(19) Deutsches Patent- und Markenamt

(10) **DE 11 2013 004 783 T5** 2015.06.11

(12) # Veröffentlichung

der internationalen Anmeldung mit der
(87) Veröffentlichungs-Nr.: **WO 2014/051733**
in deutscher Übersetzung (Art. III § 8 Abs. 2 IntPatÜG)
(21) Deutsches Aktenzeichen: **11 2013 004 783.7**
(86) PCT-Aktenzeichen: **PCT/US2013/045429**
(86) PCT-Anmeldetag: **12.06.2013**
(87) PCT-Veröffentlichungstag: **03.04.2014**
(43) Veröffentlichungstag der PCT Anmeldung
in deutscher Übersetzung: **11.06.2015**

(51) Int Cl.: ***G06F 9/30*** (2006.01)
G06F 9/06 (2006.01)

(30) Unionspriorität:
13/630,118 **28.09.2012** **US**

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

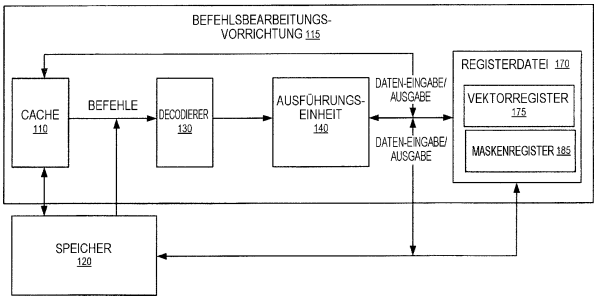
(74) Vertreter:
BOEHMERT & BOEHMERT Anwaltspartnerschaft mbB - Patentanwälte Rechtsanwälte, 28209 Bremen, DE

(72) Erfinder:
Plotnikov, Mikhail, Nizhny Novgorod, RU; Naraikin, Andrey, Nizhny Novgorod, RU; Hughes, Christopher, Santa-Clara, Calif., US

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

(54) Bezeichnung: **Durch Lese- und Schreibmasken gesteuerter Vektor-Verschiebebefehl**

(57) Zusammenfassung: Ein Prozessor führt einen Vektor-Verschiebebefehl zum Verschieben von Datenelementen von einem zweiten Vektorregister zu einem ersten Vektorregister gemäß der Steuerung eines ersten Maskenregisters und eines zweiten Maskenregisters aus. Eine Registerdatei innerhalb des Prozessors enthält das erste Vektorregister, das zweite Vektorregister, das erste Maskenregister und das zweite Maskenregister. Die Ausführungsschaltungsanordnung in dem Prozessor soll eine gegebene Anzahl von Zieldatenelementen in dem ersten Vektorregister in Reaktion auf den Vektor-Verschiebebefehl durch die gegebene Anzahl von Quelldatenelementen in dem zweiten Vektorregister ersetzen. Jedes Quelldatenelement entspricht einem Maskenbit in dem zweiten Maskenregister mit einem zweiten Bitwert und wobei jedes Zieldatenelement einem Maskenbit in dem ersten Maskenregister mit einem ersten Bitwert entspricht.



Beschreibung

TECHNISCHES GEBIET

[0001] Die vorliegende Offenbarung betrifft das Gebiet der Verarbeitungslogik, der Mikroprozessoren und der zugeordneten Befehlssatzarchitektur, die, wenn sie durch den Prozessor oder durch eine andere Verarbeitungslogik ausgeführt wird, logische, mathematische oder andere Funktionsoperationen ausführt.

STAND DER TECHNIK

[0002] Ein Befehlssatz oder eine Befehlssatzarchitektur (ISA) ist derjenige Teil der Computerarchitektur, der sich auf die Programmierung bezieht, und kann die systemspezifischen Datentypen, die systemspezifischen Befehle, die systemspezifischen Register, die systemspezifische Architektur, die systemspezifischen Adressierungsbetriebsarten, die systemspezifische Speicherarchitektur, die systemspezifische Unterbrechungs- und Ausnahmebehandlung und die systemspezifische externe Eingabe und Ausgabe (E/A) enthalten. Der Begriff Befehl bezieht sich hier allgemein auf Makrobefehle – d. h. auf Befehle, die für den Prozessor (oder für einen Befehlsumsetzer, der einen Befehl in einen oder mehrere Befehle, die durch den Prozessor verarbeitet werden sollen, (z. B. unter Verwendung einer statischen binären Übersetzung, einer dynamischen binären Übersetzung, die die dynamische Kompilierung enthält) übersetzt, morpht, emuliert oder auf andere Weise in einen oder in mehrere Befehle, die durch den Prozessor verarbeitet werden sollen, umsetzt) zur Ausführung bereitgestellt werden – im Gegensatz zu Mikrobefehlen oder Mikrooperationen (Mikro-Ops) – die das Ergebnis dessen sind, dass der Decodierer des Prozessors Makrobefehle decodiert.

[0003] Die ISA wird von der Mikroarchitektur unterschieden, die der interne Entwurf des Prozessors ist, der den Befehlssatz implementiert. Prozessoren mit unterschiedlichen Mikroarchitekturen können einen gemeinsamen Befehlssatz gemeinsam nutzen. Zum Beispiel implementieren Intel®-Core™-Prozessoren sowie Prozessoren von Advanced Micro Devices, Inc., aus Sunnyvale, CA, nahezu gleiche Versionen des x86-Befehlssatzes (mit einigen Erweiterungen, die bei neueren Versionen hinzugefügt worden sind), weisen aber unterschiedliche interne Entwürfe auf. Zum Beispiel kann dieselbe Registerarchitektur der ISA in unterschiedlichen Mikroarchitekturen unter Verwendung gut bekannter Techniken einschließlich dedizierter physikalischer Register, eines oder mehrerer dynamisch zugewiesener physikalischer Register, die einen Registerumbenennungsmechanismus verwenden, usw. auf unterschiedliche Arten implementiert werden.

[0004] Viele moderne ISAs unterstützen Einzelbefehls-Mehrdaten-Operationen (SIMD-Operationen). Anstelle eines skalaren Befehls, der nur eines oder zwei Datenelemente verarbeitet, kann ein Vektorbefehl (auch als ein gepackter Datenbefehl oder SIMD-Befehl bezeichnet) mehrere Datenelemente oder mehrere Paare von Datenelementen gleichzeitig oder parallel verarbeiten. Der Prozessor kann Parallelausführungshardware aufweisen, die auf den Vektorbefehl anspricht, um die mehreren Operationen gleichzeitig oder parallel auszuführen. Eine SIMD-Operation verarbeitet mehrere Datenelemente, die in einer Operation innerhalb eines Vektorregisters oder Speicherplatzes gepackt sind. Diese Datenelemente werden als gepackte Daten oder Vektordaten bezeichnet. Jedes der Vektorelemente kann ein getrenntes einzelnes Datum (z. B. eine Farbe eines Pixels usw.) repräsentieren, das getrennt oder unabhängig von den anderen verarbeitet werden kann.

[0005] In einigen Szenarien kann eine SIMD-Operation unabhängige Vektordatenelemente auf rekursive Weise verarbeiten, wobei sich die Anzahl der Iterationen für unterschiedliche Datenelemente unterscheidet. Somit kann die Berechnung für einige Datenelemente abgeschlossen werden, während einige andere Datenelemente noch weitere Iterationen benötigen. Ein Beispiel der rekursiven Berechnung ist eine WHILE-Schleifen-Operation. In diesem Beispiel wird ein Datenfeld X[i] (i = 0, ..., N – 1) von N Elementen einer rekursiven Berechnung unterworfen, während die Bedingung (X[i]) wahr (erfüllt) ist. Die Berechnung für X[i] endet, wenn die Bedingung (X[i]) falsch wird. Ein Beispiel der Bedingung kann X[i] > 0 sein.

```
for (i=0; i<N; i++){
   while (condition(X[i])){
      X[i]=computation(X[i]);}}
```

[0006] Falls die Anzahl der WHILE-Schleifen-Iterationen für unterschiedliche Datenelemente von X[i] unterschiedlich ist, kann die obige Berechnung nicht leicht vektorisiert werden. Eine mögliche Herangehensweise ist, dass ein Prozessor die Berechnung über jene Elemente ausführt, die die Bedingung nicht erfüllen, und die aus diesen Elementen hergeleiteten Ergebnisse daraufhin verwirft. Allerdings weist diese Herangehensweise eine niedrige Effizienz auf, da der Prozessor nicht nur eine unnötige Berechnung über diese Elemente ausführt, sondern auch nicht in der Lage ist, die von diesen Elementen belegten Vektorregisterschlitze zu nutzen.

KURZBESCHREIBUNG DER ZEICHNUNGEN

[0007] Ausführungsformen sind beispielhaft und nicht einschränkend in den Figuren der beigefügten Zeichnungen dargestellt:

[0008] Fig. 1 ist ein Blockschaltplan einer Befehlsverarbeitungsvorrichtung, die Vektorregister und Maskenregister enthält, in Übereinstimmung mit einer Ausführungsform.

[0009] Fig. 2 ist ein Blockschaltplan einer Registerarchitektur in Übereinstimmung mit einer Ausführungsform.

[0010] Fig. 3 veranschaulicht ein Beispiel einer Vektoroperationsfolge in Übereinstimmung mit einer Ausführungsform.

[0011] Fig. 4A veranschaulicht ein Beispiel von Pseudocode für Befehle, die veranlassen, dass ein Prozessor Operationen an Vektorregistern und Maskenregistern ausführt, in Übereinstimmung mit einer Ausführungsform.

[0012] Fig. 4B veranschaulicht ein Beispiel eines Codesegments zur Verwendung der Befehle aus **Fig.** 4A in Übereinstimmung mit einer Ausführungsform.

[0013] Fig. 5A ist ein Ablaufplan, der Operationen, die in Reaktion auf ein Codesegment, das den Maskenaktualisierungsbefehl und den Vektor-Verschiebebefehl verwendet, ausgeführt werden sollen, in Übereinstimmung mit einer Ausführungsform darstellt.

[0014] Fig. 5B ist ein Ablaufplan, der Operationen, die in Reaktion auf einen Maskenaktualisierungsbefehl ausgeführt werden sollen, in Übereinstimmung mit einer Ausführungsform darstellt.

[0015] Fig. 5C ist ein Ablaufplan, der Operationen, die in Reaktion auf einen Vektor-Verschiebebefehl ausgeführt werden sollen, in Übereinstimmung mit einer Ausführungsform darstellt.

[0016] Fig. 6 ist ein Blockschaltplan, der die Verwendung eines Softwarebefehlsumsetzers zum Umsetzen binärer Befehle in einem Quellbefehlssatz in binäre Befehle in einem Zielbefehlssatz in Übereinstimmung mit einer Ausführungsform darstellt.

[0017] Fig. 7A ist ein Blockschaltplan einer In-order- und einer Out-of order-Pipeline in Übereinstimmung mit einer Ausführungsform.

[0018] Fig. 7B ist ein Blockschaltplan eines In-order- und eines Out-of order-Kerns in Übereinstimmung mit einer Ausführungsform.

[0019] Fig. 8A–B sind Blockschaltpläne einer spezielleren beispielhaften In-order-Kern-Architektur in Übereinstimmung mit einer Ausführungsform.

[0020] Fig. 9 ist ein Blockschaltplan eines Prozessors in Übereinstimmung mit einer Ausführungsform.

[0021] Fig. 10 ist ein Blockschaltplan eines Systems in Übereinstimmung mit einer Ausführungsform.

[0022] Fig. 11 ist ein Blockschaltplan eines zweiten Systems in Übereinstimmung mit einer Ausführungsform.

[0023] Fig. 12 ist ein Blockschaltplan eines dritten Systems in Übereinstimmung mit einer Ausführungsform der Erfindung.

[0024] Fig. 13 ist ein Blockschaltplan eines Ein-Chip-Systems (SoC) in Übereinstimmung mit einer Ausführungsform.

Beschreibung der Ausführungsformen

[0025] In der folgenden Beschreibung sind zahlreiche spezifische Einzelheiten dargelegt. Allerdings können Ausführungsformen der Erfindung selbstverständlich ohne diese spezifischen Einzelheiten verwirklicht werden. In anderen Fällen sind gut bekannte Schaltungen, Strukturen und Techniken nicht ausführlich gezeigt, um das Verständnis dieser Beschreibung nicht zu verdecken.

[0026] Hier beschriebene Ausführungsformen bieten Befehle zum Verbessern der Effizienz einer rekursiven Vektorberechnung über unabhängigen Datenelementen. Die Befehle nutzen ein Paar Vektorregister und ein Paar Maskenregister, um eine rekursive Vektorberechnung auszuführen, wobei ein erstes Vektorregister als ein Akkumulator zum Akkumulieren von Vektorrechenergebnissen dient und ein zweites Vektorregister neue Datenelemente zum Füllen in die ungenutzten Schlitze (Positionen ungenutzter oder fertiger Datenelemente) des ersten Vektorregisters bereitstellt. Die Maskenregister werden verwendet, um anzugeben, welche Datenelemente in den entsprechenden Vektorregistern eine weitere Berechnung benötigen.

[0027] In einer Ausführungsform akkumuliert das erste Vektorregister (d. h. der Akkumulator) Eingangsdatenelemente, bis das Register mit einem vollen Vektor gefüllt ist. Daraufhin führt der Prozessor unter Verwendung nicht maskierter (d. h. dichter) Vektoroperationen eine Berechnung an diesen Datenelementen aus. Nach der Berechnung können einige Elemente (für die die Berechnung abgeschlossen ist) in dem Akkumulator an den Speicher oder an andere Ablageorte zurückgesendet werden und können andere Elemente (für die die Berechnung nicht abgeschlossen ist) für eine zusätzliche Anzahl von Iterationen in dem Akkumulator behalten werden. Die Datenelementpositionen der abgeschlossenen Berechnungen in dem Akkumulator können von

neuen Datenelementen genutzt werden, die ebenfalls dieselbe rekursive Berechnung benötigen.

[0028] Hier werden zwei Befehle RWMASKUPDATE und SPARSEMOV beschrieben. Diese Befehle verbessern in vielen Szenarien die Effizienz der Vektorisierung. Zum Beispiel können die Eingangsdatenelemente in einem Szenarium von einem oder von mehreren dünn besiedelten Vektordatensätzen kommen, von denen jeder nicht genügend Datenelemente besitzt, um den gesamten Akkumulator (d. h. das erste Vektorregister) zu füllen. Darüber hinaus können Eingangsdatenelemente von unterschiedlichen Datensätzen in der Berechnung unterschiedliche Anzahlen von Iterationen benötigen. Somit verbleiben in dem Akkumulator von diesen Datenelementen, die keine weitere Berechnung benötigen, ungenutzte Schlitze. Die hier beschriebenen Befehle ermöglichen, dass diese ungenutzten Schlitze mit nutzbaren Elementen gefüllt werden, und ermöglichen somit die rekursive Berechnung über einen vollen Vektor. Wie im Folgenden ausführlicher beschrieben wird, ist der SPARSEMOV-Befehl ein Vektor-Verschiebebefehl, der nutzbare Datenelemente (d. h. Datenelemente, die eine Berechnung benötigen) aus einem zweiten Vektorregister in den Akkumulator verschiebt. Der RWMASKUPDATE-Befehl aktualisiert sowohl ein Lesemaskenregister (das dem zweiten Vektorregister zugeordnet ist) als auch ein Schreibmaskenregister (das dem Akkumulator zugeordnet ist), um die Positionen nutzbarer Datenelemente in diesen zwei Vektorregistern zu identifizieren.

[0029] Die Verwendung von RWMASKUPDATE in Verbindung mit SPARSEMOV verringert die Gesamtzahl der Befehle, die in einer rekursiven Berechnung notwendig sind, und vereinfacht die Überlauf- und Unterlauffälle, in denen die Anzahl nutzbarer Datenelemente (d. h. Quelldatenelemente) in dem zweiten Vektorregister nicht mit der Anzahl ungenutzter Schlitze (d. h. Zielpositionen) in dem ersten Vektorregister übereinstimmt. Die aktualisierten Lese- und Schreibmasken werden verwendet, um die Datenverschiebung zwischen den zwei Vektorregistern zu steuern; insbesondere werden Schreibmaskenbits von Nullen verwendet, um die Zielpositionen in dem Akkumulator zu identifizieren, und werden Lesemaskenbits von Einsen verwendet, um die Quelldatenelemente in dem zweiten Vektorregister zu identifizieren. Die Verwendung invertierter Schreibmaskenbits zum Identifizieren der Zielpositionen vereinfacht die Datenakkumulation bei der Vektorisierung einer dünn besetzten und rekursiven Berechnung.

[0030] Fig. 1 ist ein Blockschaltplan einer Ausführungsform einer Befehlsverarbeitungsvorrichtung **115**, die eine Ausführungseinheit **140** aufweist, die eine Schaltung enthält, die betreibbar ist, um Befehle einschließlich des RWMASKUPDATE- und des SPARSEMOV-Befehls, auszuführen. In einigen Ausführungsformen kann die Befehlsverarbeitungsvorrichtung **115** ein Prozessor, ein Prozessorkern eines Mehrkernprozessors oder ein Verarbeitungselement in einem elektronischen System sein.

[0031] Ein Decodierer **130** empfängt ankommende Befehle in Form höherer Maschinenbefehle oder Makrobefehle und decodiert sie, um niedrigere Mikrooperationen, Mikrocodeeinsprungpunkte, Mikrobefehle oder andere niedrigere Befehle oder Steuersignale zu erzeugen, die den ursprünglichen höheren Befehl widerspiegeln und/oder von ihm hergeleitet sind. Die niedrigeren Befehle oder Steuersignale können die Operation des höheren Befehls durch niedrigere Operationen (z. B. Operationen auf der Schaltungsebene oder auf der Hardwareebene) implementieren. Der Decodierer **130** kann unter Verwendung vieler verschiedener Mechanismen implementiert werden. Beispiele geeigneter Mechanismen enthalten Mikrocode, Nachschlagetabellen, Hardwareimplementierungen, programmierbare Logikanordnungen (PLAs), andere im Gebiet bekannte Mechanismen, die zum Implementieren von Decodierern verwendet werden, usw., sind darauf aber nicht beschränkt.

[0032] Der Decodierer **130** kann ankommende Befehle für einen Cache **110**, für einen Speicher **120** oder für andere Quellen empfangen. Die decodierten Befehle werden an die Ausführungseinheit **140** gesendet. Die Ausführungseinheit **140** kann von dem Decodierer **130** eine oder mehrere Mikrooperationen, einen oder mehrere Mikrocodeeinsprungpunkte, einen oder mehrere Mikrobefehle, einen oder mehrere andere Befehle oder eines oder mehrere andere Steuersignale, die die empfangenen Befehle widerspiegeln oder von ihnen hergeleitet sind, empfangen. Die Ausführungseinheit **140** empfängt Daten, die von einer Registerdatei **170**, von dem Cache **110** und/oder von dem Speicher **120** eingegeben werden, und erzeugt Daten, die an diese ausgegeben werden.

[0033] In einer Ausführungsform enthält die Registerdatei **170** Architekturregister, die ebenfalls als Register bezeichnet werden. Sofern nicht etwas anderes spezifiziert ist oder deutlich sichtbar ist, werden die Ausdrücke Architekturregister, Registerdatei und Register hier zur Bezugnahme auf Register, die für die Software und/oder für den Programmierer sichtbar sind (z. B. für Software sichtbar sind), und/oder auf die Register, die durch Makrobefehle spezifiziert werden, um Operanden zu identifizieren, verwendet. Diese Register stehen im Gegensatz zu anderen, Nicht-Architektur-Registern in einer gegebenen Mikroarchitektur (z. B. temporären Registern, Umordnungspuffern, Ausscheideregistern usw.).

[0034] Um eine Verdeckung der Beschreibung zu vermeiden, ist eine verhältnismäßig einfache Befehlsverarbeitungsvorrichtung **115** gezeigt und be-

schrieben. Es wird gewürdigt werden, dass andere Ausführungsformen mehr als eine Ausführungseinheit aufweisen können. Zum Beispiel kann die Vorrichtung **115** mehrere verschiedene Typen von Ausführungseinheiten wie etwa z. B. Arithmetikeinheiten, Arithmetik-Logik-Einheiten (ALUs), Ganzzahleinheiten, Gleitkommaeinheiten usw. enthalten. Nochmals andere Ausführungsformen einer Befehlsverarbeitungsvorrichtung oder von Prozessoren können mehrere Kerne, Logikprozessoren oder Ausführungsmaschinen aufweisen. In Bezug auf die **Fig.** 7–**Fig.** 13 werden später eine Anzahl von Ausführungsformen der Befehlsverarbeitungsvorrichtung **115** gegeben.

[0035] In Übereinstimmung mit einer Ausführungsform enthält die Registerdatei **170** einen Satz von Vektorregistern **175** und einen Satz von Maskenregistern **185**, die beide die Operanden des RWMASKUPDATE- und des SPARSEMOV-Befehls speichern. Jedes Vektorregister **175** kann 512 Bits, 256 Bits oder 128 Bits breit sein oder es kann eine andere Vektorbreite verwendet werden. Jedes Maskenregister **185** enthält eine Anzahl von Maskenbits, wobei jedes Maskenbit einem Datenelement eines der Vektorregister **175** entspricht. Da jedes Maskenbit zum Maskieren eines Datenelements eines Vektorregisters verwendet wird, kann ein Maskenregister mit 64 Bits zum Maskieren von vierundsechzig 8-Bit-Datenelementen eines 512-Bit-Registers verwendet werden. Für ein Vektorregister mit einer anderen Breite (z. B. 256 Bits oder 128 Bits) und für Datenelemente mit einer anderen Größe (z. B. 16 Bits, 32 Bits oder 64 Bits) können in Verbindung mit einer Vektoroperation eine andere Anzahl von Maskenbits verwendet werden.

