran 0 - 63 para las direcciones de 64 bits. Dentro de cualquier otro formato de longitud fija de múltiples bytes, los bits que componen el formato se numeran de forma consecutiva empezando desde 0. Para fines de detección de errores y, preferiblemente, por corrección, se pueden transmitir uno o más bits de verificación con cada byte o con un grupo de bytes. Tales bits de verificación son generados de forma automática por la máquina y no pueden ser controlados directamente por el programa. Las capacidades de almacenamiento se expresan en número de bytes. Cuando la longitud de un campo de operando de almacenamiento es implicada por el código de operación de una instrucción, se dice que el campo tiene una longitud fija, que puede ser de uno, dos, cuatro, ocho o dieciséis bytes. Se pueden implicar unos campos más grandes para algunas instrucciones. Cuando la longitud de un campo de operando de almacenamiento no se implica sino que se afirma de forma explícita, se dice que el campo tiene una longitud variable. Los operandos de longitud variable pueden variar de longitud en incrementos de un byte (o, con algunas instrucciones, en múltiplos de dos bytes u otros múltiplos). Cuando la información se coloca en el almacenamiento, solo se sustituye el contenido de aquellos sitios de byte que se incluyen en el campo designado, incluso aunque el ancho de la ruta física al almacenamiento puede ser mayor que la longitud del campo que se está almacenando.

Ciertas unidades de información van a estar en una frontera integral en el almacenamiento. Una frontera 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 dan nombres especiales a campos de 2, 4, 8 y 16 bytes sobre una frontera integral. Una media palabra es un grupo de dos bytes consecutivos sobre una frontera de dos bytes y es el bloque de creación básico de las instrucciones. Una palabra es un grupo de cuatro bytes consecutivos sobre una frontera de cuatro bytes. Una doble palabra es un grupo de ocho bytes consecutivos sobre una frontera de ocho bytes. Una cuadrupalabra es un grupo de dieciséis bytes consecutivos sobre una frontera de dieciséis bytes. Cuando las direcciones de almacenamiento designan medias palabras, palabras, dobles palabras y cuadrupalabras, la representación binaria de la dirección contiene uno, dos, tres o cuatro bits cero más a la derecha, de forma respectiva. Las instrucciones se han de encontrar sobre fronteras integrales de dos bytes. Los operandos de almacenamiento de la mayor parte de las instrucciones no tienen requisitos de alineación de frontera.

En dispositivos que implementan memorias caché separadas para instrucciones y operandos de datos, se puede experimentar un retardo significativo si el programa almacena en una línea de memoria caché de la cual se extraen posteriormente las ejecuciones, con independencia de si el almacenamiento altera las instrucciones que se extraen posteriormente.

En una forma de realización, la invención se puede poner en práctica por soporte lógico (al que se hace referencia algunas veces como código interno con licencia, soporte lógico inalterable, microcódigo, milicódigo, picocódigo y similares, cualquiera de los cuales sería consistente con uno o más aspectos de la presente invención). Haciendo referencia a la figura 8, al código de soporte lógico que implementa uno o más aspectos de la presente invención se puede acceder por medio del procesador 5001 del sistema de anfitrión 5000 a partir de los dispositivos de medios de almacenamiento de largo plazo 5011, tales como una unidad de CD-ROM, una unidad de cinta o un disco duro. El código de programa de soporte lógico se puede implementar en cualquiera de una diversidad de medios conocidos para su uso con un sistema de procesamiento de datos, tal como un disco flexible, un disco duro o un CD-ROM. El código se puede distribuir en tales medios o se puede distribuir a los usuarios a partir de la memoria informática 5002 o un almacenamiento de un sistema informático a través de una red 5010 a otros sistemas informáticos para su uso por los usuarios de otros sistemas de este tipo.

El código de programa de soporte lógico 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 se pagina normalmente de los dispositivos de medios de almacenamiento 5011 al almacenamiento informático de velocidad relativamente más alta 5002 en donde se encuentra disponible para su procesamiento por el procesador 5001. Las técnicas y métodos para implementar código de programa de soporte lógico en memoria, en medios físicos y / o distribuir código de soporte lógico a través de redes son bien conocidos y no se analizarán adicionalmente en la presente memoria. A menudo se hace referencia al código de programa, cuando se crea y se almacena en un medio tangible (incluyendo, pero sin limitarse a, módulos de memoria electrónicos (RAM), memoria flash, discos compactos (CD, *Compact Disc*), DVD, cinta magnética y similares como "producto de programa informático"). El producto de programa informático puede, por lo general, ser leído por un circuito de procesamiento preferiblemente en un sistema informático para su ejecución por el circuito de procesamiento.

La figura 9 ilustra una estación de trabajo o sistema de soporte físico de servidor representativo en el que se pueden poner en práctica uno más aspectos de la presente invención. El sistema 5020 de la figura 9 comprende un sistema informático de base 5021 representativo, tal como un ordenador personal, una estación de trabajo o un servidor, incluyendo dispositivos periféricos opcionales. El sistema informático de base 5021 incluye uno o más procesadores 5026 y un bus empleado para conectar y habilitar 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 el almacenamiento a largo plazo 5027 que puede incluir un disco duro (incluyendo cualquiera de medios magnéticos, CD, DVD y memoria flash, por ejemplo) o una unidad distinta, por ejemplo. El sistema 5021 también podría incluir un adaptador de interfaz de usuario, que conecta el microprocesador 5026 a través del bus a uno o más dispositivos de interfaz, tal como un teclado 5024, un ratón 5023, una impresora / escáner 5030 y / u otros dispositivos de interfaz, que pueden ser cualquier dispositivo de interfaz de usuario, tal como una pantalla sensible al tacto, un teclado de entrada digitalizada, etc. El bus también se conecta a un dispositivo de pantalla 5022, tal como una pantalla o monitor de LCD, al microprocesador 5026 a través de un adaptador de pantalla.

El sistema 5021 se puede comunicar con otros ordenadores o redes de ordenadores por medio de un adaptador de red que se puede comunicar 5028 con una red 5029. Adaptadores de red a modo de ejemplo son canales de comunicaciones, un anillo con paso de testigo, Ethernet o módems. Como alternativa, el sistema 5021 se puede comunicar usando una interfaz inalámbrica tal como una tarjeta de CDPD (*cellular digital packet data*, datos de paquete digital celular). El sistema 5021 puede estar asociado con otros componentes de este tipo en una red de área local (LAN, *Local Area Network*) o una red de área extensa (WAN, *Wide Area Network*) o el sistema 5021 puede ser un cliente en una disposición de cliente / servidor con otro ordenador, etc. La totalidad de estas configuraciones, así como el soporte físico y el soporte lógico de comunicaciones apropiados, son conocidos en la técnica.

La figura 10 ilustra una red de procesamiento de datos 5040 en la que se pueden poner en práctica uno o más aspectos de la presente invención. La red de procesamiento de datos 5040 puede incluir una pluralidad de redes individuales, tales como una red inalámbrica y una red cableada, cada una de las cuales puede incluir una pluralidad de estaciones de trabajo individuales 5041, 5042, 5043, 5044. Adicionalmente, tal como apreciarán los expertos en la técnica, se pueden incluir una o más LAN, en donde una LAN puede comprender una pluralidad de estaciones de trabajo inteligentes acopladas a un procesador de anfitrión.

Haciendo todavía referencia a la figura 10, las redes también pueden incluir unos grandes sistemas o servidores, tales como un ordenador de pasarela (el servidor de cliente 5046) o un servidor de aplicaciones (el servidor remoto 5048, que puede acceder a un depósito de datos y al que también se puede acceder directamente desde una estación de trabajo 5045). Un ordenador de pasarela 5046 sirve como un punto de entrada a cada red individual. Una pasarela es necesaria cuando se conecta un protocolo de interconexión de redes a otro. La pasarela 5046 se puede acoplar preferiblemente a otra red (Internet 5047, por ejemplo) por medio de un enlace de comunicaciones. La pasarela 5046 también se puede acoplar directamente a una o más estaciones de trabajo 5041, 5042, 5043, 5044 usando un enlace de comunicaciones. El ordenador de pasarela se puede implementar usando un servidor System Z eServer™ de IBM facilitado por International Business Machines Corporation.

Haciendo referencia de forma concurrente a la figura 9 y la figura 10, al código de programación de soporte lógico que puede implementar uno o más aspectos de la presente invención se puede acceder por medio del procesador 5026 del sistema 5020 a partir de los medios de almacenamiento a largo plazo 5027, tales como una unidad de CD-ROM o un disco duro. El código de programación de soporte lógico se puede implementar en cualquiera de una diversidad de medios conocidos para su uso con un sistema de procesamiento de datos, tales como un disco flexible, una unidad de disco duro o un CD-ROM. El código se puede distribuir en tales medios o se puede distribuir a los usuarios 5050, 5051 a partir de la memoria o el almacenamiento de un sistema informático a través de una red a otro sistema informático para su uso por usuarios de otros sistemas de este tipo.

Como alternativa, el código de programación se puede implementar en la memoria 5025 y el procesador 5026 puede acceder al mismo 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 de aplicación 5032. El código de programa se pagina normalmente de los medios de almacenamiento 5027 a la memoria de alta velocidad 5025 en donde se encuentra disponible para su procesamiento por el procesador 5026. Las técnicas y

métodos para implementar código de programación de soporte lógico en memoria, en medios físicos y / o distribuir código de soporte lógico a través de redes, que son bien conocidos y no se analizarán adicionalmente en la presente memoria. A menudo, se hace referencia al código de programa, cuando se crea y se almacena en un medio tangible (incluyendo, pero sin limitarse a, módulos de memoria electrónicos (RAM), memoria flash, discos compactos (CD), DVD, cinta magnética y similares como "producto de programa informático". El medio de producto de programa informático es, por lo general, legible por un circuito de procesamiento preferiblemente en un sistema informático para su ejecución por el circuito de procesamiento.

La memoria caché que está más fácilmente disponible para el procesador (normalmente más rápida y más pequeña que otras memorias caché del procesador) es la memoria caché más baja (L1 o nivel 1) y el almacenamiento principal (memoria principal) es la memoria caché de nivel más alto (L3 si hay tres niveles). La memoria caché de nivel más bajo se divide a menudo en una memoria caché de instrucciones (memoria caché I) que contiene instrucciones de máquina que se van a ejecutar y una memoria caché de datos (memoria caché D) que contiene operandos de datos.

Haciendo referencia a la figura 11, se ilustra una forma de realización de procesador a modo de ejemplo para el procesador 5026. Por lo general, se emplean uno o más niveles de memoria caché 5053 para almacenar en memoria intermedia bloques de memoria con el fin de mejorar el desempeño del procesador. La memoria caché 5053 es una memoria caché de alta velocidad que contiene líneas de memoria caché de datos de memoria cuyo uso es probable. Las líneas de memoria caché típicas son de 64, 128 o 256 bytes de datos de memoria. A menudo se emplean memorias caché separadas para almacenar en memoria caché instrucciones más que para almacenar en memoria caché datos. La coherencia de memoria caché (sincronización de copias de líneas en memoria y memoria caché) es proporcionada, a menudo, por diversos algoritmos "espía" bien conocidos en la técnica. A menudo, se hace referencia al almacenamiento de memoria principal 5025 de un sistema de procesador como memoria caché. En un sistema de procesador que tiene 4 niveles de memoria caché 5053, algunas veces se hace referencia al almacenamiento principal 5025 como memoria caché de nivel 5 (L5) debido a que es, por lo general, más rápida y contiene solo una porción del almacenamiento no volátil (DASD, cinta, etc.) que se encuentra disponible para un sistema informático. El almacenamiento principal 5025 "guarda en memoria caché" páginas de los datos paginados a, y fuera de, el almacenamiento principal 5025 por el sistema operativo.

Un contador de programa (contador de instrucciones) 5061 mantiene un seguimiento de la dirección de la instrucción actual que se va a ejecutar. Un contador de programa en un procesador de arquitectura z/Architecture es de 64 bits y se puede truncar a 31 o 24 bits para soportar límites de direccionamiento anteriores. Un contador de programa se implementa, por lo general, en una PSW (*program status word*, palabra de estatus de programa) de un ordenador, de tal forma que persiste durante el cambio de contexto. De este modo, un programa en avance, que tiene un valor de contador de programa, puede ser interrumpido por ejemplo por el sistema operativo (cambio de contexto del entorno de programa al entorno del sistema operativo). La PSW del programa mantiene el valor del contador de programa mientras el programa no está activo y el contador de programa (en la PSW) del sistema operativo se usa mientras el sistema operativo se está ejecutando. Por lo general, el contador de programa es implementado por una cantidad igual al número de bytes de la instrucción actual. Las instrucciones de RISC (*Complex Instruction Set Computing*, cómputo de conjunto de instrucciones complejas) son, por lo general, de longitud variable. Las instrucciones de la arquitectura z/Architecture de IBM son instrucciones de CISC que tienen una longitud de 2, 4 o 6 bytes. El contador de programa 5061 es modificado o bien por una operación de cambio de contexto o bien por una operación de ramificación tomada de una instrucción de ramificación por ejemplo. En una operación de cambio de contexto, el valor del contador de programa actual se guarda en la palabra de estatus de programa junto con otra información de estado acerca del programa que se ejecuta (tales 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 que se va a ejecutar. Una operación de ramificación tomada se realiza con el fin de permitir que el programa tome decisiones o realicé bucles dentro del programa al cargar el resultado de la instrucción de ramificación en el contador del programa 5061.

Por lo general, se usa una unidad de extracción de instrucciones 5055 para extraer instrucciones en nombre del procesador 5026. La unidad de extracción extrae o bien "siguientes instrucciones secuenciales", instrucciones objetivo de instrucciones de ramificación tomadas o bien primeras instrucciones de un programa a continuación de un cambio de contexto. Las unidades de extracción de instrucciones modernas a menudo emplean técnicas de extracción para pre-extraer de forma especulativa instrucciones basadas en la probabilidad de que se pudieran usar las instrucciones pre-extraídas. Por ejemplo, una unidad de extracción puede extraer 16 bytes de instrucción que incluyen la siguiente instrucción secuencial y bytes adicionales de instrucciones secuenciales adicionales.

