(19) Deutsches Patent- und Markenamt

(10) **DE 10 2014 003 659 A1** 2014.09.18

(12) # Offenlegungsschrift

(21) Aktenzeichen: **10 2014 003 659.6**
(22) Anmeldetag: **14.03.2014**
(43) Offenlegungstag: **18.09.2014**

(51) Int Cl.: ***G06F 9/38*** (2006.01)

(30) Unionspriorität:
13/840,809 **15.03.2013** **US**

(71) Anmelder:
Intel Corporation, Santa Clara, Calif., US

(74) Vertreter:
BOEHMERT & BOEHMERT Anwaltspartnerschaft mbB - Patentanwälte Rechtsanwälte, 28209 Bremen, DE

(72) Erfinder:
Hughes, Christopher J., Santa Clara, Calif., US; Charney, Mark J., Lexington, Mass., US; Corbal, Jesus, Barcelona, ES; Girkar, Milind B., Sunnyvale, Calif., US; Ould-Ahmed-Vall, Elmoustapha, Chandler, Ariz., US; Toll, Bret L., Hillsboro, Oreg., US; Valentine, Robert, Kiryat Tivon, Haifa, IL

Prüfungsantrag gemäß § 44 PatG ist gestellt.

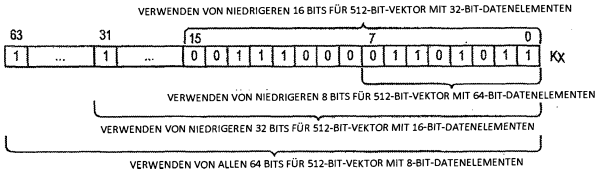
Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

(54) Bezeichnung: **SYSTEME, VORRICHTUNGEN UND VERFAHREN ZUM BESTIMMEN EINES FOLGENDEN NIEDRIGSTWERTIGEN MASKIERUNGSBITS EINES SCHREIBMASKENREGISTERS**

(57) Zusammenfassung: Die Ausführung eines KZBTZ ermittelt eine folgende niedrigstwertige Nullbitposition in einer ersten Eingabemaske und setzt eine Ausgabemaske so, dass sie die Werte der ersten Eingabemaske aufweist, aber mit allen Bitpositionen näher zur höchstwertigen Bitposition als die folgende niedrigstwertige Nullbitposition in einer ersten Eingabemaske, die auf null gesetzt ist. In einigen Ausführungsformen wird eine zweite Eingabemaske als Schreibmaske verwendet, derart dass Bitpositionen der ersten Eingabemaske bei der Berechnung der folgenden niedrigstwertigen Nullbitposition nicht berücksichtigt werden, die von einer entsprechenden Bitposition in der zweiten Eingabemaske abhängt.

ANZAHL VON EIN-BIT-VEKTOR-SCHREIBMASKENELEMENTEN

DATENELEMENT-GRÖSSE FÜR VEKTOR	VEKTORGRÖSSE		
	128 BITS	256 BITS	512 BITS
8-BIT-BYTES	16	32	64
16-BIT-WÖRTER	8	16	32
32-BIT-D-WÖRTER/SP	4	8	16
64-BIT-Q-WÖRTER/DP	2	4	8



Beschreibung

GEBIET DER ERFINDUNG

[0001] Das Gebiet der Erfindung betrifft im Allgemeinen Computerprozessorarchitektur und insbesondere Befehle, welche, wenn ausgeführt, ein bestimmtes Ergebnis zur Folge haben.

HINTERGRUND

[0002] Ein Befehlssatz oder eine Befehlssatzarchitektur (ISA für engl. instruction set architecture) ist der Teil der Computerarchitektur, der die Programmierung betrifft, und kann die nativen Datentypen, Befehle, Registerarchitektur, Adressierungsmodi, Speicherarchitektur, Unterbrechungs- und Ausnahmebehandlung und externe Eingabe und Ausgabe (I/O für engl. input/output) umfassen. Es ist zu erwähnen, dass sich der Begriff „Befehl" hierin im Allgemeinen auf einen Makrobefehl – das heißt auf Befehle, die dem Prozessor zur Ausführung zur Verfügung gestellt werden – im Gegensatz zu Mikrobefehlen oder Mikro-Operationen bezieht, die aus Decodier-Makrobefehlen eines Decoders eines Prozessors resultieren.

KURZE BESCHREIBUNG DER ZEICHNUNGEN

[0003] Die vorliegende Erfindung wird in den Figuren der beiliegenden Zeichnungen, in welchen gleiche Bezugszeichen ähnliche Elemente anzeigen, lediglich als Beispiel und nicht als Einschränkung veranschaulicht, wobei:

[0004] **Fig.** 1 eine Korrelation zwischen der Anzahl von Vektorschreibmaskenelementen mit einem aktiven Bit und der Vektorgröße und der Datenelementgröße gemäß einer Ausführungsform der Erfindung veranschaulicht.

[0005] **Fig.** 2 ist ein Blockdiagramm einer beispielhaften Ausführungsform eines Prozessors (Prozessorkerns) zum Ausführen eines oder mehrerer Befehle.

[0006] **Fig.** 3(A) und (B) veranschaulichen beispielhafte KZBTZ-Operationen.

[0007] **Fig.** 4 veranschaulicht eine Ausführungsform der Ausführung eines KZBTZ-Befehls in einem Prozessor.

[0008] **Fig.** 5 veranschaulicht eine Ausführungsform für ein Verfahren zur Verarbeitung eines KZBTZ-Befehls.

[0009] **Fig.** 6 ist ein Blockdiagramm einer Registerarchitektur **600** gemäß einer Ausführungsform der vorliegenden Erfindung.

[0010] **Fig.** 7A ist ein Blockdiagramm, das sowohl eine beispielhafte In-Order-Pipeline als auch eine beispielhafte Registerumbenennungs- und Out-of-Order-Ausgabe-/Ausführungs-Pipeline gemäß Ausführungsformen der Erfindung veranschaulicht.

[0011] **Fig.** 7B ist ein Blockdiagramm, das sowohl eine beispielhafte Ausführungsform eines Kerns mit In-Order-Architektur als auch eines beispielhaften Kerns mit Registerumbenennungs- und Out-of-Order-Ausgabe-/Ausführungsarchitektur veranschaulicht, die in einen Prozessor gemäß Ausführungsformen der Erfindung integriert werden sollen.

[0012] **Fig.** 8(A) und (B) veranschaulichen ein Blockdiagramm einer spezifischeren beispielhaften In-Order-Kernarchitektur, wobei der Kern einer von mehreren Logikblöcken (welche andere Kerne des gleichen Typs und/oder verschiedener Typen umfassen) in einem Chip wäre.

[0013] **Fig.** 9 ist ein Blockdiagramm eines Prozessors **900** gemäß Ausführungsformen der Erfindung, der mehr als einen Kern aufweisen kann, eine integrierte Speichersteuereinheit aufweisen kann, und integrierte Grafik aufweisen kann.

[0014] **Fig.** 10 bis **Fig.** 13 sind Blockdiagramme von beispielhaften Computerarchitekturen.

[0015] **Fig.** 14 ist ein Blockdiagramm, das die Verwendung eines Softwarebefehlsumsetzers zum Umsetzen von Binärbefehlen in einem Quellenbefehlssatz in Binärbefehle in einem Zielbefehlssatz gemäß Ausführungsformen der Erfindung vergleicht.

AUSFÜHRLICHE BESCHREIBUNG

[0016] In der folgenden Beschreibung werden zahlreiche spezifische Einzelheiten dargelegt. Es versteht sich jedoch von selbst, dass Ausführungsformen der Erfindung ohne diese spezifischen Einzelheiten in die Praxis umgesetzt werden können. In anderen Fällen wurden allgemein bekannte Schaltungen, Strukturen und Techniken zum besseren Verständnis dieser Beschreibung nicht im Detail dargestellt.

[0017] Bezugnahmen in der Spezifikation auf „eine (konkrete) Ausführungsform", „(irgend) eine Ausführungsform", „eine beispielhafte Ausführungsform" usw. weisen darauf hin, dass die beschriebene Ausführungsform eine bestimmte Funktion, Struktur oder Charakteristik umfassen kann, aber nicht unbedingt jede Ausführungsform die bestimmte Funktion, Struktur oder Charakteristik umfassen muss. Außerdem beziehen sich solche Ausdrücke nicht unbedingt auf die gleiche Ausführungsform. Es wird ferner der Anspruch erhoben, dass, wenn eine bestimmte Funktion, Struktur oder Charakteristik in Verbindung mit einer Ausführungsform beschrieben wird, Fachleute über die Kenntnisse verfügen, um solch eine Funktion, Struktur oder Charakteristik in Verbindung mit anderen Ausführungsformen nachzuvollziehen, einerlei ob ausdrücklich beschrieben oder nicht.

Übersicht

[0018] Die Befehlssatzarchitektur ist von der Mikroarchitektur zu unterscheiden, bei der es sich um den inneren Aufbau des Prozessors handelt, der die ISA implementiert. Prozessoren mit verschiedenen Mikroarchitekturen können einen gemeinsamen Befehlssatz gemeinsam benutzen. Zum Beispiel können Intel Pentium 4 Prozessoren, Intel Core Prozessoren und Prozessoren der Advanced Micro Devices, Inc., Sunnyvale, Kalifornien, zwar nahezu identische Versionen des x86-Befehlssatzes (wobei neueren Versionen einige Erweiterungen hinzugefügt wurden) implementieren, weisen aber jeweils einen unterschiedlichen inneren Aufbau auf. Zum Beispiel kann die gleiche Registerarchitektur der ISA unter Verwendung allgemein bekannten Techniken, welche dedizierte physikalische Register, ein oder mehrere dynamisch zugeordnete physikalische Register, die einen Registerumbenennungsmechanismus verwenden (z. B. die Verwendung einer Register-Alias-Tabelle (RAT), eines Neuordnungspuffers (ROB für engl. Reorder Buffer) und einer Ausscheideregisterdatei, wie in US-Pat. Nr. 5,446,912 beschrieben; die Verwendung von mehreren Verzeichnissen und eines Registerpools, wie in US-Pat. Nr. 5,207,132 beschrieben), usw. umfassen, auf verschiedene Arten und Weisen in verschiedenen Mikroarchitekturen implementiert werden. Sofern nicht anders angegeben, beziehen sich die Ausdrücke „Registerarchitektur", „Registerdatei" und „Register" auf das, was für die Software bzw. den Programmierer sichtbar ist, und auf die Art und Weise, in welcher Befehle Register spezifizieren. Wo Genauigkeit erwünscht ist, werden die Adjektive „logisch", „architektonisch" oder „für Software sichtbar" verwendet, um Register/Dateien in der Registerarchitektur anzugeben, während andere Adjektive verwendet werden, um Register in einer bestimmten Mikroarchitektur zu bezeichnen (z. B. physikalisches Register, Neuordnungspuffer, Ausscheideregister, Registerpool).

[0019] Ein Befehlssatz umfasst ein oder mehrere Befehlsformate. Ein bestimmtes Befehlsformat definiert verschiedene Felder (Anzahl von Bits, Stelle von Bits), um u. a. die auszuführende Operation und den bzw. die Operanden zu spezifizieren, an welchem/welchen die Operation ausgeführt werden soll. Ein bestimmter Befehl wird unter Verwendung eines bestimmten Befehlsformats ausgedrückt und spezifiziert die Operation und die Operanden. Ein Befehlsstrom ist eine spezifische Folge von befehlen, wobei jeder Befehl in der Folge eine Ausprägung eines Befehls in einem Befehlsformat ist.

[0020] Wissenschaftliche, finanzielle, autovektorisierende allzweck RMS(Erkennung, Auswertung und Synthese für engl. recognition, mining, and synthesis)-/visuelle und Multimedia-Universalanwendungen (z. B. 2D-/3D-Grafik, Bildverarbeitung, Videokompression/dekompression, Spracherkennungsalgorithmen und Audiobearbeitung) erfordern häufig, dass die gleiche Operation an einer großen Anzahl von Datenelementen ausgeführt wird (was als „Datenparellelismus" bezeichnet wird). Einzelbefehl-Mehrfachdaten (SIMD für engl. Single Instruction Multiple Data) bezieht sich auf einen Befehlstyp, der einen Prozessor veranlasst, die gleiche Operation an mehreren Datenelementen auszuführen. Die SIMD-Technologie ist insbesondere für Prozessoren geeignet, welche die Bits in einem Register logisch in einer Anzahl von Datenelementen mit fester Größe teilen können, die jeweils einen separaten Wert darstellen. Zum Beispiel können die Bits in einem 64-Bit-Register als ein Quellenoperand spezifiziert werden, der als vier separate 16-Bit-Datenelemente verarbeitet werden soll, die jeweils einen separaten 16-Bit-Wert darstellen. Als ein anderes Beispiel können die Bits in einem 256-

Bit-Register als ein Quellenoperand spezifiziert werden, der als vier separate gepackte 64-Bit-Datenelemente (Datenelemente mit Quad- oder Vierfachwort(Q)-Größe), acht separate gepackte 32-Bit-Datenelemente (Datenelemente mit Doppelwort(D)-Größe), sechzehn separate gepackte 16-Bit-Datenelemente (Datenelement mit Wort(W)-Größe) oder zweiunddreißig separate 8-Bit-Datenelemente (Datenelemente mit Byte(B)-Größe) verarbeitet werden soll. Dieser Datentyp wird als gepackter Datentyp oder Vektordatentyp bezeichnet, und Operanden dieses Datentyps werden als Operanden gepackter Daten oder Vektoroperanden bezeichnet. Mit anderen Worten bezieht sich gepacktes Datenelement oder Vektor auf eine Folge von gepackten Datenelementen; und ein Operand gepackter Daten oder ein Vektoroperand ist ein Quellen- oder Zieloperand eines SIMD-Befehls (auch als Befehl für gepackte Daten oder Vektorbefehl bekannt).

[0021] Als Beispiel spezifiziert ein SIMD-Befehlstyp eine einzige Vektoroperation, die an zwei Quellenvektoroperanden in einer vertikalen Weise ausgeführt werden soll, um einen Zielvektoroperanden (auch als Ergebnisvektoroperand bezeichnet) der gleichen Größe, mit der gleichen Anzahl von Datenelementen und in der gleichen Datenelementreihenfolge zu erzeugen. Die Datenelemente in den Quellenvektoroperanden werden als Quellendatenelemente bezeichnet, während die Datenelemente im Zielvektoroperanden als Ziel- oder Ergebnisdatenelemente bezeichnet werden. Diese Quellenvektoroperanden sind von der gleichen Größe und enthalten Datenelemente der gleichen Breite, und demnach enthalten sie die gleiche Anzahl von Datenelementen. Die Quellendatenelemente in den gleichen Bitpositionen in den beiden Quellenvektoroperanden bilden Paare von Datenelementen (auch als entsprechende Datenelemente bezeichnet; das heißt, die Datenelemente in Datenelementposition 0 jedes Quellenoperanden entsprechen einander, die Datenelemente in Datenelementposition 1 jedes Quellenoperanden entsprechen einander, usw.). Die durch diesen SIMD-Befehl spezifizierte Operation wird an jedem dieser Paare von Quellendatenelementen getrennt ausgeführt, um eine übereinstimmende Anzahl von Ergebnisdatenelementen zu generieren, und demnach weist jedes Paar von Quellendatenelementen ein entsprechendes Ergebnisdatenelement auf. Da die Operation vertikal ist, und da der Ergebnisvektoroperand von der gleichen Größe ist, die gleiche Anzahl von Datenelementen aufweist, und die Ergebnisdatenelemente in der gleichen Datenelementreihenfolge wie die Quellenvektoroperanden gespeichert werden, sind die Ergebnisdatenelemente in den gleichen Bitpositionen des Ergebnisvektoroperanden wie ihr entsprechendes Paar von Quellendatenelementen in den Quellenvektoroperanden. Zusätzlich zu diesem beispielhaften SIMD-Befehlstyp gibt es eine Vielzahl von anderen SIMD-Befehlstypen (z. B. solche, die nur einen oder mehr als zwei Quellenvektoroperanden aufweisen; die in einer horizontalen Weise funktionieren; die einen Ergebnisvektoroperanden generieren, der von einer anderen Größe ist, der eine andere Größe von Datenelementen aufweist, und/oder der eine andere Datenelementreihenfolge aufweist). Es versteht sich von selbst, dass der Begriff „Zielvektoroperand" (oder „Zieloperand") als das direkte Ergebnis des Ausführens der durch einen Befehl spezifizierten Operation definiert ist, welche die Speicherung dieses Zieloperanden an einer Stelle (sei es in einem Register oder an einer Speicheradresse, die durch diesen Befehl spezifiziert wird) umfasst, so dass auf ihn durch einen anderen Befehl (durch Spezifikation der gleichen Stelle durch den anderen Befehl) als Quellenoperand zugegriffen werden kann.

[0022] Die SIMD-Technologie, wie beispielsweise jene, die von den Intel® Core™ Prozessoren mit einem Befehlssatz eingesetzt wird, der x86-, MMX™-, Streaming SIMD Extensions (SSE)-, SSE2-, SSE3-, SSE4.1- und SSE4.2-Befehle umfasst, hat eine erhebliche Verbesserung der Anwendungsleistung ermöglicht (Core™ und MMX™ sind eingetragene Marken oder Marken der Intel Corporation, Santa Clara, Kalifornien). Es wurde ein weiterer Satz von SIMD-Erweiterungen herausgegeben und/oder veröffentlicht, der als Advanced Vector Extensions (AVX) (AVX1 und AVX2) bezeichnet wird und das VEX-Codierungsschema verwendet (siehe z. B. Intel® 64 und IA-32 Architectures Software Developers Manual, Oktober 2011; und siehe Intel® Advanced Vector Extensions Programming Reference, Juni 2011).

[0023] In der folgenden Beschreibung gibt es einige Elemente, die einer Erläuterung bedürfen, bevor die Operationen dieses konkreten Befehls in der Befehlssatzarchitektur beschrieben werden. Ein solches Element wird „Schreibmaskenregister" genannt, das im Allgemeinen verwendet wird, um einen Operanden so zu prädizieren, dass er eine Pro-Element-Rechenoperation bedingt steuert (im Folgenden kann auch der Begriff „Maskenregister" verwendet werden, der sich auf ein Schreibmaskenregister, wie beispielsweise die im Folgenden erörterten „k"-Register, bezieht). Wie im Folgenden verwendet, speichert ein Schreibmaskenregister eine Mehrzahl von Bits (16, 32, 64 usw.), wobei jedes aktive Bit des Schreibmaskenregisters die Operation/Aktualisierung eines gepackten Datenelements eines Vektorregisters während einer SIMD-Verarbeitung bestimmt. Typischerweise ist mehr als ein Schreibmaskenregister zur Verwendung durch einen Prozessorkern verfügbar.