[0036] **Fig.** 2 veranschaulicht eine Ausführungsform einer zugrundeliegenden Registerarchitektur **200**, die die hier beschriebenen Befehle unterstützt. Die Registerarchitektur **200** beruht auf den Intel®-Core™-Prozessoren, die einen Befehlssatz implementieren, der x86-, MMX™-, Streaming-SIMD-Extensions-(SSE-), SSE2-, SSE3-, SSE4.1- und SSE4.2-Befehle sowie einen zusätzlichen Satz von SIMD-Erweiterungen, auf die als die Advanced Vector Extensions (AVX) (AVX1 und AVX2) Bezug genommen wird, enthält. Allerdings ist festzustellen, dass ebenfalls eine andere Registerarchitektur verwendet werden kann, die andere Registerlängen, andere Registertypen und/oder andere Anzahlen von Registern unterstützt.

[0037] In der dargestellten Ausführungsform gibt es zweiunddreißig Vektorregister **210**, die 512 Bits breit sind; auf diese Register wird als zmm0 bis zmm 31 Bezug genommen. Die niederwertigen 256 Bits der unteren sechzehn zmm-Register sind den Registern ymm0-16 überlagert. Die höherwertigen 128 Bits der unteren sechzehn zmm-Register (die niederwertigen 128 Bits der ymm-Register) sind den Registern xmm0-15 überlagert. In der dargestellten Ausführungsform gibt es acht Maskenregister **220** (k0 bis k7), jeweils mit einer Länge von 64 Bits. In einer alternativen Ausführungsform sind die Maskenregister **220** 16 Bits breit.

[0038] In der dargestellten Ausführungsform enthält die Registerarchitektur **200** ferner sechzehn 64-Bit-Mehrzweckregister (GP-Register) **230**. In einer Ausführungsform werden sie zusammen mit den vorhandenen x86-Adressierungsbetriebsarten zum Adressieren von Speicheroperanden verwendet. Außerdem zeigt die Ausführungsform RFLAGS-Register **260**, RIP-Register **270** und MXCSR-Register **280**.

[0039] Außerdem veranschaulicht die Ausführungsform eine skalare Gleitkomma-Stapelregisterdatei (FP-Stapelregisterdatei) (x87-Stapel) **240**, auf der die gepackte ganzzahlige flache MMX-Ganzzahlregisterdatei **250** als Alias verwendet ist. In der dargestellten Ausführungsform ist der x87-Stapel ein Acht-Element-Stapel, der dafür verwendet wird, an 32/64/80-Bit-Gleitkommadaten unter Verwendung der x87-Befehlssatzerweiterung skalare Gleitkommaoperationen auszuführen; währenddessen werden die MMX-Register dazu verwendet, an gepackten 64-Bit-Ganzzahldaten Operationen auszuführen sowie Operanden für einige zwischen den MMX- und den xmm-Registern ausgeführten Operationen zu halten.

[0040] Alternative Ausführungsformen der Erfindung können breitere oder schmalere Register verwenden. Außerdem können alternative Ausführungsformen der Erfindung mehr, weniger oder andere Registerdateien und Register verwenden.

[0041] **Fig.** 3 ist eine schematische Darstellung, die ein Beispiel der Operationen darstellt, die durch einen Prozessor (d. h. durch die Befehlsverarbeitungsvorrichtung **115**) ausgeführt werden, um die Berechnung über unabhängige Datenelemente effizient zu vektorisieren. Um die Darstellung zu vereinfachen, ist jedes Vektorregister in diesem Beispiel in der Weise gezeigt, dass es nur acht Datenelemente aufweist. Alternative Ausführungsformen können in den Vektorregistern eine andere Anzahl von Datenelementen aufweisen. Die Vektorregister können 128 Bits, 256 Bits oder 512 Bits breit (z. B. die xmm-, die ymm- oder die zmm-Register aus **Fig.** 2) sein oder es kann eine andere Breite verwendet werden. Da es in jedem Vektorregister acht Datenelemente gibt, werden in Verbindung mit jedem Vektorregister nur acht Maskenbits verwendet.

[0042] In diesem Beispiel wird das Vektorregister V1 als ein Akkumulator verwendet und wird das Vektorregister V2 zum Bereitstellen neuer Datenelemente für V1 verwendet. Die Maskenregister K1 (die Schreibmaske) und K2 (die Lesemaske) werden zum

Maskieren der Datenelemente in V1 bzw. V2 verwendet. In diesem Beispiel gibt ein Maskenbit null an, dass das entsprechende Datenelement vor der Berechnung maskiert wird (d. h., dass keine weitere Berechnung notwendig ist), und gibt ein Maskenbit eins an, dass das entsprechende Datenelement eine weitere Berechnung benötigt. In einer alternativen Ausführungsform kann die Bedeutung des Maskenbitwerts umgekehrt sein; z. B. kann ein Maskenbit eins verwendet werden, um anzugeben, dass das entsprechende Datenelement keine weitere Berechnung benötigt, und kann ein Maskenbit null verwendet werden, um anzugeben, dass das entsprechende Datenelement eine weitere Berechnung benötigt.

[0043] Anfangs wird angenommen, dass der Akkumulator V1 zwei Sätze von Daten als den Eingangsvektor speichert: A und B, von denen jeder Teil eines dünn besiedelten Datenfelds sein kann. Der Index j von Aj und Bj gibt die Anzahl der Iterationen an, die ein Datenelement durchlaufen hat; z. B. ist A0 das Element A vor irgendwelchen Iterationen und A1 das Element A nach einer ersten Iteration **310**. Um die Darstellung zu vereinfachen, sind unterschiedliche Datenelemente aus demselben Datensatz in der selben Iteration mit derselben Kennung gezeigt; z. B. sind A0 an der Position 0 und A0 an der Position 2 des Eingangsvektors zwei unterschiedliche Elemente und können sie dieselben oder unterschiedliche Werte aufweisen und sind B0 an der Position 1 und B0 an der Position 3 des Eingangsvektors zwei unterschiedliche Elemente und können sie dieselben oder unterschiedliche Werte aufweisen. Die Anfangswerte der Maskenbits in dem Maskenregister K1 sind alle Einsen, die angeben, dass der Anfangseingangsvektor in V1 ein voller Vektor ist und dass jedes Element von V1 an der ersten Iteration **310** der Vektorberechnung beteiligt sein kann.

[0044] In diesem Beispiel repräsentiert jede Iteration eine Iteration einer WHILE-Schleife, in der eine rekursive Vektorberechnung ausgeführt wird. Nach der ersten Iteration **310** enthält der Akkumulator V1 einen Satz von A_1-en und B_1-en, wobei der Index angibt, dass diese Elemente die erste Iteration abgeschlossen haben. Es wird angenommen, dass Elemente von A nur eine Iteration der WHILE-Schleife benötigen und dass Elemente von B zwei Iterationen benötigen. Somit ist die Berechnung für die Elemente A nach einer Iteration der WHILE-Schleife abgeschlossen, während für die Elemente B eine weitere Iteration notwendig ist. An diesem Punkt ist die Bedingung für jedes der Elemente A falsch (da sie die Bedingung für eine weitere Berechnung nicht erfüllen) und ist die Bedingung für jedes der Elemente B wahr (da sie die Bedingung für eine weitere Berechnung erfüllen). Somit werden die Maskenbitwerte in K1 für jene Maskenbits, die A_1-en entsprechen, auf Nullen gesetzt und diejenigen für jene Maskenbits, die B_1-en entsprechen, auf Einsen gesetzt.

[0045] In einer Ausführungsform gibt ein Maskenbit null an, dass das Ergebnis an der entsprechenden Elementposition nach einer Vektoroperation über das gesamte Vektorregister (in diesem Fall V1) verworfen wird. In alternativen Ausführungsformen gibt ein Maskenbit null an, dass die Berechnung für die entsprechende Elementposition nicht ausgeführt wird und diese Elementposition somit ungenutzt ist. In beiden Szenarien ist es eine Verschwendung von Vektorbetriebsmitteln und verringert es die Effizienz der Vektorberechnung, A1-en in dem Akkumulator V1 zu halten. Somit wird in Übereinstimmung mit einer Ausführungsform der Erfindung ein zweites Vektorregister V2 verwendet, um neue Datenelemente für V1 bereitzustellen, um die von A1-en gelassenen ungenutzten Schlitze (d. h. Datenelementpositionen) zu füllen. Die Datenelemente von A1-en können im Speicher, im Cache oder in einer anderen Datenablage gesichert werden.

[0046] In dem Beispiel aus **Fig.** 3 speichert das Vektorregister V2 Elemente eines Datensatzes C, der Teil eines anderen dünnbesiedelten Vektorfelds sein kann. Die in V2 mit "*" gekennzeichneten Positionen repräsentieren "unbedeutend", d. h., dass sie keine für die rekursive Vektorberechnung nutzbaren Datenelemente enthalten. Es wird angenommen, dass jedes Datenelement von C drei Iterationen der WHILE-Schleife durchlaufen muss. Anstelle der oder zusätzlich zu den Elementen von C kann V2 neue Datenelemente A und/oder B (z. B. A_0-en, B_0-en und/oder B_1-en) bereitstellen, die eine oder mehrere Iterationen der WHILE-Schleife (und somit eine weitere Berechnung) durchlaufen müssen. Auf diese Datenelemente in V2, die eine weitere Berechnung benötigen, wird als "Quelldatenelemente" Bezug genommen. Diese Quelldatenelemente in V2 können die von A_1-en gelassenen ungenutzten Schlitze in V1 füllen (als "Zieldatenelemente" bezeichnet). Zur Erleichterung der Beschreibung wird auf Datenelemente in V1 und/oder in V2, die eine weitere Berechnung benötigen, als "nutzbare Datenelemente" Bezug genommen. Somit wird eine Zusammenführungsoperation **320** ausgeführt, um die nutzbaren Datenelemente in V1 und V2 zusammenzuführen, so dass die Quelldatenelemente in V2 an die von den Zieldatenelementen belegten Positionen in V1 verschoben werden und dass die rekursive Berechnung zu einer zweiten Iteration **330** mit zusätzlichen nutzbaren Datenelementen in V1 übergehen kann.

[0047] In einer solchen Zusammenführungsoperation können drei Szenarien auftreten: Überlauf, Unterlauf und genaue Übereinstimmung. Eine genaue Übereinstimmung gibt an, dass es in V2 dieselbe Anzahl nutzbarer Datenelemente wie die Anzahl in V1 verbliebener ungenutzter Schlitze gibt. Somit werden in einer genauen Übereinstimmung alle Quelldatenelemente in V2 in die in V1 gelassenen ungenutzten Schlitze verschoben (d. h. ersetzt). Im Ergebnis be-

sitzt V1 einen vollen Vektor, um die nächste Iteration zu beginnen, und wird K1 aktualisiert, um alles Einsen zu enthalten. In V2 ist kein weiteres Quelldatenelement verblieben und K2 wird aktualisiert, um alles Nullen zu enthalten.

[0048] Die Zusammenführungsoperation **320** veranschaulicht ein Überlaufszenarium, in dem die Anzahl neuer Datenelemente (C0) größer als die Anzahl von Maskenbits mit dem Wert null in K1 (d. h. die Anzahl von A1) ist. Somit können nicht alle neuen Datenelemente in V2 in V1 verschoben werden. In diesem Beispiel ist das eingekreiste C0 an Position 7 aus V2 in V2 verblieben, während die anderen C0-en an den Positionen 2, 4 und 6 in V1 verschoben worden sind. In dieser Ausführungsform werden die niederwertigen Elemente aus V2 in V1 verschoben; in alternativen Ausführungsformen werden die höherwertigen Elemente aus V2 in V1 verschoben. Außerdem aktualisiert die Zusammenführungsoperation **320** die entsprechenden Maskenbits in K1 und K2.

[0049] Nach der Zusammenführungsoperation **320** enthält V1 einen vollen Vektor mit acht Elementen, um die zweite Iteration **330** zu beginnen, und besitzt V2 nur ein C0, das an Position 7 verblieben ist. Das entsprechende Maskenregister K1 enthält an diesem Punkt (nach der Zusammenführungsoperation **320**) alles Einsen und K2 enthält nur ein Maskenbit mit einem Wert von eins an der Position 7.

[0050] Nach der zweiten Iteration **330** enthält der Akkumulator V1 eine Kombination von B2-en und C1-en. Während die Berechnung für die Elemente B nach dieser Iteration abgeschlossen ist, können diese B2-en im Speicher, im Cache oder in einer anderen Datenablage gesichert werden. Somit ist die Bedingung für jedes der Elemente B falsch (da sie die Bedingung für eine weitere Berechnung nicht erfüllen) und ist die Bedingung für jedes der Elemente C wahr (da sie die Bedingung für die weitere Berechnung erfüllen). Somit werden die Maskenbitwerte in K1 für jene Maskenbits, die B2-en entsprechen, auf Nullen und für jene Maskenbits, die C1-en entsprechen, auf Einsen gesetzt.

[0051] Die durch B2-en gelassenen ungenutzten Schlitze können durch die verbleibenden Quelldatenelemente in V2 gefüllt werden; in diesem Fall C0 an der Position 7 von V2. Allerdings tritt in einer nachfolgenden Zusammenführungsoperation **340** ein Unterlauf auf, da es eine kleinere Anzahl von C0-en als die Anzahl von B2-en gibt. In dem in **Fig.** 3 gezeigten Unterlaufszenarium wird das niedrigstwertige B2 in V1 durch C0 ersetzt; in alternativen Ausführungsformen kann das höchstwertige B2 in V1 durch C0 ersetzt werden. Außerdem aktualisiert die Zusammenführungsoperation **340** die entsprechenden Maskenbits in K1 und K2.

[0052] Nach der Zusammenführungsoperation **340** ist der Akkumulator V1 nicht vollständig gefüllt und besitzt V2 keine nutzbaren Datenelemente mehr, die in V1 verschoben werden können. An diesem Punkt (nach der Zusammenführungsoperation **340**) enthält das Maskenregister K1 an den den C-Elementen entsprechenden Positionen Einsen und enthält K2 alles Nullen. V2 kann zusätzliche nutzbare Datenelemente laden, die in V1 verschoben werden sollen, und die Zusammenführungsoperationen von **320** und/oder **340** können wiederholt werden, bis alle nutzbaren Datenelemente verarbeitet worden sind und keine Quelldatenelemente mehr in V2 verblieben sind. An diesem Punkt kann V1 eine Anzahl zusätzlicher Iterationen durchlaufen, bis alle Elemente in V1 die geforderte Anzahl von Iterationen erreicht haben.

[0053] Es ist zu verstehen, dass die Bedeutung der Maskenbitwerte von Nullen und Einsen gegenüber der in dem Beispiel aus **Fig.** 3 gezeigten umgekehrt werden kann; z. B. kann ein Maskenbitwert null in der Bedeutung verwendet werden, dass eine Bedingung erfüllt ist, und kann ein Maskenbitwert eins in der Bedeutung verwendet werden, dass eine Bedingung nicht erfüllt ist. In einigen Ausführungsformen kann die Bedeutung der Maskenbitwerte von K1 gegenüber der Bedeutung der Maskenbitwerte K2 umgekehrt sein; z. B. kann ein Maskenbitwert K1 in der Bedeutung verwendet werden, dass eine Bedingung nicht erfüllt ist, und kann ein Maskenbitwert von K2 in der Bedeutung verwendet werden, dass die Bedingung erfüllt ist. Somit können in dem Beispiel aus **Fig.** 3 für dasselbe Szenarium unterschiedliche Maskenbitwerte verwendet werden, solange die Bedeutung jedes Maskenbit in jedem Maskenregister konsistent definiert ist, um eine konsistente Interpretation zu ermöglichen.

[0054] In Übereinstimmung mit einer Ausführungsform der Erfindung werden die im Zusammenhang mit **Fig.** 3 beschriebenen Operationen in Reaktion auf die Vektorbefehle, die die Befehle RWMASKUPDATE und SPARSEMOV enthalten, durch einen Prozessor (z. B. durch die Befehlsverarbeitungsvorrichtung **115**) ausgeführt. Der SPARSEMOV-Befehl kann verwendet werden, um Quelldatenelemente von dem Vektorregister V2 in das Vektorregister V1 zu verschieben, wobei diejenigen Zielelemente in V1, die eine Bedingung nicht erfüllen (z. B. Elemente, die keine Berechnung mehr benötigen), ersetzt werden. Der RWMASKUPDATE-Befehl kann verwendet werden, um die Maskenregister K1 und K2 zu aktualisieren und dadurch die Positionen der Datenelemente in V1 bzw. V2, die eine Bedingung erfüllen (z. B. Elemente, die eine weitere Berechnung benötigen), zu identifizieren. In einer Ausführungsform besitzt RWMASKUPDATE zwei Operanden K1 und K2 und besitzt SPARSEMOV vier Operanden K1, V1, K2 und V2. In alternativen Ausführungsformen können eini-

ge der Operationen von RWMASKUPDATE und/oder SPARSEMOV implizit sein.

[0055] Fig. 4A zeigt ein Beispiel für Pseudocode **401** und **402** für die Befehle RWMASKUPDATE und SPARSEMOV in Übereinstimmung mit einer Ausführungsform. In dem Pseudocode **401** und **402** repräsentiert KL die Vektorlänge, die die Gesamtzahl von Datenelementen in jedem Vektorregister (z. B. sowohl in V1 als auch in V2) ist. Falls ein zmm-Register als der Akkumulator mit 8-Bit-Datenelementen verwendet wird, ist KL = 512/8 = 64. Der Pseudocode **401** beschreibt den RWMASKUPDATE-Befehl und der Pseudocode **402** beschreibt den SPARSEMOV-Befehl. Es wird angemerkt, dass ein Prozessor den RWMASKUPDATE- und den SPARSEMOV-Befehl mit anderen Operationen oder mit einer anderen Logik implementieren kann, als dies in dem Pseudocode **401** und **402** gezeigt ist.

[0056] Der RWMASKUPDATE- und der SPARSEMOV-Befehl aktualisieren Maskenregister bzw. verschieben Datenelemente zwischen Vektorregistern. Es können zusätzliche Befehle ausgeführt werden, um Ergebnisse dieser Befehle zu nutzen, um dadurch eine rekursive Vektorberechnung effizienter auszuführen. **Fig.** 4B veranschaulicht ein Beispiel eines Codesegments **400**, das den RWMASKUPDATE- und den SPARSEMOV-Befehl in Übereinstimmung mit einer Ausführungsform verwendet. Wenn das Codesegment **400** durch einen Prozessor ausgeführt wird, veranlasst es, dass der Prozessor eine rekursive Vektorberechnung über unabhängige Datenelemente eines Felds X ausführt. Das Feld X kann in dem Speicher, in dem Cache oder an anderen Datenablageplätzen gespeichert sein. Das Codesegment **400** enthält einen Initialisierungsabschnitt **410**, einen Anfangs-Zusammenführungsabschnitt **420**, einen nachfolgenden Zusammenführungsabschnitt **430**, einen Rechenabschnitt **440** und einen Restabschnitt **450**. Die Operationen in jedem der Abschnitte **410**–**450** werden im Folgenden anhand des Ablaufplans aus **Fig.** 5A beschrieben, der eine Ausführungsform eines durch einen Prozessor (z. B. durch die Befehlsverarbeitungsvorrichtung **115** aus **Fig.** 1) ausgeführten Verfahrens **500** darstellt.

[0057] In dem Initialisierungsabschnitt **410** werden beide Maskenregister K1 und K2 auf null initialisiert, was angibt, dass keine nutzbaren Datenelemente in ihren entsprechenden Vektorregistern V1 und V2 sind. Der Begriff ”nutzbare Datenelemente” bedeutet Datenelemente, die eine Berechnung benötigen. Die Iterationen beginnen bei dem Anfangs-Zusammenführungsabschnitt **420**, wo zunächst K2 geprüft wird, um zu bestimmen, ob irgendwelche nutzbaren Datenelemente in V2 verblieben sind (Block **531**). Falls es keine nutzbaren Datenelemente in V2 gibt, werden Eingangsdatenelemente von dem Feld X in V2 geladen (Block **532**) und werden ihre entsprechenden Maskenbits in K2 entsprechend eingestellt.

[0058] Der nachfolgende Zusammenführungsabschnitt **430** behandelt das Szenarium, in dem V2 nutzbare Datenelemente enthält. Die nutzbaren Datenelemente können in V2 von einem vorhergehenden Überlauf verblieben sein oder können im Block **532** in V2 geladen werden. In Reaktion auf den SPARSEMOV-Befehl **431** werden diese nutzbaren Datenelemente in V2 in Übereinstimmung mit den Maskenbits in K1 und K2 in V1 verschoben (Block **533**). In Reaktion auf den RWMASKUPDATE-Befehl **433** werden nach der Verschiebung im Block **533** die Maskenregister K1 und K2 aktualisiert, um die aktuellen Positionen der nutzbaren Datenelemente in V1 bzw. V2 zu identifizieren (Block **534**).