Las instrucciones extraídas son ejecutadas entonces por el procesador 5026. En una forma de realización, la instrucción o instrucciones extraídas se hacen pasar a una unidad de despacho 5056 de la unidad de extracción. La unidad de despacho descodifica la instrucción o instrucciones y envía información acerca de la instrucción o instrucciones descodificadas a unas unidades 5057, 5058, 5060 apropiadas. Una unidad de ejecución 5057 recibirá, por lo general, información acerca de instrucciones aritméticas descodificadas de la unidad de extracción de instrucciones 5055 y realizará operaciones aritméticas sobre operandos de acuerdo con el código de operación de la instrucción. Los operandos se proporcionan a la unidad de ejecución 5057 preferiblemente o bien de la memoria 5025, o bien registros diseñados 5059 o bien de un campo inmediato de la instrucción que se está ejecutando. Los

resultados de ejecución, cuando se almacenan, se almacenan o bien en la memoria 5025, o bien en el registro 5059 o bien en otro soporte físico de máquina (tal como registros de control, registros de PSW y similares).

Un procesador 5026 tiene, por lo general, una o más unidades 5057, 5058, 5060 para ejecutar la función de la instrucción. Haciendo referencia a la figura 12A, una unidad de ejecución 5057 se puede comunicar con los registros generales diseñados 5059, una unidad de descodificación / despacho 5056, una unidad de almacenamiento de carga 5060 y otras unidades de procesador 5065 por medio de la lógica de interconexión 5071. Una unidad de ejecución 5057 puede emplear diversos circuitos de registro 5067, 5068, 5069 para contener información sobre la que operará la unidad lógica aritmética (ALU, *arithmetic logic unit*) 5066. La ALU realiza operaciones aritméticas tales como suma, resta, multiplicación y división, así como función lógica, tales como Y, O y O EXCLUSIVO (XOR), girar y desplazar. Preferiblemente, la ALU soporta operaciones especializadas que son dependientes del diseño. Otros circuitos pueden proporcionar otros recursos de arquitectura 5072 incluyendo códigos de condición y lógica de soporte de recuperación, por ejemplo. Por lo general, el resultado de una operación de ALU se mantiene en un circuito de registro de salida 5070 que puede enviar el resultado a una diversidad de otras funciones de procesamiento. Hay muchas disposiciones de unidades de procesador, la presente invención solo tiene por objeto proporcionar una comprensión representativa de una forma de realización.

Una instrucción de ADD, por ejemplo, se ejecutaría en una unidad de ejecución 5057 que tiene una funcionalidad aritmética y lógica mientras que una instrucción de coma flotante, por ejemplo, se ejecutaría en una ejecución de coma flotante que tiene una capacidad de coma flotante especializada. Preferiblemente, una unidad de ejecución opera sobre operandos que se identifican por medio de una instrucción al realizar una función definida por código de operación sobre los operandos. Por ejemplo, una instrucción de ADD puede ser ejecutada por una unidad de ejecución 5057 sobre operandos encontrados en dos registros 5059 que se identifican por medio de campos de registro de la instrucción.

La unidad de ejecución 5057 realiza la adición aritmética sobre dos operandos y almacena el resultado en un tercer operando, en donde el tercer operando puede ser un tercer registro o uno de los dos registros de origen. La unidad de ejecución usa preferiblemente una unidad lógica aritmética (ALU, *arithmetic logic unit*) 5066 que puede realizar una diversidad de funciones lógicas tales como desplazamiento, rotación, Y, O y XOR así como una diversidad de funciones algebraicas incluyendo cualquiera de suma, resta, multiplicación, división. Algunas ALU 5066 están diseñadas para operaciones escalares y algunas para coma flotante. Los datos pueden ser Big Endian (en donde el byte menos significativo se encuentra en la dirección de byte más alta) o Little Endian (en donde el byte menos significativo se encuentra en la dirección de byte más baja), dependiendo de la arquitectura. La arquitectura z/Architecture de IBM es Big Endian. Los campos con signo pueden ser signo y magnitud, complemento a 1 o complemento a 2, dependiendo de la arquitectura. Un número de complemento a 2 es ventajoso ya que la ALU no necesita diseñar una capacidad de resta debido a que o bien un valor negativo o bien un valor positivo en el complemento a 2 requiere solo una adición dentro de la ALU. Los números se describen comúnmente en forma abreviada, en donde un campo de 12 bits define una dirección de un bloque de 4096 bytes y se describe, por lo general, como un bloque de 4 kbytes (kilobytes), por ejemplo.

Haciendo referencia a la figura 12B, la información de instrucción de ramificación para ejecutar una instrucción de ramificación se envía, por lo general, a una unidad de ramificación 5058 que a menudo emplea un algoritmo de predicción de ramificación, tal como una tabla de historia de ramificación 5082 para predecir el resultado de la ramificación antes de que se hayan completado otras operaciones condicionales. El objetivo de la instrucción de ramificación actual se extraerá y se ejecutará de forma especulativa antes de que se hayan completado las operaciones condicionales. Cuando se han completado las operaciones condicionales, las instrucciones de ramificación ejecutadas de forma especulativa o bien se completan o bien se descartan sobre la base de las condiciones de la operación condicional y el resultado especulado. Una instrucción de ramificación típica puede probar códigos de condición y ramificarse a una dirección objetivo si los códigos de condición cumplen con el requisito de ramificación en la instrucción de ramificación, una dirección objetivo se puede calcular sobre la base de diversos números incluyendo los encontrados en los campos de registro o un campo inmediato de la instrucción, por ejemplo. La unidad de ramificación 5058 puede emplear una ALU 5074 que tiene una pluralidad de circuitos de registro de entrada 5075, 5076, 5077 y un circuito de registro de salida 5080. La unidad de ramificación 5058 se puede comunicar con los registros generales 5059, la unidad de despacho de descodificación 5056 u otros circuitos 5073, por ejemplo.

La ejecución de un grupo de instrucciones se puede interrumpir por una diversidad de razones, incluyendo un cambio de contexto iniciado por un sistema operativo, un error o excepción de programa que da lugar a un cambio de contexto, una señal de interrupción de E / S que da lugar a un cambio de contexto o actividad de múltiples subprocesos de una pluralidad de programas (en un entorno de múltiples subprocesos), por ejemplo. Preferiblemente, una acción de cambio de contexto guarda información de estado acerca de un programa que se está ejecutando en la actualidad y entonces carga información de estado acerca de otro programa que se está invocando. La información de estado se puede guardar en registros de soporte físico o en memoria, por ejemplo. La información de estado comprende preferiblemente un valor de contador de programa que apunta a una siguiente instrucción que se va a ejecutar, códigos de condición, información de traducción de memoria y contenidos de registro diseñado. Una actividad de cambio de contexto puede ser puesta en práctica por circuitos de soporte físico,

programas de aplicación, programas de sistema operativo o código de soporte lógico inalterable (microcódigo, picocódigo o código interno con licencia (LIC, *licensed internal code*)) solos o en combinación.

Un procesador accede a operandos de acuerdo con métodos definidos por la instrucción. La instrucción puede proporcionar un operando inmediato usando el valor de una porción de la instrucción, puede proporcionar uno o más campos de registro que apuntan de forma explícita o bien a registros de propósito general o bien a registros de propósito especial (registros de coma flotante, por ejemplo). La instrucción puede usar registros implicados que se identifican por medio de un campo de código de operación como operandos. La instrucción puede usar sitios de memoria para operandos. Un sitio de memoria de un operando puede ser proporcionado por un registro, un campo inmediato o una combinación de registros y campo inmediato, tal como es ilustrado por el recurso de desplazamiento largo de arquitectura 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 conjuntamente para proporcionar la dirección del operando en memoria, por ejemplo). Sitio en la presente memoria implica, por lo general, un sitio en memoria principal (almacenamiento principal) a menos que se indique lo contrario.

Haciendo referencia a la figura 12C, un procesador accede al almacenamiento usando una unidad de carga / almacenamiento 5060. La unidad de carga / almacenamiento 5060 puede realizar una operación de carga al obtener la dirección del operando objetivo en memoria 5053 y cargar el operando en un registro 5059 u otro sitio de memoria 5053 o puede realizar una operación de almacenamiento al obtener la dirección del operando objetivo en memoria 5053 y almacenar los datos obtenidos de un registro 5059 u otro sitio de memoria 5053 en el sitio de operando objetivo en la memoria 5053. La unidad de carga / almacenamiento 5060 puede ser especulativa y puede acceder a la memoria en una secuencia que esta fuera de orden en relación con la secuencia de instrucciones, no obstante la unidad de carga / almacenamiento 5060 va a mantener la apariencia para los programas de que las instrucciones se ejecutaron en orden. Una unidad de carga / almacenamiento 5060 se puede comunicar con los registros generales 5059, la unidad de descodificación / despacho 5056, la interfaz de memoria caché / memoria 5053 u otros elementos 5083 y comprende diversos circuitos de registro, la ALU 5085 y la lógica de control 5090 para calcular direcciones de almacenamiento y para proporcionar una secuenciación de canalización para mantener las operaciones en orden. Algunas operaciones pueden estar fuera de orden pero la unidad de carga / almacenamiento proporciona una funcionalidad para hacer que parezca al programa que las operaciones fuera de orden se han realizado en orden, como es bien conocido en la técnica.

Preferiblemente, se hace referencia a menudo a las direcciones que “ve” un programa de aplicación como direcciones virtuales. Algunas veces se hace referencia a las direcciones virtuales como “direcciones lógicas” y “direcciones efectivas”. Estas direcciones virtuales son virtuales ya que son redirigidas al sitio de memoria físico por cualquiera de una diversidad de técnicas de traducción dinámica de direcciones (DAT, *dynamic address translation*), incluyendo, pero sin limitarse a, precisar simplemente una dirección virtual con un valor de desplazamiento, traducir la dirección virtual a través de una o más tablas de traducción, comprendiendo las tablas de traducción preferiblemente al menos una tabla de segmento y una tabla de página sola o en combinación, teniendo la tabla de segmento preferiblemente una entrada que apunta a la tabla de página. En la arquitectura z/Architecture, se proporciona una jerarquía de traducción que incluye una primera región de tabla, una segunda región de tabla, una tercera región de tabla, una tabla de segmento y una tabla de página opcional. A menudo, el desempeño de la traducción de direcciones se mejora al usar una memoria de almacenamiento intermedio de traducción anticipada (TLB, *translation lookaside buffer*) que comprende entradas que establecen una correspondencia de una dirección virtual con un sitio de memoria físico asociado. Las entradas se crean cuando la DAT traduce una dirección virtual usando las tablas de traducción. El uso subsiguiente de la dirección virtual puede usar entonces la entrada de la TLB rápida en lugar de los accesos de tabla de traducción secuencial lenta. El contenido de TLB puede ser gestionado por una diversidad de algoritmos de sustitución incluyendo LRU (*Least Recently used*, ultimo recientemente usado).

En el caso en el que el procesador es un procesador de un sistema de multiprocesador, cada procesador tiene la responsabilidad de mantener los recursos compartidos, tales como la E / S, la memoria caché, la TLB y la memoria, entrelazados por coherencia. Por lo general, se usarán tecnologías “espía” para mantener la coherencia de la memoria caché. En un entorno espía, cada línea de memoria caché se puede marcar como que se encuentra en cualquiera de un estado compartido, un estado exclusivo, un estado cambiado, un estado no valido y similares con el fin de facilitar la compartición.

Las unidades de E / S 5054 (figura 11) dotan al procesador de medios para acoplarse a dispositivos periféricos incluyendo cinta, disco, impresoras, pantallas y redes, por ejemplo. A menudo, las unidades de E / S son presentadas al programa informático por controladores de soporte lógico. En los grandes sistemas, tales como System Z de IBM®, los adaptadores de canal y adaptadores de sistema abierto son unidades de E / S del gran sistema que proporcionan las comunicaciones entre el sistema operativo y los dispositivos periféricos.

Además, otros tipos de entornos informáticos se pueden beneficiar de uno o más aspectos de la presente invención. Como un ejemplo, un entorno puede incluir un emulador (por ejemplo, soporte lógico u otros mecanismos de emulación), en el que se emula una arquitectura particular (incluyendo, por ejemplo, ejecución de instrucciones, funciones diseñadas tales como traducción de direcciones y registros diseñados) o un subconjunto de la misma (por ejemplo, en un sistema informático nativo que tiene un procesador y memoria). En tal entorno, una o más funciones de emulación en el emulador pueden implementar uno o más aspectos de la presente invención, a pesar de que un

ordenador que ejecuta el emulador puede tener una arquitectura diferente de las capacidades que se están emulando. Como un ejemplo, en el modo de emulación, la instrucción u operación específica que se está emulando se descodifica, y se construye 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 extracción de instrucciones para extraer instrucciones de la memoria y para proporcionar opcionalmente almacenamiento en memoria caché local para la instrucción extraída; una unidad de descodificación de instrucciones para recibir las instrucciones extraídas y para determinar el tipo de instrucciones que se han extraído y una unidad de ejecución de instrucciones para ejecutar las instrucciones. La ejecución puede incluir cargar datos en un registro de memoria; almacenar datos de vuelta a la memoria de un registro o realizar algún tipo de operación aritmética o lógica, según sea determinado por la unidad de descodificación. En un ejemplo, cada unidad se implementa en soporte lógico. Por ejemplo, las operaciones que son realizadas por las unidades se implementan como una o más subrutinas dentro del soporte lógico de emulador.