[0024] Die Befehlssatzarchitektur umfasst mindestens einige SIMD-Befehle, welche Vektoroperationen spezifizieren und welche Felder zum Auswählen von Quellenregistern und/oder Zielregistern aus diesen Vektorregistern aufweisen (ein beispielhafter SIMD-Befehl kann eine Vektoroperation spezifizieren, die an den Inhalten

eines oder mehrerer der Vektorregister ausgeführt werden soll, und das Ergebnis dieser Vektoroperation wird in einem der Vektorregister gespeichert). Verschiedene Ausführungsformen der Erfindung können Vektorregister verschiedener Größen aufweisen und größere, kleinere, verschieden große Datenelemente unterstützen.

[0025] Die Größe der durch einen SIM-Befehl spezifizierten Mehrbit-Datenelemente (z. B. Byte, Wort, Doppelwort, Quadwort) bestimmt die Bitstellen der „Datenelementpositionen" innerhalb eines Vektorregisters, und die Größe des Vektoroperanden bestimmt die Anzahl von Datenelementen. Gepacktes Datenelement bezieht sich auf die Daten, die in einer bestimmten Position gespeichert sind. Mit anderen Worten ändern sich in Abhängigkeit von der Größe der Datenelemente im Zieloperanden und der Größe des Zieloperanden (der Gesamtzahl von Bits im Zieloperanden) (oder anders ausgedrückt, in Abhängigkeit von der Größe des Zieloperanden und der Anzahl von Datenelementen innerhalb des Zieloperanden) die Bitstellen der Mehrbit-Datenelementpositionen innerhalb des resultierenden Vektoroperanden (wenn z. B. das Ziel für den resultierenden Vektoroperanden ein Vektorregister ist (in dieser Erörterung werden Vektorregister und Register gepackter Datenelementen austauschbar verwendet), dann ändern sich die Bitstellen der Mehrbit-Datenelementpositionen innerhalb des Zielvektorregisters). Zum Beispiel sind die Bitstellen der Mehrbit-Datenelemente zwischen einer Vektoroperation, welche 32-Bit-Datenelemente (die Datenelementposition 0 belegt die Bitstellen 31:0, die Datenelementposition 1 belegt die Bitstellen 63:32, und so weiter) verarbeitet, und einer Vektoroperation, welche 64-Bit-Datenelemente (die Datenelementposition 0 belegt die Bitstellen 63:0, die Datenelementposition belegt die Bitstellen 127:64, und so weiter), verschieden.

[0026] Außerdem besteht eine Korrelation zwischen der Anzahl von Vektorschreibmaskenelementen mit einem aktiven Bit und der Vektorgröße und der Datenelementgröße gemäß einer Ausführungsform der Erfindung, wie in **Fig.** 1 dargestellt. Es sind Vektorgrößen von 128-Bits, 256-Bits und 512-Bits dargestellt, obwohl auch andere Breiten möglich sind. Es sind Datenelementgrößen von 8-Bit-Bytes (B), 16-Bit-Wörtern (W), 32-Bit-Doppelwörtern (D) oder Gleitkomma (FP für engl. floating point) einfacher Genauigkeit und 64-Bit-Quadwörtern (Q) oder Gleitkomma doppelter Genauigkeit berücksichtigt, obwohl auch andere Breiten möglich sind. Wie dargestellt, können, wenn die Vektorgröße 128 Bits beträgt, 16 Bits zur Maskierung verwendet werden, wenn die Vektor-Datenelementgröße 8 Bits beträgt, 8 Bits können zur Maskierung verwendet werden, wenn die Vektor-Datenelementgröße 16 Bits beträgt, 4 Bits können zur Maskierung verwendet werden, wenn die Vektor-Datenelementgröße 32 Bits beträgt, und 2 Bits können zur Maskierung verwendet werden, wenn die Vektor-Datenelementgröße 64 Bits beträgt. Wenn die Vektorgröße 256 Bits beträgt, können 32 Bits zur Maskierung verwendet werden, wenn die Breite des gepackten Datenelements 8 Bits beträgt, 16 Bits können zur Maskierung verwendet werden, wenn die Vektor-Datenelementgröße 16 Bits beträgt, 8 Bits können zur Maskierung verwendet werden, wenn die Vektor-Datenelementgröße 32 Bits beträgt, und 4 Bits können zur Maskierung verwendet werden, wenn die Vektor-Datenelementgröße 64 Bits beträgt. Wenn die Vektorgröße 512 Bits beträgt, können 64 Bits zur Maskierung verwendet werden, wenn die Vektor-Datenelementgröße 8 Bits beträgt, 32 Bits können zur Maskierung verwendet werden, wenn die Vektor-Datenelementgröße 16 Bits beträgt, 16 Bits können zur Maskierung verwendet werden, wenn die Vektor-Datenelementgröße 32 Bits beträgt, und 8 Bits können zur Maskierung verwendet werden, wenn die Vektor-Datenelementgröße 64 Bits beträgt.

[0027] In Abhängigkeit von der Kombination der Vektorgröße und der Datenelementgröße können entweder alle 64 Bits oder nur ein Teilsatz der 64 Bits als Schreibmaske verwendet werden. Im Allgemeinen entspricht, wenn ein einziges Pro-Element-Maskierungskontrollbit verwendet wird, die Anzahl von Bits im Vektorschreibmaskenregister, die zur Maskierung verwendet werden (aktive Bits), der Vektorgröße in Bits geteilt durch die Vektor-Datenelementgröße in Bits.

[0028] Wie bereits erwähnt, enthalten Schreibmaskenregister Maskenbits, welche Elementen in einem Vektorregister (oder einer Speicherstelle) entsprechen und die Elemente verfolgen, an welche Operation ausgeführt werden sollten. Aus diesem Grund ist es wünschenswert, über gemeinsame Operationen zu verfügen, welche ein ähnliches Verhalten bei diesen Maskenbits wie für die Vektorregister duplizieren und es im Allgemeinen ermöglichen, diese Maskenbits innerhalb der Maskenschreibregister anzupassen.

[0029] **Fig.** 2 ist ein Blockdiagramm einer beispielhaften Ausführungsform eines Prozessors (Prozessorkerns) **200** zum Ausführen eines oder mehrerer KZBTZ-Befehle **204**. In einigen Ausführungsformen kann der Prozessor ein Universalprozessor (z. B. von dem Typ, der in Tisch-, Laptop-, Server- und dergleichen Computern verwendet wird) sein. Alternativ kann der Prozessor ein Spezialprozessor sein. Beispiele von geeigneten Spezialprozessoren umfassen, ohne darauf beschränkt zu sein, Netzwerkprozessoren, Kommunikationsprozessoren, kryptografische Prozessoren, Grafikprozessoren, Co-Prozessoren, eingebettete Prozessoren, Digitalsignalprozessoren (DSPs) und Steuereinheiten, um nur einige Beispiele zu nennen. Der Prozessor kann ein beliebiger von verschiedenen Prozessoren zum Rechnen mit komplexem Befehlssatz (CISC für engl. complex

instruction set computing), verschiedenen Prozessoren mit reduziertem Befehlssatz (RISC für engl. reduced instruction set computing), verschiedenen Prozessoren für sehr lange Befehlswörter (VLIW für engl. very long instruction word), verschiedenen Hybriden davon oder ganz anderen Typen von Prozessoren sein.

[0030] Der Prozessor **200** umfasst architektonisch sichtbare Register (z. B. eine architektonische Registerdatei) **205**. Die architektonischen Register können hierin auch einfach als Register bezeichnet sein. Sofern nicht anders festgelegt oder ersichtlich, werden die Ausdrücke „architektonisches Register", „Registerdatei" und „Register" hierin zur Bezeichnung von Registern, die für die Software und/oder den Programmierer sichtbar sind, und/oder der Register verwendet, die durch Makrobefehle oder Befehle der Assemblersprache zum Identifizieren von Operanden spezifiziert werden. Diese Register stehen im Gegensatz anderen, nicht-architektonischen oder nicht-architektonisch sichtbaren Registern in einer bestimmten Mikroarchitektur (z. B. temporären Registern, die von Befehlen verwendet werden, Neuordnungspuffern, Ausscheideregistern usw.). Die Register stellen im Allgemeinen chipintegrierte Prozessorspeicherstellen dar. Die veranschaulichten architektonischen Register umfassen Register gepackter Daten **206**. Jedes der Register gepackter Daten kann so betrieben werden, dass es gepackte oder Vektordaten speichert. Die veranschaulichten architektonischen Register umfassen außerdem Operationsmaskenregister gepackter Daten **207**. Jedes der Operationsmaskenregister gepackter Daten kann so betrieben werden, dass es eine Operationsmaske gepackter Daten speichert. Diese Register können in dieser Beschreibung als Schreibmaskenregister bezeichnet sein. Operanden gepackter Daten können in den Registern gepackter Daten **207** gespeichert werden.

[0031] Der Prozessor umfasst außerdem eine Ausführungslogik **208**. Die Ausführungslogik kann so betrieben werden, dass sie den einen oder die mehreren KZBTZ-Befehle **204** ausführt oder verarbeitet. In einigen Ausführungsformen kann die Ausführungslogik bestimmte Logik (z. B. bestimmte Schaltungsanordnung oder Hardware möglicherweise kombiniert mit Firmware) zum Ausführen dieser Befehle umfassen.

[0032] Ein wichtiges Algorithmusmuster zum effizienten Vektorisieren sind Rechenvorgänge, die sowohl Lese- als auch Schreiboperationen aus indirekten bzw. in indirekte Speicherstellen umfassen. Zum Beispiel Kopieren von A[B[i]] nach A[C[i]]. Das Vektorisieren dieses Schleifentyps umfasst ein Ausführen von Sammel- und Streuoperationen an mehreren Indexvektoren (z. B. B[i] und C[i]). Diese Vektorisierung setzt jedoch voraus, dass keine Speicherabhängigkeiten durch das simultane Ausführen von mehreren Lese- und Schreiboperationen verletzt werden. Wenn zum Beispiel eine Gruppe von Elementen mit SIMD-Breite aus B[i] einen gemeinsamen Wert mit C[i] enthält, dann kann eine Lesen-nach-Schreiben-Abhängigkeit verletzt werden. Genauer gesagt, muss dann, wenn B[0] = 0, B[1] = 1, C[0] = 1, und C[1] = 2, das Auslesen von A[B[1]] auf das Schreiben in A[C[0]] folgen. Wenn alle der Leseoperationen bei einem Sammelbefehl simultan ausgeführt werden, dann verletzen alle der Schreiboperation bei einem Streubefehl diese Abhängigkeit, was zu einer inkorrekten Antwort führen kann.

[0033] Um dieses Problem lösen zu helfen, vergleicht ein Befehl, der vconflict genannt wird, jedes Element eines ersten Vektors mit allen Elementen eines zweiten Vektors und gibt die Vergleichsergebnisse als einen Satz von Bitvektoren in ein Vektorregister aus. Der Gedanke dabei ist, „Konflikte" oder übereinstimmende Indizes über verschiedene Sammlungen/Streuungen zu erkennen. Wenn es Konflikte gibt, dann wird ein Rechenvorgang an einer bestimmten Gruppe von Elementen mit SIMD-Breite iterativ ausgeführt, wobei so viele Elemente als möglich simultan ausgeführt werden, wie in dem folgenden Pseudocode veranschaulicht.

```
for (i=0; i<N; i+=SIMD_WIDTH)
gather_indices = vload(&B[i]); scatter_indices = vload(&C[i]);
comparisons = vconflict(gather_indices, scatter_indices);
elements_left_mask = all_ones;
do {
do_these = Compute_Mask_of_Non_Conflicting_Elements(comparisons,
elements_left_mask);
Gather_Compute_Scatter(gather_indices, scatter_indices, do_these);
```

```
elements_left_mask A= do_these;
} while (elements_left_mask != 0);
}
```

[0034] „Compute_Mask_of_Non_Conflicting_Elements" (Berechnen von Maske von nicht-konfliktiven Elementen) ist eine nicht-triviale Operation. Bestehende Befehle in Verbindung mit vconflict haben alle ein Problem gemein – sie können nicht alle möglichen Datenabhängigkeiten innerhalb zweier Sätze von Indizes ohne Ausführen mehrerer vconflict-Instanzen und zusätzliche Bearbeitung der Vergleichsergebnisse erkennen. Insbesondere wird im vorstehenden Beispiel gezeigt, wie doppelte Indizes zwischen den Sammel- und Streu-Indexvektoren erkannt werden müssen, um Lesen-nach-Schreiben(RAW für engl. read-after-write)-Abhängigkeiten zu erkennen. Die vorgeschlagene Lösung, um diese Abhängigkeiten durchzusetzen, wobei die Verarbeitung einiger Elemente verzögert wird, kann jedoch eine Verletzung von Schreiben-nach-Schreiben(WAW für engl. write-after-write)- oder Schreiben-nach-Lesen(WAR für engl. write-after-read)-Abhängigkeiten verursachen. Man nehme zum Beispiel an, dass B[0] = 0, B[1] = 1, B[2] = 2, C[0] = 1, C[1] = 3 und C[2] = 3. Genau wie im vorstehenden Beispiel weist die zweite Iteration eine RAW-Abhängigkeit von der ersten Iteration auf, so dass sie verzögert werden muss. Die dritte Iteration weist keine RAW-Abhängigkeit auf, so dass eine Wahl getroffen werden kann, sie simultan mit der ersten Iteration auszuführen. Wenn dies geschieht, dann findet jedoch das Schreiben in A[C[2]] (=A[3]) vor dem Schreiben in A[C[1]] (=A[3]) statt und verletzt eine WAW-Abhängigkeit.

[0035] Im Folgenden sind Ausführungsformen eines Befehls, der generisch Nullmaske-vorfolgender-Null („KZBTZ" für engl. zero mask before trailing zero)-Befehl des Befehlssatzes genannt wird, und Ausführungsformen von Systemen, Architekturen, Befehlsformaten usw., die zum Ausführen solch eines Befehls verwendet werden können. Die Ausführung eines KZBTZ ermittelt eine folgende niedrigstwertige Nullbitposition in einer ersten Eingabemaske und setzt eine Ausgabemaske so, dass sie die Werte der ersten Eingabemaske aufweist, aber mit allen Bitpositionen näher zur höchstwertigen Bitposition als die folgende niedrigstwertige Nullbitposition in einer ersten Eingabemaske, die auf null gesetzt ist. In einigen Ausführungsformen wird eine zweite Eingabemaske als Schreibmaske verwendet, derart dass Bitpositionen der ersten Eingabemaske bei der Berechnung der abschließenden niedrigstwertigen Nullbitposition nicht berücksichtigt werden, die von einer entsprechenden Bitposition in der zweiten Eingabemaske abhängt.

[0036] **Fig.** 3(A) und (B) veranschaulichen beispielhafte KZBTZ-Operationen. In **Fig.** 3(A) gibt es zwei Quellenschreibmaskenregister **301** und **303** und eine Zielschreibmaske **305**. Der erste „0"-Wert im ersten Quellenschreibmaskenregister **301** erscheint in der dritten Bitposition (SRC1[2]). In einer entsprechenden Bitposition des zweiten Quellenschreibmaskenregisters **303** ist der Wert eine „1", was bedeutet, dass diese Bitposition die erste Bitposition ist, die sowohl eine Null im ersten Quellenschreibmaskenregister **301** als auch aktiviert im zweiten Schreibmaskenregister **303** war. Das Zielschreibmaskenregister **305** weist daher die Inhalte des ersten Schreibmaskenregisters **301** bis zur Bitposition 3 auf, und diese Bitposition und Bitpositionen im Zielschreibmaskenregister **305**, die höherwertig sind, sind auf 0 gesetzt.

[0037] In **Fig.** 3(B) gibt es zwei Quellenschreibmaskenregister **307** und **309** und eine Zielschreibmaske **311**. Der erste „0"-Wert im ersten Quellenschreibmaskenregister **307** erscheint in der dritten Bitposition (SRC1[2]). In einer entsprechenden Bitposition des zweiten Quellenschreibmaskenregisters **309** ist der Wert eine „0", was bedeutet, dass diese Bitposition nicht als die folgende niedrigstwertige Nullbitposition gewertet wird. Die erste Bitposition, die beide Anforderungen erfüllt, ist Bitposition 4 (SRC1[3] und SRC2[3]). Das Zielschreibmaskenregister **311** weist daher die Inhalte des ersten Schreibmaskenregisters **307** bis zur Bitposition 4 auf, und diese Bitposition und Bitpositionen im Zielschreibmaskenregister **311**, die höherwertig sind, sind auf 0 gesetzt.

Beispielhaftes KZBTZ-Format

[0038] Ein beispielhaftes Format dieses Befehls ist „KZBTZ K1, K2, K3", wobei der Zieloperand K1 ein Schreibmaskenregister ist, K2 und K3 Quellenschreibmaskenregister sind, und KZBTZ der Operationscode des Befehls ist. In einigen Ausführungsformen sind K1, K2 und K3 dedizierte Schreibmaskenregister, wie zuvor genauer beschrieben. In anderen Ausführungsformen sind K1, K2 und K3 Universalregister.

Beispielhafte Verfahren zur Ausführung von KZBTZ

[0039] **Fig.** 4 veranschaulicht eine Ausführungsform der Ausführung eines KZBTZ-Befehls in einem Prozessor. Ein KZBTZ-Befehl mit einem ersten und einem zweiten Quellenschreibmaskenoperanden, einem Zielschreibmaskenoperanden und einem Operationscode wird bei **401** abgerufen.

[0040] Der KZBTZ-Befehl wird bei **403** durch die Decodierlogik **403** decodiert.

[0041] Die Werte der Quellenoperanden werden bei **405** abgerufen/ausgelesen. Zum Beispiel werden die Quellenschreibmaskenregister ausgelesen.

[0042] Der decodierte KZBTZ-Befehl (oder Operationen, die solch einen Befehl umfassen, wie beispielsweise Mikrooperationen) wird durch Ausführungsbetriebsmittel, wie beispielsweise eine oder mehrere Funktionseinheiten, bei **407** ausgeführt, um eine niedrigstwertige Nullbitposition im ersten Quellenschreibmaskenoperanden zu ermitteln, die einen Eins-Wert in einer entsprechenden Bitposition des zweiten Quellenschreibmaskenoperanden aufweist. Diese Bitposition bedeutet eine folgende niedrigstwertige Nullbitposition. Beispiele dafür sind in **Fig.** 3 zu finden.