[0059] In dem nachfolgenden Zusammenführungsabschnitt **430** wird ein zweiter SPARSEMOV-Befehl **432** ausgeführt, um die Indizes (Positionen) derjenigen Datenelemente in dem Feld X, die von V2 in V1 verschoben wurden, zu speichern, so dass Ergebnisse der Berechnung an ihre ursprünglichen Positionen in dem Feld X zurückgespeichert werden können.

[0060] Der Rechenabschnitt **440** behandelt die Vektorberechnung eines vollen Vektors (wie dadurch angegeben ist, dass die entsprechende Maske alles Einsen sind; d. h., wenn IsFullMask(K1) wahr ist). Falls V1 keinen vollen Vektor nutzbarer Datenelemente besitzt (Block **535**) und falls es Eingangsdatenelemente gibt, die nicht in V1 geladen worden sind (Block **538**), gibt es an, dass zusätzliche Eingangsdatenelemente über V2 in V1 geladen werden können (Blöcke **532**–**534**). Falls V1 keinen vollen Vektor besitzt und falls es keine weiteren Eingangselemente gibt, die in V1 geladen werden können (Block **538**), gibt es an, dass die Operationen zu dem Restabschnitt **450** übergehen, wo die verbleibenden Datenelemente in V1 berechnet werden, bis die Berechnung abgeschlossen ist und die Ergebnisse in dem Feld X zurückgesichert werden (Block **539**).

[0061] Falls V1 einen vollen Vektor nutzbarer Datenelemente besitzt (Block **535**), kann die Vektorberechnung an V1 ausgeführt werden (Block **536**). Falls irgendwelche Datenelemente in V1 keine weitere Berechnung benötigen, wird das Maskenregister K1 aktualisiert. Die Vektorberechnung wird fortgesetzt, bis ein oder mehrere Datenelemente in V1 (wie durch die entsprechenden Maskenbits mit dem Wert null in K1 angegeben ist) keine weitere Berechnung benötigen; an diesem Punkt werden diese Datenelemente in dem Feld X zurückgesichert (Block **537**). In der wie gezeigten Ausführungsform können die Datenelemente mit einem SCATTER-Befehl gesichert werden und können Maskenbits mit dem Wert null in K1 unter Verwendung einer Funktion knot (K1) identifiziert werden. Mit Ausnahme des RWMASKUPDATE-

und des SPARSEMOV-Befehls können die in dem Codesegment **400** verwendeten spezifischen Befehle und Funktionen wie etwa SCATTER, knot, IsFullMask usw. durch alternative Befehlssequenzen emuliert werden.

[0062] Die Operationen der Blöcke **531**–**537** werden wiederholt, bis es keine weiteren über V2 in VI zu ladenden Eingangsdatenelemente mehr gibt (Block **538**); d. h., wenn alle Eingangsdatenelemente im Feld X in V2 geladen worden sind und wenn alle nutzbaren Datenelemente in V2 in V1 verschoben worden sind. Das heißt, wenn der Restabschnitt **450** beginnt. An diesem Punkt kann V1 keinen vollen Vektor nutzbarer Datenelemente enthalten, während jene Datenelemente in V1 eine weitere Berechnung benötigen. Die Vektorberechnung wird fortgesetzt, bis alle verbleibenden Datenelemente in V1 die geforderte Anzahl von Iterationen erreicht haben (Block **539**). An diesem Punkt kann das Rechenergebnis in V1 (z. B. unter Verwendung eines SCATTER-Befehls) in dem Feld X zurückgesichert werden (Block **539**).

[0063] **Fig.** 5B ist ein Blockablaufplan eines Verfahrens **510** zum Ausführen des RWMASKUPDATE-Befehls in Übereinstimmung mit einer Ausführungsform. Das Verfahren **510** beginnt damit, dass ein Prozessor (z. B. die Befehlsverarbeitungsvorrichtung **115** aus **Fig.** 1) einen Maskenaktualisierungsbefehl empfängt, der ein erstes Maskenregister und ein zweites Maskenregister spezifiziert (Block **511**). Der Prozessor decodiert den Maskenaktualisierungsbefehl (Block **512**). In Reaktion auf den decodierten Maskenaktualisierungsbefehl führt der Prozessor die Operationen aus, die Folgendes enthalten: Invertieren einer gegebenen Anzahl von Maskenbits in dem ersten Maskenregister; z. B. dadurch, dass diese Maskenbits von einem ersten Bitwert (z. B. null) auf einen zweiten Bitwert (z. B. eins) gesetzt werden (Block **513**); und Invertieren der gegebenen Anzahl von Maskenbits in dem zweiten Maskenregister; z. B. dadurch, dass diese Maskenbits von dem zweiten Bitwert (z. B. eins) auf den ersten Bitwert (z. B. null) gesetzt werden (**514**). Die gegebene Anzahl ist die kleinere der Anzahl der Maskenbits in dem ersten Maskenregister mit dem ersten Bitwert und der Anzahl der Maskenbits in dem zweiten Maskenregister mit dem zweiten Bitwert. In einer alternativen Ausführungsform kann der erste Bitwert eins sein und kann der zweite Bitwert null sein.

[0064] **Fig.** 5C ist ein Blockablaufplan eines Verfahrens **520** zum Ausführen des SPARSEMOV-Befehls in Übereinstimmung mit einer Ausführungsform. Das Verfahren **520** beginnt damit, dass ein Prozessor (z. B. die Befehlsverarbeitungsvorrichtung **115** aus **Fig.** 1) einen Vektor-Verschiebebefehl empfängt, der ein erstes Maskenregister, ein zweites Maskenregister, ein erstes Vektorregister und ein zweites Vektorregister spezifiziert (Block **521**). Der Prozessor decodiert die Vektor-Verschiebeoperation (Block **522**). In Reaktion auf den decodierten Vektor-Verschiebebefehl und auf der Grundlage der Maskenbitwerte in dem ersten und in dem zweiten Maskenregister ersetzt der Prozessor eine gegebene Anzahl von Zieldatenelementen in dem ersten Vektorregister durch eine gegebene Anzahl von Quelldatenelementen in dem zweiten Vektorregister (Block **523**). In einer Ausführungsform entspricht jedes Quelldatenelement einem Maskenbit in dem zweiten Maskenregister mit einem zweiten Bitwert (z. B. eins) und wobei jedes Zieldatenelement einem Maskenbit in dem ersten Maskenregister mit einem ersten Bitwert (z. B. null) entspricht. In einer alternativen Ausführungsform kann der erste Bitwert eins sein und kann der zweite Bitwert null sein. Die gegebene Anzahl ist die kleinere der Anzahl der Maskenbits in dem ersten Maskenregister mit dem ersten Bitwert und der Anzahl der Maskenbits in dem zweiten Maskenregister mit dem zweiten Bitwert.

[0065] In verschiedenen Ausführungsformen können die Verfahren aus **Fig.** 5A–C durch einen Mehrzweckprozessor, durch einen Spezialprozessor (z. B. durch einen Graphikprozessor oder durch einen digitalen Signalprozessor) oder durch einen anderen Typ einer digitalen Logikvorrichtung oder Befehlsverarbeitungsvorrichtung ausgeführt werden. In einigen Ausführungsformen können die Verfahren aus **Fig.** 5A–C durch die Befehlsverarbeitungsvorrichtung **115** aus **Fig.** 1 oder durch einen ähnlichen Prozessor, eine ähnliche Vorrichtung oder ein ähnliches System wie die in **Fig.** 7–**Fig.** 13 gezeigten Ausführungsformen ausgeführt werden. Darüber hinaus können die Befehlsverarbeitungsvorrichtung **115** aus **Fig.** 1 sowie der Prozessor, die Vorrichtung oder das System, die in **Fig.** 7–**Fig.** 13 gezeigt sind, entweder dieselben, ähnliche oder andere Ausführungsformen von Operationen und Verfahren als die Verfahren aus **Fig.** 5A–C ausführen.

[0066] In einigen Ausführungsformen kann die Befehlsverarbeitungsvorrichtung **115** aus **Fig.** 1 mit einem Befehlsumsetzer zusammenarbeiten, der einen Befehl von einem Quellbefehlssatz in einen Zielbefehlssatz umsetzt. Zum Beispiel kann der Befehlsumsetzer einen Befehl in einen oder mehrere andere Befehle, die durch den Kern verarbeitet werden sollen, (z. B. unter Verwendung einer statischen binären Übersetzung, einer dynamischen binären Übersetzung einschließlich einer dynamischen Kompilierung) übersetzen, morphen, emulieren oder auf andere Weise umsetzen. Der Befehlsumsetzer kann in Software, in Hardware, in Firmware oder in einer Kombination davon implementiert werden. Der Befehlsumsetzer kann sich auf einem Prozessor, außerhalb eines Prozessors oder teilweise auf einem und teilweise außerhalb eines Prozessors befinden.

[0067] Fig. 6 ist ein Blockschaltplan, der die Verwendung eines Softwarebefehlsumsetzers in Übereinstimmung mit Ausführungsformen der Erfindung gegenüberstellt. In der dargestellten Ausführungsform ist der Befehlsumsetzer ein Softwarebefehlsumsetzer, obwohl der Befehlsumsetzer alternativ in Software, in Firmware, in Hardware oder in verschiedenen Kombinationen davon implementiert werden kann. **Fig.** 6 zeigt, dass ein Programm in einer höheren Sprache **602** unter Verwendung eines x86-Compilers **604** kompiliert werden kann, um binären x86-Code **606** zu erzeugen, der systemspezifisch durch einen Prozessor mit wenigstens einem x86-Befehlssatzkern **616** ausgeführt werden kann. Der Prozessor mit wenigstens einem x86-Befehlssatzkern **616** repräsentiert irgendeinen Prozessor, der im Wesentlichen dieselben Funktionen wie ein Intel-Prozessor mit wenigstens einem x86-Befehlssatzkern ausführen kann, indem er (1) einen wesentlichen Teil des Befehlssatzes des Intel-x86-Befehlssatzkerns oder (2) Objektcodeversionen von Anwendungen oder anderer Software, die darauf gerichtet sind, auf einem Intel-Prozessor mit wenigstens einem x86-Befehlssatzkern ausgeführt zu werden, kompatibel ausführt oder auf andere Weise verarbeitet, um im Wesentlichen dasselbe Ergebnis wie ein Intel-Prozessor mit wenigstens einem x86-Befehlssatzkern zu erzielen. Der x86-Compiler **604** repräsentiert einen Compiler, der betreibbar ist, um x86-Binärcode **606** (z. B. Objektcode) zu erzeugen, der mit einer oder ohne eine zusätzliche Bindeverarbeitung in dem Prozessor mit wenigstens einem x86-Befehlssatzkern **616** ausgeführt werden kann. Ähnlich zeigt **Fig.** 6, dass das Programm in der höheren Sprache **602** unter Verwendung eines Compilers **608** mit einem alternativen Befehlssatz kompiliert werden kann, um alternativen Befehlssatzbinärcode **610** zu erzeugen, der durch einen Prozessor ohne wenigstens einen x86-Befehlssatzkern **614** (z. B. einen Prozessor mit Kernen, die den MIPS-Befehlssatz von MIPS Technologies aus Sunnyvale, CA, ausführen und/oder die den ARM-Befehlssatz der ARM Holdings aus Sunnyvale, CA, ausführen) systemspezifisch ausgeführt werden kann. Der Befehlsumsetzer **612** wird verwendet, um den x86-Binärcode **606** in Code umzusetzen, der durch den Prozessor ohne einen x86-Befehlssatzkern **614** systemspezifisch ausgeführt werden kann. Dieser umgesetzte Code ist wahrscheinlich nicht derselbe wie der Binärcode **610** des alternativen Befehlssatzes, da ein Befehlsumsetzer, der dazu fähig ist, schwierig herzustellen ist; allerdings führt der umgesetzte Code dieselbe allgemeine Operation aus und kann er aus Befehlen von dem alternativen Befehlssatz bestehen. Somit repräsentiert der Befehlsumsetzer **612** Software, Firmware, Hardware oder eine Kombination davon, die durch Emulation, Simulation oder irgendeinen Prozess ermöglichen, dass ein Prozessor oder eine andere elektronische Vorrichtung, die keinen x86-Befehlssatz-Prozessor oder x86-Befehlssatz-Kern besitzt, den x86-Binärcode **606** ausführt.

Beispielhafte Kernarchitekturen

In-Order- und Out-of Order-Kern-Blockschaltplan

[0068] Fig. 7A ist ein Blockschaltplan, der sowohl eine beispielhafte In-order-Pipeline als auch eine beispielhafte Registerumbenennungs-Out-of-Order-Ausgabe/Ausführungs-Pipeline in Übereinstimmung mit Ausführungsformen der Erfindung darstellt. **Fig.** 7B ist ein Blockschaltplan, der sowohl eine beispielhafte Ausführungsform eines In-order-Architektur-Kerns als auch eines beispielhaften Registerumbenennungs-Out-of-Order-Ausgabe/Ausführungs-Architektur-Kerns zur Aufnahme in einen Prozessor in Übereinstimmung mit Ausführungsformen der Erfindung darstellt. Die Kästen in durchgezogenen Linien in **Fig.** 7A und **Fig.** 7B veranschaulichen die In-order-Pipeline und den In-Order-Kern, während die optionale Hinzufügung der Kästen in Strichlinien die Registerumbenennungs-Out-of-Order-Ausgabe/Ausführungs-Pipeline und den Registerumbenennungs-Out-of-Order-Ausgabe/Ausführungs-Kern veranschaulichen. Ausgehend davon, dass der In-order-Aspekt eine Teilmenge des Out-of-Order-Aspekts ist, wird der Out-of-Order-Aspekt beschrieben.

[0069] In **Fig.** 7A enthält eine Prozessorpipeline **700** eine Abrufstufe **702**, eine Längendecodierungsstufe **704**, eine Decodierungsstufe **706**, eine Zuordnungsstufe **708**, eine Umbenennungsstufe **710**, eine Planungsstufe (auch als eine Absende- oder Ausgabestufe bekannt) **712**, eine Registerlese/Speicherlese-Stufe **714**, eine Ausführungsstufe **716**, eine Zurückschreib/Speicherschreib-Stufe **718**, eine Ausnahmebehandlungsstufe **722** und eine Festschreibstufe **724**.

[0070] Fig. 7B zeigt einen Prozessorkern **790**, der eine Frontend-Einheit **730** enthält, die mit einer Ausführungsmaschineneinheit **750** gekoppelt ist, wobei beide mit einer Speichereinheit **770** gekoppelt sind. Der Kern **790** kann ein Reduced-Instruction-Set-Computing-Kern (RISC-Kern), ein Complex-Instruction-Set-Computing-Kern (CISC-Kern), ein Very-Long-Instruction-Word-Kern (VLIW-Kern) oder ein Hybridkerntyp oder ein alternativer Kerntyp sein. Als eine abermals andere Option kann der Kern **790** ein Spezialkern wie etwa z. B. ein Netz- oder Kommunikationskern, eine Kompressionsmaschine, ein Coprozessorkern, ein Mehrzweck-Computergraphikverarbeitungseinheits-Kern (GPGPU-Kern), ein Graphikkern oder dergleichen sein.

[0071] Die Frontend-Einheit **730** enthält eine Verzweigungsvorhersageeinheit **732**, die mit einer Befehls-Cache-Einheit **734** gekoppelt ist, die mit einem

Befehls-Translation-Lookaside-Puffer (TLB) **736** gekoppelt ist, der mit einer Befehlsabrufeinheit **738** gekoppelt ist, die mit einer Decodiereinheit **740** gekoppelt ist. Die Decodiereinheit **740** (oder der Decodierer) kann Befehle decodieren und als eine Ausgabe eine oder mehrere Mikrooperationen, Mikrocodeeinsprungpunkte, Mikrobefehle, andere Befehle oder andere Steuersignale, die aus den ursprünglichen Befehlen decodiert werden oder die diese auf andere Weise widerspiegeln oder von ihnen hergeleitet sind, erzeugen. Die Decodiereinheit **740** kann unter Verwendung vieler verschiedener Mechanismen implementiert werden. Beispiele geeigneter Mechanismen enthalten Nachschlagetabellen, Hardwareimplementierungen, programmierbare Logikanordnung (PLAs), Mikrocode-Nur-Lese-Speicher (ROMs) usw., sind darauf aber nicht beschränkt. In einer Ausführungsform enthält der Kern **790** einen Mikrocode-ROM oder ein anderes Medium, der bzw. das Mikrocode für bestimmte Makrobefehle (z. B. in der Decodiereinheit **740** oder auf andere Weise innerhalb der Frontend-Einheit **730**) speichert. Die Decodiereinheit **740** ist mit einer Umbenennungs/Zuweisungs-Einheit **752** in der Ausführungsmaschineneinheit **750** gekoppelt.

[0072] Die Ausführungsmaschineneinheit **750** enthält die Umbenennungs/Zuweisungs-Einheit **752**, die mit einer Ausscheideeinheit **754** und mit einem Satz einer oder mehrerer Planungseinheiten **756** gekoppelt ist. Die eine oder die mehreren Planungseinheiten **756** repräsentieren irgendeine Anzahl unterschiedlicher Planer einschließlich Reservierungsstationen, eines zentralen Befehlsfensters usw. Die eine oder die mehreren Planungseinheiten **756** sind mit der bzw. mit den physikalischen Registerdateieinheiten **758** gekoppelt. Jede der einen oder mehreren physikalischen Registerdateieinheiten **758** repräsentiert eine oder mehrere physikalische Registerdateien, von denen verschiedene einen oder mehrere verschiedene Datentypen wie etwa skalar-ganzzahlig, skalar-Gleitkomma, gepackt-ganzzahlig, gepackt-Gleitkomma, Vektor-ganzzahlig, Vektor-Gleitkomma, Status (z. B. einen Befehlszeiger, der die Adresse des nächsten auszuführenden Befehls ist) usw. speichern. In einer Ausführungsform umfasst die physikalische Registerdateieinheit **758** eine Vektorregistereinheit, eine Schreibmaskenregistereinheit und eine Skalarregistereinheit. Diese Registereinheiten können Architekturvektorregister, Vektormaskenregister und Mehrzweckregister bereitstellen. Die eine oder die mehreren Registerdateieinheiten **758** sind mit der Ausscheideeinheit **754** überlappt, um verschiedene Arten darzustellen, in denen die Registerumbenennung und die Out-of-Order-Ausführung (z. B. unter Verwendung eines oder mehrerer Umordnungspuffer und einer oder mehrerer Ausscheideregisterdateien; unter Verwendung einer oder mehrerer Zukunftsdateien, eines oder mehrerer Historienpuffer und einer oder mehrerer Ausscheideregisterdateien; unter Verwendung von Registerkarten und eines Pools von Registern; usw.) implementiert werden können. Die Ausscheideeinheit **754** und die eine oder die mehreren physikalischen Registerdateieinheiten **758** sind mit dem einen oder den mehreren Ausführungsclustern **760** gekoppelt. Der eine oder die mehreren Ausführungscluster **760** enthalten einen Satz einer oder mehrerer Ausführungseinheiten **762** und einen Satz einer oder mehrerer Speicherzugriffseinheiten **764**. Die Ausführungseinheiten **762** können verschiedene Operationen z. B. Stellenversetzungsoperationen, Addition, Subtraktion, Multiplikation) und diese an verschiedenen Datentypen (z. B. skalar-Gleitkomma, gepackt-ganzzahlig, gepackt-Gleitkomma, Vektor-ganzzahlig, Vektor-Gleitkomma) ausführen. Obwohl einige Ausführungsformen eine Anzahl von Ausführungseinheiten enthalten können, die für spezifische Funktionen oder Sätze von Funktionen vorgesehen sind, können andere Ausführungsformen nur eine Ausführungseinheit oder mehrere Ausführungseinheiten, die alle sämtliche Funktionen ausführen, enthalten. Da bestimmte Ausführungsformen für bestimmte Typen von Daten/Operationen getrennte Pipelines erzeugen (z. B. eine Skalar-ganzzahlig-Pipeline, eine Skalar-Gleitkomma/Gepackt-ganzzahlig/Gepackt-Gleitkomma/Vektor-ganzzahlig/Vektor-Gleitkomma-Pipeline und/oder eine Speicherzugriffspipeline, die jeweils ihre eigene Planungseinheit, ihre eigenen eine oder mehreren physikalischen Registerdateieinheiten und/oder ihren eigenen einen oder mehrere Ausführungscluster besitzen – und da im Fall einer getrennten Speicherzugriffspipeline bestimmte Ausführungsformen implementiert werden, in denen nur der Ausführungscluster dieser Pipeline eine oder mehrere Speicherzugriffseinheiten **764** besitzt), sind die eine oder die mehreren Planungseinheiten **756**, physikalischen Registerdateieinheiten **758** und Ausführungscluster **760** in der Weise gezeigt, dass sie möglicherweise mehrere sind. Außerdem ist zu verstehen, dass dort, wo getrennte Pipelines verwendet werden, eine oder mehrere dieser Pipelines Out-of-Order-Ausgabe/Ausführungs-- und der Rest In-Order- sein können.