Más en particular, en un gran sistema, instrucciones de máquina diseñada son usadas por programadores, hoy en día habitualmente programadores de “C”, a menudo por medio de una aplicación informática. Estas instrucciones almacenadas en el medio de almacenamiento se pueden ejecutar de forma nativa en un servidor de IBM® de arquitectura z/Architecture o, como alternativa, en máquinas que ejecutan otras arquitecturas. Estas se pueden emular en los servidores de gran sistema existentes y futuros de IBM® y en otras máquinas de IBM® (por ejemplo, servidores de Power Systems o servidores de System x®). Se pueden ejecutar en máquinas que ejecutan Linux en una amplia diversidad de máquinas usando soporte físico fabricado por IBM®, Intel®, AMD® y otros. Además de la ejecución en ese soporte físico bajo una arquitectura z/Architecture, se puede usar Linux así como máquinas que usan emulación por Hercules, UMX o FSI (Fundamental Software, Inc.), en donde en general la ejecución se encuentra en un modo de emulación. En el modo de emulación, el soporte lógico de emulación es ejecutado por un procesador nativo para emular la arquitectura de un procesador emulado.

El procesador nativo ejecuta, por lo general, un soporte lógico de emulación que comprende o bien soporte lógico inalterable o bien un sistema operativo nativo para realizar la emulación del procesador emulado. El soporte lógico de emulación es responsable de extraer y de ejecutar instrucciones de la arquitectura de procesador emulada. El soporte lógico de emulación mantiene un contador de programa emulado para mantener un seguimiento de fronteras de instrucciones. El soporte lógico de emulación puede extraer una o más instrucciones de máquina emuladas de cada vez y convertir la una o más instrucciones de máquina emuladas en un grupo correspondiente de instrucciones de máquina nativas para su ejecución por el procesador nativo. Estas instrucciones convertidas se pueden guardar en memoria caché de tal forma que se puede llevar a cabo una conversión más rápida. No obstante, el soporte lógico de emulación va a mantener las reglas de arquitectura de la arquitectura de procesador emulada para asegurar que los sistemas operativos y aplicaciones escritas para el procesador emulado operen correctamente. Además, el soporte lógico de emulación va a proporcionar recursos que se identifican por medio de la arquitectura de procesador emulada incluyendo, pero sin limitarse a, registros de control, registros de propósito general, registros de coma flotante, función de traducción dinámica de direcciones incluyendo tablas de segmento y tablas de página, por ejemplo mecanismos de interrupción, mecanismos de cambio de contexto, relojes de hora del día (*Time of Day*, TOD) e interfaces diseñadas a los subsistemas de E / S de tal forma que un sistema operativo o un programa de aplicación diseñado para ejecutarse en el procesador emulado se pueda ejecutar en el procesador nativo que tiene el soporte lógico de emulación.

Una instrucción específica que se está emulando se descodifica y se llama a una subrutina para realizar la función de la instrucción individual. Una función de soporte lógico de emulación que emula una función de un procesador emulado se implementa, por ejemplo en un controlador o subrutina “C” o algún otro método para proporcionar un controlador para el soporte físico específico según esté dentro de la pericia de los expertos en la técnica después de la comprensión de la descripción de la forma de realización preferida. Diversas patentes de emulación de soporte físico y de soporte lógico incluyendo, pero sin limitarse a, la patente de EE. UU. 5.551.013, que lleva por título “*Multiprocessor for Hardware Emulation*”, de Beausoleil y col. y la patente de EE. UU. 6.009.261, que lleva por título “*Preprocessing of Stored Target Routines for Emulating Incompatible Instructions on a Target Processor*”, de Scalzi y col. y la patente de EE. UU. con n.º 5.574.873, que lleva por título “*Decoding Guest Instruction to Directly Access Emulation Routines that Emulate the Guest Instructions*”, de Davidian y col. y la patente de EE. UU. con n.º 6.308.255, que lleva por título “*Symmetrical Multiprocessing Bus and Chipset Used for Coprocessor Support Allowing Non-native Code to Run in a System*”, de Gorishek y col. y la patente de EE. UU. con n.º 6.463.582, que lleva por título “*Dynamic Optimizing Object Code Translator for Architecture Emulation and Dynamic Optimizing Object Code Translation Method*”, de Lethin y col. y la patente de EE. UU. con n.º 5.790.825, que lleva por título “*Method for Emulating Guest Instructions on a Host Computer Through Dynamic Recompilation of Host Instructions*”, de Eric Traut; y muchas otras, ilustran una diversidad de formas conocidas para obtener la emulación de un formato de instrucción diseñado para una máquina diferente para una máquina objetivo disponible para los expertos en la técnica.

En la figura 13, se proporciona un ejemplo de un sistema informático de anfitrión emulado 5092 que emula un sistema informático de anfitrión 5000’ de una arquitectura de anfitrión. En el sistema informático de anfitrión emulado 5092, el procesador de anfitrión (CPU) 5091 es un procesador de anfitrión emulado (o procesador de anfitrión virtual)

y comprende un procesador de emulación 5093 que tiene una arquitectura de conjunto de instrucciones nativas diferente de la del procesador 5091 del ordenador de anfitrión 5000'. El sistema informático de anfitrión emulado 5092 tiene una memoria 5094 a la que puede acceder el procesador de emulación 5093. En la forma de realización a modo de ejemplo, la memoria 5094 se divide en una porción de memoria informática de anfitrión 5096 y una porción de rutinas de emulación 5097. La memoria informática de anfitrión 5096 se encuentra disponible para programas del ordenador de anfitrión emulado 5092 de acuerdo con la arquitectura informática de anfitrión. El procesador de emulación 5093 ejecuta instrucciones nativas de un conjunto de instrucciones diseñadas de una arquitectura diferente de la del procesador emulado 5091, las instrucciones nativas obtenidas de la memoria de rutinas de emulación 5097 y puede acceder a una instrucción de anfitrión para la ejecución de un programa en la memoria informática de anfitrión 5096 al emplear una o más instrucciones obtenidas en una rutina de secuencia y acceso / descodificación que puede descodificar la instrucción o instrucciones de anfitrión a las que se accede para determinar una rutina de ejecución de instrucciones nativa para emular la función de la instrucción de anfitrión a la que se accede. Otros recursos que son definidos por la arquitectura del sistema informático de anfitrión 5000' pueden ser emulados por rutinas de recursos diseñados, incluyendo recursos tales como registros de propósito general, registros de control, traducción dinámica de direcciones y soporte de subsistema y / o memoria caché de procesador, por ejemplo. Las rutinas de emulación también pueden aprovechar funciones disponibles en el procesador de emulación 5093 (tales como registros generales y traducción dinámica de direcciones virtuales) para mejorar el desempeño de las rutinas de emulación. También se pueden proporcionar un soporte físico especial y motores de descarga para ayudar al procesador 5093 a emular la función del ordenador de anfitrión 5000'.

La terminología usada en la presente memoria es solo para fines de descripción de formas de realización particulares y no tiene por objeto ser limitante de la invención. Tal como se usan en la presente memoria, las formas singulares "un", "una" y "el / la" tienen por objeto incluir también las formas plurales, a menos que el contexto indique claramente lo contrario. Se entenderá además que los términos "comprende" y / o "que comprende", cuando se usan en la presente memoria descriptiva, especifican la presencia de características, números enteros, etapas, operaciones, elementos y / o componentes expuestos, pero no impiden la presencia o adición de otras una o más características, números enteros, etapas, operaciones, elementos, componentes y / o grupos de los mismos.

Las correspondientes estructuras, materiales, actos y equivalentes de todos los elementos de etapa más función o medios en las reivindicaciones en lo sucesivo, si los hay, tienen por objeto incluir cualquier estructura, material o acto para realizar la función en combinación con otros elementos reivindicados según se reivindique de forma específica. La descripción de uno o más aspectos de la presente invención se ha presentado para fines de ilustración y descripción, pero no tiene por objeto ser exhaustiva o estar limitada a la invención en la forma que se divulga. Muchas modificaciones y variaciones serán evidentes para los expertos en la técnica sin apartarse del alcance de la invención. La forma de realización se escogió y se describió con el fin de explicar del mejor modo los principios de la invención y la aplicación práctica, y para permitir que otros expertos en la técnica entiendan la invención para diversas formas de realización con diversas modificaciones según sean apropiadas para el uso particular que se contemple.

Anexo

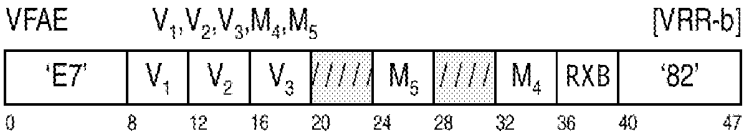
Recurso de vectores

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	Mnemónico	Características							Código de operación	Página
VECTOR DE ENCONTRAR CUALQUIER IGUAL	VFAE	VRR-b C*	VF		SP	Dv			E782	23-1
VECTOR DE ENCONTRAR CUALQUIER ELEMENTO IGUAL	VFEE	VRR-b C*	VF		SP	Dv			E780	23-2
VECTOR DE ENCONTRAR ELEMENTO NO IGUAL	VFENE	VRR-b C*	VF		SP	Dv			E781	23-3
VECTOR DE COMPARAR INTERVALO DE CADENA	VSTRC	VRR-d C*	VF		SP	Dv			E78A	23-4

VECTOR DE ENCONTRAR CUALQUIER IGUAL



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

Si la bandera de tipo de resultado (RT, *Result Type*) 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 del elemento correspondiente en el primer operando se establecen a uno, 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 7 del primer operando.

Cada instrucción tiene una sección de mnemónicos ampliados que describe mnemónicos ampliados recomendados y sus sintaxis de ensamblador de máquina correspondiente.

Nota de programación: Para todas las instrucciones que opcionalmente establecen el código de condición, el desempeño se puede deteriorar 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 encuentra ningún byte igual a cero o cero si se establece la bandera de búsqueda de cero, un índice igual al número de bytes en el vector se almacena en el byte siete del primer operando.

El campo M_4 especifica el control de tamaño de elemento (ES, *element size*). 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.

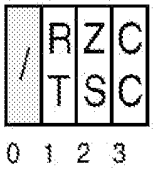
0 - Byte

1 - Media palabra

2 - Palabra

3 - 15 - Reservado

El campo M_5 tiene el siguiente formato:



Los bits del campo M_5 se definen tal como sigue:

• ***Tipo de resultado (RT):*** Si es cero, cada elemento resultante es una máscara de todas las comparaciones de intervalo en ese elemento. Si es uno, se almacena un índice de byte en el byte 7 del primer operando y se almacenan ceros en todos los otros elementos.

• ***Búsqueda de cero (ZS)****:* Si es uno, también se compara con cero cada elemento del segundo operando.

• ***Código de condición establecido (CC):*** Si es cero, el código de condición no se establece y permanece sin cambios. Si es uno, el código de condición se establece tal como se especifica en la siguiente sección.

Condiciones especiales

Se reconoce una excepción de especificación y no se lleva a cabo ninguna otra acción si tiene lugar cualquiera de los siguientes:

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

2. El bit 0 del campo M_5 no son cero.

Código de condición resultante:

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

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

0 Si se establece el bit ZS, no hubo coincidencias en un elemento indexado inferior a 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 coincidieron con al menos un elemento en el tercer operando.

3 Ningún elemento en el segundo operando coincide con elemento alguno en el tercer operando.

Excepciones de programa:

1. Datos con DXC FE, registro de vector

• Operación si el recurso de ampliación de vectores no está instalado

• Especificación (valor de ES reservado)

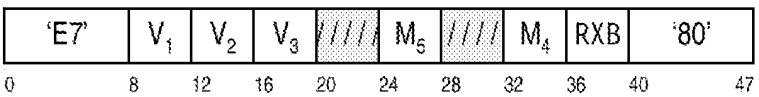
• Restricción de transacción

Mnemónicos ampliados:

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 $V_1, V_2, V_3, 1, (M_5 \mid X'3')$
VFAEZFS	V_1, V_2, V_3, M_5	VFAE $V_1, V_2, V_3, 2, (M_5 \mid X'3')$

VECTOR ENCONTRAR ELEMENTO IGUAL

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



Procediendo de izquierda a derecha, los elementos de número entero binario sin signo del segundo operando se comparan con los elementos de número entero binario sin signo correspondientes 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 7 del primer operando. Se almacenan ceros en los bytes restantes del primer operando. Si no se encuentra ningún byte que sea igual o cero si se establece la comparación de cero, entonces un índice igual al número de bytes en el vector se almacena en el byte siete del primer operando. Se almacenan ceros en los bytes restantes.

Si se establece el bite de búsqueda de cero (ZS) en el campo M_5, entonces cada elemento en el segundo operando también se compara como igualdad con cero. Si se encuentra un elemento cero en el segundo operando antes de que se encuentre cualquier que son iguales cualesquiera otros elementos del segundo y tercer operandos, el índice de byte del primer byte del elemento que se encuentra que es cero se almacena en el byte siete del primer operando y se almacenan ceros en todos los otros sitios de byte. Si la bandera de código de condición establecido (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 del vector. Si se especifica un valor, que reconoce una excepción de especificación.

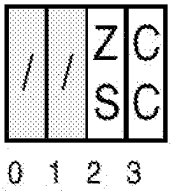
0 - Byte

1 - Media palabra

2 - Palabra

3 - 15 - Reservado

El campo M_5 tiene el siguiente formato:



Los bits del campo M_5 se definen tal como sigue:

• ***Reservado:*** Los bits 0 - 1 son reservados y deben ser cero. De lo contrario, se reconoce una excepción de especificación.

• ***Búsqueda de cero (ZS)****:* Si es uno, también se compara con cero cada elemento del segundo operando.

• ***Código de condición establecido (CC*)**: Si es cero, el código de condición permanece sin cambios. Si es uno, el código de condición se establece tal como se especifica en la siguiente sección.

Condiciones especiales

Se reconoce una excepción de especificación y no se lleva a cabo ninguna otra acción si tiene lugar cualquiera de los siguientes:

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 tal como sigue:

0 Si se establece el bit de comparación de cero, la comparación detectó un elemento cero en el segundo operando en un elemento con un índice pequeño que cualesquiera comparaciones iguales.