[0043] Die Werte bis zur folgende niedrigstwertigen Nullbitposition, ohne diese einzubeziehen, werden bei **409** im Zielschreibmaskenoperanden in entsprechenden Bitpositionen gespeichert. Außerdem werden die restlichen Bitpositionen des Zielschreibmaskenoperanden auf 0 gesetzt. Obwohl **407** und **409** getrennt veranschaulicht wurden, werden sie in einigen Ausführungsformen als ein Teil der Ausführung des Befehls zusammen ausgeführt.

[0044] **Fig.** 5 veranschaulicht eine Ausführungsform für ein Verfahren zur Verarbeitung eines KZBTZ-Befehls. In dieser Ausführungsform wird davon ausgegangen, dass einige, wenn nicht alle, der Operationen **401** bis **405** bereits vorher ausgeführt wurden, aber sie sind nicht dargestellt, um die im Folgenden dargelegten Einzelheiten nicht zu verkomplizieren. Zum Beispiel sind weder das Abrufen und Decodieren noch die Operandenabruf dargestellt. In diesem Beispiel wird jede Bitposition parallel verarbeitet, aber die Bitposition können auch seriell ausgewertet werden.

[0045] Bei **501** werden die Inhalte des ersten Quellenschreibmaskenregisters in das Zielschreibmaskenregister geschrieben. In einigen Ausführungsformen werden die Inhalte des ersten Quellschreibmaskenregisters alternativ in ein temporares Register oder in eine andere Datenstruktur geschrieben.

[0046] Eine temporäre Variable wird bei **502** auf 0 gesetzt. Diese temporare Variable wird als Zähler verwendet, um zu bestimmen, ob die Anzahl von Bestimmungen in **503** die Anzahl von Bitpositionen im ersten Quellenschreibmaskenregister überschritten hat.

[0047] Bei **503** erfolgt eine Bestimmung dessen, ob: 1) der Zähler niedriger als die Anzahl von Bitpositionen im ersten Quellenschreibmaskenregister ist; 2) ein Bitwert in einer Zählerwert-Bitposition des ersten Quellenschreibmaskenregisters 1 ist oder, 3) ein Bitwert in einer Zählerwert-Bitposition des zweiten Quellenschreibmaskenregisters 1 ist. Wenn eine dieser Bestimmungen falsch ist, dann besteht der nächste Schritt darin, bei **507** alle der Bitpositionen des Zielschreibmaskenregisters von der Zählerwert-Bitposition bis zur höchstwertigen Bitposition auf null zu setzen. Ein „falsch" für die Zählerbestimmung bedeutet, dass die gesamte erste Schreibmaske ausgewertet und keine niedrigstwertige Nullbitposition gefunden wurde. Eine Falsch-Anzeige für den in der Bitposition der ersten Quellenschreibmaske gespeicherten Wert, die nicht 1 ist, ist daher eine Anzeige eines Nullwerts. Wenn der Wert, der in der gleichen Bitposition des zweiten Quellenschreibmaskenregisters gespeichert ist, eine 1 ist, dann wurde die folgende niedrigstwertige Nullbitposition gefunden.

[0048] Wenn eine dieser Bestimmungen wahr ist, dann besteht der nächste Schritt darin, den Zähler bei **505** zu erhöhen und die Bestimmungen von **503** erneut durchzuführen.

[0049] Eine beispielhafte Verwendung von KZBTZ wird im Folgenden erörtert. In diesem Beispiel besteht eine Lösung für das Vorhergesagte darin, keine Ausführung von späteren Iterationen vor früheren Iterationen zuzulassen; demnach muss Vektor-/SIMD-Ausführung bei der ersten RAW-Abhängigkeit gestoppt werden. Wenn die Eingabemaske k2 eine Maske ist, die Bits aufweist, die für Elemente gesetzt sind, die noch berechnet werden müssen und die keine verbleibenden RAW-Abhängigkeiten aufweisen, und eine Eingabemaske k3 eine Maske ist, die alle noch zu berechnenden Elemente anzeigt, ermittelt sie das früheste Element mit einem

RAW-Konflikt und setzt alle Bits für spätere Elemente auf null. Dies führt zum folgenden Algorithmus zum Vektorisieren von Schleifen wie diese:

```
for (1=0; i<N; i+=SIMD_WIDTH) {

gather_indices = vload(&B[i]); scatter_indices = vload(&C[i]);

comparisons = vconflict(gather_indices, scatter_indices);

elements_left_mask = all_ones;

do {

no raw_mask =

Compute_Mask_of_Elements_wh_RAW_Dependence(comparisons, elements_left_mask);

stop_at_first_conf_mask = kzbtz(no_raw_mask, elements_left_mask); Gath-

er_Compute_Scatter(gather_indices, scatter_indices, stop_at_first_conf_mask);

elements_left_mask A= stop_at_first_conf_mask;

} while (elements_left_mask != 0);

}
```

Beispielhafte Registerarchitektur

[0050] **Fig.** 6 ist Blockdiagramm einer Registerarchitektur **600** gemäß einer Ausführungsform der Erfindung. In der veranschaulichten Ausführungsform gibt es 32 Vektorregister **610** mit einer Breite von 512 Bits; diese Register werden als zmm0 bis zmm031 bezeichnet. Die niedrigerwertigen 256 Bits der unteren 16 zmm-Register überlagern Register ymm0 bis ymm16. Die niedrigerwertigen Bits der unteren 16 zmm-Register (die niedrigerwertigen 128 Bits der ymm-Register) überlagern Register xmm0 bis xmm15.

[0051] Universalregister **625** – in der veranschaulichten Ausführungsform gibt es sechzehn 64-Bit-Universalregister, die zusammen mit den bestehenden x86-Adressiermodi zum Adressieren von Speicheroperanden verwendet werden. Diese Register sind mit den Benennungen RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP und R8 bis R15 gekennzeichnet.

[0052] Skalare Gleitkomma-Stapelregisterdatei (x87-Stapel) **645**, in welcher die gepackte ganzzahlige flache MMX-Registerdatei **650** aliasiert ist – in der veranschaulichten Ausführungsform ist der x87-Stapel ein Stapel von acht Elementen, der zum Durchführen von skalaren Gleitkommaoperationen an 32-/64-/80-Bit-Gleitkommadaten unter Verwendung der x87-Befehlssatzerweiterung verwendet wird; während die MMX-Register zum Ausführen von Operationen an gepackten ganzzahligen 64-Bit-Daten sowie zum Speichern von Operanden für einige Operationen verwendet werden, die zwischen den MMX- und XMM-Registern ausgeführt werden.

[0053] Alternative Ausführungen der Erfindung können breitere oder schmalere Register verwenden. Außerdem können alternative Ausführungsformen der Erfindung mehr, weniger oder andere Registerdateien und Register verwenden.

Beispielhafte Kernarchitekturen, Prozessoren und Computerarchitekturen

[0054] Prozessorkerne können auf verschiedene Arten und Weisen, für verschiedene Zwecke und in verschiedenen Prozessoren implementiert werden. Zum Beispiel können Implementierungen solcher Kerne umfassen: 1) einen In-Order-Universalkern, der für Allzweckberechnung bestimmt ist; 2) einen Out-of-Order-Hochleistungs-Universalkern, der für Allzweckberechnung bestimmt ist; 3) einen Spezialkern, der in erster Linie für Grafik- und/oder wissenschaftliche (Durchsatz-)Berechnung bestimmt ist. Implementierungen von verschiedenen Prozessoren können umfassen: 1) eine CPU, die einen oder mehrere In-Order-Universalkerne, die für Allzweckberechnung bestimmt sind, und/oder einen oder mehrere Out-of-Order-Universalkerne umfasst, die für Allzweckberechnung bestimmt sind; und 2) einen Coprozessor, der einen oder mehrere Spezialkerne umfasst, die in erster Linie für Grafik- und/oder wissenschaftliche (Durchsatz-)Berechnung bestimmt sind. Solche

verschiedene Prozessoren führen zu verschiedenen Computersystemarchitekturen, welche umfassen können: 1) den Coprozessor auf einem von der CPU separaten Chip; 2) den Coprozessor auf einem separaten Chip im gleichen Gehäuse wie eine CPU; 3) den Coprozessor auf dem gleichen Chip wie eine CPU (in welchem Fall solch ein Coprozessor manchmal als Speziallogik, wie beispielsweise integrierte Grafik- und/oder wissenschaftliche (Durchsatz-)Logik, oder als Universalkerne bezeichnet wird); und 4) System auf einem Chip, das auf dem gleichen Chip die beschriebene CPU (manchmal als Anwendungskern(e) oder Anwendungsprozessor(en) bezeichnet), den zuvor beschriebenen Coprozessor und zusätzliche Funktionalität umfassen kann. Als Nächstes werden beispielhafte Kernarchitekturen beschrieben, worauf Beschreibungen von beispielhaften Prozessoren und Computerarchitekturen folgen.

Beispielhafte Kernarchitekturen

In-Order- und Out-of-Order-Kernblockdiagramm

[0055] **Fig.** 7A ist ein Blockdiagramm, das sowohl eine beispielhafte In-Order-Pipeline als auch eine beispielhafte Registerumbenennungs- und Out-of-Order-Ausgabe-/Ausführungs-Pipeline gemäß Ausführungsformen der Erfindung veranschaulicht. **Fig.** 7B ist ein Blockdiagramm, das sowohl eine beispielhafte Ausführungsform eines Kerns mit In-Order-Architektur als auch eines beispielhaften Kerns mit Registerumbenennungs- und Out-of-Order-Ausgabe-/Ausführungsarchitektur veranschaulicht, die in einen Prozessor gemäß Ausführungsformen der Erfindung integriert werden sollen. Die Kästchen mit durchgehenden Linien in **Fig.** 7A und **Fig.** 7B veranschaulichen die In-Order-Pipeline und den In-Order-Kern, während die optionale Hinzufügung der Kästchen mit gestrichelten Linien die Pipeline und den Kern für Registerumbenennung und Out-of-Order-Ausgabe/Ausführung veranschaulicht. Da der In-Order-Aspekt ein Teilsatz des Out-of-Order-Aspekts ist, wird der Out-of-Order-Aspekt beschrieben.

[0056] In **Fig.** 7A umfasst eine Prozessor-Pipeline QAE00 eine Abrufstufe QAE02, eine Längendecodierstufe QAE04, eine Decodierstufe QAE06, eine Zuordnungsstufe QAE08, eine Umbenennungsstufe QAE10, eine Ablaufsteuerungs(auch bekannt als Abfertigungs- oder Ausgabe)-Stufe QAE12, eine Registerlese-/Speicherlesestufe QAE14, eine Ausführungsstufe QAE16, eine Zurückschreib-/Speicherschreibstufe QAE18, eine Ausnahmebehandlungsstufe QAE22 und eine Commit-Stufe QAE24.

[0057] **Fig.** 7B stellt einen Prozessorkern QAE90 dar, der eine Eingangseinheit QAE30 umfasst, die mit einer Ausführungsmaschineneinheit QAE50 gekoppelt ist, wobei beide mit einer Speichereinheit QAE70 gekoppelt sind. Der Kern QAE90 kann ein Kern zum Rechnen mit reduziertem Befehlssatz (RISC), ein Kern zum Rechnen mit komplexem Befehlssatz (CISC), ein Kern für sehr lange Befehlswörter (VLIW) oder ein Hydrid oder ein alternativer Kerntyp sein. Als noch eine andere Option kann der Kern QAE90 ein Spezialkern, wie beispielsweise ein Netzwerk- oder Kommunikationskern, eine Kompressionsmaschine, ein Coprozessorkern, ein Kern für Allzweckberechnungs-Grafikprozessoreinheit (GPGPU für engl. general purpose computing graphics processing unit), ein Grafikkern oder dergleichen sein.

[0058] Die Eingangseinheit QAE30 umfasst eine Zweigprädiktionseinheit QAE32, die mit einer Befehls-Cache-Einheit QAE34 gekoppelt ist, die mit einem Befehlsübersetzungs-Vorgriffspuffer (TLB) QAE36 gekoppelt, der mit einer Befehlsabrufeinheit QAE38 gekoppelt ist, die mit einer Decodiereinheit QAE40 gekoppelt ist. Die Decodiereinheit QAE40 (oder der Decoder) kann Befehle decodieren und als eine Ausgabe eine oder mehrere Mikrooperationen, Mikrocode-Eintrittsstellen, Mikrobefehle, andere Befehle oder andere Steuersignale generieren, die aus den Originalbefehlen decodiert oder von diesen abgeleitet werden oder diese anderweitig widerspiegeln. Die Decodiereinheit QAE40 kann unter Verwendung von mehreren verschiedenen Mechanismen implementiert sein. Beispiele von geeigneten Mechanismen umfassen, ohne darauf beschränkt zu sein, Nachschlagetabellen, Hardwareimplementierungen, programmierbare Logikfelder (PLAs für engl. programmable logic arrays), Mikrocode-Festwertspeicher (ROMs für engl. read only memories) usw. In einer Ausführungsform umfasst der Kern QAE90 einen Mikrocode-ROM oder ein anderes Medium, das Mikrocode für bestimmte Makrobefehle speichert (z. B. in der Decodiereinheit QAE40 oder andernfalls in der Eingangseinheit QAE30). Die Decodiereinheit QAE40 ist mit einer Umbenennungs-/Zuordnungseinheit QAE52 in der Ausführungsmaschineneinheit QAE50 gekoppelt.

[0059] Die Ausführungsmaschineneinheit QAE50 umfasst die Umbenennungs-/Zuordnungseinheit QAE52, die mit einer Ausscheideeinheit QAE54 und einem Satz von einer oder mehreren Scheduler-Einheit(en) QAE56 gekoppelt ist. Die Scheduler-Einheit(en) QAE56 stellt/stellen eine beliebige Anzahl von verschiedenen Schedulern dar, die Reservierungsstationen, ein zentrales Befehlsfenster usw. umfassen. Die Scheduler-Einheit (en) QAE56 ist/sind mit der/den Einheit(en) physikalischer Registerdatei(en) QAE58 gekoppelt. Jede der Ein-

heiten physikalischer Registerdatei(en) QAE58 stellt eine oder mehrere physikalische Registerdateien dar, von welchen verschiedene einen oder mehrere Datentypen speichern, wie beispielsweise skalare Ganzzahl, skalares Gleitkomma, gepackte Ganzzahl, gepacktes Gleitkomma, vektorielle Ganzzahl, vektorielles Gleitkomma, Status (z. B. einen Befehlszeiger, der die Adresse des nächsten Befehls ist, der ausgeführt werden soll) usw. In einer Ausführungsform umfasst die Einheit physikalischer Registerdatei(en) QAE58 eine Vektorregistereinheit und eine Skalarregistereinheit. Diese Registereinheiten können architektonische Vektorregister, Vektormaskenregister und Universalregister bereitstellen. Die Einheit(en) physikalischer Registerdatei(en) QAE58 sind von der Ausscheideeinheit QAE54 überlappt, um verschiedene Möglichkeiten zu veranschaulichen, in welchen Registerumbenennung und Out-of-Order-Ausführung (z. B. unter Verwendung von Neuordnungspuffer(n) und Ausscheideregisterdatei(en); unter Verwendung von Zukunftsdatei(en), Verlaufspuffer(n) und Ausscheideregisterdatei(en); unter Verwendung von Registerverzeichnissen und eines Registerpools usw.) implementiert werden können. Die Ausscheideeinheit QAE54 und die Einheit(en) physikalischer Registerdatei(en) QAE58 sind mit Ausführungs-Clustern QAE60 gekoppelt. Der/die Ausführungs-Cluster QAE60 umfasst/umfassen einen Satz von einer oder mehreren Ausführungseinheiten QAE62 und einen Satz von einer oder mehreren Speicherzugriffseinheiten QAE64. Die Ausführungseinheiten QAE62 können verschiedene Operationen (z. B. Verschiebungen, Addition, Subtraktion, Multiplikation) und an verschiedenen Datentypen (z. B. skalarem Gleitkomma, gepackter Ganzzahl, gepacktem Gleitkomma, vektorieller Ganzzahl, vektoriellem Gleitkomma) ausführen. Obwohl einige Ausführungsformen eine Anzahl von Ausführungseinheiten umfassen können, die spezifischen Funktionen oder Sätzen von Funktionen gewidmet sind, können andere Ausführungsformen nur eine Ausführungseinheit oder mehrere Ausführungseinheiten umfassen, die allesamt alle Funktionen ausführen. Die Scheduler-Einheit(en) QAE56, die Einheit(en) physikalischer Registerdatei(en) QAE58 und der/die Ausführungscluster QAE60 sind so dargestellt, dass sie möglicherweise mehrere sind, da bestimmte Ausführungsformen separate Pipelines für bestimmte Typen von Daten/Operationen erstellen (z. B. eine skalare Ganzzahl-Pipeline, eine skalare Gleitkomma-/gepackte Ganzzahl-/gepackte Gleitkomma-/vektorielle Ganzzahl-/vektorielle Gleitkomma-Pipeline und/oder eine Speicherzugriffs-Pipeline, die jeweils ihre eigene Scheduler-Einheit, ihre eigene Einheit physikalischer Registerdatei(en) und/oder ihren eigenen Ausführungs-Cluster aufweisen – und im Falle einer separaten Speicherzugriffs-Pipeline sind bestimmte Ausführungsformen implementiert, in welche nur der Ausführungs-Cluster dieser Pipeline die Speicherzugriffseinheit(en) QAE64 aufweist). Es versteht sich außerdem von selbst, dass bei Verwenden separater Pipelines eine oder mehrere dieser Pipelines mit Out-of-Order- und die restlichen mit In-Order-Ausführung sein können.

[0060] Der Satz von Speicherzugriffseinheiten QAE64 ist mit der Speichereinheit QAE70 gekoppelt, welche eine Daten-TLB-Einheit QAE72 umfasst, die mit einer Daten-Cache-Einheit QAE74 gekoppelt ist, die mit einer Cache-Einheit QAE76 der Ebene 2 (L2) gekoppelt ist. In einer beispielhaften Ausführungsform können die Speicherzugriffseinheiten QAE64 eine Lasteinheit, eine Speicheradresseinheit und eine Speicherdateneinheit umfassen, die jeweils mit der Daten-TLB-Einheit QAE72 in der Speichereinheit QAE70 gekoppelt sind. Die Befehls-Cache-Einheit QAE34 ist ferner mit der Cache-Einheit QAE76 der Ebene 2 (L2) in der Speichereinheit QAE70 gekoppelt. Die L2-Cache-Einheit QAE76 ist mit einer oder mehreren anderen Cache-Ebenen und schließlich mit einem Hauptspeicher gekoppelt.