[0073] Der Satz von Speicherzugriffseinheiten **764** ist mit der Speichereinheit **770** gekoppelt, die eine Daten-TLB-Einheit **762** enthält, die mit einer Daten-Cache-Einheit **774** gekoppelt ist, die mit einer Cache-Einheit **776** der 2. Ebene (L2-Cache-Einheit) gekoppelt ist. In einer beispielhaften Ausführungsform können die Speicherzugriffseinheiten **764** eine Ladeeinheit, eine Speicheradresseneinheit und eine Speicherdateneinheit enthalten, von denen jede mit der Daten-TLB-Einheit **772** in der Speichereinheit **770** gekoppelt ist. Ferner ist die Befehls-Cache-Einheit **734** mit einer Cache-Einheit **776** der 2. Ebene (L2-Cache-Einheit) in der Speichereinheit **770** gekoppelt. Die L2-Cache-Einheit **776** ist mit einer oder mit meh-

reren weiteren Cache-Ebenen und schließlich mit einem Hauptspeicher gekoppelt.

[0074] Beispielsweise kann die beispielhafte Registerumbenennungs-Out-of-Order-Ausgabe/Ausführungs-Kern-Architektur die Pipeline **700** Folgendes implementieren: 1) Der Befehlsabruf **738** führt die Abruf- und Längendecodierungsstufe **702** und **704** aus; 2) die Decodiereinheit **740** führt die Decodierungsstufe **706** aus; 3) die Umbenennungs/Zuweisungs-Einheit **752** führt die Zuweisungsstufe **708** und die Umbenennungsstufe **710** aus; 4) die eine oder die mehreren Planungseinheiten **756** führen die Planungsstufe **712** aus; 5) die eine oder die mehreren physikalischen Registerdateieinheiten **758** und die Speichereinheit **770** führen die Registerlese/Speicherlese-Stufe **714** aus; der Ausführungscluster **760** führt die Ausführungsstufe **716** aus; 6) die Speichereinheit **770** und die eine oder die mehreren physikalischen Registerdateieinheiten **758** führen die Zurückschreib/Speicherschreib-Stufe **718** aus; 7) verschiedene Einheiten können an der Ausnahmebehandlungsstufe **722** beteiligt sein; und 8) die Ausscheideeinheit **754** und die eine oder die mehreren physikalischen Registerdateieinheiten **758** führen die Festschreibstufe **724** aus.

[0075] Der Kern **790** kann einen oder mehrere Befehlssätze (z. B. den x86-Befehlssatz (mit einigen Erweiterungen, die bei neueren Versionen hinzugefügt worden sind); den MIPS-Befehlssatz der MIPS Technologies aus Sunnyvale, CA; den ARM-Befehlssatz (mit optionalen zusätzlichen Erweiterungen wie etwa NEON) der ARM Holdings aus Sunnyvale, CA) einschließlich des einen oder der mehreren hier beschriebenen Befehle unterstützen. In einer Ausführungsform enthält der Kern **790** eine Logik zum Unterstützen einer Befehlssatzerweiterung für gepackte Daten (z. B. SSE, AVX1, AVX2 usw.) und ermöglicht er dadurch, dass die von vielen Multimediaanwendungen verwendeten Operationen unter Verwendung gepackter Daten ausgeführt werden.

[0076] Es ist zu verstehen, dass der Kern Multithreading (die Ausführung zweier oder mehrerer paralleler Sätze von Operationen oder Threads) unterstützen kann und dass er dies auf eine Vielzahl von Arten einschließlich Zeitschlitz-Multithreading, gleichzeitigem Multithreading (wobei ein einzelner physikalischer Kern einen Logikkern für jeden der Threads bereitstellt, den der physikalische Kern gleichzeitig im Multithreading ausführt) oder einer Kombination davon (z. B. Zeitschlitz-Abrufen und Zeitschlitz-Decodieren und gleichzeitiges Multithreading danach wie etwa in der Intel®-Hyperthreading-Technologie) tun kann.

[0077] Obwohl die Registerumbenennung im Kontext der Out-of-Order-Ausführung beschrieben ist, ist zu verstehen, dass die Registerumbenennung in einer In-order-Architektur verwendet werden kann. Obwohl die dargestellte Ausführungsform des Prozessors ebenfalls getrennte Befehls- und Daten-Cache-Einheiten **734/774** und eine gemeinsam genutzte L2-Cache-Einheit **776** enthält, können alternative Ausführungsformen einen einzelnen internen Cache sowohl für Anwendungen als auch für Daten wie etwa z. B. einen internen Cache der 1. Ebene (L1-Cache) oder mehrere Ebenen eines internen Caches aufweisen. In einigen Ausführungsformen kann das System eine Kombination eines internen Caches und eines externen Caches, der extern gegenüber dem Kern und/oder dem Prozessor ist, enthalten. Alternativ kann der gesamte Cache gegenüber dem Kern und/oder dem Prozessor extern sein.

Spezifische beispielhafte In-order-Kern-Architektur

[0078] Die **Fig.** 8A–B veranschaulichen einen Blockschaltplan einer spezielleren beispielhaften In-order-Kern-Architektur, wobei der Kern einer von mehreren Logikblöcken (einschließlich anderer Kerne vom selben Typ und/oder anderer Typen) in einem Chip ist. Die Logikblöcke kommunizieren über ein Verdrahtungsnetz mit hoher Bandbreite (z. B. über ein Ringnetz) mit einiger Festfunktionslogik, mit Speicher-E/A-Schnittstellen und mit anderer notwendiger E/A-Logik, die von der Anwendung abhängen.

[0079] **Fig.** 8A ist ein Blockschaltplan eines Einprozessorkerns zusammen mit seiner Verbindung mit dem Verdrahtungsnetz **802** auf dem Chip und mit seiner lokalen Teilmenge des Caches **804** der 2. Ebene (L2-Caches) in Übereinstimmung mit Ausführungsformen der Erfindung. In einer Ausführungsform unterstützt ein Befehlsdecodierer **800** den x86-Befehlssatz mit einer Befehlssatzerweiterung für gepackte Daten. Ein L1-Cache **806** ermöglicht Zugriffe mit niedriger Latenzzeit auf den Cache-Speicher in die Skalar- und Vektoreinheiten. Obwohl in einer Ausführungsform eine Skalareinheit **808** und eine Vektoreinheit **810** getrennte Registersätze (die Skalarregister **812** bzw. Vektorregister **814**) verwenden und Daten, die zwischen ihnen übertragen werden, in den Speicher geschrieben und daraufhin von einem Cache der 1. Ebene **806** (L1-Cache) zurückgelesen werden (um den Entwurf zu vereinfachen), können alternative Ausführungsformen der Erfindung eine andere Herangehensweise verwenden (z. B. einen einzelnen Registersatz verwenden oder einen Kommunikationsweg enthalten, der ermöglicht, dass Daten zwischen den zwei Registerdateien übertragen werden, ohne geschrieben und zurückgelesen zu werden).

[0080] Die lokale Teilmenge des L2-Caches **804** ist Teil eines globalen L2-Caches, der in getrennte lokale Teilmengen, eine pro Prozessorkern, unterteilt ist. Jeder Prozessorkern weist einen direkten Zugriffsweg auf seine eigene lokale Teilmenge des L2-Caches **804** auf. Daten, die von einem Prozessor-

kern gelesen werden, werden in seiner L2-Cache-Teilmenge **804** gespeichert und auf sie kann parallel dazu, dass andere Prozessorkerne auf ihre eigenen lokalen L2-Cache-Teilmengen zugreifen, schnell zugegriffen werden. Die durch einen Prozessorkern geschriebenen Daten werden in seiner eigenen L2-Cache-Teilmenge **804** gespeichert und bei Bedarf aus anderen Teilmengen geräumt. Das Ringnetz stellt die Kohärenz für gemeinsam genutzte Daten sicher. Das Ringnetz ist bidirektional, um zu ermöglichen, dass Agenten wie etwa Prozessorkerne, L2-Caches und andere Logikblöcke innerhalb des Chips miteinander kommunizieren. Jeder Ringdatenweg ist pro Richtung 1012 Bits breit.

[0081] **Fig.** 8B ist eine erweiterte Ansicht eines Teils des Prozessorkerns in **Fig.** 8A in Übereinstimmung mit Ausführungsformen der Erfindung. **Fig.** 8B enthält einen L1-Daten-Cache **806A**, der Teil des L1-Caches **804** ist, sowie weitere Einzelheiten hinsichtlich der Vektoreinheit **810** und der Vektorregister **814**. Genauer ist die Vektoreinheit **810** eine 16-breite Vektorverarbeitungseinheit (VPU) (siehe die 16-breite ALU **828**), die einen oder mehrere Ganzzahlbefehle, einfach genaue Gleitkommabefehle und doppelt genaue Gleitkommabefehle ausführt. Die VPU unterstützt mit der Swizzle-Einheit **820** Swizzling der Registereingaben, numerische Umsetzung mit den Einheiten **822A**–B für numerische Umsetzung und Replikation mit der Replikationseinheit **824** an dem Speichereingang. Schreibmaskenregister **826** ermöglichen das Vorhersagen resultierender Vektorschreibvorgänge.

Prozessor mit integriertem Speichercontroller und integrierter Graphik

[0082] **Fig.** 9 ist ein Blockschaltplan eines Prozessors **900**, der mehr als einen Kern aufweisen kann, einen integrierten Speichercontroller aufweisen kann und eine integrierte Graphik aufweisen kann, in Übereinstimmung mit Ausführungsformen der Erfindung. Die Kästen in durchgezogenen Linien in **Fig.** 9 veranschaulichen einen Prozessor **900** mit einem einzelnen Kern **902A**, einem Systemagenten **910** und einem Satz einer oder mehrerer Buscontrollereinheiten **916**, während die optionale Hinzufügung der Kästen in Strichlinien einen alternativen Prozessor **900** mit mehreren Kernen **902A**–N, einem Satz einer oder mehrerer integrierter Speichercontrollereinheiten **914** in der Systemagenteneinheit **910** und einer Speziallogik **908** darstellt.

[0083] Somit können verschiedene Implementierungen des Prozessors **900** Folgendes enthalten: 1) eine CPU mit der Speziallogik **908**, die eine integrierte Graphik und/oder eine wissenschaftliche (Durchsatz-)Logik ist (die einen oder mehrere Kerne enthalten kann) und mit den Kernen **902A**–N, die einer oder mehrere Mehrzweckkerne (z. B. Mehrzweck-In-order-Kerne, Mehrzweck-Out-of-Order-Kerne, eine Kombination der zwei) sind; 2) einen Coprozessor mit den Kernen **902A**–N, die eine große Anzahl von Spezialkernen sind, die primär für Graphik und/oder für Wissenschaft (Durchsatz) bestimmt sind; und 3) einen Coprozessor mit den Kernen **902A**–N, die eine große Anzahl von Mehrzweck-In-order-Kernen sind. Somit kann der Prozessor **900** ein Mehrzweckprozessor, ein Coprozessor oder ein Spezialprozessor wie etwa z. B. ein Netz- oder Kommunikationsprozessor, eine Kompressionsmaschine, ein Graphikprozessor, eine GPGPU (Mehrzweck-Graphikverarbeitungseinheit), ein Coprozessor mit vielen integrierten Kernen (MIC-Coprozessor) mit hohem Durchsatz (der 30 oder mehr Kerne enthält), ein eingebetteter Prozessor oder dergleichen sein. Der Prozessor kann in einem oder in mehreren Chips implementiert sein. Der Prozessor **900** kann unter Verwendung einer Anzahl von Prozesstechnologien wie etwa z. B. BiCMOS, CMOS oder NMOS ein Teil eines oder mehrerer Substrate oder auf ihnen implementiert sein.

[0084] Die Speicherhierarchie enthält eine oder mehrere Cache-Ebenen innerhalb der Kerne, einen Satz oder eine oder mehrere gemeinsam genutzte Cache-Einheiten **906** und externen Speicher (nicht gezeigt), der mit dem Satz integrierter Speichercontrollereinheiten **914** gekoppelt ist. Der Satz gemeinsam genutzter Cache-Einheiten **906** kann einen oder mehrere Caches mittlerer Ebene wie etwa einen Cache der 2. Ebene (L2-Cache), einen Cache der 3. Ebene (L3-Cache), einen Cache der 4. Ebene (L4-Cache) oder Caches anderer Ebenen, einen Cache der letzten Ebene (LLC-Cache) und/oder Kombinationen davon enthalten. Obwohl in einer Ausführungsform eine ringgestützte Verdrahtungseinheit **912** die integrierte Graphiklogik **908**, den Satz gemeinsam genutzter Cache-Einheiten **906** und die Systemagenteneinheit **910**/die eine oder die mehreren integrierten Speichercontrollereinheiten **914** verbindet, können alternative Ausführungsformen irgendeine Anzahl gut bekannter Techniken zum Verbinden solcher Einheiten verwenden. In einer Ausführungsform wird die Kohärenz zwischen einer oder mehreren Cache-Einheiten **906** und Kernen **902A**–N aufrechterhalten.

[0085] In einigen Ausführungsformen sind einer oder mehrere der Kerne **902A**–N Multithreadingfähig. Der Systemagent **910** enthält jene Komponenten, die die Kerne **902A**–N koordinieren und betreiben. Die Systemagenteneinheit **910** kann z. B. eine Leistungssteuereinheit (PCU) und eine Anzeigeeinheit enthalten. Die PCU kann Logik und Komponenten, die notwendig sind, um den Leistungszustand der Kerne **902A**–N und der integrierten Graphiklogik **908** zu regeln, sein oder enthalten. Die Anzeigeeinheit dient zum Ansteuern einer oder mehrerer extern verbundener Anzeigen.

[0086] Die Kerne **902A**–N können hinsichtlich des Architekturbefehlssatzes homogen oder heterogen sein; d. h., zwei oder mehr der Kerne **902A**–N können zur Ausführung desselben Befehlssatzes fähig sein, während andere zur Ausführung nur einer Teilmenge dieses Befehlssatzes oder eines anderen Befehlssatzes fähig sein können.

Beispielhafte Computerarchitekturen

[0087] Die **Fig.** 10–**Fig.** 13 sind Blockschaltpläne beispielhafter Computerarchitekturen. Andere Systementwürfe und Systemkonfigurationen, die auf den Gebieten der Laptops, der Desktops, der Hand-PCs, der Personal Digital Assistants, der technischen Workstations, der Server, der Netzvorrichtungen, der Netz-Hubs, der Switches, der eingebetteten Prozessoren, der digitalen Signalprozessoren (DSPs), der Graphikvorrichtungen, der Videospielvorrichtungen, der Set-Top-Boxen, der Mikrocontroller, der Mobiltelephone, der tragbaren Medienspieler, der Handvorrichtungen und verschiedener anderer elektronischer Vorrichtungen bekannt sind, sind ebenfalls geeignet. Allgemein sind eine breite Vielfalt von Systemen oder elektronischen Vorrichtungen, die einen Prozessor und/oder eine andere Ausführungslogik, wie sie hier offenbart sind, enthalten können, allgemein geeignet.

[0088] Nun anhand von **Fig.** 10 ist ein Blockschaltplan eines Systems **1000** in Übereinstimmung mit einer Ausführungsform der vorliegenden Erfindung gezeigt. Das System **1000** kann einen oder mehrere Prozessoren **1010**, **1015** enthalten, die mit einem Controller-Hub **1020** gekoppelt sind. In einer Ausführungsform enthält der Controller-Hub **1020** einen Graphikspeichercontroller-Hub (GMCH) **1090** und einen Eingabe/Ausgabe-Hub (IOH) **1050** (die sich auf getrennten Chips befinden können); enthält der GMCH **1090** Speicher und Graphikcontroller, mit denen der Speicher **1040** und ein Coprozessor **1045** gekoppelt sind; koppelt der IOH **1050** die Eingabe/Ausgabe-Vorrichtungen (E/A-Vorrichtungen) **1060** mit dem GMCH **1090**. Alternativ sind der Speichercontroller und/oder der Graphikcontroller innerhalb des Prozessors (wie hier beschrieben) integriert, wobei der Speicher **1040** und der Coprozessor **1045** mit dem Prozessor **1010** und mit dem Controller-Hub **1020** in einem einzelnen Chip mit dem IOH **1050** direkt gekoppelt sind.

[0089] Das optionale Wesen zusätzlicher Prozessoren **1015** ist in **Fig.** 10 mit Strichlinien bezeichnet. Jeder Prozessor **1010**, **1015** kann einen oder mehrere der hier beschriebenen Prozessorkerne enthalten und kann dieselbe Version des Prozessors **900** sein.

[0090] Der Speicher **1040** kann z. B. dynamischer Schreib-Lese-Speicher (DRAM), Phasenwechselspeicher (PCM) oder eine Kombination der zwei sein. Für wenigstens eine Ausführungsform kommuniziert der Controller-Hub **1020** mit dem einen oder mit den mehreren Prozessoren **1010**, **1015** über einen Mehr-Abzweig-Bus wie etwa einen Frontside-Bus (FSB), über eine Punkt-zu-Punkt-Schnittstelle wie etwa QuickPath Interconnect (QPI) oder über eine ähnliche Verbindung **1095**.

[0091] In einer Ausführungsform ist der Coprozessor **1045** ein Mehrzweckprozessor wie etwa z. B. ein MIC-Prozessor mit hohem Durchsatz, ein Netz- oder Kommunikationsprozessor, eine Kompressionsmaschine, ein Graphikprozessor, eine GPGPU, ein eingebetteter Prozessor oder dergleichen. In einer Ausführungsform kann der Controller-Hub **1020** einen integrierten Graphikbeschleuniger enthalten.

[0092] Hinsichtlich eines Spektrums von Gütemetriken einschließlich Architektureigenschaften, Mikroarchitektureigenschaften, thermischer Eigenschaften, Leistungsverbrauchseigenschaften und dergleichen kann es zwischen den physikalischen Betriebsmitteln **1010**, **1015** eine Vielzahl von Unterschieden geben.

[0093] In einer Ausführungsform führt der Prozessor **1010** Befehle aus, die Datenverarbeitungsoperationen eines allgemeinen Typs steuern. In die Befehle können Coprozessorbefehle eingebettet sein. Der Prozessor **1010** erkennt diese Coprozessorbefehle als einen Typ, der durch den angeschlossenen Coprozessor **1045** ausgeführt werden sollte. Dementsprechend gibt der Prozessor **1010** diese Coprozessorbefehle (oder Steuersignale, die Coprozessorbefehle repräsentieren) auf einem Coprozessorbus oder auf einer anderen Verdrahtung an den Coprozessor **1045** aus. Der eine oder die mehreren Coprozessoren **1045** nehmen die empfangenen Coprozessorbefehle an und führen sie aus.

[0094] Nun in **Fig.** 11 ist ein Blockschaltplan eines ersten spezielleren beispielhaften Systems **1100** in Übereinstimmung mit einer Ausführungsform der vorliegenden Erfindung gezeigt. Wie in **Fig.** 11 gezeigt ist, ist das Multiprozessorsystem **1100** ein Punkt-zu-Punkt-Verdrahtungssystem und enthält es einen ersten Prozessor **1170** und einen zweiten Prozessor **1180**, die über eine Punkt-zu-Punkt-Verdrahtung **1150** gekoppelt sind. Jeder der Prozessoren **1170** und **1180** kann eine selbe Version des Prozessors **900** sein. In einer Ausführungsform der Erfindung sind die Prozessoren **1170** und **1180** die Prozessoren **1010** bzw. **1015**, während der Coprozessor **1138** der Coprozessor **1045** ist. In einer anderen Ausführungsform sind die Prozessoren **1170** und **1180** der Prozessor **1010** bzw. der Coprozessor **1045**.

[0095] Die Prozessoren **1170** und **1180** sind in der Weise gezeigt, dass sie integrierte Speichercontrollereinheiten (IMC-Einheiten) **1172** bzw. **1182** enthalten. Außerdem enthält der Prozessor **1170** als Teil seiner Buscontrollereinheiten Punkt-zu-Punkt-

Schnittstellen (P-P-Schnittstellen) **1176** und **1178**; ähnlich enthält der zweite Prozessor **1180** P-P-Schnittstellen **1186** und **1188**. Die Prozessoren **1170**, **1180** können unter Verwendung der P-P-Schnittstellenschaltungen **1178**, **1188** über eine Punkt-zu-Punkt-Schnittstelle (P-P-Schnittstelle) **1150** Informationen austauschen. Wie in **Fig.** 11 gezeigt ist, koppeln IMCs **1172** und **1182** die Prozessoren mit jeweiligen Speichern, d. h. mit einem Speicher **1132** und mit einem Speicher **1134**, die Abschnitte des Hauptspeichers sein können, der lokal an die jeweiligen Prozessoren angeschlossen ist.

[0096] Die Prozessoren **1170**, **1180** können über einzelne P-P-Schnittstellen **1152**, **1154** unter Verwendung von Punkt-zu-Punkt-Schnittstellenschaltungen **1176**, **1194**, **1186**, **1198** mit einem Chipsatz **1190** Informationen austauschen. Der Chipsatz **1190** kann über eine Hochleistungsschnittstelle **1139** optional Informationen mit dem Coprozessor **1138** austauschen. In einer Ausführungsform ist der Coprozessor **1138** ein Spezialprozessor wie etwa z. B. ein MIC-Prozessor mit hohem Durchsatz, ein Netz- oder Kommunikationsprozessor, eine Kompressionsmaschine, ein Graphikprozessor, eine GPGPU, ein eingebetteter Prozessor oder dergleichen.