1 La comparación detectó una coincidencia entre los segundo y en algún elemento. Si el bit de comparación con cero s establecidos, esta coincidencia se presentó en un elemento con un índice menor o igual al elemento de comparación con cero.

2 --

3 Ningún elemento comparado es igual.

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

Excepciones de programa

• Datos con DXC FE, registro de vector

• Operación si el recurso de ampliación de vectores no está instalado

• Especificación (valor de ES reservado)

• Restricción de transacción

Mnemónicos ampliados

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 | X'1')
VFEEHS	V_1, V_2, V_3, M_5	VFEE V_1, V_2, V_3, 1, (M_5 | X'1')
VFEEFS	V_1, V_2, V_3, M_5	VFEE V_1, V_2, V_3, 2, (M_5 | X'1')
VFEEZB	V_1, V_2, V_3, M_5	VFEE V_1, V_2, V_3, 0, (M_5 | X'2')
VFEEZH	V_1, V_2, V_3, M_5	VFEE V_1, V_2, V_3, 1, (M_5 | X'2')
VFEEZF	V_1, V_2, V_3, M_5	VFEE V_1, V_2, V_3, 2, (M_5 | X'2')
VFEEZBS	V_1, V_2, V_3, M_5	VFEE V_1, V_2, V_3, 0, (M_5 | X'3')
VFEEZHS	V_1, V_2, V_3, M_5	VFEE V_1, V_2, V_3, 1, (M_5 | X'3')
VFEEZFS	V_1, V_2, V_3, M_5	VFEE V_1, V_2, V_3, 2, (M_5 | X'3')

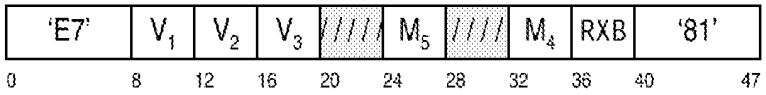
Notas de programación

1. Siempre se almacena un índice de byte en el primer operando para cualquier tamaño de elemento. Por ejemplo, si el tamaño de elemento se estableció a media palabra y la 2ª media palabra indexada se compara como igual, entonces se almacenaría un índice de byte de 4.

2. El tercer operando no debería contener elementos con un valor de cero. Si el tercer operando contiene de hecho un cero y coincide con un elemento cero en el segundo operando antes de cualesquiera otras comparaciones iguales, se establece el código de condición uno con independencia del establecimiento de los bits de comparación cero.

VECTOR ENCONTRAR ELEMENTO NO IGUAL

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



Procediendo de izquierda a derecha, los elementos de número entero binario sin signo del segundo operando se comparan con los elementos de número entero binario sin signo correspondientes del tercer operando. Si dos elementos no son iguales, el índice de byte del elemento más a la izquierda no igual se coloca en el byte 7 del primer operando y se almacenan ceros a todos los otros bytes. Si el bit de código de condición establecido (CC) en el campo M_5 se establece a uno, el código de condición se establece para indicar qué operando fue mayor. Si todos los elementos fueron iguales, entonces un índice de byte igual al tamaño de vector se coloca en el byte siete del primer operando y se colocan ceros en todos los otros sitios de byte. Si el bit CC es uno, se establece el código de condición tres.

Si el bit de búsqueda de cero (ZS) se establece en el campo M_5, cada elemento en el segundo operando también se compara como igualdad con cero. Si se encuentra un elemento cero en el segundo operando antes de que se encuentre que son no iguales cualesquiera otros elementos del segundo operando, el índice de byte del primer byte del elemento que se encontró 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 del elemento (ES). El control de ES específica el tamaño de los elementos en los operandos de registro de vector. Si se específica un valor reservado, se reconoce una excepción de especificación.

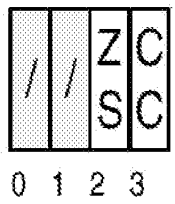
0 - Byte

1 - Media palabra

2 - Palabra

3 - 15 - Reservado

El campo M_5 tiene el siguiente formato:



Los bits del campo M_5 se definen tal como sigue:

• ***Búsqueda de cero (ZS):*** Si es uno, también se compara con cero cada elemento del segundo operando.

• ***Código de condición establecido (CC):*** Si es cero, el código de condición no se establece y permanece sin cambios. Si es uno, el código de condición se establece tal como se especifica en la siguiente sección.

Condiciones especiales

Se reconoce una excepción de especificación y no se lleva a cabo ninguna otra acción si tiene lugar cualquiera de los siguientes:

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 tal como sigue:

0 Si se establece el bit de comparación de cero, la comparación detectó un elemento cero en ambos operandos en un elemento indexado inferior a cualquier otra comparación no igual.

1 Se detectó una falta de coincidencia de elementos y el elemento en VR2 es menor que el elemento en VR3.

2 Se detectó una falta de coincidencia de elementos y el elemento en VR2 es mayor que en elemento en VR3.

3 Todos los elementos se comparan como iguales y, si se establece el bit de comparación cero, no se encontraron elementos cero en el segundo operando.

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

Excepciones de programa

• Datos con DXC FE, registro de vector

• Operación si el recurso de ampliación de vectores no está instalado

• Especificación (valor de ES reservado)

Restricción de transacción

Mnemónicos ampliados

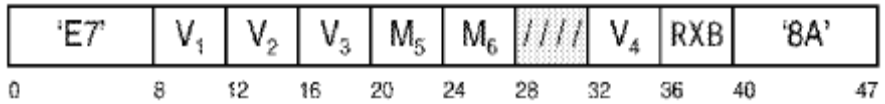
VFENEB V_1, V_2, V_3, M_5 VFENE $V_1, V_2, V_3, 0, M_5$

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 | X'1')
VFENEHS	V_1, V_2, V_3, M_5	VFENE V_1, V_2, V_3, 1, (M_5 | X'1')
VFENEFS	V_1, V_2, V_3, M_5	VFENE V_1, V_2, V_3, 2, (M_5 | X'1')
VFENEZB	V_1, V_2, V_3, M_5	VFENE V_1, V_2, V_3, 0, (M_5 | X'2')
VFENEZH	V_1, V_2, V_3, M_5	VFENE V_1, V_2, V_3, 1, (M_5 | X'2')
VFENEZF	V_1, V_2, V_3, M_5	VFENE V_1, V_2, V_3, 2, (M_5 | X'2')
VFENEZBS	V_1, V_2, V_3, M_5	VFENE V_1, V_2, V_3, 0, (M_5 | X'3')
VFENEZHS	V_1, V_2, V_3, M_5	VFENE V_1, V_2, V_3, 1, (M_5 | X'3')
VFENEZFS	V_1, V_2, V_3, M_5	VFENE V_1, V_2, V_3, 2, (M_5 | X'3')

VECTOR COMPARAR INTERVALO DE CADENAS

VSTRC $V_1, V_2, V_3, V_4, M_5, M_6$ [VRR-d]



Procediendo de izquierda a derecha, los elementos de número entero binario sin signo en el segundo operando se comparan con intervalos de valores definidos por pares de elementos par - impar en el tercer y el cuarto operandos. Combinados con valores de control del cuarto operando definen el intervalo de comparaciones que se van a realizar. Si un elemento coincide con cualquiera de los intervalos especificados por el tercer y el 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 bit del elemento en el primer operando que se corresponde con el elemento que se está comparando en el segundo operando se establecen a uno si el elemento coincide con cualquiera de los intervalos, de lo contrario se establecen a cero.

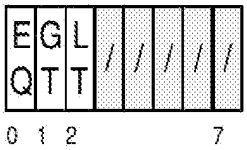
Si la bandera de tipo de resultado (RT) en el campo M_6 se establece a uno, el índice de byte del primer elemento en el segundo operando que coincide con cualquiera de los intervalos especificados por el tercer y el cuarto operandos o una comparación de cero, si la bandera ZS se establece a uno, se coloca en el byte siete del primer operando y se almacenan ceros en los bytes restantes. Si no coincide elemento alguno, entonces un índice igual al número de bytes en un vector se coloca 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 con cero de los elementos de segundo operando a los intervalos proporcionados por el tercero y el cuarto operandos. Si una comparación de cero en un elemento indexado inferior a cualquier otra comparación verdadera, entonces el código de condición se establece a cero.

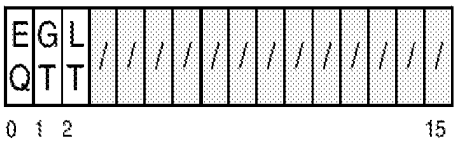
Los operandos contienen elementos del tamaño especificado por el control de tamaño de elemento en el campo M_5.

Los elementos de cuarto operando tienen el siguiente formato:

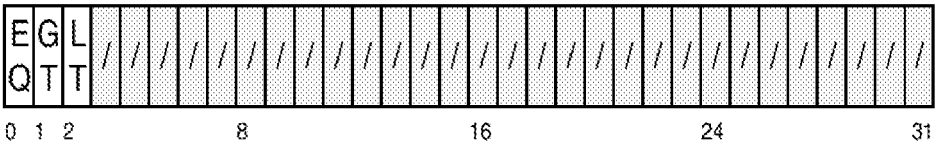
Si ES es igual a 0:



Si ES es igual a 1:



Si ES es igual a 2:



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

- ***Igual (EQ):*** Cuando se hace una comparación de 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 son reservados y deben ser cero para asegurar la compatibilidad futura.

Los bits de control se pueden usar 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 de elemento (ES). El control de ES específica el tamaño de los elementos en los operandos de registro del vector. Si se especifica un valor reservado, se reconoce una excepción de especificación.

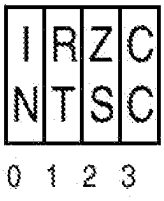
0 - Byte

1 - Media palabra

2 - Palabra

3 - 15 - Reservado

El campo M_6 tiene el siguiente formato:



Los bits del campo M_6 se definen tal como sigue:

- ***Resultado invertido (IN)****:* Si es cero, la comparación procede 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 intervalos.
- ***Tipo de resultado (RT):*** Si es cero, cada elemento resultante es una máscara de todas las comparaciones de intervalo en ese elemento. Si es uno, se almacena un índice en el byte siete del primer operando. Se almacenan ceros en los bytes restantes.
- ***Búsqueda de cero (ZS)****:* Si es uno, también se compara con cero cada elemento del segundo operando.
- ***Código de condición establecido (CC)****:* Si es cero, el código de condición no se establece y permanece sin cambios. Si es uno, el código de condición se establece tal como se especifica en la siguiente sección.

Condiciones especiales

Se reconoce una excepción de especificación y no se lleva a cabo ninguna otra acción si tiene lugar cualquiera de los siguientes:

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

Código de condición resultante:

0 Si ZS = 1 y se encuentra un cero en un elemento indexado inferior a cualquier comparación.

1 Comparación encontrada.

2 --

3 Ninguna comparación encontrada.

Excepciones de programa

• Datos con DXC FE, registro de vector

• Operación si el recurso de ampliación de vectores no está instalado

• Especificación (valor de ES reservado)

• Restricción de transacción

Mnemónicos ampliados

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 | 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 | 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 | 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 | 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 | 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 | 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 | 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 | 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 | X'3')

	VR2 →	A	b	C	d	e	F	1	2
GE	A	T	T	T	T	T	T	F	F
LE	Z	T	F	T	F	F	T	F	F
GE	a	F	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	T	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

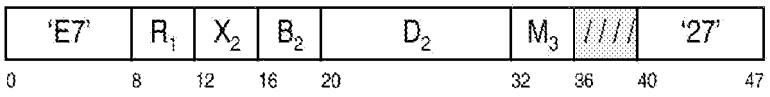
ES = 1, ZS = 0

VR1 (a) resultados con RT = 0

VR1 (b) resultados con RT = 1

RECUENTO DE CARGA A FRONTERA DE BLOQUE

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



Un número entero binario sin signo de 32 bits que contiene el número de bytes que es posible cargar desde el segundo sitio de operando sin cruzar una frontera de bloque especificada, limitado como máximo a dieciséis, se coloca en el primer operando.

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

La dirección del segundo operando no se usa para direccionar datos.

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

Código de frontera

0 64 bytes

1 128 bytes

2 256 bytes

3 512 bytes

4 1 kbyte

5 2 kbytes

6 4 kbytes

7 - 15 Reservado

Código de condición resultante:

0 Operando uno si es cero

1 --

2 --

3 Operando uno menos que dieciséis

Código de condición resultante:

Excepciones de programa:

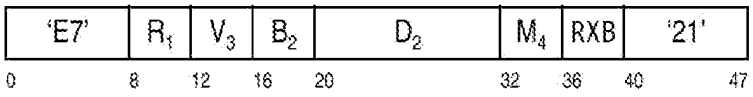
• Operación si el recurso de ampliación de vectores no está instalado.

• Especificación.

Nota de programación: Se espera que el recuento de carga a frontera de bloque se use en conjunción con vector de carga a frontera de bloque para determinar el número de bytes que se cargaron.

VECTOR DE CARGA DE GR DESDE ELEMENTO DE 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 de ES en el campo M_4 e indexado por la dirección del segundo operando se coloca en el primer sitio de operando. El tercer operando es un registro de vector. El primer operando es un registro general. Si el índice especificado por la dirección del segundo operando es mayor que el elemento de número más alto en el tercer operando, del tamaño de elemento especificado, los datos en el primer operando no se pueden predecir.

Si el elemento de registro de vector es más pequeño que una doble palabra, el elemento se alinea a la derecha en el registro general de 64 bits y los bits restantes se llenan con ceros.

La dirección del segundo operando no se usa para direccionar datos; en su lugar, los 12 bits más a la derecha de la dirección se usan 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 del vector. Si se especifica un valor reservado, se reconoce una excepción de especificación.

0 - Byte

1 - Media palabra

2 - Palabra

3 - Doble palabra

4 - 15 - Reservado sin cambios

Código de condición resultante: El código permanece sin cambios.