[0061] Als Beispiel kann die beispielhafte Registerumbenennungs- und Out-of-Order-Ausgabe-/Ausführungs-Kernarchitektur die Pipeline AQE00 folgendermaßen implementieren: 1) die Befehlsabrufeinheit QAE38 führt die Abruf- und Längendecodierstufen QAE02 und QAE04 aus; 2) die Decodiereinheit QAE40 führt die Decodierstufe QAE06 aus; 3) die Umbenennungs-/Zuordnungseinheit QAE52 führt die Zuordnungsstufe QAE08 und die Umbenennungsstufe QAE10 aus; 4) die Scheduler-Einheit(en) QAE56 führt/führen die Ablaufsteuerungsstufe QAE12 aus; 5) die Einheit(en) physikalischer Registerdatei(en) QAE58 und die Speichereinheit QAE70 führen die Registerlese-/Speicherlesestufe QAE14 aus; der Ausführungs-Cluster QAE60 führt die Ausführungsstufe QAE16 aus; 6) die Speichereinheit QAE70 und die Einheit(en) physikalischer Registerdatei(en) QAE58 führen die Zurückschreib-/Speicherschreibstufe QAE18 aus; 7) verschiedene Einheiten können an der Ausnahmebehandlungsstufe QAE22 beteiligt sein; und 8) die Ausscheideeinheit QAE54 und die Einheit(en) physikalischer Registerdatei(en) QAE58 führen die Commit-Stufe QAE24 aus.

[0062] Der Kern QAE90 kann einen oder mehrere Befehlssätze (z. B. den x86-Befehlssatz (mit einigen Erweiterungen, welche neueren Version hinzugefügt wurden); den MIPS-Befehlssatz von MIPS Technologies, Sunnyvale, Kalifornien; den ARM-Befehlssatz (mit optionalen zusätzlichen Erweiterungen, wie beispielsweise NEON) von der ARM Holdings, Sunnyvale, Kalifornien), einschließlich des hierin beschrieben Befehls bzw. der hierin beschriebenen Befehle, unterstützen. In einer Ausführungsform umfasst der Kern QAE90 Logik zur Unterstützung einer Befehlssatzerweiterung für gepackte Daten (z. B. AVX1, AVX2 und/oder irgendeine Form des zuvor beschriebenen generischen vektorfreundlichen Befehlsformats (U = 0 und/oder U = 1)), um dadurch

zu ermöglichen, dass die durch viele Multimedia-Anwendungen verwendeten Operationen unter Verwendung von gepackten Daten ausgeführt werden.

[0063] Es versteht sich von selbst, dass der Kern Multithreading (Ausführen von zwei oder mehr parallelen Sätzen von Operationen oder Threads) unterstützten kann und dies auf verschiedene Arten und Weisen tun kann, welche Zeitscheiben-Multithreading, simultanes Multithreading (wobei ein einziger physikalischer Kern einen logischen Kern für jeden der Threads bereitstellt, für die der physikalische Kern simultanes Multithreading ausführt) oder eine Kombination davon (z. B. Zeitscheiben-Abruf und -Decodierung und anschließendes simultanes Multithreading, wie beispielsweise in der Intel® Hyperthreading-Technologie) umfassen.

[0064] Es versteht sich von selbst, dass, obwohl Registerumbenennung im Kontext von Out-of-Order-Ausführung beschrieben wird, Registerumbenennung auch in einer In-order-Architektur verwendet werden kann. Obwohl die veranschaulichte Ausführungsform des Prozessors getrennte Befehls- und Daten-Cache-Einheiten QAE34/QAE74 und eine gemeinsam benutzte L2-Cache-Einheit QAE76 umfasst, können alternative Ausführungsformen einen einzigen internen Cache sowohl für Befehle als auch Daten aufweisen, wie beispielsweise einen internen Cache der Ebene 1 (L1) oder einen internen Cache mehrerer Ebenen. In einigen Ausführungsformen kann das System eine Kombination eines internen Caches und eines externen Caches aufweisen, der außerhalb des Kerns und/oder des Prozessors ist. Alternativ können alle der Caches außerhalb des Kerns und/oder des Prozessors sein.

Spezifische beispielhafte In-Order-Kernarchitektur

[0065] **Fig.** 8(A) und (B) veranschaulichen ein Blockdiagramm einer spezifischeren beispielhaften In-Order-Kernarchitektur, wobei der Kern einer von mehreren Logikblöcken (welche andere Kerne des gleichen Typs und/oder verschiedener Typen umfassen) in einem Chip wäre. Die Logikblöcke kommunizieren durch ein Zwischenverbindungsnetzwerk mit hoher Bandbreite (z. B. ein Ringnetzwerk) mit Logik mit irgendeiner bestimmten Funktion, Speicher-I/O-Schnittstellen und anderer erforderlichen I/O-Logik je nach der Anwendung.

[0066] **Fig.** 8A ist ein Blockdiagramm eines einzigen Prozessorkerns zusammen mit seiner Verbindung zum chipinternen Zwischenverbindungsnetzwerk **802** und mit seinem lokalen Teilsatz des Caches **804** der Ebene 2 (L2) gemäß Ausführungsformen der Erfindung. In einer Ausführungsform unterstützt ein Befehlsdecoder **800** den x86-Befehlssatz mit einer Befehlssatzerweiterung für gepackte Daten. Ein L1-Cache **806** ermöglicht Zugriffe mit kurzen Latenzzeiten auf Cache-Speicher in die Skalar- und Vektoreinheiten. Obwohl in einer Ausführungsform (zur Vereinfachung des Aufbaus) eine Skalareinheit **808** und eine Vektoreinheit **810** separate Registersätze (Skalarregister **812** bzw. Vektorregister **814**) verwenden, und zwischen ihnen übertragene Daten in den Speicher geschrieben und dann aus einem Cache **806** der Ebene 1 (L1) zurückeingelesen werden, können alternative Ausführungsformen der Erfindung einen anderen Ansatz verwenden (z. B. einen einzigen Registersatz verwenden oder einen Kommunikationspfad einbeziehen, der es ermöglicht, dass Daten zwischen den beiden Registerdateien übertragen werden, ohne geschrieben und zurückgelesen zu werden).

[0067] Der lokale Teilsatz des L2-Caches **804** ist Teil eines globalen L2-Caches, der in separate lokale Teilsätze, einen pro Prozessorkern, geteilt ist. Jeder Prozessorkern weist einen Direktzugriffspfad zu seinem eigenen lokalen Teilsatz des L2-Caches **804** auf. Daten, die durch einen Prozessorkern ausgelesen werden, werden in seinem L2-Cache-Teilsatz **804** gespeichert, und parallel zu anderen Prozessoren, die auf ihre eigenen lokalen L2-Cache-Teilsätze zugreifen, kann schnell darauf zugegriffen werden. Daten, die durch einen Prozessorkern geschrieben werden, werden in seinem eigenen L2-Cache-Teilsatz **804** gespeichert und nötigenfalls aus anderen Teilsätzen entleert. Das Ringnetzwerk gewährleistet Kohärenz für gemeinsam benutzte Daten. Das Ringnetzwerk ist bidirektional, um Agenten, wie beispielsweise Prozessorkernen, L2-Caches und anderen Logikblöcken, zu ermöglichen, innerhalb des Chips miteinander zu kommunizieren. Jeder Ring-Datenpfad ist pro Richtung **1012** Bits breit.

[0068] **Fig.** 8B ist eine erweiterte Ansicht des Prozessorkerns in **Fig.** 8A gemäß Ausführungsformen der Erfindung. **Fig.** 8B umfasst einen L1-Daten-Cache **806A**, der ein Teil des L1-Caches **804** ist, sowie genauere Einzelheiten bezüglich der Vektoreinheit **810** und der Vektorregister **814**. Konkret ist die Vektoreinheit **810** eine 16 Bit breite Vektorverarbeitungseinheit (VPU für engl. vector processing unit) (siehe 16 Bit breite ALU **828**), welche einen oder mehrere von Ganzzahlbefehlen, Gleitkommabefehlen mit einfacher Genauigkeit und Gleitkommabefehlen mit doppelter Genauigkeit ausführt. Die VPU unterstützt Umordnen der Registereingänge mit einer Umordnungseinheit **820**, numerische Umsetzung mit Einheiten für numerische Umsetzung **822-A** und Duplizierung mit einer Dupliziereinheit **824** am Speichereingang.

Prozessor mit integrierter Speichersteuereinheit und Grafik

[0069] **Fig.** 9 ist ein Blockdiagramm eines Prozessors **900**, der mehr als einen Kern aufweisen kann, eine integrierte Speichersteuereinheit aufweisen kann, und integrierte Grafik gemäß Ausführungsformen der Erfindung aufweisen kann. Die Kästchen mit durchgehenden Linien in **Fig.** 9 veranschaulichen einen Prozessor **900** mit einem einzigen Kern **902A**, einer Systemagenteneinheit **910**, einem Satz von einer oder mehreren Bussteuereinheiten **916**, während die optionale Hinzufügung der Kästchen mit gestrichelten Linien einen alternativen Prozessor **900** mit mehreren Kernen **902A**–N, einen Satz von einer oder mehreren integrierten Speichersteuereinheit(en) **914** in der Systemagenteneinheit **910** und Speziallogik **908** umfasst.

[0070] Demnach können verschiedene Implementierungen des Prozessor **900** Folgendes umfassen: 1) eine CPU mit der Speziallogik **908**, wobei es sich um integrierte Grafik- und/oder wissenschaftliche (Durchsatz-) Logik handelt (welche einen oder mehrere Kerne umfassen kann), und die Kerne **902A**–N, wobei es sich um einen oder mehrere Universalkerne handelt (z. B. In-Order-Universalkerne, Out-of-Order-Universalkerne, eine Kombination der beiden); 2) einen Coprozessor mit den Kernen **902A**–N, wobei es sich um eine große Anzahl von Spezialkernen handelt, die in erster Linie für Grafik und/oder Wissenschaft (Durchsatz) bestimmt sind; und 3) einen Coprozessor mit den Kernen **902A**–N, wobei es sich um eine große Anzahl von In-Order-Universalkernen handelt. Demnach kann der Prozessor **900** ein Universalprozessor, ein Coprozessor oder Spezialprozessor, wie beispielsweise ein Netzwerk- oder Kommunikationsprozessor, eine Kompressionsmaschine, ein Grafikprozessor, eine GPGPU (Universal-Grafikverarbeitungseinheit), ein Coprozessor mit hohem Durchsatz und vielen integrierten Kernen (MIC für engl. many integrated core) (der 30 oder mehr Kerne umfasst), ein eingebetteter Prozessor oder dergleichen sein. Der Prozessor kann auf einem oder mehreren Chips implementiert sein. Der Prozessor **900** kann Teil eines oder mehrerer Substrate sein und/oder er kann auf einem oder mehreren Substraten unter Verwendung einer beliebigen von einer Anzahl von Prozesstechnologien, wie beispielsweise BiCMOS, CMOS oder NMOs, implementiert sein.

[0071] Die Speicherhierarchie umfasst eine oder mehrere Cache-Ebenen innerhalb der Kerne, einen Satz oder einen oder mehrere gemeinsam benutzte Cache-Einheiten **906** und einen externen Speicher (nicht dargestellt), der mit dem Satz von integrierten Speichersteuereinheiten **914** gekoppelt ist. Der Satz der gemeinsam benutzten Cache-Einheiten **906** kann einen oder mehrere Caches mittlerer Ebene, wie beispielsweise Ebene 2 (L2), Ebene 3 (L3), Ebene 4 (L4) oder anderer Cache-Ebenen, einen Cache der letzten Ebene (LLC für engl. last level cache) und/oder Kombinationen davon umfassen. Obwohl in einer Ausführungsform eine ringbasierte Zwischenverbindungseinheit **912** die integrierte Grafiklogik **908**, den Satz von gemeinsam benutzten Cache-Einheiten **906** und die Systemagenteneinheit **910** bzw. die integrierten Speichersteuereinheit(en) **914** miteinander verbindet, können alternative Ausführungsformen eine beliebige Anzahl von allgemein bekannten Techniken für die Verbindung solcher Einheiten miteinander verwenden. In einer Ausführungsform wird Kohärenz zwischen einer oder mehreren Cache-Einheiten **906** und den Kernen **902A**–N aufrechterhalten.

[0072] In einigen Ausführungsformen sind ein oder mehrere der Kerne **902A**–N zu Multithreading imstande. Der Systemagent **910** umfasst jene Komponenten, welche die Kerne **902A**–N koordinieren und steuern. Die Systemagenteneinheit **910** kann zum Beispiel eine Leistungssteuerungseinheit (PCU für engl. power control unit) und eine Anzeigeeinheit umfassen. Die PCU kann Logik sein oder Logik und Komponenten umfassen, die zum Regeln des Leistungszustands der Kerne **902A**–N und der integrierten Grafiklogik **908** benötigt werden. Die Anzeigeeinheit ist zum Ansteuern einer oder mehrerer extern angeschlossener Anzeigen.

[0073] Die Kerne **902A**–N können homogen oder heterogen in Bezug auf den Architekturbefehlssatz sein; das heißt, zwei oder mehr der Kerne **902A**–N können zur Ausführung des gleichen Befehlssatzes imstande sein, während andere nur zur Ausführung eines Teilsatzes dieses Befehlssatzes oder eines anderen Befehlssatzes imstande sein können.

Beispielhafte Computerarchitekturen

[0074] **Fig.** 10 bis **Fig.** 13 sind Blockdiagramme von beispielhaften Computerarchitekturen. Andere auf dem Fachgebiet bekannte Systemkonstruktionen und -konfigurationen für Laptops, Tischcomputer, Hand-PCs, persönliche digitale Assistenten, Engineering-Workstations, Server, Netzwerkgeräte, Netzkontrollstationen, Switches, eingebettete Prozessoren, Digitalsignalprozessoren (DSPs), Grafikgeräte, Videospielgeräte, Set-Top-Boxen, Mikro-Controller, Zellulartelefone, tragbare Media Player, Hand-Geräte und verschiedene andere elektronische Geräte sind ebenfalls geeignet. Im Allgemeinen eignet sich grundsätzlich eine große Vielzahl von Systemen oder elektronischen Geräten, die zum Integrieren eines Prozessors und/oder anderer Ausführungslogik, wie hierin offenbart, imstande sind.

[0075] Nunmehr unter Bezugnahme auf **Fig.** 10 ist ein Blockdiagramm eines Systems **1000** gemäß einer Ausführungsform der vorliegenden Erfindung dargestellt. Das System **1000** kann einen oder mehrere Prozessoren **1010**, **1015** umfassen, welche mit einem Steuerhub **1020** gekoppelt sind. In einer Ausführungsform umfasst der Steuerhub **1020** einen Grafikspeicher-Steuerhub (GMCH für engl. graphics memory controller hub) **1090** und einen Eingangs-/Ausgangs-Hub (IOH für engl. Input/Output Hub) **1050** (die auf separaten Chips sein können); der GMCH **1090** umfasst Speicher- und Grafiksteuereinheiten, mit welchen ein Speicher **1040** und ein Coprozessor **1045** gekoppelt sind; der IOH **1050** koppelt Eingabe-/Ausgabe(I/O)-Einrichtungen **1060** mit dem GMCH **1090**. Alternativ sind eine oder beide der Speicher- und Grafiksteuereinheiten in den Prozessor integriert (wie hierin beschrieben), der Speicher **1040** und der Coprozessor **1045** sind direkt mit dem Prozessor **1010** gekoppelt, und der Steuerhub **1020** in einem einzigen Chip mit dem IOH **1050**.

[0076] Die optionale Beschaffenheit von zusätzlichen Prozessoren **1015** ist in **Fig.** 10 durch gestrichelte Linien dargestellt. Jeder Prozessor **1010**, **1015** kann einen oder mehrere der hierin beschriebenen Prozessorkerne umfassen und kann eine Version des Prozessors **900** sein.

[0077] Der Speicher **1040** kann zum Beispiel ein dynamischer Direktzugriffsspeicher (DRAM für engl. dynamic random access memory), ein Phasenwechselspeicher (PCM für engl. phase change memory) oder eine Kombination der beiden sein. Für mindestens eine Ausführungsform kommuniziert der Steuerhub **1020** mit dem/den Prozessor(en) **1010**, **1015** über einen Mehrpunktbus, wie beispielsweise einem Frontside-Bus (FSB), eine Punkt-zu-Punkt-Schnittstelle, wie beispielsweise eine QuickPath Interconnect (QPI), oder eine ähnliche Verbindung **1095**.

[0078] In einer Ausführungsform ist der Coprozessor **1045** ein Spezialprozessor, wie beispielsweise ein MIC-Prozessor mit hohem Durchsatz, ein Netzwerk- oder Kommunikationsprozessor, eine Kompressionsmaschine, ein Grafikprozessor, eine GPGPU, ein eingebetteter Prozessor oder dergleichen. In einer Ausführungsform kann der Steuerhub **1020** einen integrierten Grafikbeschleuniger umfassen.

[0079] Es kann eine Vielzahl von Unterschieden zwischen den physikalischen Betriebsmitteln **1010**, **1015** in Bezug auf ein Spektrum von Bewertungsmetriken geben, die architektonische, mikroarchitektonische, thermische und Leistungsverbrauchscharakteristiken und dergleichen umfassen.

[0080] In einer Ausführungsform führt der Prozessor **1010** Befehle aus, welche Datenverarbeitungsoperationen eines allgemeinen Typs steuern. In die Befehle können Coprozessorbefehle eingebettet sein. Der Prozessor **1010** erkennt diese Coprozessorbefehle als einen Typ, der vom angeschlossenen Coprozessor **1045** ausgeführt werden sollte. Demgemäß gibt der Prozessor **1010** diese Coprozessorbefehle (oder Steuersignale, welche die Coprozessorbefehle darstellen) auf einem Coprozessorbus oder einer anderen Zwischenverbindung an den Coprozessor **1045** aus. Der oder die Coprozessor(en) **1045** nehmen die empfangenen Coprozessorbefehle an und führen sie aus.