[0097] Ein gemeinsam genutzter Cache (nicht gezeigt) kann in einem der Prozessoren oder außerhalb beider Prozessoren enthalten sein, aber dennoch mit den Prozessoren über eine P-P-Verdrahtung verbunden sein, so dass die lokalen Cache-Informationen eines oder beider Prozessoren in dem gemeinsam genutzten Cache gespeichert werden können, falls ein Prozessor in eine Betriebsart mit niedrigem Leistungsverbrauch versetzt wird.

[0098] Der Chipsatz **1190** kann über eine Schnittstelle **1196** mit einem ersten Bus **1116** gekoppelt sein. In einer Ausführungsform kann der erste Bus **1116** ein Peripheral-Component-Interconnect-Bus (PCI-Bus) oder ein Bus wie etwa ein PCI-Express-Bus oder ein anderer E/A-Verdrahtungsbus der dritten Generation sein, obwohl der Schutzumfang der vorliegenden Erfindung darauf nicht beschränkt ist.

[0099] Wie in **Fig.** 11 gezeigt ist, können mit dem ersten Bus **1116** zusammen mit einer Busbrücke **1118**, die den ersten Bus **1116** mit einem zweiten Bus **1120** koppelt, verschiedene E/A-Vorrichtungen **1114** gekoppelt sein. In einer Ausführungsform sind einer oder mehrere zusätzliche Prozessoren **1115** wie etwa Coprozessoren, MIC-Prozessoren mit hohem Durchsatz, GPGPUs, Beschleuniger (wie etwa z. B. Graphikbeschleuniger oder digitale Signalverarbeitungseinheiten (DSP-Einheiten)), frei programmierbare logische Anordnungen oder irgendein anderer Prozessor mit dem ersten Bus **1116** gekoppelt. In einer Ausführungsform kann der zweite Bus **1120** ein Bus mit niedriger Anschlussstiftanzahl (LPC) sein. In einer Ausführungsform können verschiedene Vorrichtungen einschließlich z. B. einer Tastatur und/oder einer Maus **1122**, Kommunikationsvorrichtungen **1127** und einer Ablageeinheit **1128** wie etwa eines Plattenlaufwerks oder einer anderen Massenablagevorrichtung, die Befehle/Code und Daten **1130** enthalten kann, mit einem zweiten Bus **1120** gekoppelt sein. Ferner kann mit dem zweiten Bus **1120** eine Audio-E/A **1124** gekoppelt sein. Es wird angemerkt, dass andere Architekturen möglich sind. Zum Beispiel kann ein System anstelle der Punkt-zu-Punkt-Architektur aus **Fig.** 11 einen Mehr-Abzweig-Bus oder eine andere solche Architektur implementieren.

[0100] Nun in **Fig.** 12 ist ein Blockschaltplan eines zweiten spezielleren beispielhaften Systems **1200** in Übereinstimmung mit einer Ausführungsform der vorliegenden Erfindung gezeigt. Gleiche Elemente in **Fig.** 11 und **Fig.** 12 tragen gleiche Bezugszeichen und bestimmte Aspekte aus **Fig.** 11 sind aus **Fig.** 12 weggelassen worden, um die Verdeckung anderer Aspekte aus **Fig.** 12 zu vermeiden.

[0101] **Fig.** 12 veranschaulicht, dass die Prozessoren **1170**, **1180** einen integrierten Speicher und eine E/A-Steuerlogik (”CL”) **1172** bzw. **1182** enthalten können. Somit enthält die CL **1172**, **1182** integrierte Speichercontrollereinheiten und enthält sie eine E/A-Steuerlogik. **Fig.** 12 stellt dar, dass nicht nur die Speicher **1132**, **1134** mit der CL **1172**, **1182** gekoppelt sind, sondern dass auch die E/A-Vorrichtungen **1214** mit der Steuerlogik **1172**, **1182** gekoppelt sind. Mit dem Chipsatz **1190** sind Legacy-E/A-Vorrichtungen **1215** gekoppelt.

[0102] Nun in **Fig.** 13 ist ein Blockschaltplan eines SoC **1300** in Übereinstimmung mit einer Ausführungsform der vorliegenden Erfindung gezeigt. Ähnliche Elemente in **Fig.** 9 tragen gleiche Bezugszeichen. Außerdem sind Kästen in Strichlinien optionale Merkmale in fortgeschritteneren SoCs. In **Fig.** 13 sind eine oder mehrere Verdrahtungseinheiten **1302** mit Folgendem gekoppelt: einem Anwendungsprozessor **1310**, der einen Satz eines oder mehrerer Kerne **202A**–N und eine oder mehrere gemeinsam genutzte Cache-Einheiten **906** enthält; einer Systemagenteneinheit **910**; einem oder mehreren Buscontrollereinheiten **916**; einer oder mehrerer integrierter Speichercontrollereinheiten **914**; einem Satz oder einem oder mehreren Coprozessoren **1320**, die eine integrierte Graphiklogik, einen Bildprozessor, einen Audioprozessor und einen Videoprozessor enthalten können; einer statischen Schreib-Lese-Speichereinheit (SRAM-Einheit) **1330**; einer Einheit **1332** für direkten Speicherzugriff (DMA); und einer Anzeigeeinheit **1340** zum Koppeln mit einer oder mit mehreren externen Anzeigen. In einer Ausführungsform enthalten der eine oder die mehreren Coprozessoren **1320** einen Spezialprozessor wie etwa z.

B. einen Netz- oder Kommunikationsprozessor, eine Kompressionsmaschine, eine GPGPU, einen MIC-Prozessor mit hohem Durchsatz, einen eingebetteten Prozessor oder dergleichen.

[0103] Ausführungsformen der hier offenbarten Mechanismen können in Hardware, in Software, in Firmware oder in einer Kombination solcher Implementierungsherangehensweisen implementiert werden. Ausführungsformen der Erfindung können als Computerprogramme oder als Programmcode implementiert werden, die in programmierbaren Systemen ausgeführt werden, die wenigstens einen Prozessor, ein Ablagesystem (einschließlich flüchtiger und nichtflüchtiger Speicher- und/oder Ablageelemente), wenigstens eine Eingabevorrichtung und wenigstens eine Ausgabevorrichtung umfassen.

[0104] Programmcode wie etwa der in **Fig.** 11 dargestellte Code **1130** kann angewendet werden, um Befehle einzugeben, um die hier beschriebenen Funktionen auszuführen und Ausgabeinformationen zu erzeugen. Die Ausgabeinformationen können auf bekannte Weise an eine oder mehrere Ausgabevorrichtungen angelegt werden. Für diese Anmeldung enthält ein Verarbeitungssystem irgendein System, das einen Prozessor aufweist, wie etwa z. B.: einen digitalen Signalprozessor (DSP), einen Mikrocontroller, eine anwendungsspezifische integrierte Schaltung (ASIC) oder einen Mikroprozessor.

[0105] Der Programmcode kann in einer höheren prozeduralen oder objektorientierten Programmiersprache zum Kommunizieren mit einem Verarbeitungssystem implementiert werden. Auf Wunsch kann der Programmcode ebenfalls in Assembler- oder Maschinensprache implementiert werden. Tatsächlich sind die hier beschriebenen Mechanismen in Bezug auf den Schutzumfang nicht auf irgendeine bestimmte Programmiersprache beschränkt. Auf jeden Fall kann die Sprache eine kompilierte oder interpretierte Sprache sein.

[0106] Einer oder mehrere Aspekte wenigstens einer Ausführungsform können durch repräsentative Befehle implementiert werden, die in einem maschinenlesbaren Medium gespeichert sind, das verschiedene Logik innerhalb des Prozessors repräsentiert, die, wenn sie durch eine Maschine gelesen wird, veranlasst, dass die Maschine Logik herstellt, um die hier beschriebenen Techniken auszuführen. Solche Darstellungen, die als ”IP-Kerne” bekannt sind, können in einem konkreten, maschinenlesbaren Medium gespeichert sein und an verschiedene Kunden oder Fertigungseinrichtungen geliefert werden, um sie in die Fertigungsmaschinen zu laden, die die Logik oder den Prozessor tatsächlich herstellen.

[0107] Solche maschinenlesbaren Ablagemedien können ohne Beschränkung nichtflüchtige, konkrete Anordnungen von Artikeln, die durch eine Maschine oder Vorrichtung hergestellt oder gebildet sind, einschließlich Ablagemedien wie etwa Festplatten, irgendeinem anderen Typ von Platten einschließlich Disketten, optischer Platten, Compact-Disc-Nur-Lese-Speicher (CD-ROMs), Compact-Disc-Rewritables (CD-RWs) und magnetooptischer Platten, Halbleitervorrichtungen wie etwa Nur-Lese-Speicher (ROMs), Schreib-Lese-Speicher (RAMs) wie etwa dynamische Schreib-Lese-Speicher (DRAMs), statische Schreib-Lese-Speicher (SRAMs), löschbare programmierbare Nur-Lese-Speicher (EPROMs), Flash-Speicher, elektrisch löschbare programmierbare Nur-Lese-Speicher (EEPROMs), Phasenwechselspeicher (PCM), magnetischer oder optischer Karten oder irgendeines anderen Medientyps, der zum Speichern elektronischer Befehle geeignet ist, enthalten.

[0108] Dementsprechend enthalten Ausführungsformen der Erfindung ebenfalls nichtflüchtige konkrete maschinenlesbare Medien, die Befehle enthalten oder die Entwurfsdaten wie etwa eine Hardware Description Language (HDL), die die hier beschriebenen Strukturen, Schaltungen, Vorrichtungen, Prozessoren und/oder Systemmerkmale definiert, enthalten. Solche Ausführungsformen können auch als Programmprodukte bezeichnet werden.

[0109] Obwohl bestimmte beispielhafte Ausführungsformen beschrieben und in den beigefügten Zeichnungen gezeigt sind, sind diese Ausführungsformen selbstverständlich lediglich veranschaulichend und nicht einschränkend für die umfassende Erfindung und ist diese Erfindung selbstverständlich nicht auf die spezifischen gezeigten und beschriebenen Konstruktionen und Anordnungen beschränkt, da dem Durchschnittsfachmann auf dem Gebiet beim Studium dieser Offenbarung verschiedene andere Änderungen einfallen können. In einem Gebiet der Technologie wie etwa diesem, in dem das Wachstum schnell ist und weitere Fortschritte nicht leicht vorherzusehen sind, können die offenbarten Ausführungsformen in Bezug auf Anordnung und Einzelheit, wie es durch technologische Fortschritte ermöglicht wird, leicht änderbar sein, ohne von den Prinzipien der vorliegenden Offenbarung oder von dem Schutzumfang der beigefügten Ansprüche abzuweichen.

Patentansprüche

1. Vorrichtung, die Folgendes umfasst:
eine Registerdatei, die ein erstes Maskenregister, ein zweites Maskenregister, ein erstes Vektorregister und ein zweites Vektorregister enthält; und
eine Ausführungsschaltungsanordnung, die mit der Registerdatei gekoppelt ist, wobei die Ausführungsschaltungsanordnung einen Befehl zum Ersetzen einer gegebenen Anzahl von Zieldatenelementen in

dem ersten Vektorregister durch die gegebene Anzahl von Quelldatenelementen in dem zweiten Vektorregister ausführen soll, wobei jedes Quelldatenelement einem Maskenbit in dem zweiten Maskenregister mit einem zweiten Bitwert entspricht und wobei jedes Zieldatenelement einem Maskenbit in dem ersten Maskenregister mit einem ersten Bitwert entspricht.

2. Vorrichtung nach Anspruch 1, wobei die Ausführungsschaltungsanordnung für jedes Maskenbit des ersten Maskenregisters mit dem ersten Bitwert das zweite Vektorregister nach einem Datenelement, das ein entsprechendes Maskenbit mit dem zweiten Bitwert in dem zweiten Maskenregister als eines der Quelldatenelemente aufweist, durchsuchen soll.

3. Vorrichtung nach Anspruch 1, wobei die Ausführungsschaltungsanordnung eine rekursive Berechnung über Datenelemente in dem ersten Vektorregister ausführen soll und Ergebnisse der rekursiven Berechnung in dem ersten Vektorregister akkumulieren soll.

4. Vorrichtung nach Anspruch 1, wobei die Ausführungsschaltungsanordnung eine rekursive Berechnung über Datenelemente in dem ersten Vektorregister ausführen soll, bis ein oder mehrere Datenelemente in dem ersten Vektorregister keine weitere Berechnung benötigen, und das eine oder die mehreren Datenelemente unter Verwendung des ersten Maskenregisters als die Zieldatenelemente kennzeichnen soll.

5. Vorrichtung nach Anspruch 1, wobei die gegebene Anzahl von Zieldatenelementen in dem ersten Vektorregister entweder niederwertige Elemente oder höherwertige Elemente in dem ersten Vektorregister sind und wobei die gegebene Anzahl von Quelldatenelementen in dem zweiten Vektorregister entweder niederwertige Elemente oder höherwertige Elemente in dem zweiten Vektorregister sind.

6. Vorrichtung nach Anspruch 1, wobei der erste Bitwert ein inverser des zweiten Bitwerts ist.

7. Vorrichtung nach Anspruch 1, wobei der erste Bitwert derselbe wie der zweite Bitwert ist.

8. Vorrichtung nach Anspruch 1, wobei die gegebene Anzahl die kleinere der Anzahl der Maskenbits in dem ersten Maskenregister mit dem ersten Bitwert und der Anzahl der Maskenbits in dem zweiten Maskenregister mit dem zweiten Bitwert ist.

9. Verfahren, das Folgendes umfasst:
Empfangen eines Vektor-Verschiebebefehls, der ein erstes Maskenregister, ein zweites Maskenregister, ein erstes Vektorregister und ein zweites Vektorregister spezifiziert, durch einen Prozessor; und
Ersetzen einer gegebenen Anzahl von Zieldatenelementen in dem ersten Vektorregister durch die gegebene Anzahl von Quelldatenelementen in dem zweiten Vektorregister in Reaktion auf den Vektor-Verschiebebefehl, wobei jedes Quelldatenelement einem Maskenbit in dem zweiten Maskenregister mit einem zweiten Bitwert entspricht und wobei jedes Zieldatenelement einem Maskenbit in dem ersten Maskenregister mit einem ersten Bitwert entspricht.

10. Verfahren nach Anspruch 9, das ferner Folgendes umfasst:
Durchsuchen des zweiten Vektorregisters nach einem Datenelement, das ein entsprechendes Maskenbit des zweiten Bitwerts in dem zweiten Maskenregister als eines der Quelldatenelemente aufweist, für jedes Maskenbit des ersten Maskenregisters mit dem ersten Bitwert.

11. Verfahren nach Anspruch 9, das ferner Folgendes umfasst:
Ausführen einer rekursiven Berechnung über Datenelemente in dem ersten Vektorregister; und
Akkumulieren von Ergebnissen der rekursiven Berechnung in dem ersten Vektorregister.

12. Verfahren nach Anspruch 9, das ferner Folgendes umfasst:
Ausführen einer rekursiven Berechnung über Datenelemente in dem ersten Vektorregister, bis eines oder mehrere Datenelemente in dem ersten Vektorregister keine weitere Berechnung benötigen; und
Kennzeichnen des einen oder der mehreren Datenelemente als die Zieldatenelemente unter Verwendung des ersten Maskenregisters.

13. Verfahren nach Anspruch 9, wobei die gegebene Anzahl von Quelldatenelementen in dem zweiten Vektorregister entweder niederwertige Datenelemente oder höherwertige Datenelemente in dem zweiten Vektorregister sind und wobei die gegebene Anzahl von Zieldatenelementen in dem ersten Vektorregister entweder niederwertige Datenelemente oder höherwertige Datenelemente in dem ersten Vektorregister sind.

14. Verfahren nach Anspruch 9, wobei der erste Bitwert ein inverser des zweiten Bitwerts ist.

15. Verfahren nach Anspruch 9, wobei der erste Bitwert derselbe wie der zweite Bitwert ist.

16. Verfahren nach Anspruch 9, wobei die gegebene Anzahl die kleinere der Anzahl von Maskenbits in dem ersten Maskenregister mit dem ersten Bitwert und der Anzahl von Maskenbits in dem zweiten Maskenregister mit dem zweiten Bitwert ist.

17. System, das Folgendes umfasst:

Speicher zum Speichern eines Eingangsdatenfelds, das mehrere Eingangsdatenelemente enthält;
eine Registerdatei, die ein erstes Maskenregister, ein zweites Maskenregister, ein erstes Vektorregister und ein zweites Vektorregister enthält; und
eine Ausführungsschaltungsanordnung, die mit dem Speicher und mit der Registerdatei gekoppelt ist, wobei die Ausführungsschaltungsanordnung an dem ersten Vektorregister eine rekursive Berechnung für mehrere Iterationen ausführen soll, wobei mehr als eine der Iterationen die folgenden Operationen enthält:
eine Vektorladeoperation zum Laden wenigstens eines Abschnitts der Eingangsdatenelemente von dem Eingangsdatenfeld in das zweite Vektorregister,
eine Vektor-Verschiebeoperation zum Verschieben der Eingangsdatenelemente in dem zweiten Vektorregister in das erste Vektorregister,
eine Maskenaktualisierungsoperation zum Aktualisieren des ersten und des zweiten Maskenregisters, um in dem ersten bzw. in dem zweiten Vektorregister Datenelemente zu identifizieren, die eine weitere Berechnung benötigen,
eine Vektorberechnungsoperation zum Verarbeiten der Datenelemente in dem ersten Vektorregister, und eine Vektorspeicheroperation zum Speichern von Ergebnissen der Vektorberechnungsoperation in dem Speicher.

18. System nach Anspruch 17, wobei die Ausführungsschaltungsanordnung die Vektor-Verschiebeoperation zum Ersetzen einer gegebenen Anzahl von Zieldatenelementen in dem ersten Vektorregister durch die gegebene Anzahl von Datenelementen in dem zweiten Vektorregister ausführt, wobei jedes Quelldatenelement einem Maskenbit in dem zweiten Maskenregister mit einem zweiten Bitwert entspricht und wobei jedes Zieldatenelement einem Maskenbit in dem ersten Maskenregister mit einem ersten Bitwert entspricht.

19. System nach Anspruch 17, wobei der erste Bitwert ein inverser des zweiten Bitwerts ist.

20. System nach Anspruch 17, wobei der erste Bitwert derselbe wie der zweite Bitwert ist.

21. System nach Anspruch 17, wobei die gegebene Anzahl die kleinere der Anzahl der Maskenbits in dem ersten Maskenregister mit dem ersten Bitwert und der Anzahl der Maskenbits in dem zweiten Maskenregister mit dem zweiten Bitwert ist.

22. System nach Anspruch 17, wobei die Ausführungsschaltungsanordnung die Maskenaktualisierungsoperation zum Aktualisieren des ersten Maskenregisters und des zweiten Maskenregisters ausführt, so dass die Maskenbits des ersten Maskenregisters und des zweiten Maskenregisters entsprechende Datenelemente des ersten Vektorregisters bzw. des zweiten Vektorregisters, die eine weitere Berechnung benötigen, identifizieren.

23. System nach Anspruch 17, wobei die Ausführungsschaltungsanordnung die Vektor-Verschiebeoperation in Reaktion auf einen Vektor-Verschiebebefehl, der das erste Vektorregister, das erste Maskenregister, das zweite Vektorregister und das zweite Maskenregister als Operanden spezifiziert, ausführt.

24. System nach Anspruch 17, wobei die Ausführungsschaltungsanordnung die Maskenaktualisierungsoperation in Reaktion auf einen Maskenaktualisierungsbefehl, der das erste Maskenregister und das zweite Maskenregister als Operanden spezifiziert, ausführt.