Excepciones de programa

• Datos con DXC FE, registro de vector

• Operación si el recurso de ampliación de vectores no está instalado

• Especificación (valor de ES reservado)

• Restricción de transacción

Mnemónicos ampliados

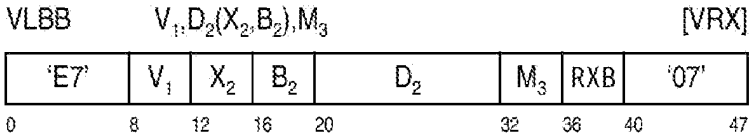
VLGVB R1, V3, D2(B2) VLGV R1, V3, D2 (B2), 0

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

VECTOR DE CARGA A FRONTERA 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 frontera, el resto del primer operando no se puede predecir. No se reconocen excepciones de acceso en bytes no cargados.

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

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

Código de frontera

0 64 bytes

1 128 bytes

2 256 bytes

3 512 bytes

4 1 kbyte

5 2 kbytes

6 4 kbytes

7 - 15 Reservado

Código de condición resultante: El código permanece sin cambios.

Excepciones de programa*:*

• Acceso (extracción, operando 2)

• Datos con DXC FE, registro de vector

• Operación si el recurso de ampliación de vectores no está instalado.

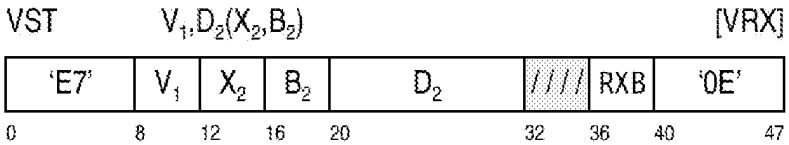
• Especificación (código de frontera de bloque reservado).

• Restricción de transacción.

Notas de programación:

1. En ciertas circunstancias, los datos se pueden cargar más allá de la frontera de bloque. No obstante, esto solo tendrá lugar si no hay excepciones de acceso en esos datos.

VECTOR GUARDAR



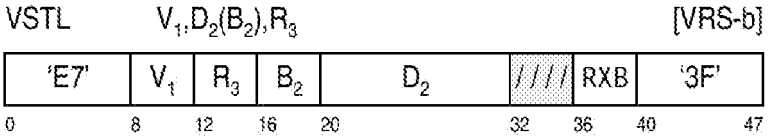
El valor de 128 bits en el primer operando se almacena en el sitio de almacenamiento especificado por el segundo operando. El desplazamiento para VST se trata como un número entero sin signo de 12 bits.

Código de condición resultante: El código permanece sin cambios.

Excepciones de programa:

- Acceso (extracción, operando 2)
- Datos con DXC FE, registro de vector
- Operación si el recurso de ampliación de vectores no está instalado.
- Restricción de transacción.

VECTOR DE ALMACENAR CON LONGITUD



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

Las excepciones de acceso solo se reconocen en los bytes almacenados.

El desplazamiento para el vector de almacenar con longitud se trata como un número entero sin signo de 12 bits.

Código de condición resultante: El código permanece sin cambios.

Excepciones de programa:

- Acceso (almacenar, operando 2)
- Datos con DXC FE, registro de vector
- Operación si el recurso de ampliación de vectores no está instalado.
- Restricción de transacción

Descripción de 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 designados de registro de vector. Los bits para las designaciones de registro no especificadas por la instrucción son reservados y, por lo tanto, se deberían establecer a cero; de lo contrario, el programa puede no operar de forma 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 tal como sigue:

0. Bit más significativo para la designación de registro de vector en los bits 8 - 11 de la instrucción.

1. Bit más significativo para la designación de registro de vector en los bits 12 - 15 de la instrucción.

2. Bit más significativo para la designación de registro de vector en los bits 16 - 19 de la instrucción.

3. Bit más significativo para la designación de registro de vector en los bits 32 - 35 de la instrucción.

Control de habilitación de vector

Las instrucciones y registros de vector solo se pueden usar si tanto el control de habilitación de vector (el bit 46) como el control de registro de AFP (el bit 45) en el registro de control cero se establecen a 1. Si se instala el recurso de vectores y se ejecuta una instrucción de vector sin los bits de habilitación establecidos, se reconoce una excepción de datos con el hexadecimal DXC FE. Si no se instala el recurso de vectores, se reconoce una excepción de operación.

REIVINDICACIONES

1. Un producto de programa informático que comprende un medio de almacenamiento legible por ordenador legible por un circuito de procesamiento y que almacena código de programa para la ejecución por el circuito de procesamiento para realizar un método que comprende:

obtener, mediante un procesador, una instrucción de máquina para la ejecución, comprendiendo la instrucción de máquina:

al menos un campo de código de operación (302) para proporcionar un código de operación, identificando el código de operación una operación de carga a frontera de bloque;

un campo de registro (304) que se va a usar para designar un registro, comprendiendo el registro un primer operando;

al menos un campo (306, 308, 310) que se va a usar para indicar una dirección de partida en memoria principal; y

ejecutar la instrucción de máquina, comprendiendo la ejecución:

calcular (410) la dirección de partida (412) a partir del al menos un campo de la instrucción de máquina, indicando la dirección de partida un sitio en memoria principal del que va a comenzar la carga en el primer operando;

calcular (420) una dirección de fin (422) en memoria principal en la que se va a detener la carga en el primer operando, y cargar (430 una cantidad variable de datos desde memoria principal en el primer operando, comprendiendo la carga cargar desde memoria principal comenzando en la dirección de partida en memoria principal y terminando en la dirección de fin en memoria principal, en donde la cantidad variable de datos se carga desde memoria principal en el primer operando sin cruzar una frontera de memoria principal designada; caracterizado por:

comprender adicionalmente la instrucción de máquina un indicador de tamaño de frontera de bloque (312) para indicar un tamaño de frontera de bloque de un bloque de memoria principal; y

en donde calcular la dirección de fin incluye usar la siguiente ecuación: dirección de fin = mínimo de (dirección de partida + (tamaño de frontera - (dirección de partida Y NO máscara de frontera)), dirección de partida + tamaño de registro), en donde el tamaño de frontera es el tamaño de frontera de bloque, la máscara de frontera es igual a 0 - tamaño de frontera, y el tamaño de registro es una longitud especificada del registro que se designa en el campo de registro de la instrucción de máquina.

2. El producto de programa informático de la reivindicación 1, en donde el al menos un campo comprende un campo de desplazamiento, un campo de base y un campo de índice, el campo de base y el campo de índice para localizar registros generales que tienen contenidos que se van a añadir a contenidos del campo de desplazamiento para formar la dirección de partida, y en donde la instrucción de máquina comprende adicionalmente un campo de máscara, especificando el campo de máscara el indicador de tamaño de frontera de bloque.

3. El producto de programa informático de la reivindicación 2, en donde el tamaño de frontera de bloque es un tamaño de frontera de bloque de una pluralidad de tamaños de frontera de bloque que pueden ser especificados por el campo de máscara.

4. El producto de programa informático de cualquier reivindicación anterior, en donde la carga comprende uno de: cargar el primer operando de izquierda a derecha, o cargar el primer operando de derecha a izquierda.

5. El producto de programa informático de la reivindicación 4, en donde una dirección de la carga se proporciona en tiempo de ejecución.

6. El producto de programa informático de cualquier reivindicación anterior, en donde la instrucción de máquina comprende adicionalmente un campo de ampliación que se va a usar en la designación de uno o más registros, y en donde el campo de registro se combina con al menos una porción del campo de ampliación para designar el registro.

7. El producto de programa informático de cualquier reivindicación anterior, en donde la carga del primer operando comienza en un byte indexado 0 del primer operando, y continúa hasta un índice de byte máximo del registro.

8. Un sistema informático que comprende:

una memoria; y

un procesador en comunicación con la memoria, en donde el sistema informático está configurado para realizar un método, comprendiendo dicho método:

obtener, mediante el procesador, una instrucción de máquina para la ejecución, comprendiendo la instrucción de máquina:

al menos un campo de código de operación (302) para proporcionar un código de operación, identificando el código de operación una operación de carga a frontera de bloque;

un campo de registro (304) que se va a usar para designar un registro, comprendiendo el registro un primer operando;

al menos un campo (306, 308, 310) que se va a usar para indicar una dirección de partida en memoria principal; y

ejecutar la instrucción de máquina, comprendiendo la ejecución:

calcular (410) la dirección de partida (412) a partir del al menos un campo de la instrucción de máquina, indicando la dirección de partida un sitio en memoria principal del que va a comenzar la carga en el primer operando;

calcular (420) una dirección de fin (422) en memoria principal en la que se va a detener la carga en el primer operando;

cargar (430) una cantidad variable de datos desde memoria principal en el primer operando, comprendiendo la carga cargar desde memoria principal comenzando en la dirección de partida en memoria principal y terminando en la dirección de fin en memoria principal, en donde la cantidad variable de datos se carga desde memoria principal en el primer operando sin cruzar una frontera de memoria principal designada; caracterizado por:

comprender adicionalmente la instrucción de máquina un indicador de tamaño de frontera de bloque (312) para indicar un tamaño de frontera de bloque de un bloque de memoria principal; y

en donde calcular la dirección de fin incluye usar la siguiente ecuación: dirección de fin = mínimo de (dirección de partida + (tamaño de frontera - (dirección de partida Y NO máscara de frontera)), dirección de partida + tamaño de registro), en donde el tamaño de frontera es el tamaño de frontera de bloque, la máscara de frontera es igual a 0 - tamaño de frontera, y el tamaño de registro es una longitud especificada del registro que se designa en el campo de registro de la instrucción de máquina.

9. El sistema informático de la reivindicación 8, en donde el al menos un campo comprende un campo de desplazamiento, un campo de base y un campo de índice, el campo de base y el campo de índice para localizar registros generales que tienen contenidos que se van a añadir a contenidos del campo de desplazamiento para formar la dirección de partida, y en donde la instrucción de máquina comprende adicionalmente un campo de máscara, especificando el campo de máscara el indicador de tamaño de frontera de bloque.

10. El sistema informático de cualquiera de las reivindicaciones 8 a 9, en donde la carga comprende uno de: cargar el primer operando de izquierda a derecha, o cargar el primer operando de derecha a izquierda.

11. El sistema informático de la reivindicación 10, en donde una dirección de la carga se proporciona en tiempo de ejecución.

12. El sistema informático de la reivindicación 8, en donde la instrucción de máquina comprende adicionalmente un campo de ampliación que se va a usar en la designación de uno o más registros, y en donde el campo de registro se combina con al menos una porción del campo de ampliación para designar el registro.

13. Un método que comprende:

obtener, mediante un procesador, una instrucción de máquina para la ejecución, comprendiendo la instrucción de máquina:

al menos un campo de código de operación (302) para proporcionar un código de operación, identificando el código de operación una operación de carga a frontera de bloque;

un campo de registro (304) que se va a usar para designar un registro, comprendiendo el registro un primer operando;

al menos un campo (306, 308, 310) que se va a usar para indicar una dirección de partida en memoria principal; y

ejecutar la instrucción de máquina, comprendiendo la ejecución:

calcular (410) la dirección de partida (412) a partir del al menos un campo de la instrucción de máquina, indicando la dirección de partida un sitio en memoria principal del que va a comenzar la carga en el primer operando;

calcular (420) una dirección de fin (422) en memoria principal en la que se va a detener la carga en el primer operando; y

cargar (430) una cantidad variable de datos desde memoria principal en el primer operando, comprendiendo la carga cargar desde memoria principal comenzando en la dirección de partida en memoria principal y terminando en la dirección de fin en memoria principal, en donde la cantidad variable de datos se carga desde memoria principal en el primer operando sin cruzar una frontera de memoria principal designada; caracterizado por:

comprender adicionalmente la instrucción de máquina un indicador de tamaño de frontera de bloque (312) para indicar un tamaño de frontera de bloque de un bloque de memoria principal; y

en donde calcular la dirección de fin incluye usar la siguiente ecuación: dirección de fin = mínimo de (dirección de partida + (tamaño de frontera - (dirección de partida Y NO máscara de frontera)), dirección de partida + tamaño de registro), en donde el tamaño de frontera es el tamaño de frontera de bloque, la máscara de frontera es igual a 0 - tamaño de frontera, y el tamaño de registro es una longitud especificada del registro que se designa en el campo de registro de la instrucción de máquina.