[0081] Nunmehr unter Bezugnahme auf **Fig.** 11 ist ein Blockdiagramm eines ersten spezifischeren beispielhaften Systems **1100** gemäß einer Ausführungsform der vorliegenden Erfindung dargestellt. Wie in **Fig.** 11 dargestellt, ist das Multiprozessorsystem **1100** ist ein Punkt-zu-Punkt-Zwischenverbindungssystem **1170** und umfasst einen ersten Prozessor **1170** und einen zweiten Prozessor **1180**, die über eine Punkt-zu-Punkt-Zwischenverbindung **1150** gekoppelt sind. Jeder der Prozessoren **1170** und **1180** können eine Version des Prozessors **900** sein. In einer Ausführungsform der Erfindung sind die Prozessoren **1170** und **1180** sind Prozessoren **1010** bzw. **1015**, während ein Coprozessor **1138** ist ein Coprozessor **1045** ist. In einer Ausführungsform sind die Prozessoren **1170** und **1180** ein Prozessor **1010** bzw. ein Coprozessor **1045**.

[0082] Die Prozessoren **1170** und **1180** sind so dargestellt, dass sie integrierte Speichersteuer(IMC für engl. integrated memory controller)-Einheiten **1172** bzw. **1182** aufweisen. Der Prozessor **1170** umfasst außerdem als Teil seiner Bussteuereinheiten Punkt-zu-Punkt(P-P)-Schnittstellen **1176** und **1178**; ähnlich umfasst der zweite Prozessor **1180** P-P-Schnittstellen **1186** und **1188**. Die Prozessoren **1170**, **1180** können über eine Punkt-zu-Punkt(P-P)-Schnittstelle **1150** unter Verwendung von P-P-Schnittstellenschaltungen **1178**, **1188** Informationen austauschen. Wie in **Fig.** 11 dargestellt, koppeln die IMCs **1172** und **1182** die Prozessoren mit jeweiligen Speichern, nämlich mit einem Speicher **1132** und einem Speicher **1134**, welche Abschnitte eines Hauptspeichers sein können, der lokal an die jeweiligen Prozessoren angeschlossen ist.

[0083] Die Prozessoren **1170**, **1180** können jeweils über individuelle P-P-Schnittstellen P-P **1152**, **1154** unter Verwendung von Punkt-zu-Punkt-Schnittstellenschaltungen **1176**, **1194**, **1186**, **1198** Informationen mit einem Chipsatz **1190** austauschen. Der Chipsatz **1190** kann optional über eine Hochleistungsschnittstelle **1139** Infor-

mationen mit dem Coprozessor **1138** austauschen. In einer Ausführungsform ist der Coprozessor **1138** ein Spezialprozessor, wie beispielsweise ein MIC-Prozessor mit hohem Durchsatz, ein Netzwerk- oder Kommunikationsprozessor, eine Kompressionsmaschine, ein Grafikprozessor, eine GPGPU, ein eingebetteter Prozessor oder dergleichen.

[0084] Ein gemeinsam benutzter Cache (nicht dargestellt) kann in jedem Prozessor enthalten oder außerhalb von beiden Prozessoren, aber dennoch über eine P-P-Zwischenverbindung mit den Prozessoren verbunden sein, derart dass Informationen des lokalen Caches eines oder beider Prozessoren im gemeinsam benutzten Cache gespeichert werden können, wenn ein Prozessor in einen Stromsparmodus versetzt wird.

[0085] Der Chipsatz **1190** kann über eine Schnittstelle **1196** mit einem ersten Bus **1116** gekoppelt sein. In einer Ausführungsform kann der erste Bus **1114** ein Bus zur Verbindung von Peripheriekomponenten (PCI für engl. Peripheral Component Interconnect) oder solch ein Bus wie beispielsweise ein PCI Express-Bus oder ein anderer I/O-Zwischenverbindungsbus der dritten Generation sein, obwohl der Schutzumfang der vorliegenden Erfindung nicht darauf beschränkt ist.

[0086] Wie in **Fig.** 11 dargestellt, können verschiedene I/O-Einrichtungen **1114** mit dem ersten Bus **1116** zusammen mit einer Busbrücke **1118** gekoppelt sein, welche den ersten Bus **1116** mit einem zweiten Bus **1120** koppelt. In einer Ausführungsform sind ein oder mehrere zusätzliche Prozessor(en) **1115**, wie beispielsweise Coprozessoren, MIC-Prozessoren mit hohem Durchsatz, GPGPUs, Beschleuniger (wie etwa z. B. Grafikbeschleuniger oder Digitalsignalverarbeitungs(DSP)-Einheiten), feldprogrammierbare Gate-Arrays oder jeglicher andere Prozessor, mit dem ersten Bus **1116** gekoppelt. In einer Ausführungsform kann der zweite Bus **1120** ein Bus mit geringer Anschlusszahl (LPC für engl. low pin count) sein. Verschiedene Einrichtungen können an einen zweiten Bus **1120** angeschlossen sein, die in einer Ausführungsform zum Beispiel eine Tastatur und/oder eine Maus **1122**, Kommunikationseinrichtungen **1127** und eine Speichereinheit **1128**, wie beispielsweise ein Plattenlaufwerk oder eine Massenspeichereinrichtung, umfassen, die Befehle/Code und Daten **1130** enthalten kann. Ferner kann Audio-I/O mit dem zweiten Bus **1120** gekoppelt sein. Es ist zu erwähnen, dass auch eine andere Architektur möglich ist. Zum Beispiel kann ein System anstelle der Punkt-zu-Punkt-Architektur von **Fig.** 11 eine Mehrpunktbus- oder andere derartige Architektur implementieren.

[0087] Nunmehr unter Bezugnahme auf **Fig.** 12 ist ein Blockdiagramm eines zweiten spezifischeren beispielhaften Systems **1200** gemäß einer Ausführungsform der vorliegenden Erfindung dargestellt. Gleiche Elemente in **Fig.** 11 und **Fig.** 12 tragen gleiche Bezugszeichen, und bestimmte Aspekte von **Fig.** 11 wurden in **Fig.** 12 weggelassen, um ein Verkomplizieren anderer Aspekte von **Fig.** 12 zu vermeiden.

[0088] **Fig.** 12 veranschaulicht, dass die Prozessoren **1170**, **1180** integrierte Speicher und I/O-Steuerlogik („CL" für engl. control logic) **1172** bzw. **1182** umfassen können. Demnach umfassen die CL **1172**, **1182** integrierte Speichersteuereinheiten und I/O-Steuerlogik. **Fig.** 12 veranschaulicht, dass nicht nur die Speicher **1132**, **1134** mit der CL **1172**, **1182** gekoppelt sind, sondern dass auch I/O-Einrichtungen **1214** mit der Steuerlogik **1172**, **1182** gekoppelt sind. Alte I/O-Einrichtungen **1215** sind mit dem Chipsatz **110** gekoppelt.

[0089] Nunmehr unter Bezugnahme auf **Fig.** 13 ist ein Blockdiagramm eines Systemchips (SoC für engl. system an a chip) **1300** gemäß einer Ausführungsform der vorliegenden Erfindung dargestellt. Ähnliche Elemente in **Fig.** 9 tragen gleiche Bezugszeichen. Außerdem sind Kästchen mit gestrichelten Linien optionale Funktionen auf weiterentwickelten SoCs. In **Fig.** 13 sind eine oder mehrere Zwischenverbindungseinheiten **1302** mit Folgendem gekoppelt: einem Anwendungsprozessor **1310**, der einen Satz von einem oder mehreren Kernen **202A**–N und einer oder mehreren gemeinsam benutzte Cache-Einheit(en) **906** umfasst; einer Systemagenteneinheit **910**; Bussteuereinheit(en) **916**; integrierten Speichersteuereinheit(en) **914**; einem Satz oder einem oder mehreren Coprozessoren **1320**, welche integrierte Grafiklogik, einen Bildprozessor, einen Audioprozessor und einen Videoprozessor umfassen können; einer statischen Direktzugriffsspeicher(SRAM für engl. static random access memory)-Einheit **1330**; einer Speicherdirektzugriffs(DMA für engl. direct memory access)-Einheit **1332**; und einer Anzeigeeinheit **1340** zur Kopplung mit einer oder mehreren externen Anzeigen. In einer Ausführungsform umfasst/umfassen der/die Coprozessor(en) **1320** einen Spezialprozessor, wie beispielsweise einen Netzwerk- oder Kommunikationsprozessor, eine Kompressionsmaschine, eine GPGPU, einen MIC-Prozessor mit hohem Durchsatz, einen eingebetteten Prozessor oder dergleichen.

[0090] Ausführungsformen der hierin offenbarten Mechanismen können in Hardware, Software, Firmware oder einer Kombination solcher Implementierungslösungen implementiert sein. Ausführungsformen der Erfindung können als Computerprogramme oder Programmcode implementiert sein, die auf programmierbaren Systemen ausgeführt werden, die mindestens einen Prozessor, ein Speichersystem (das flüchtige und nicht-

flüchtige Speicher und/oder Speicherelemente umfasst), mindestens eine Eingabevorrichtung und mindestens eine Ausgabevorrichtung umfassen.

[0091] Programmcode, wie beispielsweise der in **Fig.** 11 veranschaulichte Code **1130**, kann auf Eingabebefehle angewendet werden, um die hierin beschriebenen Funktionen auszuführen und Ausgabeinformationen zu generieren. Die Ausgabeinformationen können in bekannter Weise auf eine oder mehrere Ausgabevorrichtungen angewendet werden. Zum Zwecke dieser Anmeldung umfasst ein Verarbeitungssystem jedes System, das einen Prozessor, wie beispielsweise einen Digitalsignalprozessor (DSP), einen Mikrocontroller, eine anwendungsspezifische integrierte Schaltung (ASIC für engl. application specific integrated circuit) oder einen Mikroprozessor, aufweist.

[0092] Der Programmcode kann in einer verfahrens- oder objektorientierten Programmiersprache implementiert sein, um mit einem Verarbeitungssystem zu kommunizieren. Der Programmiercode kann außerdem in Assembler- oder Maschinensprache implementiert sein, falls gewünscht. Tatsächlich sind die hierin beschriebenen Mechanismen nicht auf eine bestimmte Programmiersprache beschränkt. Auf jeden Fall kann die Sprache eine kompilierte oder interpretierte Sprache sein.

[0093] Ein oder mehrere Aspekte mindestens einer Ausführungsform können durch repräsentative Befehle implementiert sein, die auf einem maschinenlesbaren Medium gespeichert sind, welches diverse Logik innerhalb des Prozessors darstellt, und die, wenn durch eine Maschine ausgelesen, die Maschine veranlassen, Logik zu erstellen, um die hierin beschriebenen Techniken auszuführen. Solche Darstellungen, die als „IP-Kerne" bekannt sind, können auf einem gegenständlichen, maschinenlesbaren Medium gespeichert sein und an verschiedenen Kunden oder Produktionsstätten geliefert werden, um sie in die Fertigungsmaschinen zu laden, welche die Logik oder den Prozessor tatsächlich erzeugen.

[0094] Solche maschinenlesbaren Speichermedien können, ohne darauf beschränkt zu sein, Folgende umfassen: nicht-transitorische, gegenständliche Anordnungen von Gegenständen, die durch eine Maschine oder ein Gerät hergestellt oder gebildet sind, einschließlich Speichermedien, wie beispielsweise Festplatten, jeglichen anderen Typs von Platten, einschließlich Disketten, optischer Platten, Compact-Disk-Festwertspeicher (CD-ROMs für engl. compact disk read-only memories), wiederbeschreibbarer Compact-Disk (CD-RWs für engl. compact disk rewritable's) und magnetooptischer Platten, Halbleiterbauelemente wie beispielsweise Festwertspeicher (ROMs für engl. read-only memories), Direktzugriffsspeicher (RAMs für engl. random access memories), wie beispielsweise dynamische Direktzugriffsspeicher (DRAMs für engl. dynamic random access memories), statische Direktzugriffsspeicher (SRAMs für engl. static random access memories), löschbare programmierbare Festwertspeicher (EPROMs für engl. erasable programmable read-only memories), Flash-Speicher, elektrisch löschbare programmierbare Festwertspeicher (EEPROMs für engl. electrically erasable programmable read-only memories), Phasenwechselspeicher (PCM für engl. phase change memory), magnetische oder optische Karten oder jeden anderen Typ von Medien, die zum Speichern von elektronischen Befehlen imstande sind.

[0095] Demgemäß umfassen Ausführungsformen der Erfindung auch nicht-transitorische, gegenständliche maschinenlesbare Medien, welche Befehle enthalten oder welche Entwurfsdaten enthalten, wie beispielsweise Hardwarebeschreibungssprache (HDL für engl. Hardware Description Language), welche hierin beschriebene Strukturen, Schaltungen, Vorrichtungen, Prozessoren und/oder Systemfunktionen definiert. Solche Ausführungsformen können auch als Programmprodukte bezeichnet werden.

Emulation (einschließlich Binärübersetzung, Code-Morphing usw.)

[0096] In einigen Fällen kann ein Befehlsumsetzer zum Umsetzen eines Befehls von einem Quellenbefehlssatz in einen Zielbefehlssatz verwendet werden. Zum Beispiel kann der Befehlsumsetzer einen Befehl in einen oder mehrere durch den Kern zu verarbeitende Befehle übersetzen (z. B. unter Verwendung von statischer Binärübersetzung, dynamischer Binärübersetzung, einschließlich dynamischer Kompilierung), morphen, emulieren oder anderweitig umsetzen. Der Befehlsumsetzer kann in Software, Hardware, Firmware oder einer Kombination davon implementiert sein. Der Befehlsumsetzer kann prozessorintern, prozessorextern oder teilweise prozessorintern und teilweise prozessorextern sein.

[0097] Fig. 14 ist ein Blockdiagramm, das die Verwendung eines Softwarebefehlsumsetzers zum Umsetzen von Binärbefehlen in einem Quellenbefehlssatz in Binärbefehle in einem Zielbefehlssatz gemäß Ausführungsformen der Erfindung vergleicht. In der veranschaulichten Ausführungsform ist der Befehlsumsetzer ein Softwarebefehlsumsetzer, obwohl der Befehlsumsetzer alternativ in Software, Firmware, Hardware oder verschie-

denen Kombinationen davon implementiert werden kann. **Fig.** 14 stellt ein Programm in einer höheren Programmiersprache **1402** dar, die unter Verwendung eines x86-Compilers **1404** kompiliert werden kann, um x86-Binärcode **1406** zu generieren, der durch einen Prozessor mit mindestens einem x86-Befehlssatzkern **1416** nativ ausgeführt werden kann. Der Prozessor mit mindestens einen x86-Befehlssatzkern **1416** stellt jeden Prozessor dar, der im Wesentlichen die gleichen Funktionen wie ein Intel Prozessor mit mindestens einem x86-Bfehlssatzkern durch kompatibles Ausführen oder anderweitiges Verarbeiten (1) eines wesentlichen Teils des Befehlssatzes des Intel x86-Befehlssatzkerns oder (2) von Objektcodeversionen von Anwendungen oder anderer Software, die auf einem Intel Prozessor mit mindestens einem x86-Befehlssatzkern ausgeführt werden soll, ausführen kann, um im Wesentlichen das gleiche Ergebnis wie ein Intel Prozessor mit mindestens einem x86-Befehlssatzkern zu erzielen. Der x86-Compiler **1404** stellt einen Compiler dar, der so betrieben werden kann, dass er x86-Binärcode **1406** (z. B. Objektcode) generiert, der mit oder ohne zusätzliche Verknüpfungsverarbeitung, auf dem Prozessor mit mindestens einem x86-Befehlssatzkern **1416** ausgeführt werden kann. Ähnlich stellt **Fig.** 14 das Programm in der höheren Programmiersprache **1402** dar, die unter Verwendung eines alternativen Befehlssatz-Compilers **1408** kompiliert werden kann, um alternativen Befehlssatz-Binärcode **1410** zu generieren, der durch einen Prozessor ohne mindestens einen x86-Befehlssatzkern **1414** (z. B. einen Prozessor mit Kernen, die den MIPS-Befehlssatz von MIPS Technologies, Sunnyvale, Kalifornien, ausführen, und/oder die den ARM-Befehlssatz der ARM Holdings, Sunnyvale, Kalifornien, ausführen) nativ ausgeführt werden kann. Der Befehlsumsetzer **1412** wird zum Umsetzen des x86-Binärcodes **1406** in Code verwendet, der durch den Prozessor ohne x86-Befehlssatzkern **1414** nativ ausgeführt werden kann. Dieser umgesetzte Code ist wahrscheinlich nicht der gleiche wie der alternative Befehlssatz-Binärcode **1410**, da ein Befehlsumsetzer, der dazu imstande ist, schwer herzustellen ist; der umgesetzte Code führt jedoch die allgemeine Operation aus und besteht aus Befehlen aus dem alternativen Befehlssatz. Demnach stellt der Befehlsumsetzer **1412** Software, Firmware, Hardware oder eine Kombination davon dar, die durch Emulation, Simulation oder einen beliebigen anderen Prozess einen Prozessor oder eine andere elektronische Einrichtung ermöglicht, die keinen x86-Befehlssatzprozessor oder -kern zum Ausführen des x86-Binärcodes **1406** aufweist.

ZITATE ENTHALTEN IN DER BESCHREIBUNG

Diese Liste der vom Anmelder aufgeführten Dokumente wurde automatisiert erzeugt und ist ausschließlich zur besseren Information des Lesers aufgenommen. Die Liste ist nicht Bestandteil der deutschen Patent- bzw. Gebrauchsmusteranmeldung. Das DPMA übernimmt keinerlei Haftung für etwaige Fehler oder Auslassungen.

Zitierte Patentliteratur

- US 5446912 [0018]
- US 5207132 [0018]

Patentansprüche

1. Vorrichtung, umfassend:
Decodierlogik zum Decodieren eines Nullmaske-vor-folgender-Null(KZBTZ)-Befehls, wobei der KZBTZ-Befehl einen ersten Quellenschreibmaskenoperanden und einen Zielmaskenoperanden umfasst;
Ausführungslogik zum Ausführen des decodierten KZBTZ-Befehls, um eine folgende niedrigstwertige Nullbitposition in der ersten Quellenschreibmaske zu ermitteln und die Zielschreibmaske so zu setzen, dass sie die Werte der ersten Quellenschreibmaske aufweist, aber mit allen Bitpositionen näher zur höchstwertigen Bitposition als die folgende niedrigstwertige Nullbitposition in einer Quellenschreibmaske, die auf null gesetzt ist.