Es folgen 15 Seiten Zeichnungen

Anhängende Zeichnungen

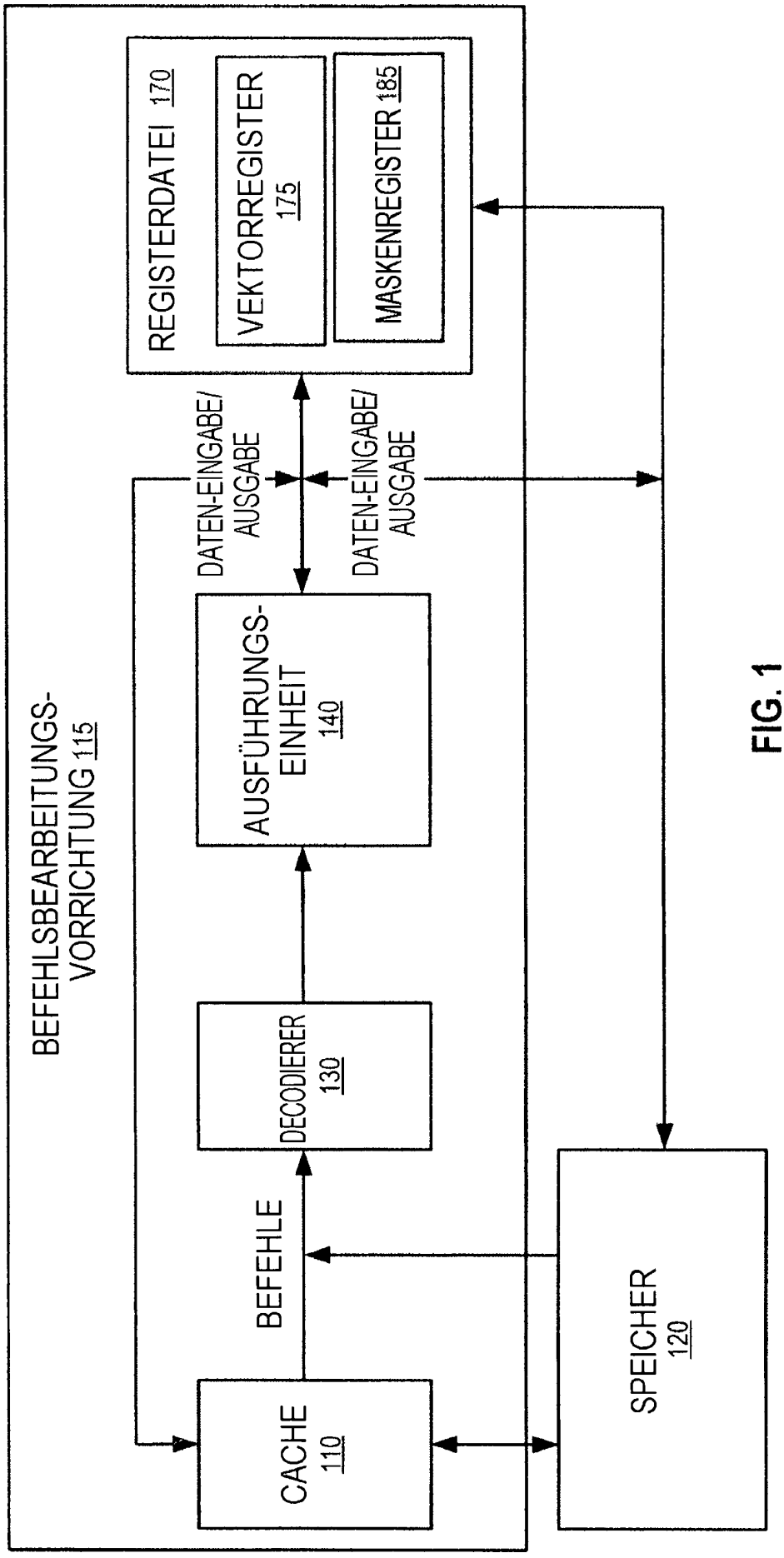


FIG. 1

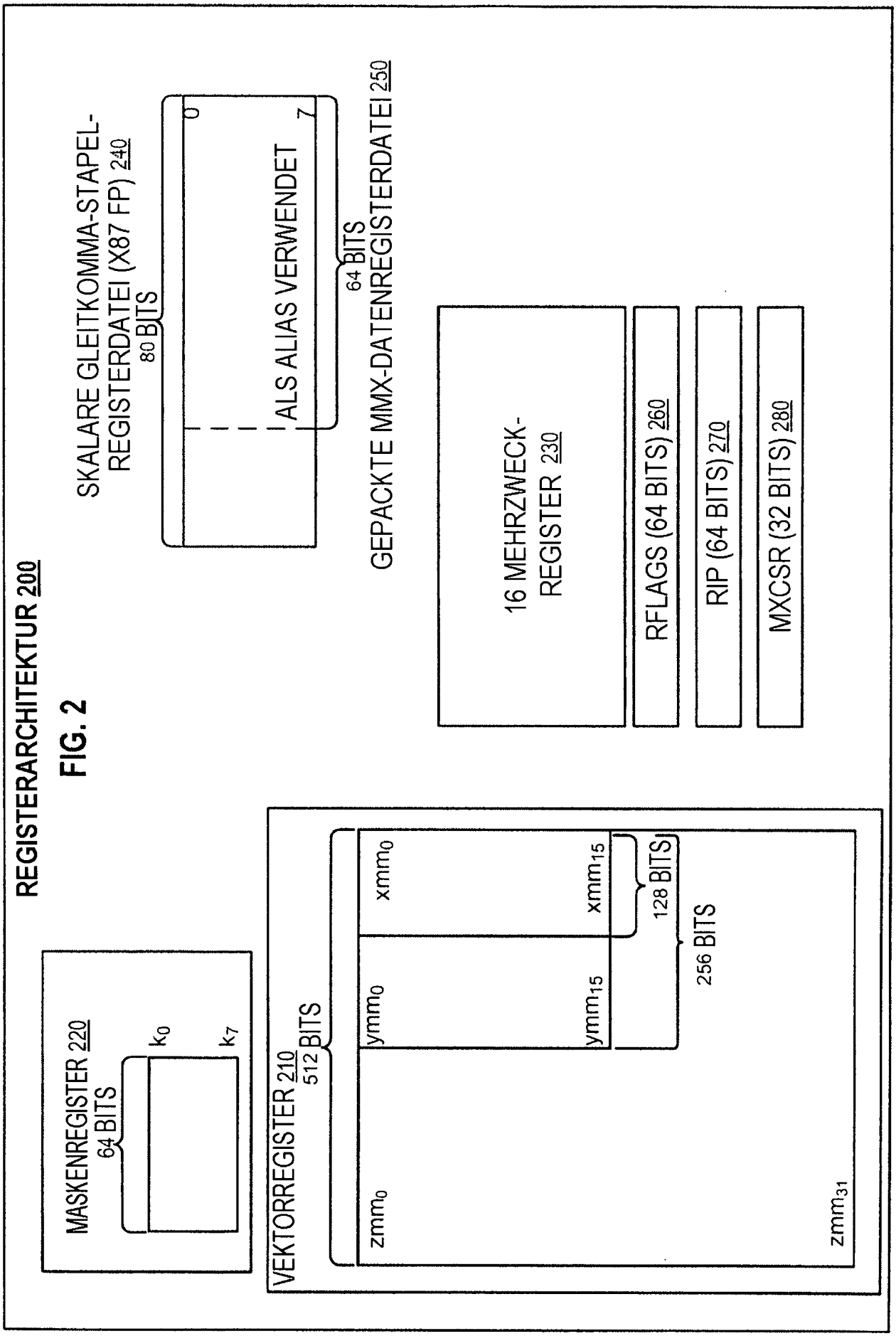

REGISTERARCHITEKTUR 200
MASKENREGISTER 220
64 BITS
k_0
k_7
VEKTORREGISTER 210
512 BITS
zmm_0
ymm_0
xmm_0
ymm_{15}
xmm_{15}
128 BITS
256 BITS
zmm_{31}
SKALARE GLEITKOMMA-STAPEL-REGISTERDATEI (X87 FP) 240
80 BITS
0
7
ALS ALIAS VERWENDET
64 BITS
GEPACKTE MMX-DATENREGISTERDATEI 250
16 MEHRZWECK-REGISTER 230
RFLAGS (64 BITS) 260
RIP (64 BITS) 270
MXCSR (32 BITS) 280

FIG. 2

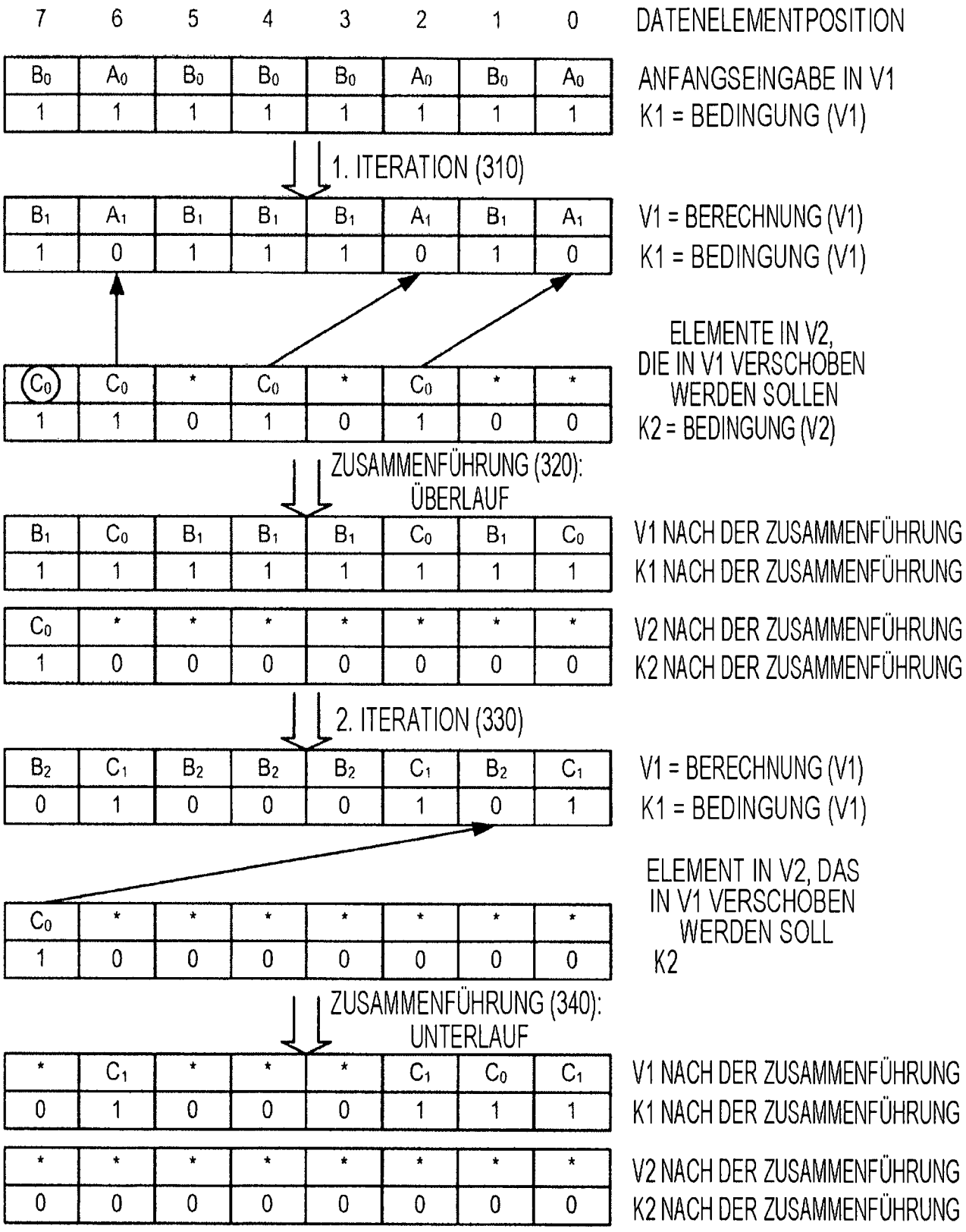

7 6 5 4 3 2 1 0
DATENELEMENTPOSITION
B0 A0 B0 B0 B0 A0 B0 A0
1 1 1 1 1 1 1 1
ANFANGSEINGABE IN V1
K1 = BEDINGUNG (V1)
1. ITERATION (310)
B1 A1 B1 B1 B1 A1 B1 A1
1 0 1 1 1 0 1 0
V1 = BERECHNUNG (V1)
K1 = BEDINGUNG (V1)
C0 C0 * C0 * C0 * *
1 1 0 1 0 1 0 0
ELEMENTE IN V2, DIE IN V1 VERSCHOBEN WERDEN SOLLEN
K2 = BEDINGUNG (V2)
ZUSAMMENFÜHRUNG (320): ÜBERLAUF
B1 C0 B1 B1 B1 C0 B1 C0
1 1 1 1 1 1 1 1
V1 NACH DER ZUSAMMENFÜHRUNG
K1 NACH DER ZUSAMMENFÜHRUNG
C0 * * * * * * *
1 0 0 0 0 0 0 0
V2 NACH DER ZUSAMMENFÜHRUNG
K2 NACH DER ZUSAMMENFÜHRUNG
2. ITERATION (330)
B2 C1 B2 B2 B2 C1 B2 C1
0 1 0 0 0 1 0 1
V1 = BERECHNUNG (V1)
K1 = BEDINGUNG (V1)
C0 * * * * * * *
1 0 0 0 0 0 0 0
ELEMENT IN V2, DAS IN V1 VERSCHOBEN WERDEN SOLL
K2
ZUSAMMENFÜHRUNG (340): UNTERLAUF
* C1 * * * C1 C0 C1
0 1 0 0 0 1 1 1
V1 NACH DER ZUSAMMENFÜHRUNG
K1 NACH DER ZUSAMMENFÜHRUNG
* * * * * * * *
0 0 0 0 0 0 0 0
V2 NACH DER ZUSAMMENFÜHRUNG
K2 NACH DER ZUSAMMENFÜHRUNG

FIG. 3

401

```
rwmaskupdate(K1,K2)
    for (i = 0, k = 0; i < KL; i++){
            if (!K1[i]){
                    for (; k < KL; k++){
                            if (K2[k]){
                                    K2[k] = 0;
                                    K1[i] = 1;
                                    k++;
                                    break;
                            }
                    }
            }
    }
```

402

```
sparsemov(K1,V1,K2,V2)
    for (i = 0, k = 0; i < KL; i++){
            if (!K1[i]){
                    for(; k < KL; k++){
                            if (K2[k]){
                                    V1[i] = V2[k];
                                    k++;
                                    break;
                            }
                    }
            }
    }
```

FIG. 4A


400
410
420
430
440
450
431
432
433
i = 0; //Initialisiere Schleifenzähler
v_index = -1:-2:...:-KL+1:-KL //Initialisiere Vektor der Indizes
v_KL = KL:KL:...:KL:KL //Inkrementiere für Vektor der Indizes
K1 = 0; //Akkumulator ist anfangs leer
K2 = 0; //noch keine Überläufe
do{
if (K2 == 0){ //falls keine Elemente vom vorhergehenden Überlauf verblieben sind
V2 = vector_load(X[i+KL-1:i]); //Lade neue KL-Elemente des X-Felds
K2 = condition(V2); //Erzeuge Lesemaske für neue Elemente
i += KL; //Inkrementiere Schleifenzähler
v_index += v_KL; //Inkrementiere Indexvektor
} //Fahre andernfalls mit der Lesemaske K2 fort
sparsemov(K1, V1, K2, V2); //Füge neue Elemente zu V1 ...
sparsemov(K1, V3, K2, v_index); //und ihre Indizes zu V3 hinzu
rwmaskupdate(K1,K2); //Aktualisiere Lese- und Schreibmaske
IsFullMask(K1){ //nur für vollen Akkumulator
do{
V1 = computation(V1); //Dichte Berechnung über akkumulierte Daten
K1 = condition(V1); //Prüfe Bedingung nach der Berechnung
}while(K1 == 0xFFFF) //Prüfe, ob Akkumulator noch voll ist (alles 1-en in K1)
scatter(knot(K1),V1,V3,X); //Streue Elemente von V1 mit Indizes V3 in das X-Feld
}
}while((i < N) || (K2 != 0)); //Setze fort, wenn es einen weiteren Eingabestrom oder
//neue Daten in V2 gibt
//Starte Restberechnungen
K3 = K1; //Speichere Restmaske für abschließende Streuung
do{
V1{K1} = computation(V1); //Berechnung gemäß Maske K1
K1 = condition(V1); //Prüfe Bedingung nach Berechnung
}while(K1 != 0) //Prüfe, ob noch etwas zu tun bleibt
scatter(K3,V1,V3,X); //Streue verbleibende Elemente aus V1 ...
//mit Indizes V3 in das X-Feld

FIG. 4B

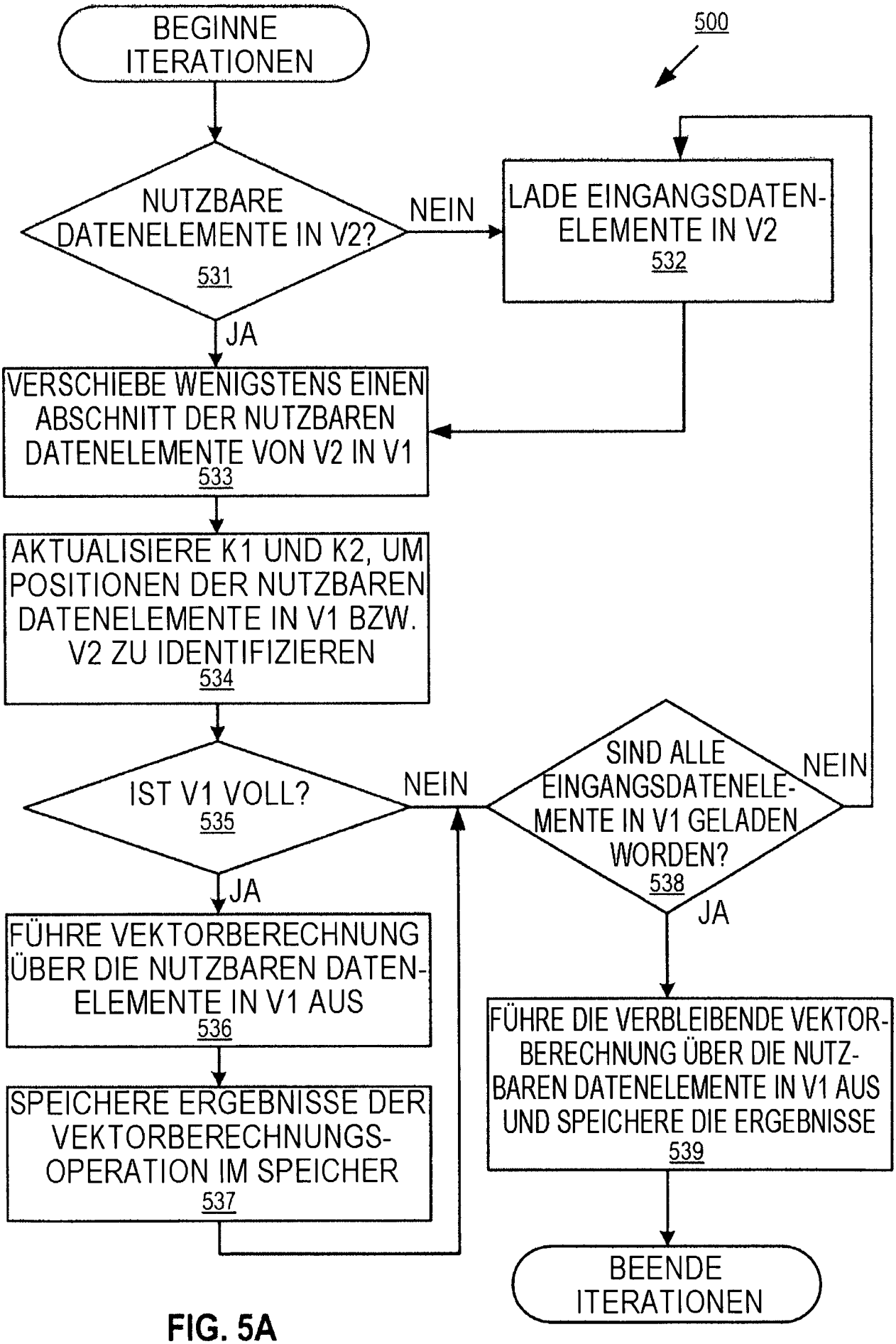

BEGINNE ITERATIONEN
500
NUTZBARE DATENELEMENTE IN V2?
531
NEIN
LADE EINGANGSDATEN-ELEMENTE IN V2
532
JA
VERSCHIEBE WENIGSTENS EINEN ABSCHNITT DER NUTZBAREN DATENELEMENTE VON V2 IN V1
533
AKTUALISIERE K1 UND K2, UM POSITIONEN DER NUTZBAREN DATENELEMENTE IN V1 BZW. V2 ZU IDENTIFIZIEREN
534
IST V1 VOLL?
535
NEIN
SIND ALLE EINGANGSDATENELE-MENTE IN V1 GELADEN WORDEN?
538
NEIN
JA
FÜHRE VEKTORBERECHNUNG ÜBER DIE NUTZBAREN DATEN-ELEMENTE IN V1 AUS
536
SPEICHERE ERGEBNISSE DER VEKTORBERECHNUNGS-OPERATION IM SPEICHER
537
FÜHRE DIE VERBLEIBENDE VEKTOR-BERECHNUNG ÜBER DIE NUTZ-BAREN DATENELEMENTE IN V1 AUS UND SPEICHERE DIE ERGEBNISSE
539
BEENDE ITERATIONEN