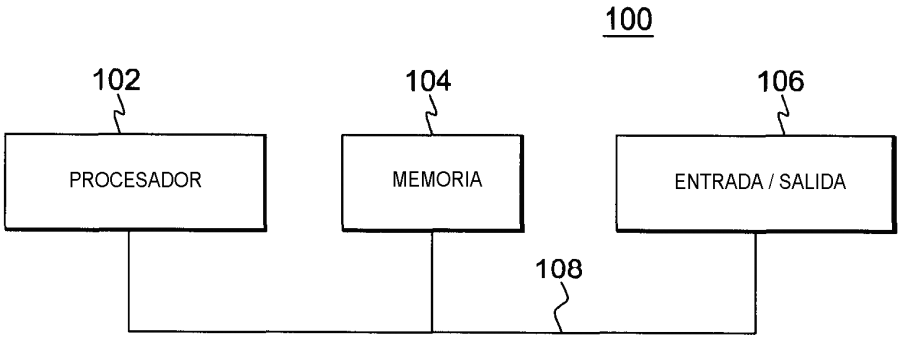


FIG. 1

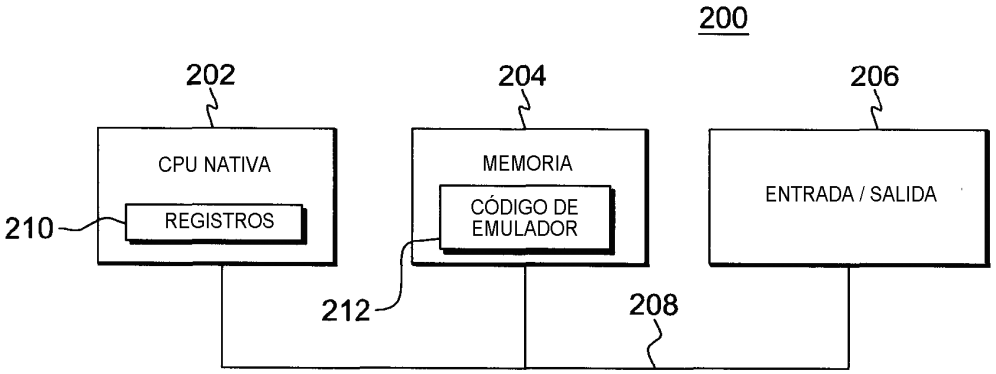


FIG. 2A

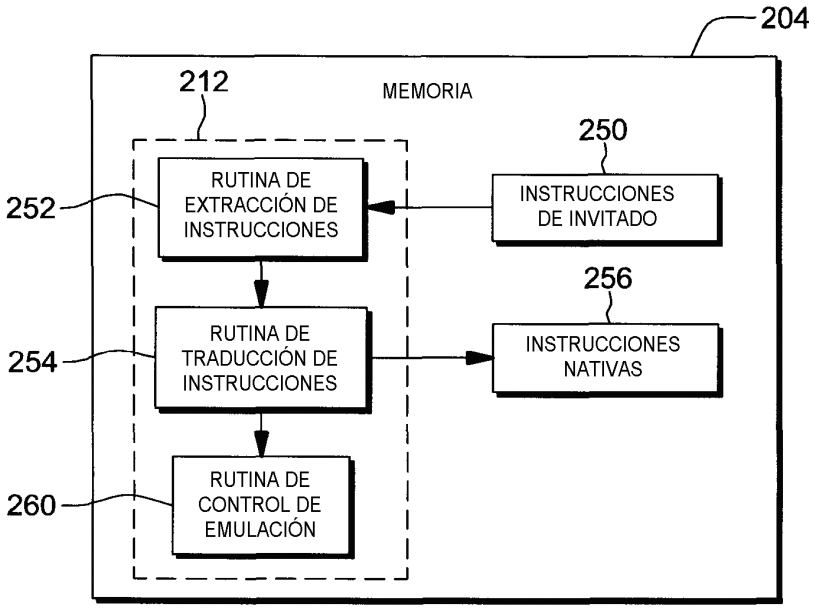


FIG. 2B

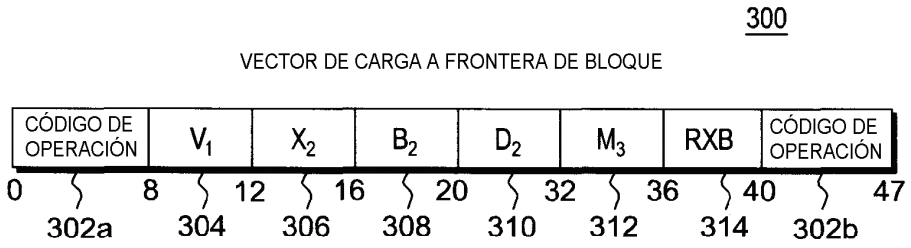

300
VECTOR DE CARGA A FRONTERA DE BLOQUE
CÓDIGO DE OPERACIÓN
V1
X2
B2
D2
M3
RXB
CÓDIGO DE OPERACIÓN
0
8
12
16
20
32
36
40
47
302a
304
306
308
310
312
314
302b

FIG. 3

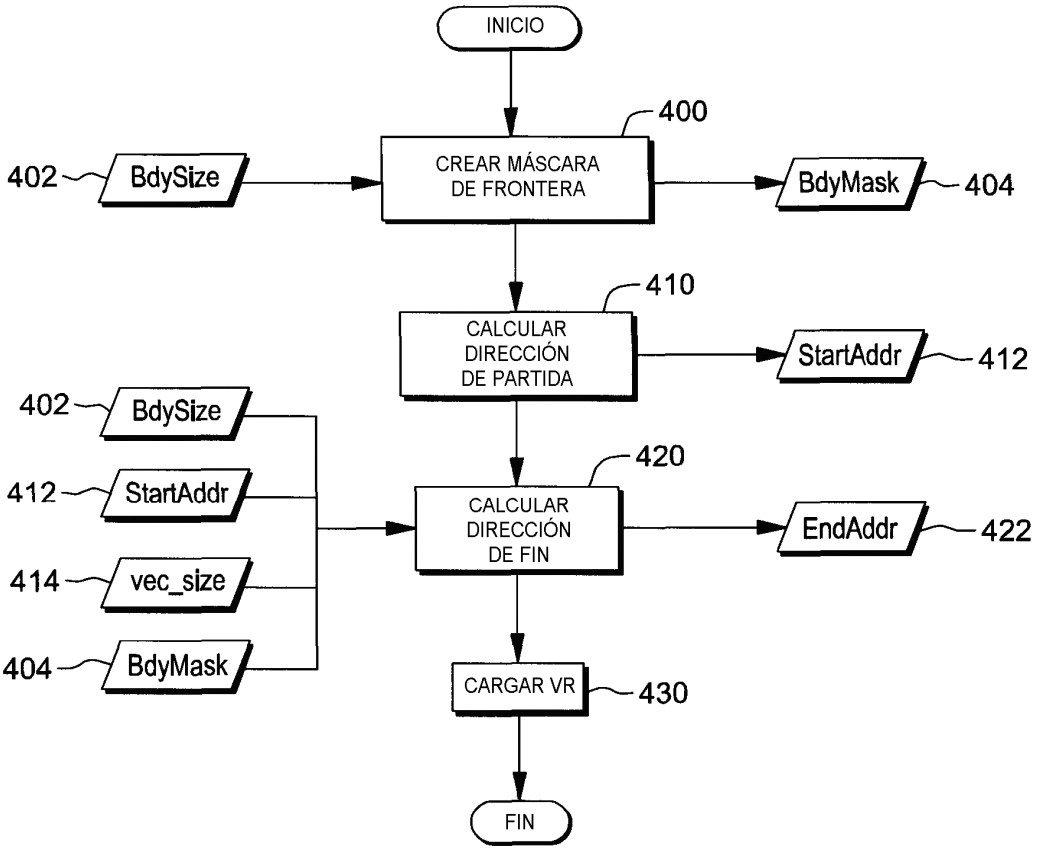

INICIO
400
402
BdySize
CREAR MÁSCARA DE FRONTERA
BdyMask
404
410
CALCULAR DIRECCIÓN DE PARTIDA
StartAddr
412
402
BdySize
412
StartAddr
420
CALCULAR DIRECCIÓN DE FIN
EndAddr
422
414
vec_size
404
BdyMask
CARGAR VR
430
FIN

FIG. 4

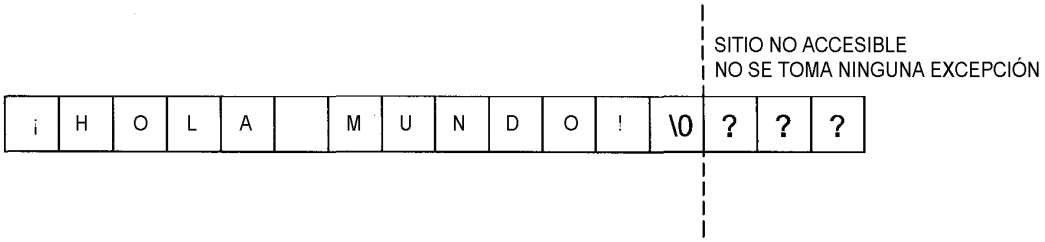

SITIO NO ACCESIBLE
NO SE TOMA NINGUNA EXCEPCIÓN
¡ H O L A M U N D O ! \0 ? ? ?

FIG. 5

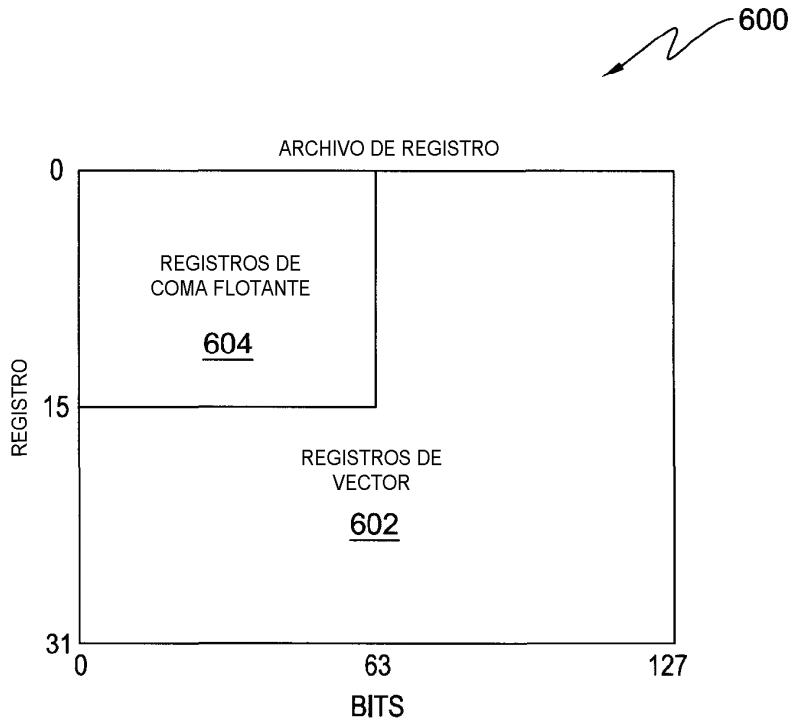

600
ARCHIVO DE REGISTRO
0
REGISTROS DE
COMA FLOTANTE
604
REGISTRO
15
REGISTROS DE
VECTOR
602
31
0
63
127
BITS

FIG. 6


PRODUCTO
DE PROGRAMA
INFORMÁTICO
700
704
LÓGICA DE CÓDIGO
DE PROGRAMA
MEDIO DE
ALMACENAMIENTO
LEGIBLE POR
ORDENADOR
702

FIG. 7

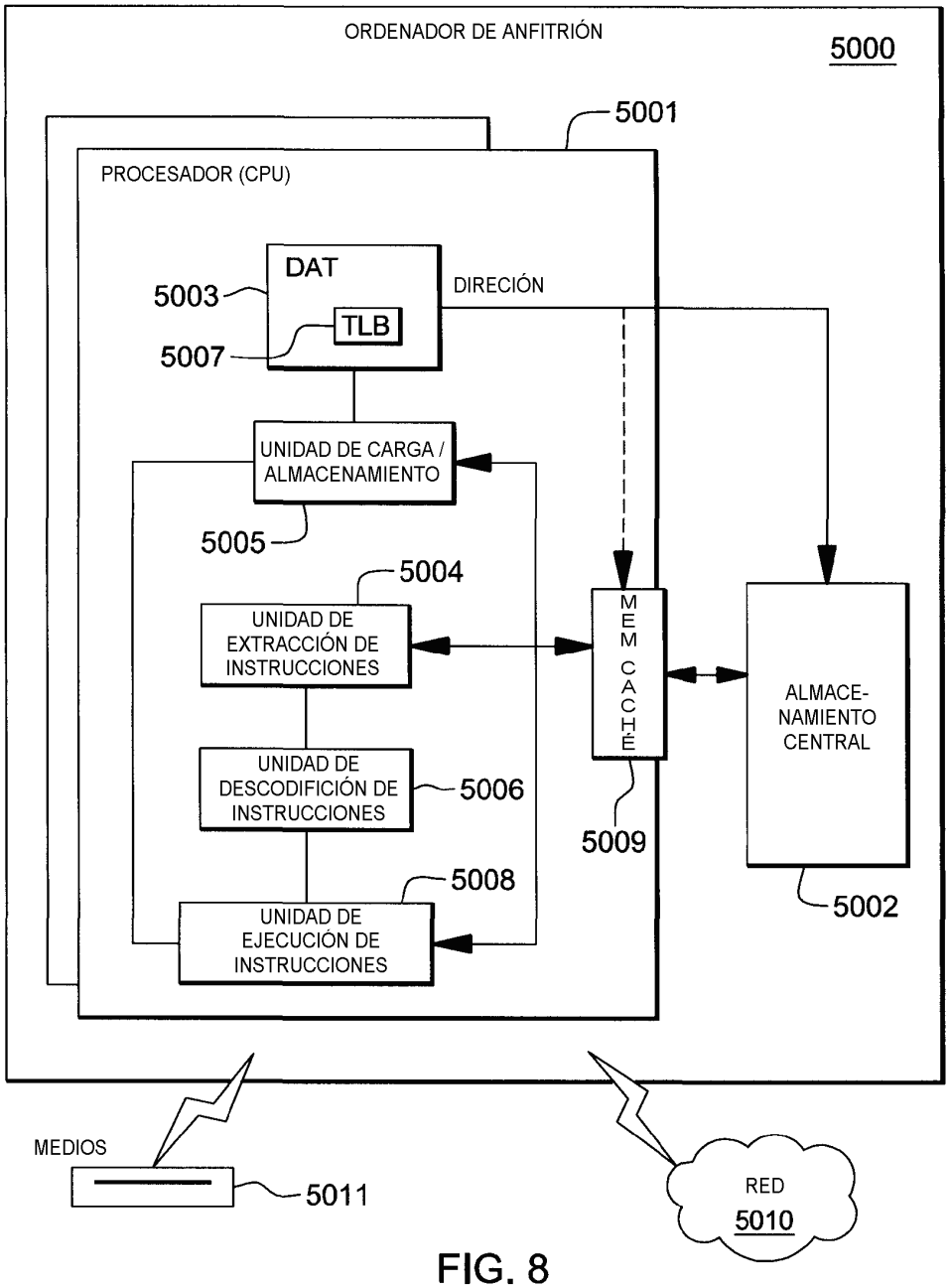

ORDENADOR DE ANFITRIÓN
5000
5001
PROCESADOR (CPU)
DAT
5003
TLB
5007
DIRECIÓN
UNIDAD DE CARGA / ALMACENAMIENTO
5005
5004
UNIDAD DE EXTRACCIÓN DE INSTRUCCIONES
UNIDAD DE DESCODIFICIÓN DE INSTRUCCIONES
5006
5008
UNIDAD DE EJECUCIÓN DE INSTRUCCIONES
MEM CACHÉ
5009
ALMACE-NAMIENTO CENTRAL
5002
MEDIOS
5011
RED
5010