2. Vorrichtung nach Anspruch 1, wobei der KZBTZ-Befehl ferner einen zweiten Quellenschreibmaskenoperanden umfasst, und die folgende niedrigstwertige Nullbitposition eine erste Bitposition im Quellenschreibmaskenoperanden ist, die auf null gesetzt ist, wenn eine entsprechende Bitposition des zweiten Quellenschreibmaskenoperanden auf eins gesetzt ist.

3. Vorrichtung nach Anspruch 1 oder 2, wobei die Schreibmaskenoperanden dedizierte Schreibmaskenregister sind.

4. Vorrichtung nach Anspruch 3, wobei die dedizierten Schreibmaskenregister 8 oder 16 Bit groß sind.

5. Vorrichtung nach Anspruch 1 oder 2, wobei die Schreibmaskenoperanden Universalregister sind.

6. Vorrichtung nach Anspruch 1 bis 5, wobei die Ausführungslogik jede Bitposition des ersten Quellenoperanden seriell von der niedrigstwertigen bis zur höchstwertigen auswertet.

7. Vorrichtung nach Anspruch 6, wobei durch die Ausführungslogik ein Zähler verwendet wird, um zu bestimmen, wann alle der Bitpositionen ausgewertet wurden.

8. Verfahren zum Ausführen eines Nullmaske-vor-folgender-Null(KZBTZ)-Befehls in einem Computerprozessor, wobei der KZBTZ-Befehl einen ersten Quellenschreibmaskenoperanden und einen Zielmaskenoperanden umfasst, wobei das Verfahren umfasst:
Ermitteln einer abschließenden niedrigstwertigen Nullbitposition in der ersten Quellenschreibmaske;
derartiges Setzen der Zielschreibmaske, dass sie die Werte der ersten Quellenschreibmaske aufweist, aber mit allen Bitpositionen näher zur höchstwertigen Bitposition als die abschließende niedrigstwertige Nullbitposition in einer ersten Quellenschreibmaske, die auf null gesetzt ist.

9. Verfahren nach Anspruch 8, wobei der KZBTZ-Befehl ferner einen zweiten Quellenschreibmaskenoperanden umfasst, und die folgende niedrigstwertige Nullbitposition eine erste Bitposition im Quellenschreibmaskenoperanden ist, die auf null gesetzt ist, wenn eine entsprechende Bitposition des zweiten Quellenschreibmaskenoperanden auf eins gesetzt ist.

10. Verfahren nach Anspruch 8 oder 9, wobei die Schreibmaskenoperanden dedizierte Schreibmaskenregister sind.

11. Verfahren nach Anspruch 10, wobei die dedizierten Schreibmaskenregister 8 oder 16 Bit groß sind.

12. Verfahren nach Anspruch 8 oder 9, wobei die Schreibmaskenoperanden Universalregister sind.

13. Verfahren nach einem der Ansprüche 8 bis 12, wobei jede Bitposition des ersten Quellenschreibmaskenoperanden seriell von der niedrigstwertigen bis zur höchstwertigen ausgewertet wird.

14. Verfahren nach Anspruch 13, um basierend auf einem Zähler zu bestimmen, wann alle der Bitpositionen ausgewertet wurden.

15. Gegenständliches maschinenlesbares Medium, umfassend Code, welcher, wenn durch einen Prozessor ausgeführt, den Prozessor veranlasst, ein Verfahren zum Ausführen eines Nullmaske-vor-abschließender-Null (KZBTZ)-Befehls auszuführen, wobei der KZBTZ-Befehl einen ersten Quellenschreibmaskenoperanden und einen Zielschreibmaskenoperanden umfasst, wobei das Verfahren umfasst:
Ermitteln einer folgenden niedrigstwertigen Nullbitposition in der ersten Quellenschreibmaske;

derartiges Setzen der Zielschreibmaske, dass sie die Werte der ersten Quellenmaske aufweist, aber mit allen Bitpositionen näher zur höchstwertigen Bitposition als die folgende niedrigstwertige Nullbitposition in einer ersten Quellenschreibmaske, die auf null gesetzt ist.

16. Verfahren nach Anspruch 15, wobei der KZBTZ-Befehl ferner einen zweiten Quellenschreibmaskenoperanden umfasst, und die folgende niedrigstwertige Nullbitposition eine erste Bitposition im Quellenschreibmaskenoperanden ist, die auf null gesetzt ist, wenn eine entsprechende Bitposition des zweiten Quellenschreibmaskenoperanden auf eins gesetzt ist.

17. Verfahren nach Anspruch 15 oder 16, wobei die Schreibmaskenoperanden dedizierte Schreibmaskenregister sind.

18. Verfahren nach Anspruch 17, wobei die dedizierten Schreibmaskenregister 8 oder 16 Bit groß sind.

19. Verfahren nach Anspruch 15 oder 16, wobei die Schreibmaskenoperanden Universalregister sind.

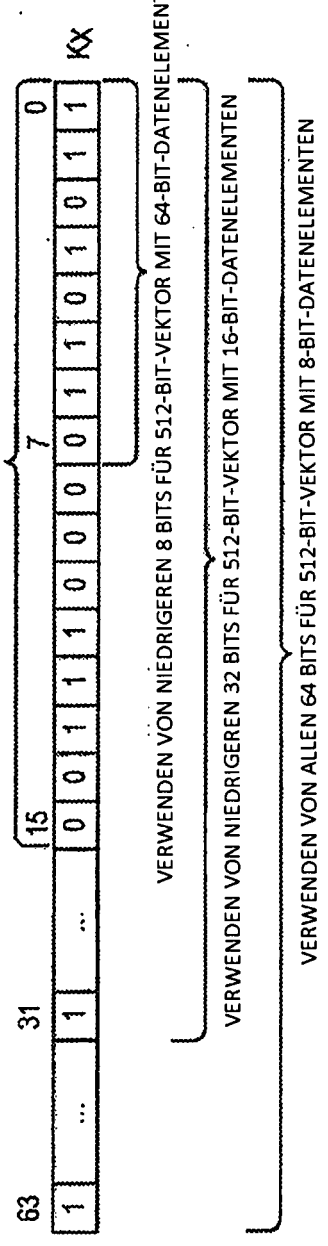
20. Verfahren nach einem der Ansprüche 15 bis 19, wobei jede Bitposition des ersten Quellenschreibmaskenoperanden seriell von der niedrigstwertigen bis zur höchstwertigen ausgewertet wird.

Es folgen 14 Seiten Zeichnungen

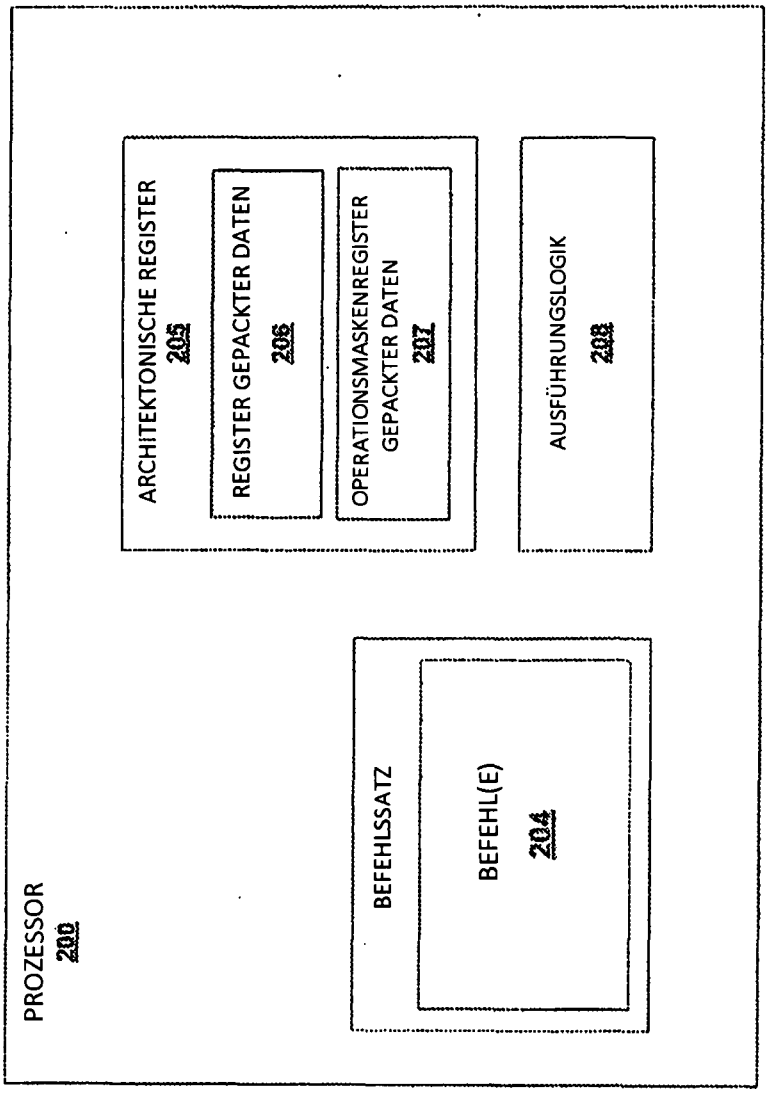
Anhängende Zeichnungen

ANZAHL VON EIN-BIT-VEKTOR-SCHREIBMASKENELEMENTEN

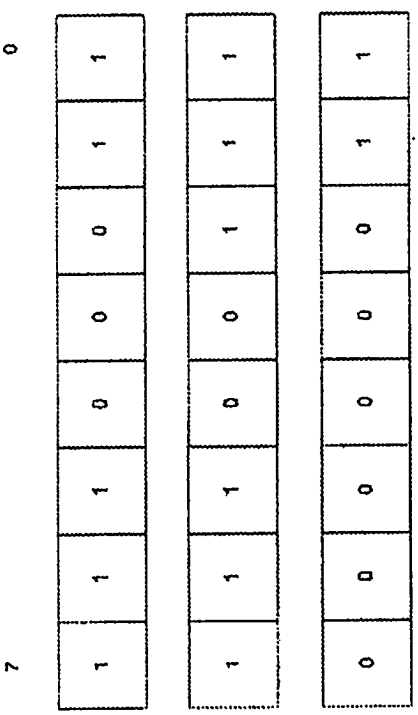
DATENELEMENT-GRÖSSE FÜR VEKTOR	VEKTORGRÖSSE		
	128 BITS	256 BITS	512 BITS
8-BIT-BYTES	16	32	64
16-BIT-WÖRTER	8	16	32
32-BIT-D-WÖRTER/SP	4	8	16
64-BIT-Q-WÖRTER/DP	2	4	8



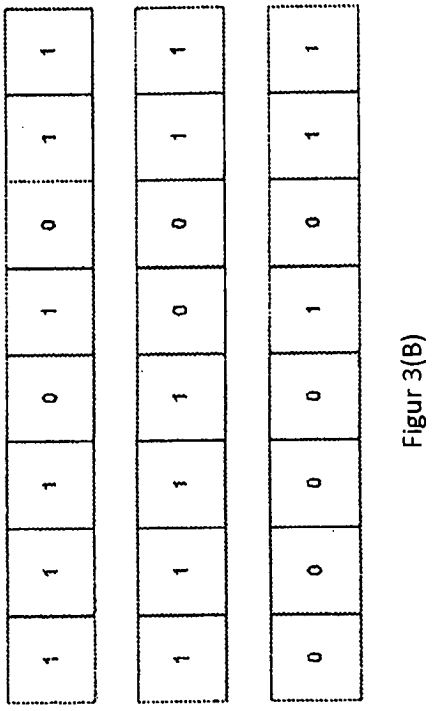
Figur 1


PROZESSOR
200
BEFEHLSSATZ
BEFEHL(E)
204
ARCHITEKTONISCHE REGISTER
205
REGISTER GEPACKTER DATEN
206
OPERATIONSMASKENREGISTER GEPACKTER DATEN
207
AUSFÜHRUNGSLOGIK
208

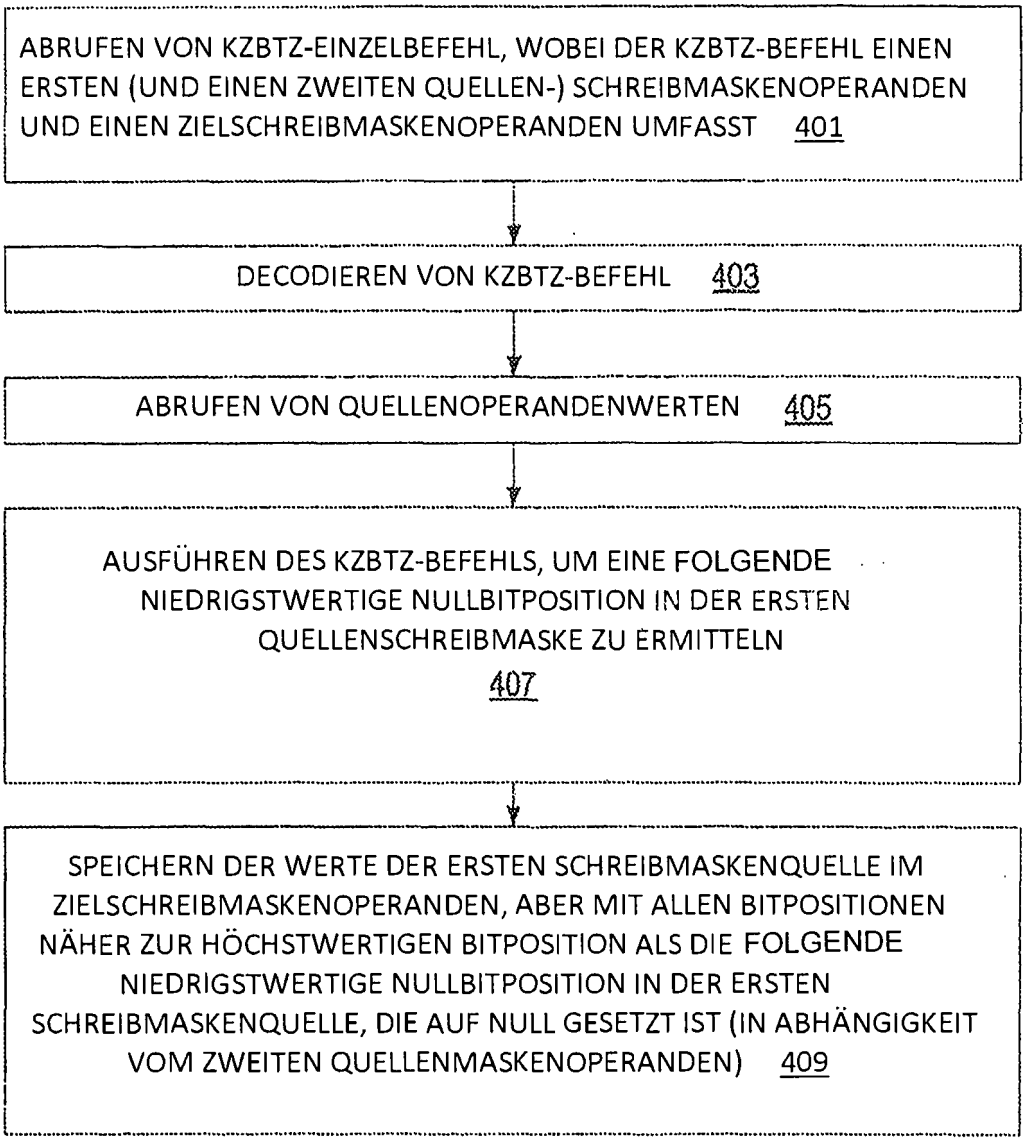
Figur 2



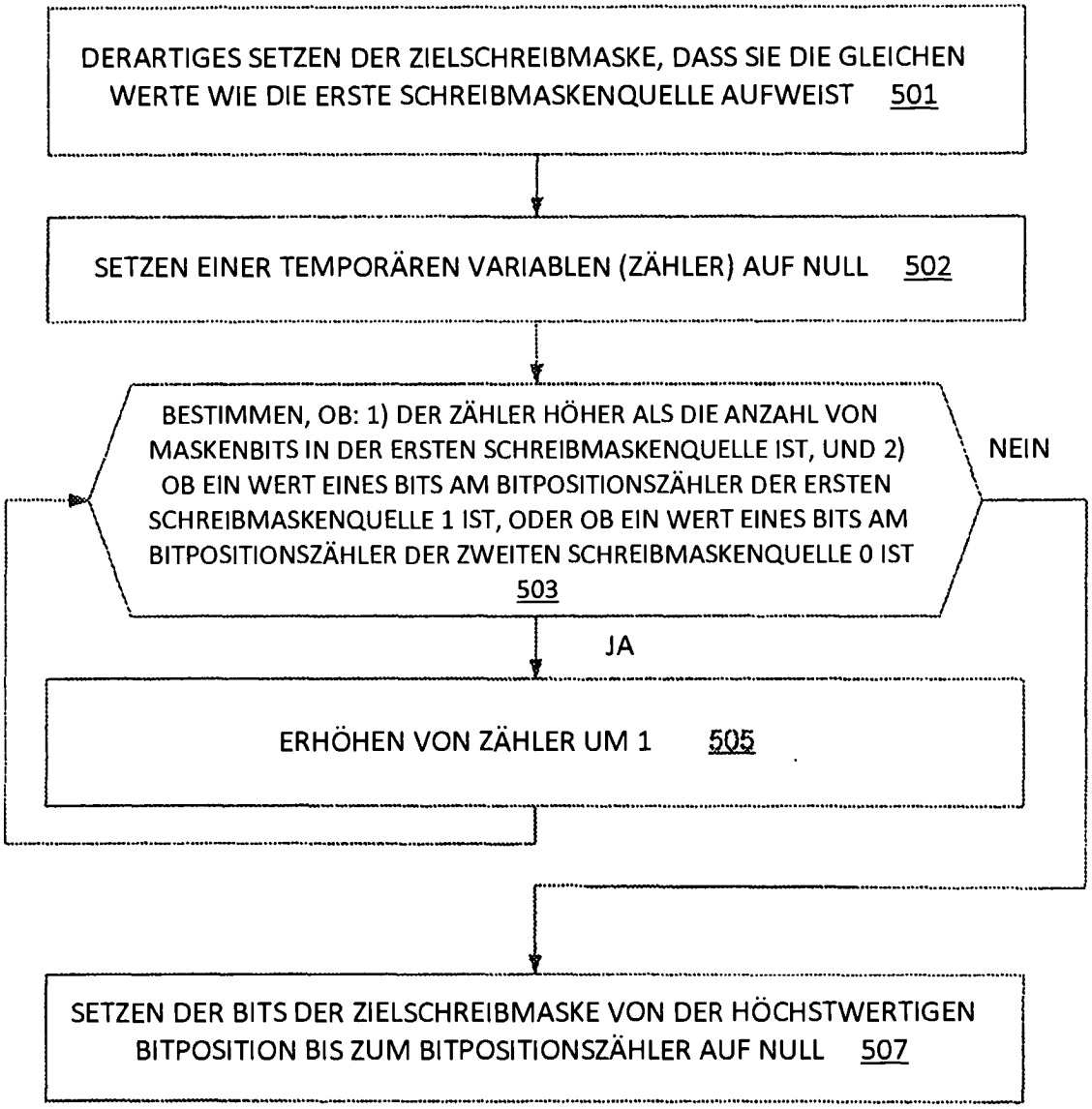
Figur 3(A)