FIG. 5A

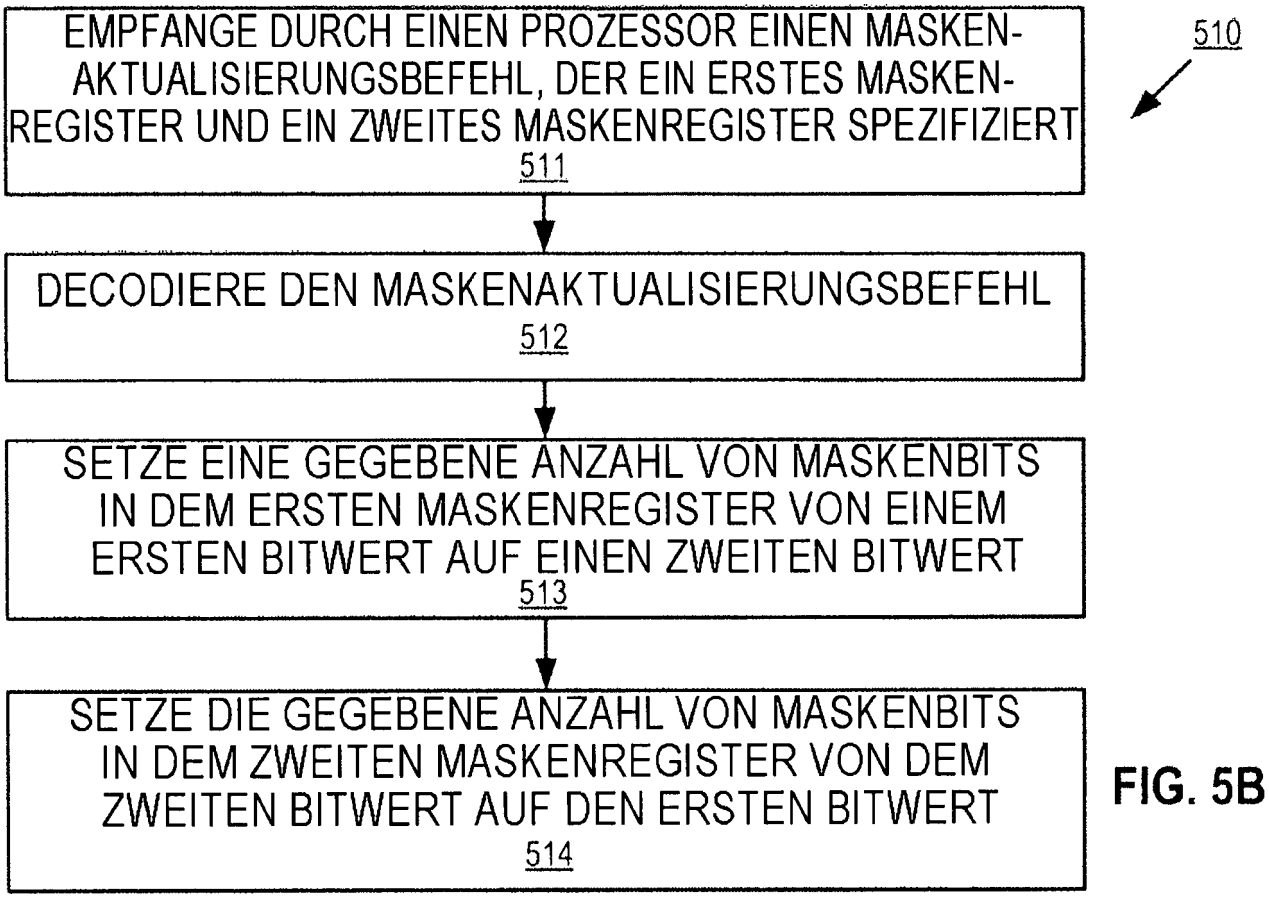

EMPFANGE DURCH EINEN PROZESSOR EINEN MASKEN-AKTUALISIERUNGSBEFEHL, DER EIN ERSTES MASKEN-REGISTER UND EIN ZWEITES MASKENREGISTER SPEZIFIZIERT
511
510
DECODIERE DEN MASKENAKTUALISIERUNGSBEFEHL
512
SETZE EINE GEGEBENE ANZAHL VON MASKENBITS IN DEM ERSTEN MASKENREGISTER VON EINEM ERSTEN BITWERT AUF EINEN ZWEITEN BITWERT
513
SETZE DIE GEGEBENE ANZAHL VON MASKENBITS IN DEM ZWEITEN MASKENREGISTER VON DEM ZWEITEN BITWERT AUF DEN ERSTEN BITWERT
514

FIG. 5B

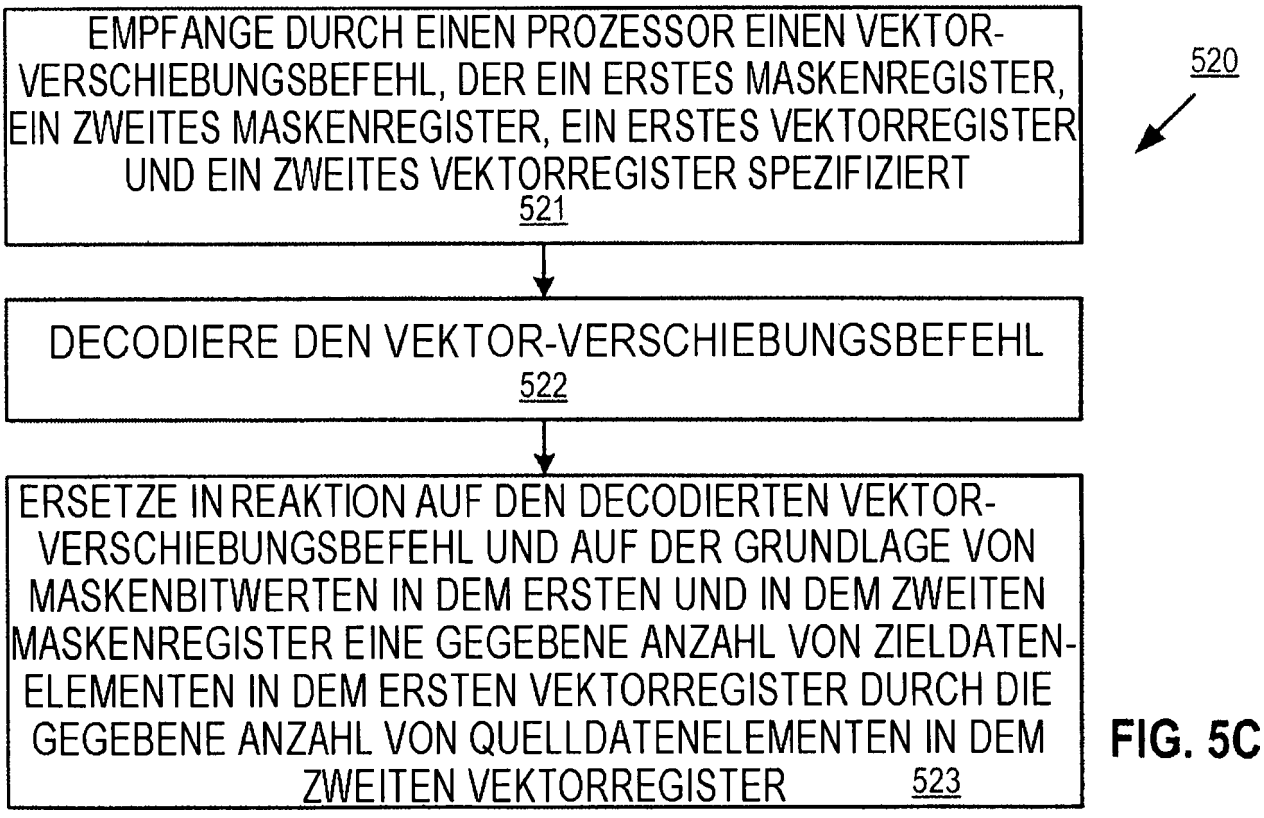

EMPFANGE DURCH EINEN PROZESSOR EINEN VEKTOR-VERSCHIEBUNGSBEFEHL, DER EIN ERSTES MASKENREGISTER, EIN ZWEITES MASKENREGISTER, EIN ERSTES VEKTORREGISTER UND EIN ZWEITES VEKTORREGISTER SPEZIFIZIERT
521
520
DECODIERE DEN VEKTOR-VERSCHIEBUNGSBEFEHL
522
ERSETZE IN REAKTION AUF DEN DECODIERTEN VEKTOR-VERSCHIEBUNGSBEFEHL UND AUF DER GRUNDLAGE VON MASKENBITWERTEN IN DEM ERSTEN UND IN DEM ZWEITEN MASKENREGISTER EINE GEGEBENE ANZAHL VON ZIELDATEN-ELEMENTEN IN DEM ERSTEN VEKTORREGISTER DURCH DIE GEGEBENE ANZAHL VON QUELLDATENELEMENTEN IN DEM ZWEITEN VEKTORREGISTER
523

FIG. 5C

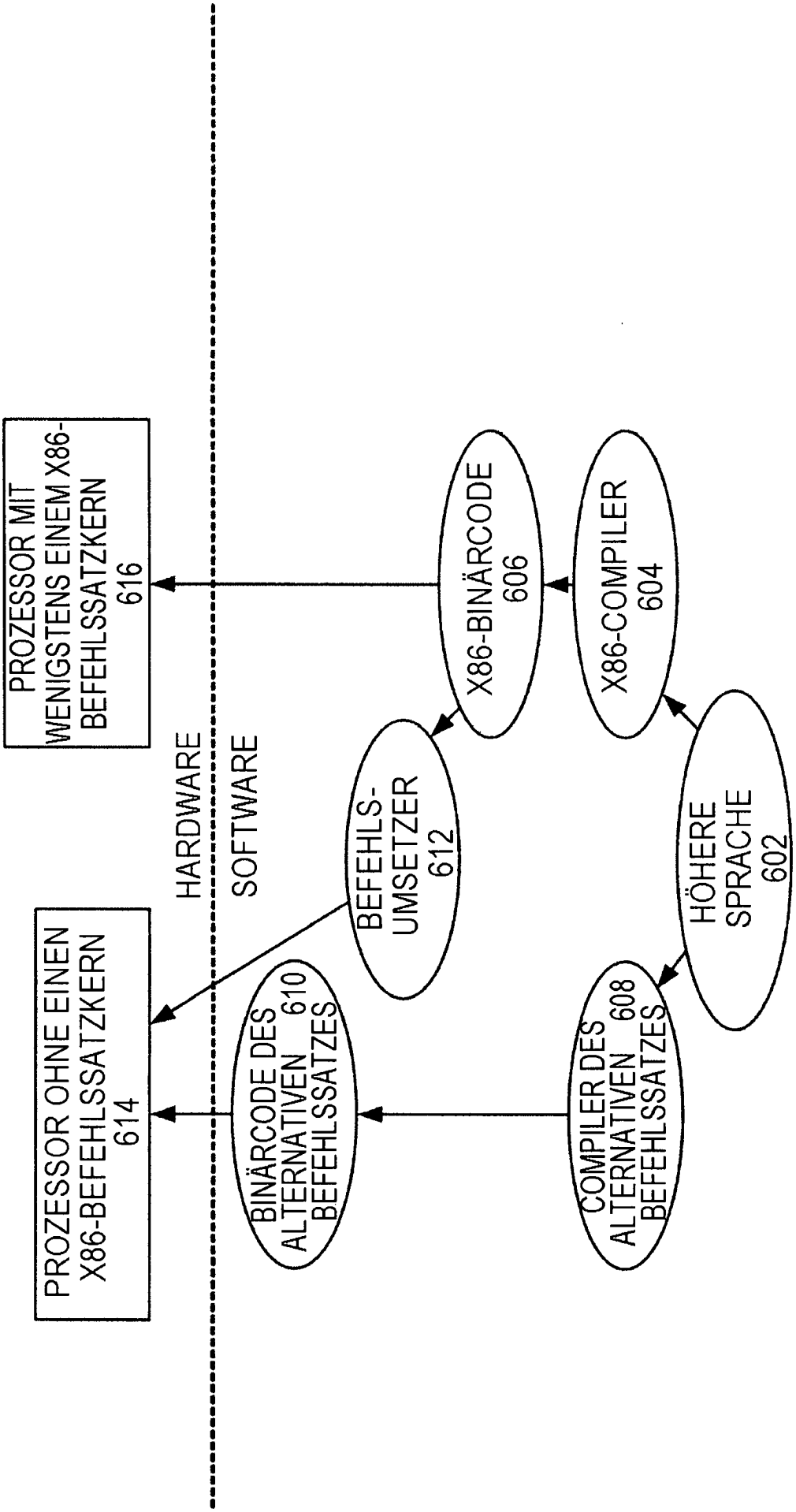
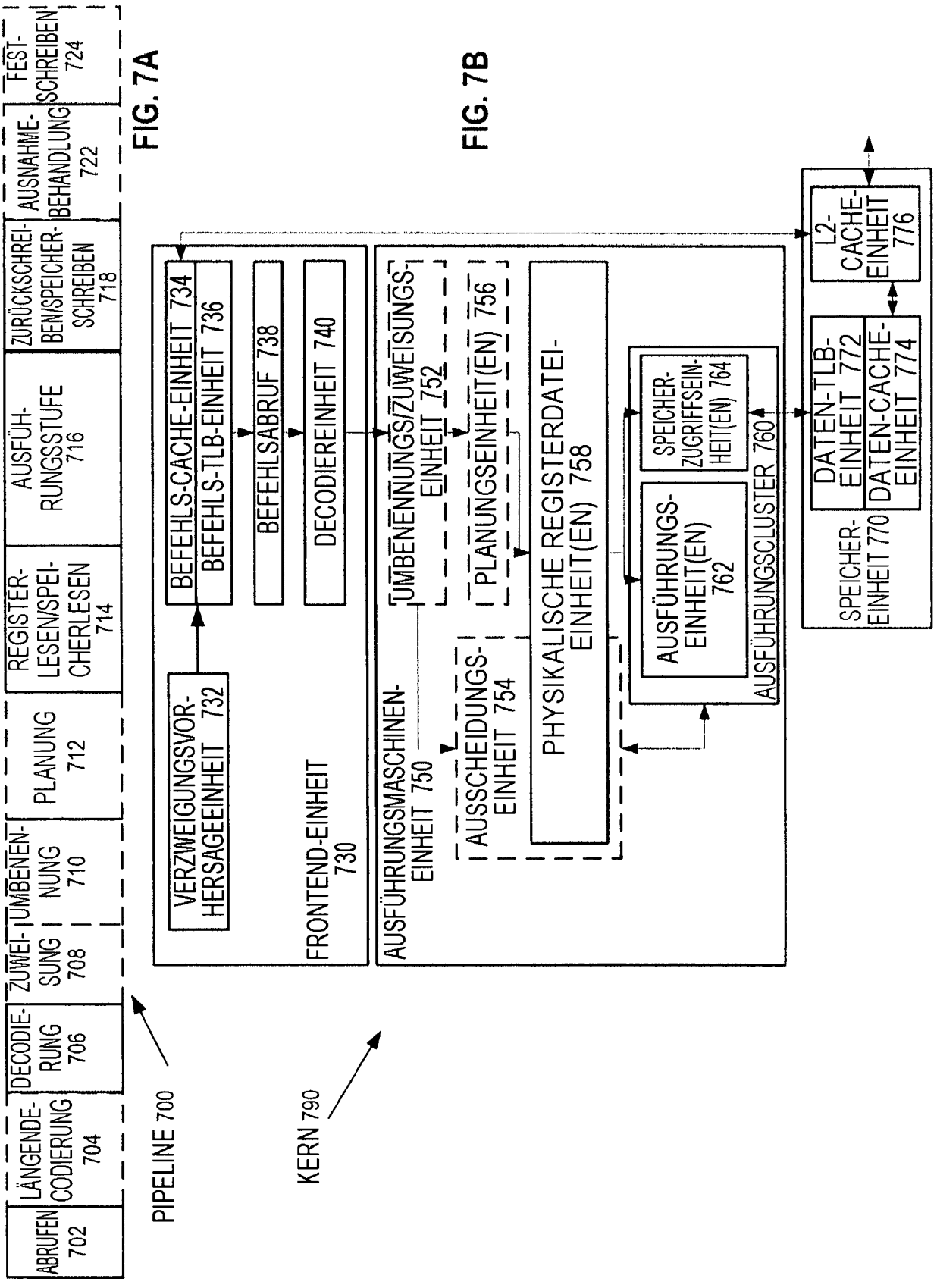

PROZESSOR MIT WENIGSTENS EINEM X86-BEFEHLSSATZKERN 616
PROZESSOR OHNE EINEN X86-BEFEHLSSATZKERN 614
HARDWARE
SOFTWARE
X86-BINÄRCODE 606
X86-COMPILER 604
BEFEHLS-UMSETZER 612
HÖHERE SPRACHE 602
BINÄRCODE DES ALTERNATIVEN BEFEHLSSATZES 610
COMPILER DES ALTERNATIVEN BEFEHLSSATZES 608

FIG. 6


ABRUFEN 702
LÄNGENDE-CODIERUNG 704
DECODIE-RUNG 706
ZUWEI-SUNG 708
UMBENEN-NUNG 710
PLANUNG 712
REGISTER-LESEN/SPEI-CHERLESEN 714
AUSFÜH-RUNGSSTUFE 716
ZURÜCKSCHREI-BEN/SPEICHER-SCHREIBEN 718
AUSNAHME-BEHANDLUNG 722
FEST-SCHREIBEN 724
PIPELINE 700
FIG. 7A
VERZWEIGUNGSVOR-HERSAGEEINHEIT 732
BEFEHLS-CACHE-EINHEIT 734
BEFEHLS-TLB-EINHEIT 736
BEFEHLSABRUF 738
DECODIEREINHEIT 740
FRONTEND-EINHEIT 730
AUSFÜHRUNGSMASCHINEN-EINHEIT 750
UMBENENNUNGS/ZUWEISUNGS-EINHEIT 752
AUSSCHEIDUNGS-EINHEIT 754
PLANUNGSEINHEIT(EN) 756
PHYSIKALISCHE REGISTERDATEI-EINHEIT(EN) 758
AUSFÜHRUNGS-EINHEIT(EN) 762
SPEICHER-ZUGRIFFSEIN-HEIT(EN) 764
AUSFÜHRUNGSCLUSTER 760
KERN 790
FIG. 7B
SPEICHER-EINHEIT 770
DATEN-TLB-EINHEIT 772
DATEN-CACHE-EINHEIT 774
L2-CACHE-EINHEIT 776

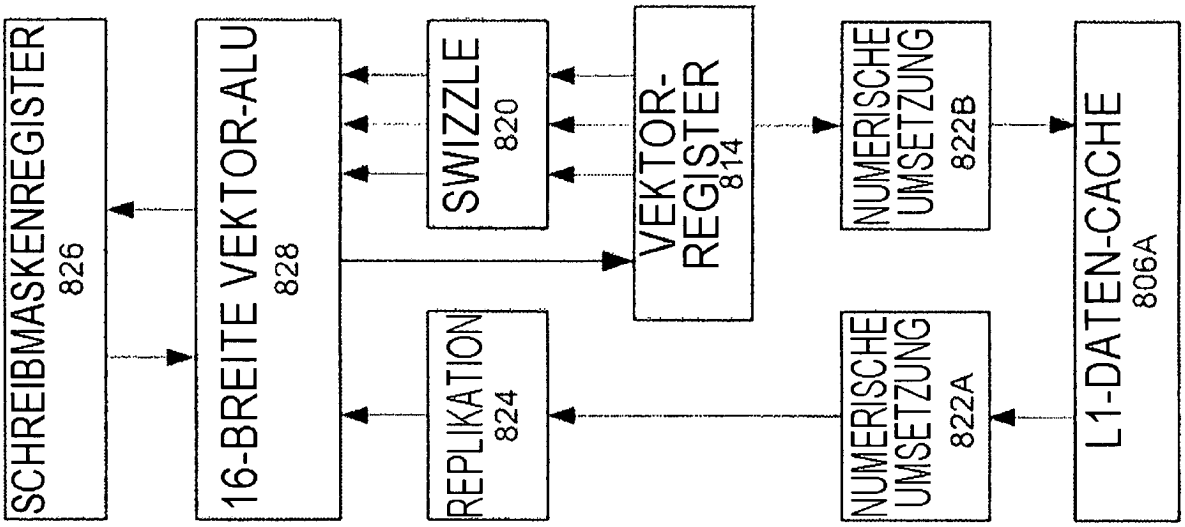

SCHREIBMASKENREGISTER
826
16-BREITE VEKTOR-ALU
828
REPLIKATION
824
SWIZZLE
820
VEKTOR-
REGISTER
814
NUMERISCHE
UMSETZUNG
822A
NUMERISCHE
UMSETZUNG
822B
L1-DATEN-CACHE
806A

FIG. 8B

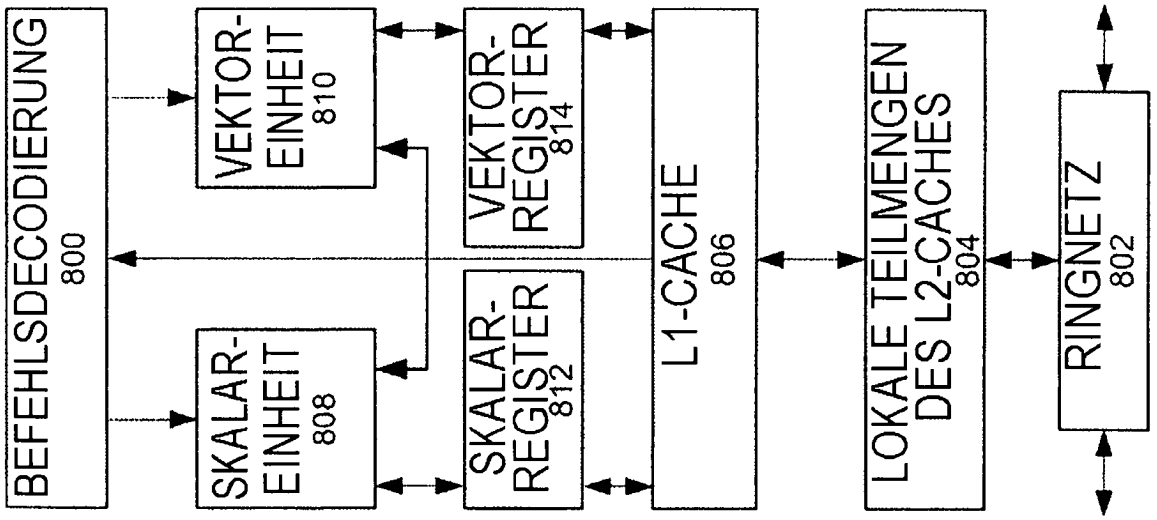

BEFEHLSDECODIERUNG
800
SKALAR-
EINHEIT
808
VEKTOR-
EINHEIT
810
SKALAR-
REGISTER
812
VEKTOR-
REGISTER
814
L1-CACHE
806
LOKALE TEILMENGEN
DES L2-CACHES
804
RINGNETZ
802