FIG. 8

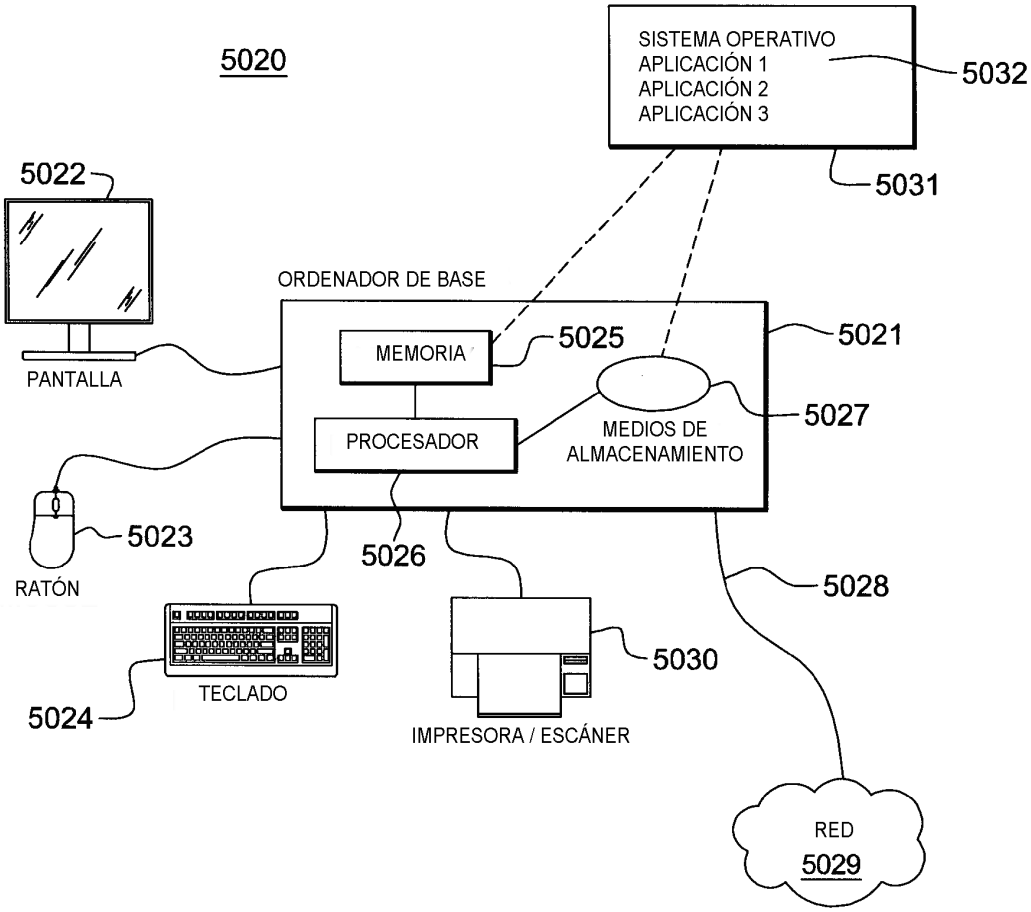

5020
SISTEMA OPERATIVO
APLICACIÓN 1
APLICACIÓN 2
APLICACIÓN 3
5032
5031
5022
ORDENADOR DE BASE
MEMORIA
5025
5021
PANTALLA
PROCESADOR
MEDIOS DE ALMACENAMIENTO
5027
5023
RATÓN
5026
5028
TECLADO
5024
5030
IMPRESORA / ESCÁNER
RED
5029

FIG. 9

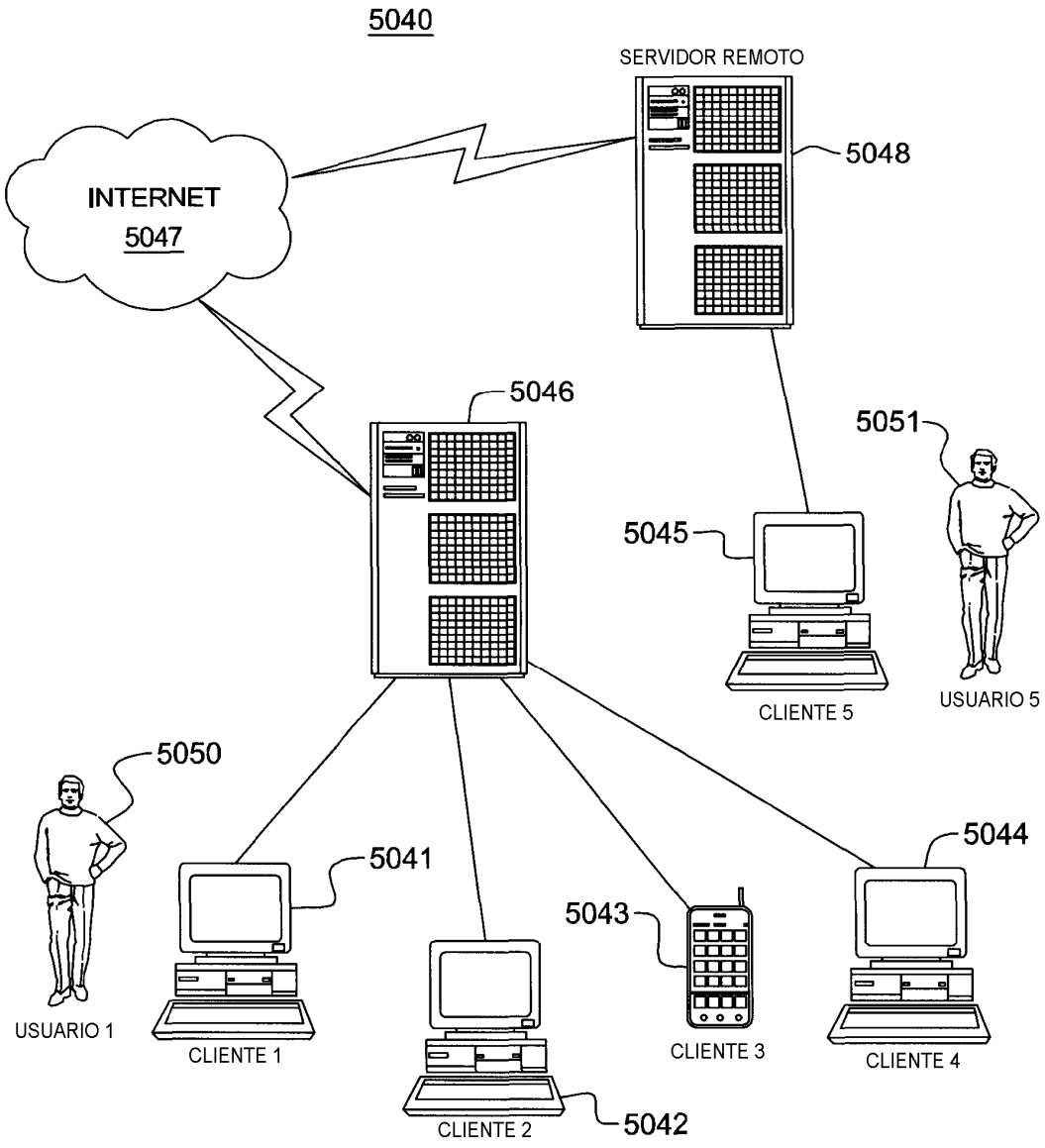

5040
SERVIDOR REMOTO
5048
INTERNET
5047
5046
5051
5045
CLIENTE 5
USUARIO 5
5050
5041
5044
5043
USUARIO 1
CLIENTE 1
CLIENTE 3
CLIENTE 4
CLIENTE 2
5042

FIG. 10

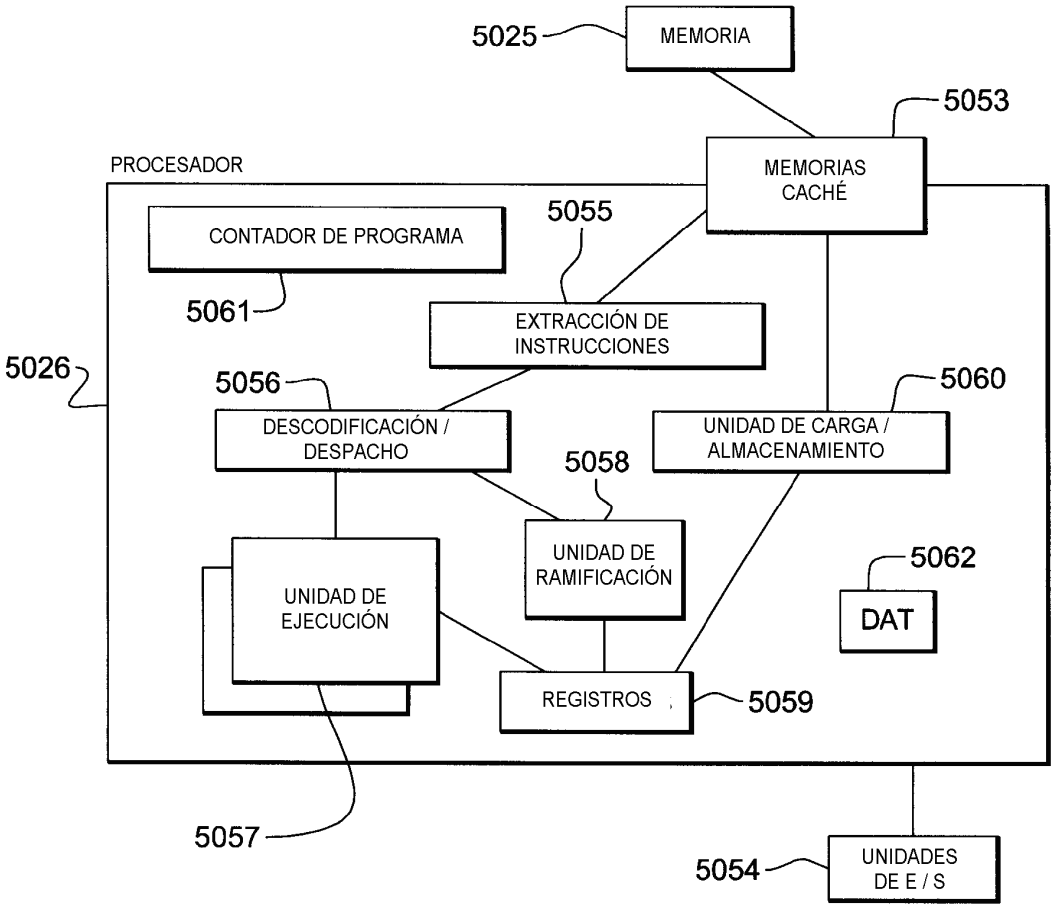

5025
MEMORIA
5053
MEMORIAS CACHÉ
PROCESADOR
CONTADOR DE PROGRAMA
5055
5061
EXTRACCIÓN DE INSTRUCCIONES
5026
5056
5060
DESCODIFICACIÓN / DESPACHO
UNIDAD DE CARGA / ALMACENAMIENTO
5058
UNIDAD DE RAMIFICACIÓN
5062
UNIDAD DE EJECUCIÓN
DAT
REGISTROS
5059
5057
5054
UNIDADES DE E / S

FIG. 11

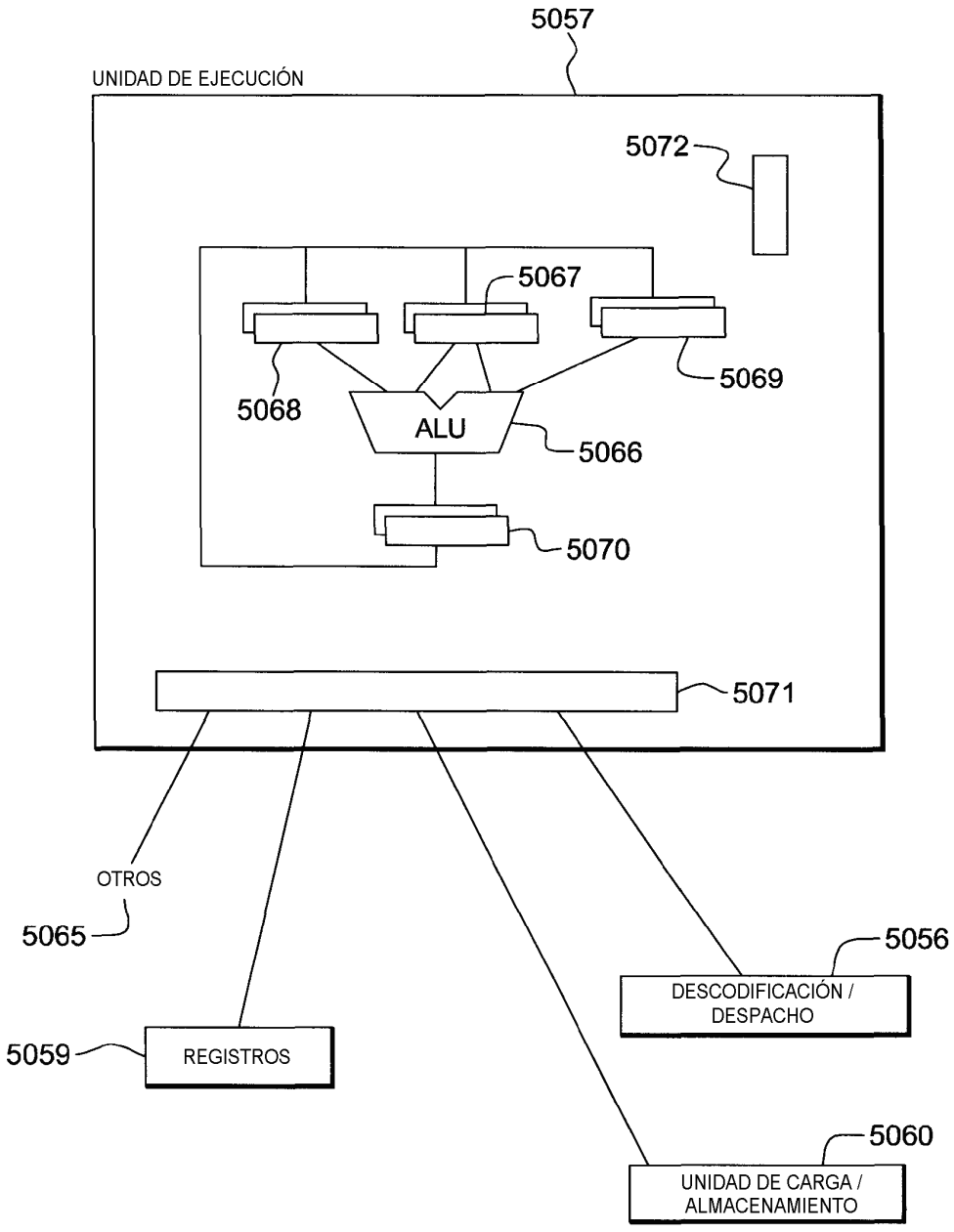

5057
UNIDAD DE EJECUCIÓN
5072
5067
5068
5069
ALU
5066
5070
5071
OTROS
5065
5056
DESCODIFICACIÓN / DESPACHO
5059
REGISTROS
5060
UNIDAD DE CARGA / ALMACENAMIENTO