Figur 3(B)

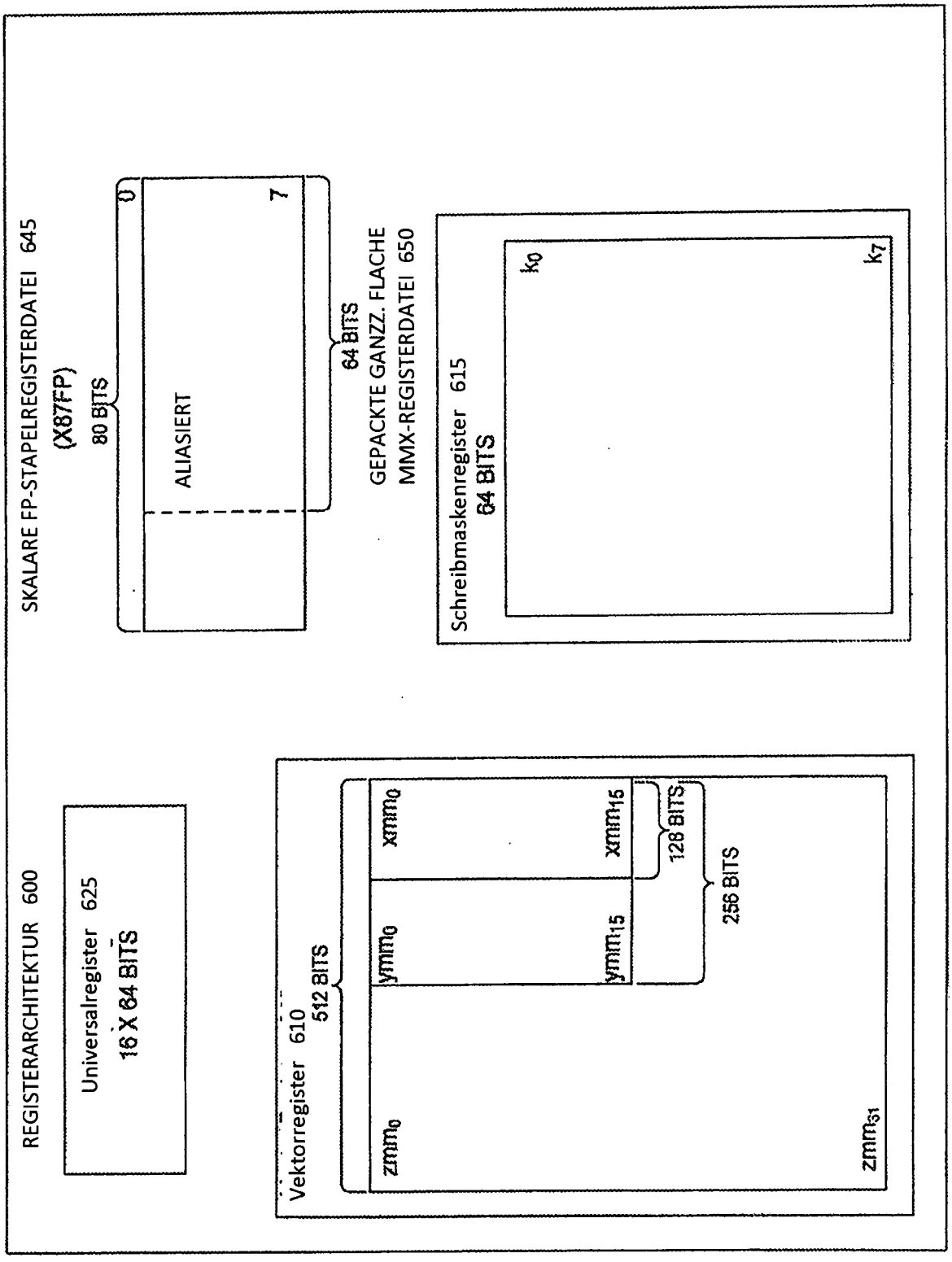

ABRUFEN VON KZBTZ-EINZELBEFEHL, WOBEI DER KZBTZ-BEFEHL EINEN ERSTEN (UND EINEN ZWEITEN QUELLEN-) SCHREIBMASKENOPERANDEN UND EINEN ZIELSCHREIBMASKENOPERANDEN UMFASST 401
DECODIEREN VON KZBTZ-BEFEHL 403
ABRUFEN VON QUELLENOPERANDENWERTEN 405
AUSFÜHREN DES KZBTZ-BEFEHLS, UM EINE FOLGENDE NIEDRIGSTWERTIGE NULLBITPOSITION IN DER ERSTEN QUELLENSCHREIBMASKE ZU ERMITTELN 407
SPEICHERN DER WERTE DER ERSTEN SCHREIBMASKENQUELLE IM ZIELSCHREIBMASKENOPERANDEN, ABER MIT ALLEN BITPOSITIONEN NÄHER ZUR HÖCHSTWERTIGEN BITPOSITION ALS DIE FOLGENDE NIEDRIGSTWERTIGE NULLBITPOSITION IN DER ERSTEN SCHREIBMASKENQUELLE, DIE AUF NULL GESETZT IST (IN ABHÄNGIGKEIT VOM ZWEITEN QUELLENMASKENOPERANDEN) 409

FIGUR 4


DERARTIGES SETZEN DER ZIELSCHREIBMASKE, DASS SIE DIE GLEICHEN WERTE WIE DIE ERSTE SCHREIBMASKENQUELLE AUFWEIST 501
SETZEN EINER TEMPORÄREN VARIABLEN (ZÄHLER) AUF NULL 502
BESTIMMEN, OB: 1) DER ZÄHLER HÖHER ALS DIE ANZAHL VON MASKENBITS IN DER ERSTEN SCHREIBMASKENQUELLE IST, UND 2) OB EIN WERT EINES BITS AM BITPOSITIONSZÄHLER DER ERSTEN SCHREIBMASKENQUELLE 1 IST, ODER OB EIN WERT EINES BITS AM BITPOSITIONSZÄHLER DER ZWEITEN SCHREIBMASKENQUELLE 0 IST 503
NEIN
JA
ERHÖHEN VON ZÄHLER UM 1 505
SETZEN DER BITS DER ZIELSCHREIBMASKE VON DER HÖCHSTWERTIGEN BITPOSITION BIS ZUM BITPOSITIONSZÄHLER AUF NULL 507

Figur 5

FIG. 6


REGISTERARCHITEKTUR 600
Universalregister 625
16 X 64 BITS
SKALARE FP-STAPELREGISTERDATEI 645
(X87FP)
80 BITS
0
7
ALIASIERT
64 BITS
GEPACKTE GANZZ. FLACHE
MMX-REGISTERDATEI 650
Schreibmaskenregister 615
64 BITS
k0
k7
Vektorregister 610
512 BITS
zmm0
ymm0
xmm0
ymm15
xmm15
128 BITS
256 BITS
zmm31

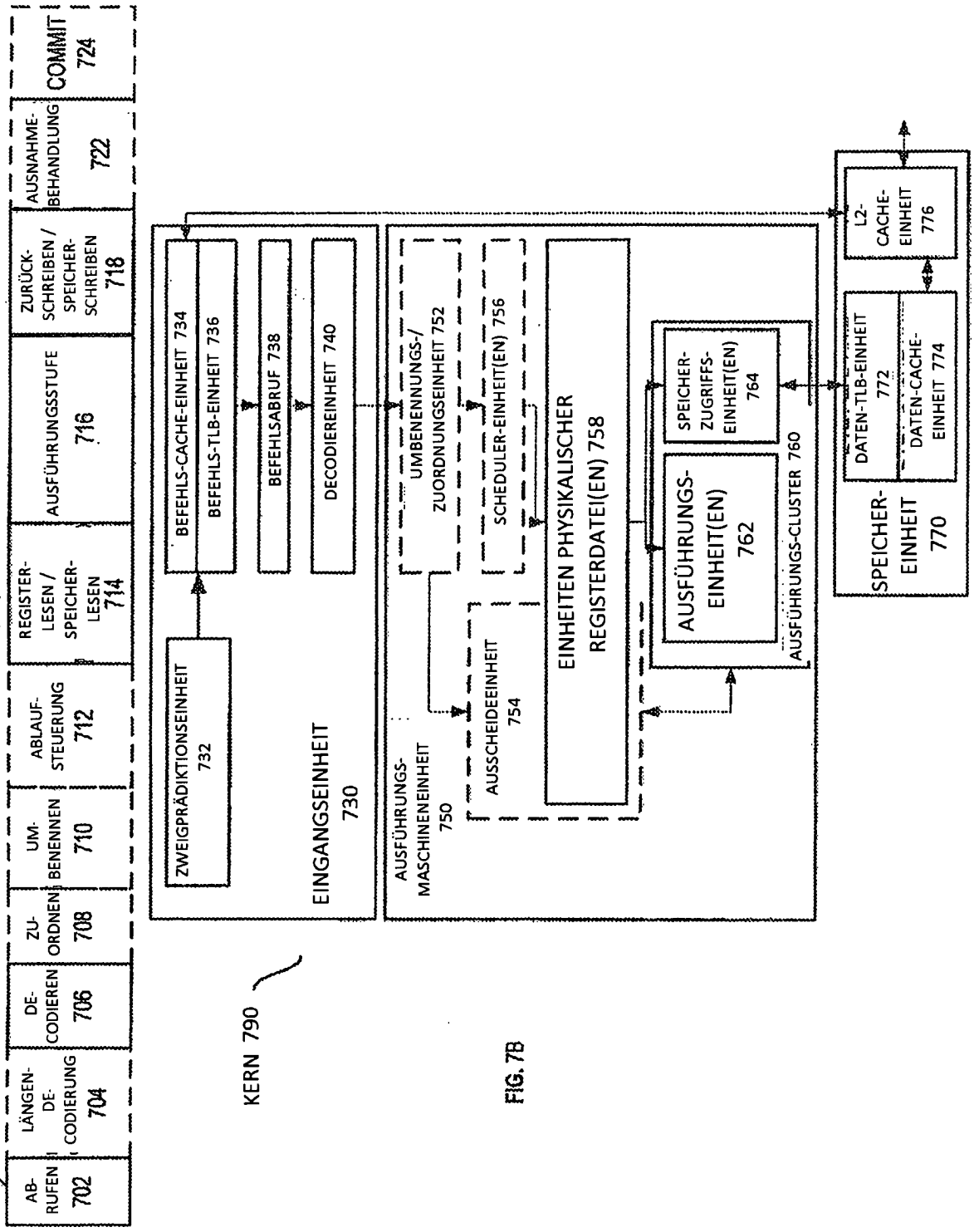

PIPELINE 700
ABRUFEN 702
LÄNGEN-DECODIERUNG 704
DECODIEREN 706
ZUORDNEN 708
UMBENENNEN 710
ABLAUFSTEUERUNG 712
REGISTER-LESEN / SPEICHER-LESEN 714
AUSFÜHRUNGSSTUFE 716
ZURÜCKSCHREIBEN / SPEICHERSCHREIBEN 718
AUSNAHMEBEHANDLUNG 722
COMMIT 724
FIG. 7A
KERN 790
EINGANGSEINHEIT 730
ZWEIGPRÄDIKTIONSEINHEIT 732
BEFEHLS-CACHE-EINHEIT 734
BEFEHLS-TLB-EINHEIT 736
BEFEHLSABRUF 738
DECODIEREINHEIT 740
AUSFÜHRUNGS-MASCHINENEINHEIT 750
UMBENENNUNGS-/ ZUORDNUNGSEINHEIT 752
AUSSCHEIDEEINHEIT 754
SCHEDULER-EINHEIT(EN) 756
EINHEITEN PHYSIKALISCHER REGISTERDATEI(EN) 758
AUSFÜHRUNGS-EINHEIT(EN) 762
SPEICHER-ZUGRIFFS-EINHEIT(EN) 764
AUSFÜHRUNGS-CLUSTER 760
SPEICHER-EINHEIT 770
DATEN-TLB-EINHEIT 772
DATEN-CACHE-EINHEIT 774
L2-CACHE-EINHEIT 776

FIG. 7B

FIG. 8B

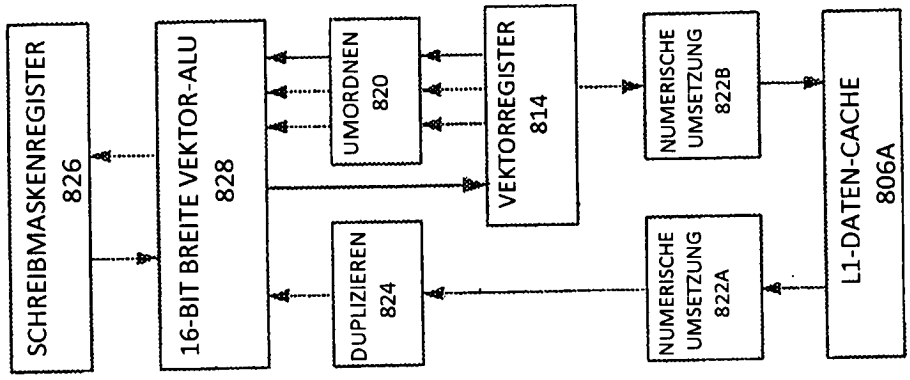
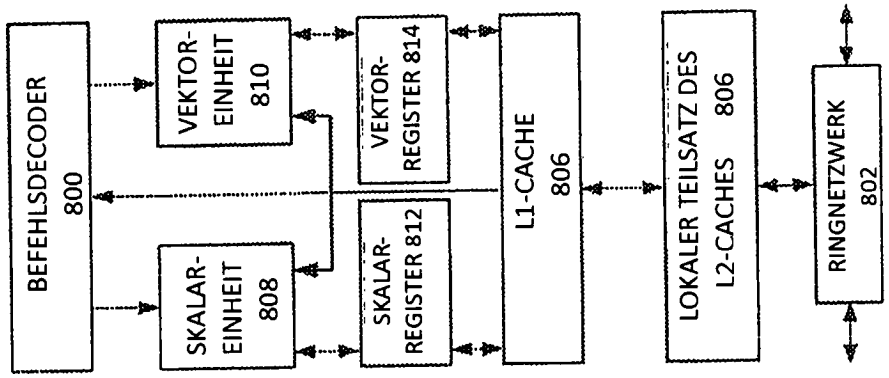

SCHREIBMASKENREGISTER 826
16-BIT BREITE VEKTOR-ALU 828
UMORDNEN 820
DUPLIZIEREN 824
VEKTORREGISTER 814
NUMERISCHE UMSETZUNG 822B
NUMERISCHE UMSETZUNG 822A
L1-DATEN-CACHE 806A

FIG. 8A


BEFEHLSDECODER 800
VEKTOR-EINHEIT 810
SKALAR-EINHEIT 808
VEKTOR-REGISTER 814
SKALAR-REGISTER 812
L1-CACHE 806
LOKALER TEILSATZ DES L2-CACHES 806
RINGNETZWERK 802

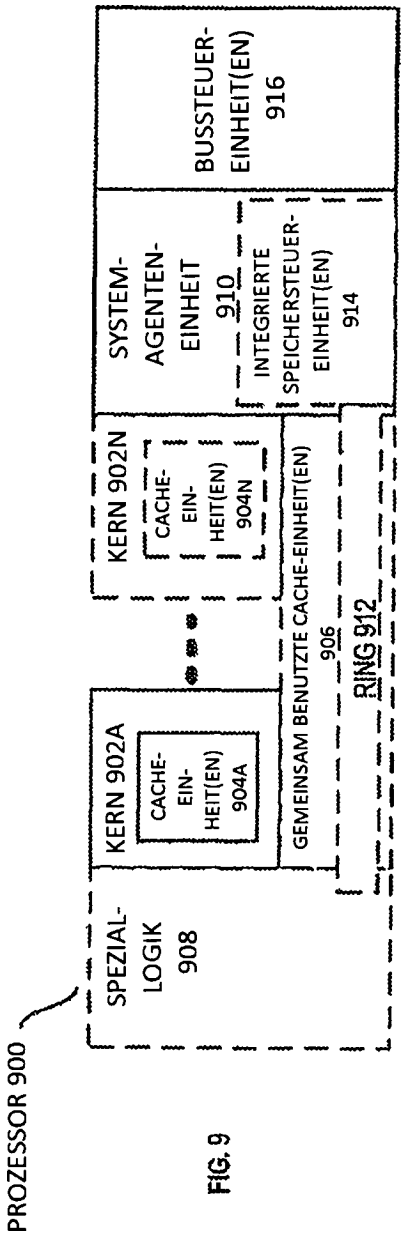

PROZESSOR 900
SPEZIAL-LOGIK 908
KERN 902A
CACHE-EIN-HEIT(EN) 904A
KERN 902N
CACHE-EIN-HEIT(EN) 904N
GEMEINSAM BENUTZTE CACHE-EINHEIT(EN) 906
RING 912
SYSTEM-AGENTEN-EINHEIT 910
INTEGRIERTE SPEICHERSTEUER-EINHEIT(EN) 914
BUSSTEUER-EINHEIT(EN) 916