FIG. 8A

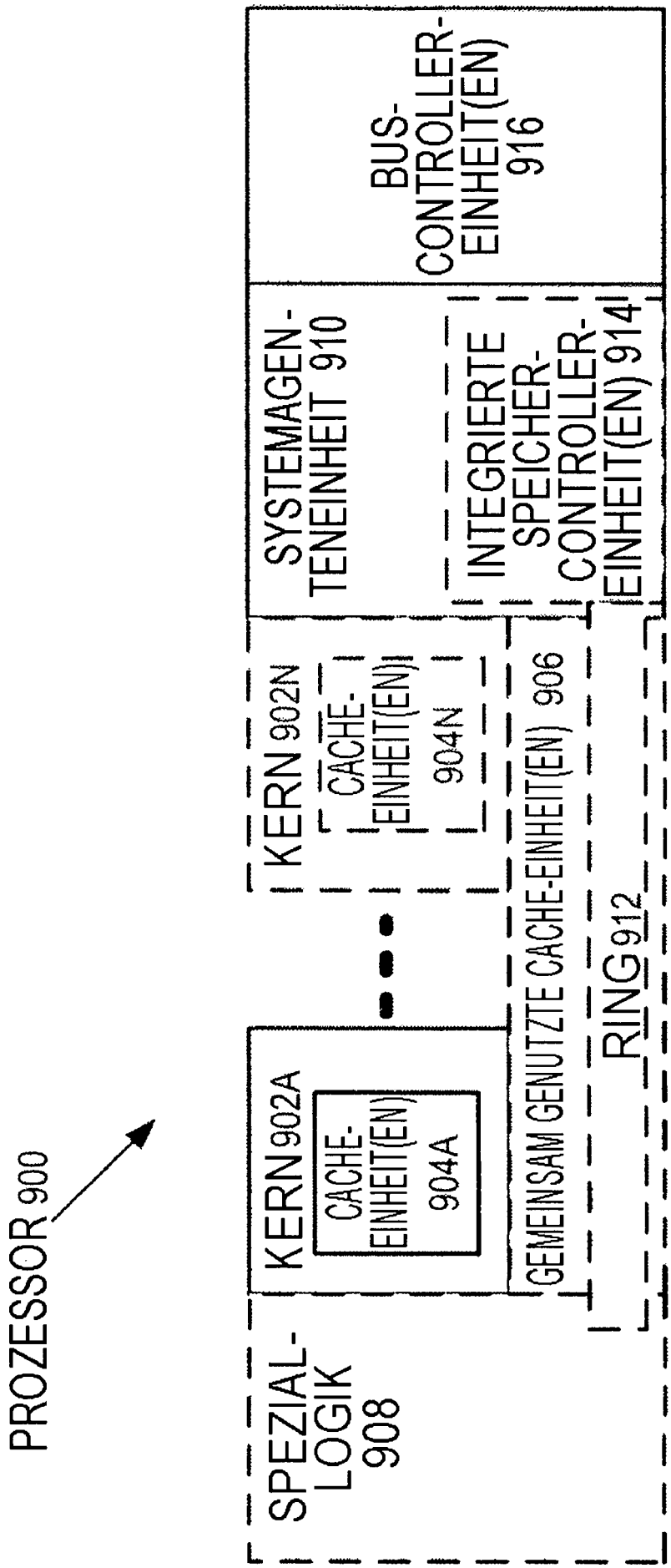

PROZESSOR 900
SPEZIAL-LOGIK 908
KERN 902A
CACHE-EINHEIT(EN) 904A
KERN 902N
CACHE-EINHEIT(EN) 904N
GEMEINSAM GENUTZTE CACHE-EINHEIT(EN) 906
RING 912
SYSTEMAGEN-TENEINHEIT 910
INTEGRIERTE SPEICHER-CONTROLLER-EINHEIT(EN) 914
BUS-CONTROLLER-EINHEIT(EN) 916

FIG. 9

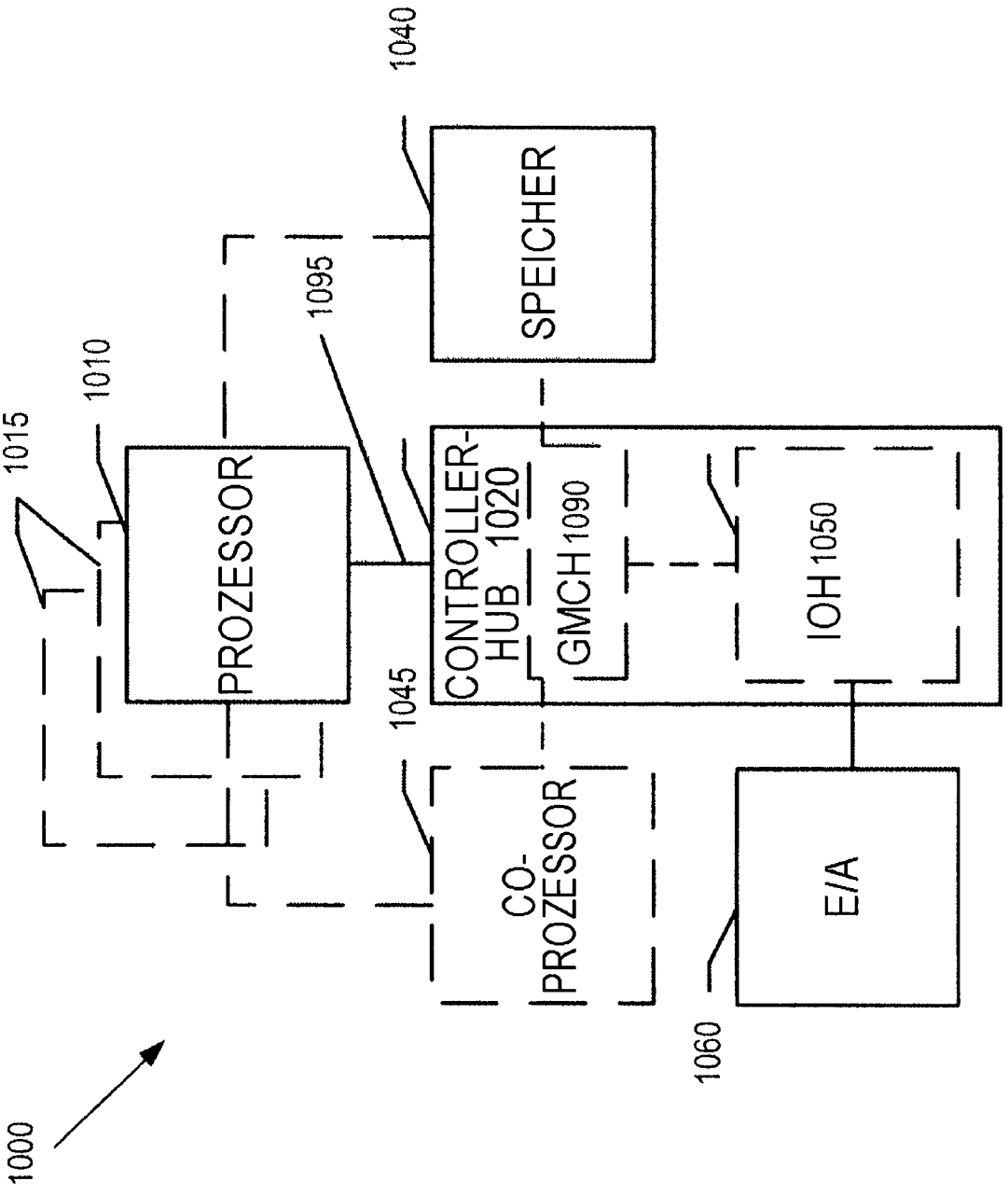

1000
1010
1015
PROZESSOR
1095
1040
SPEICHER
CONTROLLER-HUB 1020
GMCH 1090
IOH 1050
1045
CO-PROZESSOR
E/A
1060

FIG. 10

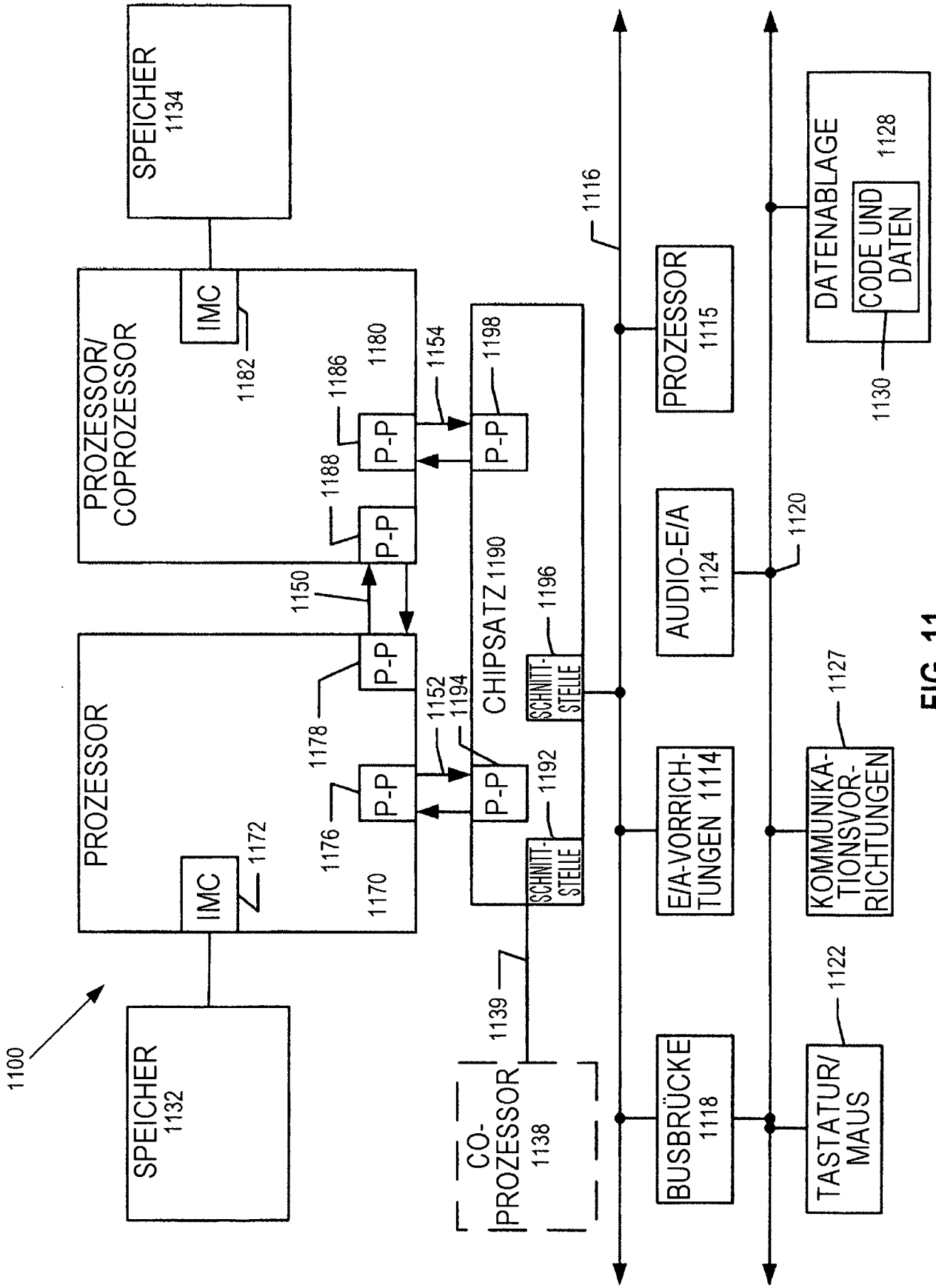

1100
SPEICHER 1132
PROZESSOR
IMC 1172
1170
P-P
P-P
1176
1178
1150
PROZESSOR/ COPROZESSOR
IMC 1182
SPEICHER 1134
1180
P-P
P-P
1186
1188
1154
1152
1194
CHIPSATZ 1190
P-P
P-P
1198
1196
SCHNITT-STELLE
SCHNITT-STELLE
1192
1139
CO-PROZESSOR 1138
1116
PROZESSOR 1115
AUDIO-E/A 1124
1120
E/A-VORRICH-TUNGEN 1114
KOMMUNIKA-TIONSVOR-RICHTUNGEN
1127
BUSBRÜCKE 1118
TASTATUR/ MAUS
1122
DATENABLAGE
CODE UND DATEN
1128
1130

FIG. 11

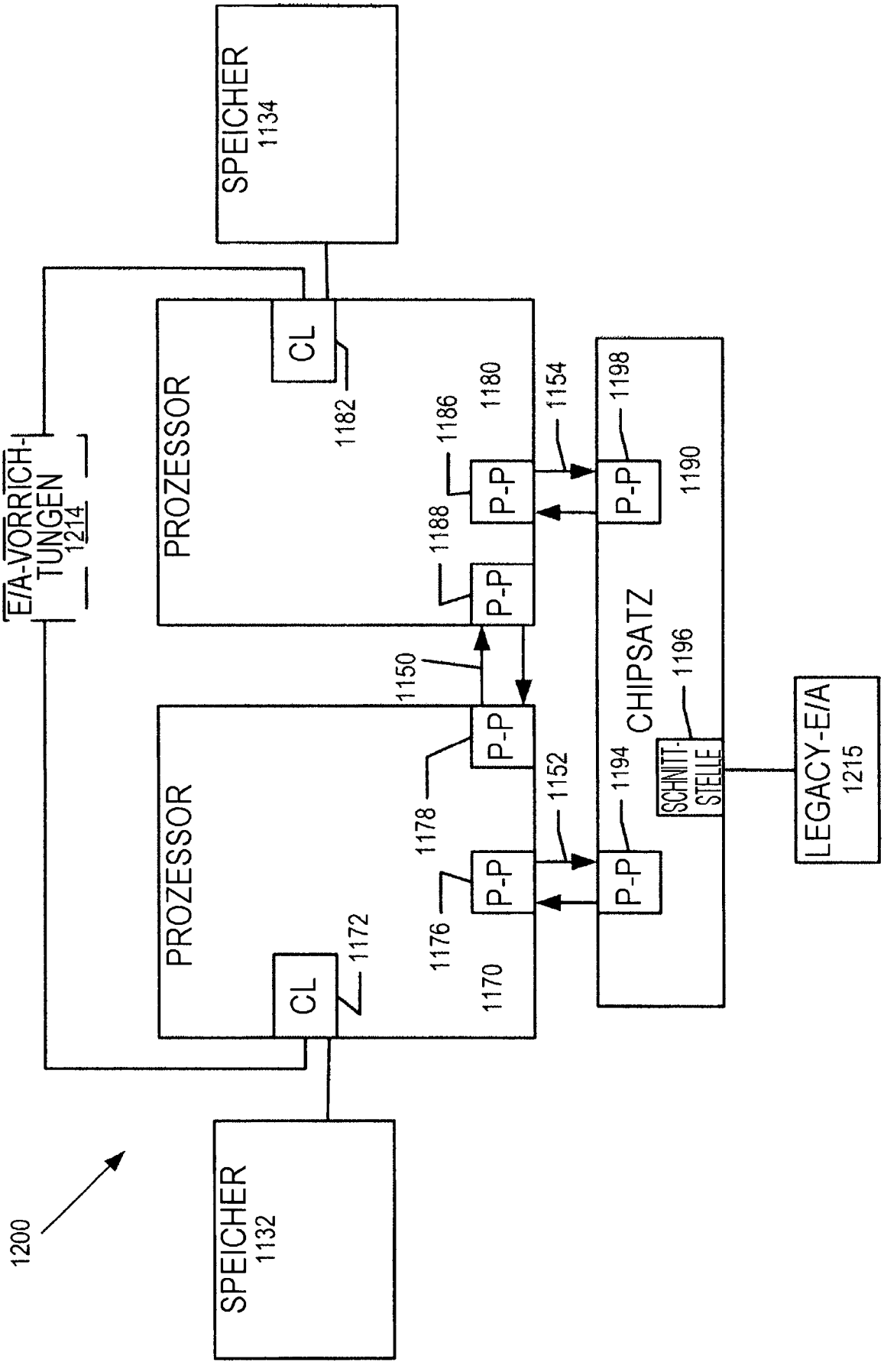

SPEICHER
1134
PROZESSOR
CL
1182
1180
1186
1188
P-P
P-P
1154
1198
1190
CHIPSATZ
1196
SCHNITT-STELLE
1194
1152
1178
1176
1170
1172
PROZESSOR
CL
SPEICHER
1132
E/A-VORRICHTUNGEN
1214
LEGACY-E/A
1215
1200
1150

FIG. 12

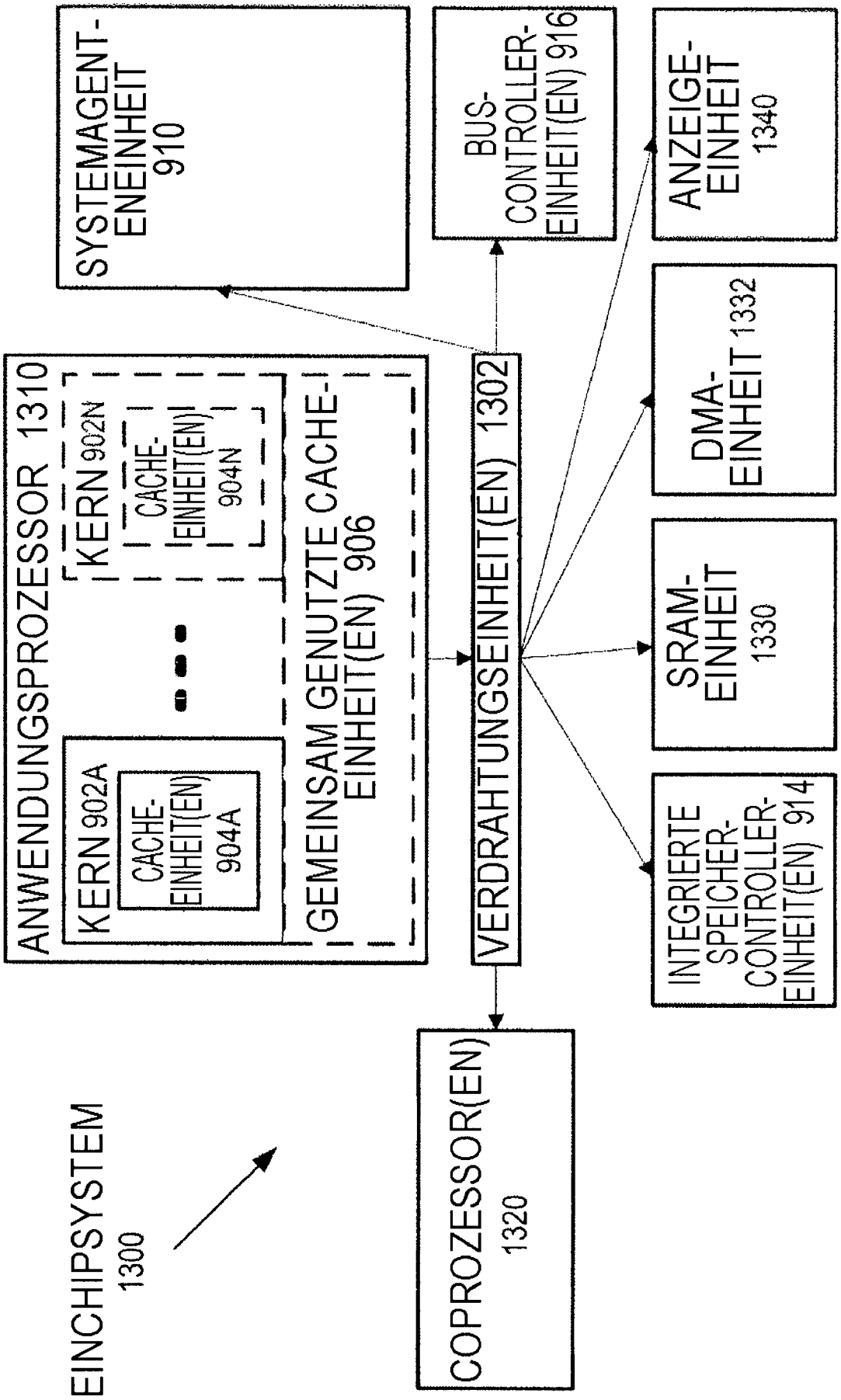

EINCHIPSYSTEM
1300
ANWENDUNGSPROZESSOR 1310
KERN 902A
CACHE-EINHEIT(EN) 904A
KERN 902N
CACHE-EINHEIT(EN) 904N
GEMEINSAM GENUTZTE CACHE-EINHEIT(EN) 906
SYSTEMAGENT-ENEINHEIT 910
BUS-CONTROLLER-EINHEIT(EN) 916
VERDRAHTUNGSEINHEIT(EN) 1302
ANZEIGE-EINHEIT 1340
DMA-EINHEIT 1332
SRAM-EINHEIT 1330
INTEGRIERTE SPEICHER-CONTROLLER-EINHEIT(EN) 914
COPROZESSOR(EN) 1320

FIG. 13