FIG. 12A

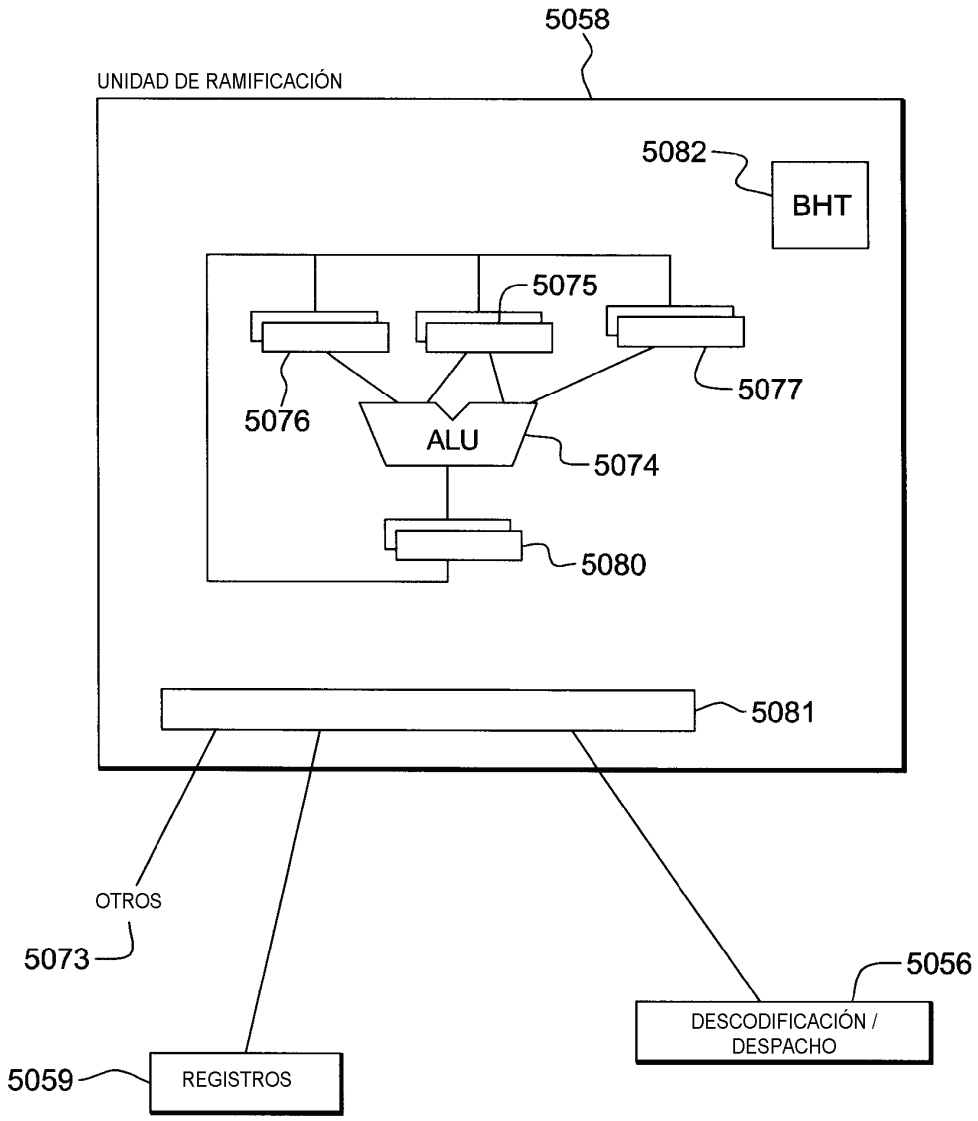

5058
UNIDAD DE RAMIFICACIÓN
5082
BHT
5075
5076
5077
ALU
5074
5080
5081
OTROS
5073
5056
DESCODIFICACIÓN / DESPACHO
5059
REGISTROS

FIG. 12B

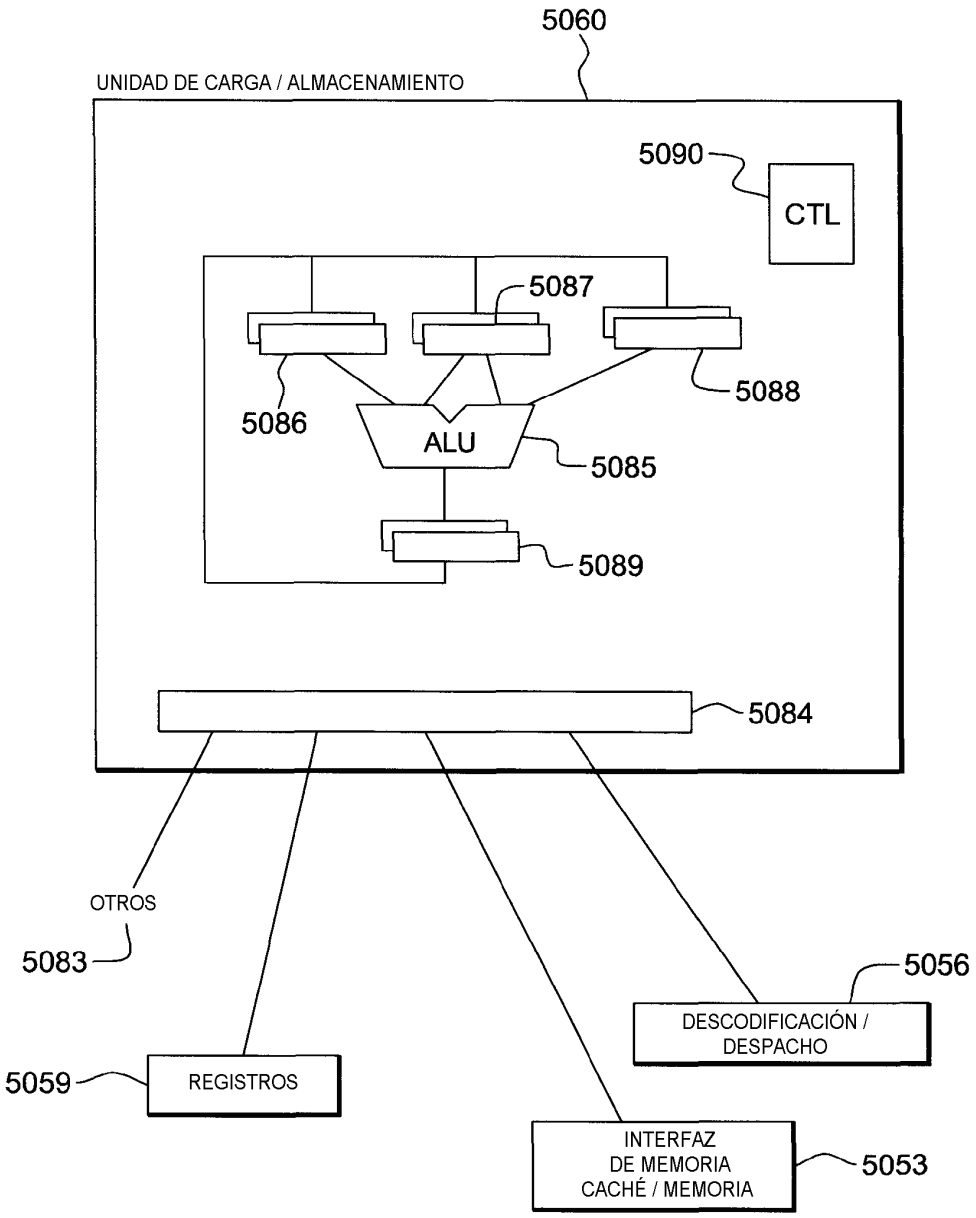

5060
UNIDAD DE CARGA / ALMACENAMIENTO
5090
CTL
5087
5086
5088
ALU
5085
5089
5084
OTROS
5083
5056
DESCODIFICACIÓN / DESPACHO
5059
REGISTROS
INTERFAZ DE MEMORIA CACHÉ / MEMORIA
5053

FIG. 12C

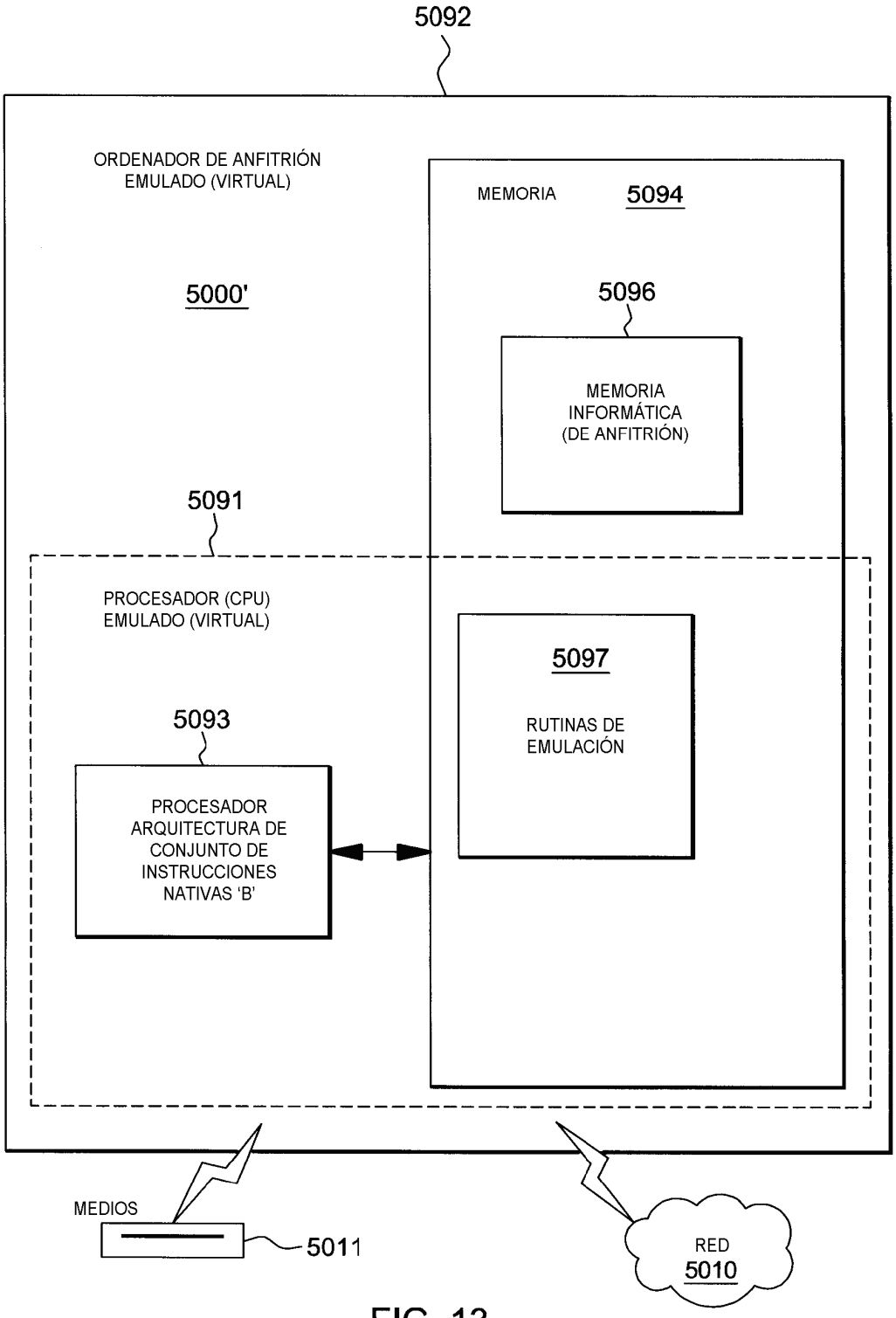

5092
ORDENADOR DE ANFITRIÓN EMULADO (VIRTUAL)
5000'
MEMORIA
5094
5096
MEMORIA INFORMÁTICA (DE ANFITRIÓN)
5091
PROCESADOR (CPU) EMULADO (VIRTUAL)
5097
RUTINAS DE EMULACIÓN
5093
PROCESADOR ARQUITECTURA DE CONJUNTO DE INSTRUCCIONES NATIVAS 'B'
MEDIOS
5011
RED
5010

FIG. 13