FIG. 9

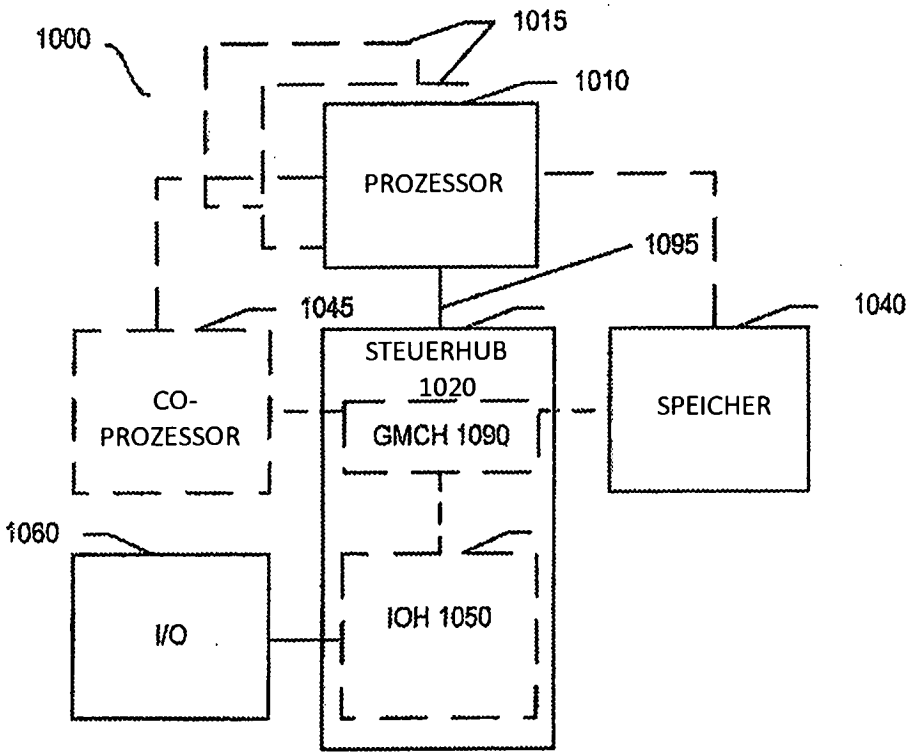

1000
1015
1010
PROZESSOR
1095
1045
1040
STEUERHUB
1020
CO-
PROZESSOR
GMCH 1090
SPEICHER
1060
I/O
IOH 1050

FIG. 10

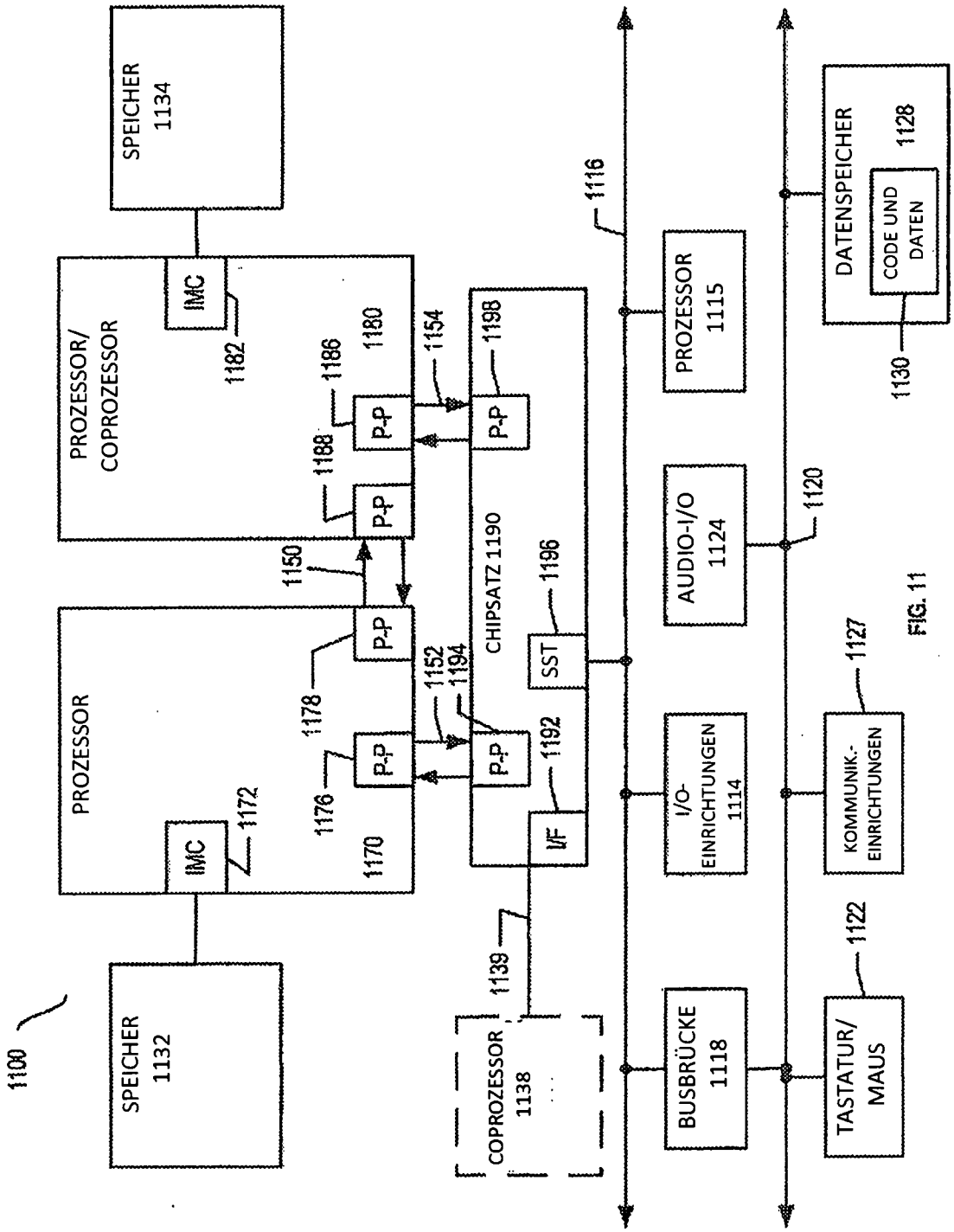

1100
SPEICHER 1132
PROZESSOR
IMC
1172
1170
1176
1178
P-P
1150
PROZESSOR/ COPROZESSOR
IMC
1182
SPEICHER 1134
1180
1186
1188
1154
1198
1152
1194
CHIPSATZ 1190
1196
SST
1192
I/F
1139
COPROZESSOR 1138
1116
PROZESSOR 1115
AUDIO-I/O 1124
1120
I/O-EINRICHTUNGEN 1114
KOMMUNIK.-EINRICHTUNGEN
1127
BUSBRÜCKE 1118
TASTATUR/ MAUS
1122
DATENSPEICHER
CODE UND DATEN
1128
1130

FIG. 11

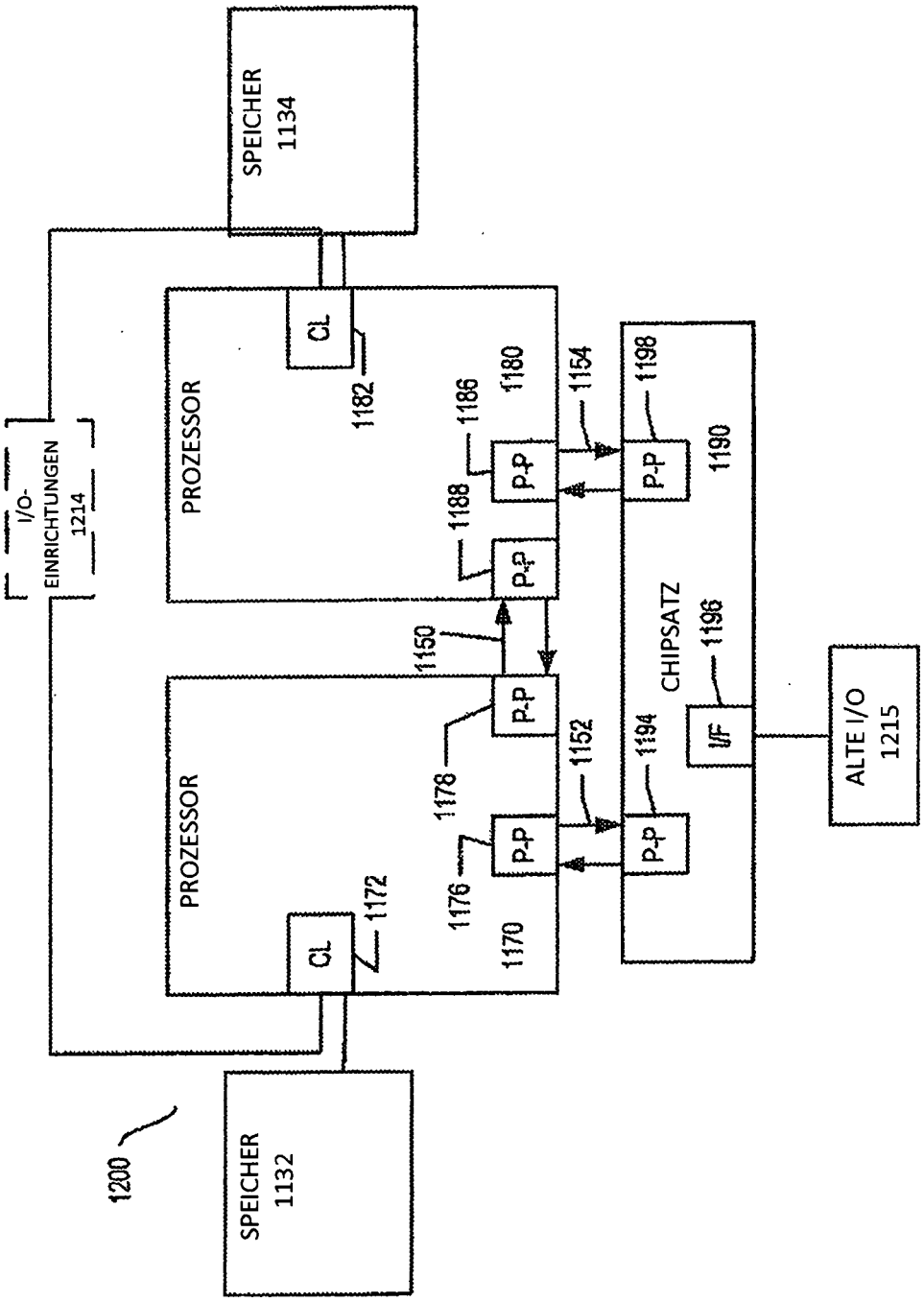

1200
SPEICHER
1132
SPEICHER
1134
I/O-
EINRICHTUNGEN
1214
PROZESSOR
CL
1172
1182
PROZESSOR
CL
1180
1186
1188
P-P
P-P
1150
1178
1176
1170
1152
1154
1198
1190
1194
CHIPSATZ
1196
I/F
ALTE I/O
1215

FIG. 12

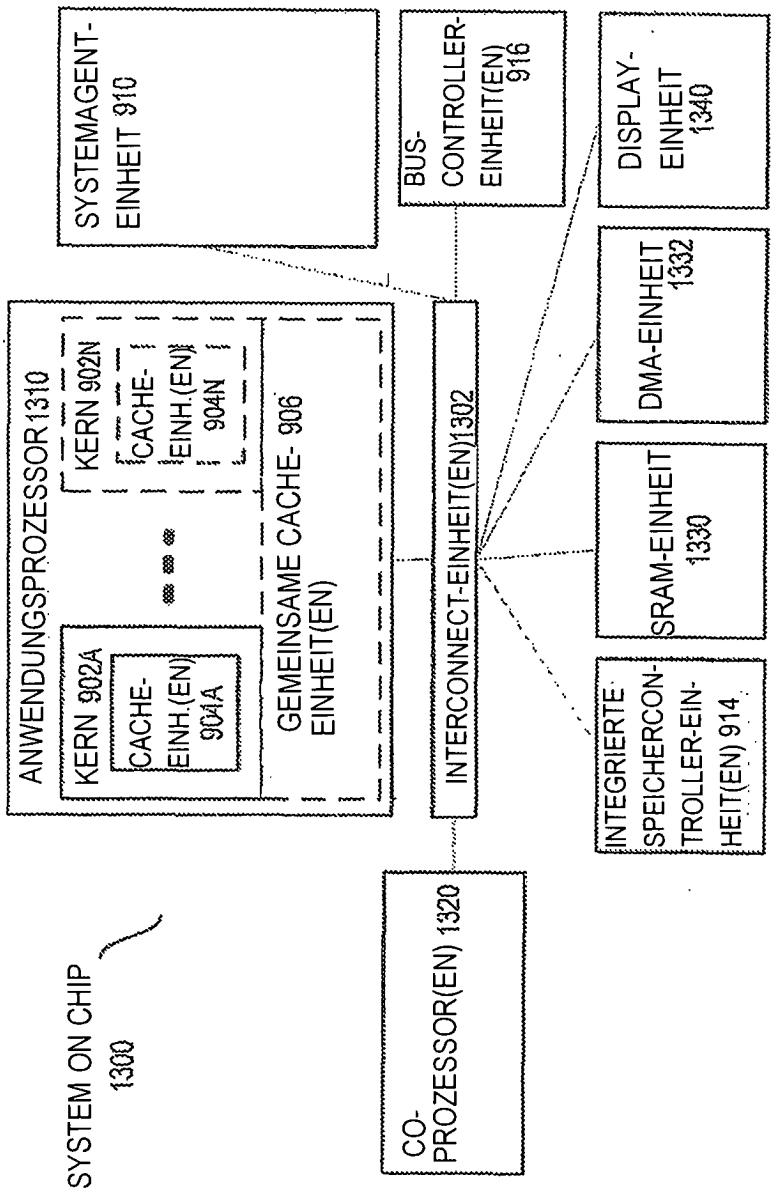

SYSTEM ON CHIP
1300
ANWENDUNGSPROZESSOR 1310
KERN 902A
CACHE-EINH.(EN) 904A
KERN 902N
CACHE-EINH.(EN) 904N
GEMEINSAME CACHE- 906 EINHEIT(EN)
SYSTEMAGENT-EINHEIT 910
BUS-CONTROLLER-EINHEIT(EN) 916
DISPLAY-EINHEIT 1340
DMA-EINHEIT 1332
SRAM-EINHEIT 1330
INTEGRIERTE SPEICHERCON-TROLLER-EIN-HEIT(EN) 914
INTERCONNECT-EINHEIT(EN)1302
CO-PROZESSOR(EN) 1320

FIG. 13

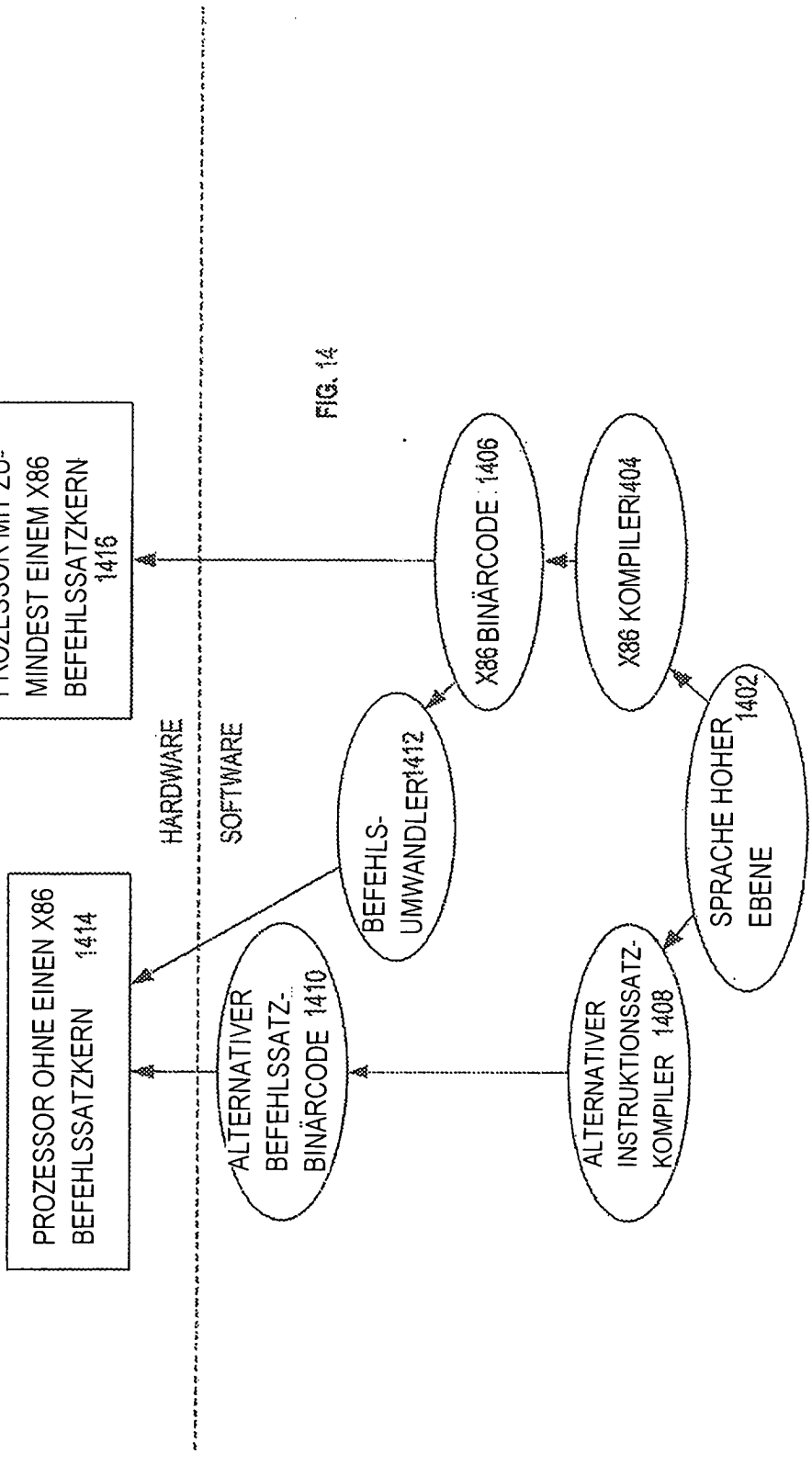

PROZESSOR MIT ZU-
MINDEST EINEM X86
BEFEHLSSATZKERN
1416
HARDWARE
SOFTWARE
X86 BINÄRCODE 1406
X86 KOMPILER1404
BEFEHLS-
UMWANDLER1412
SPRACHE HOHER
EBENE
1402
PROZESSOR OHNE EINEN X86
BEFEHLSSATZKERN 1414
ALTERNATIVER
BEFEHLSSATZ-
BINÄRCODE 1410
ALTERNATIVER
INSTRUKTIONSSATZ-
KOMPILER 1408

FIG. 14