(19) Deutsches Patent- und Markenamt

(10) **DE 10 2015 002 253 A1** 2015.10.01

(12) # Offenlegungsschrift

(21) Aktenzeichen: **10 2015 002 253.9**
(22) Anmeldetag: **23.02.2015**
(43) Offenlegungstag: **01.10.2015**

(51) Int Cl.: ***G06F 9/38*** (2006.01)
G06F 9/30 (2006.01)

(30) Unionspriorität:
14/229,183 **28.03.2014** **US**

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

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

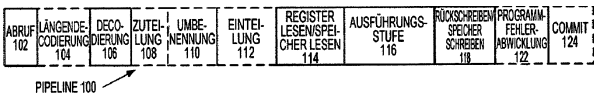
(72) Erfinder:
Sole, Guillem, Barcelona, ES; Fernandez, Manel, Barcelona, ES; Espasa, Roger, Barcelona, ES

Prüfungsantrag gemäß § 44 PatG ist gestellt.

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

(54) Bezeichnung: **Verfahren und Vorrichtung zum Ausführen mehrerer Multiplikationsoperationen**

(57) Zusammenfassung: Es werden eine Vorrichtung und ein Verfahren zur Ausführung mehrerer Multiplikationsoperationen beschrieben. Zum Beispiel umfasst eine Ausführungsform eines Prozessors eine Anweisungsabrufeinheit zum Abrufen einer Doppelmultiplikationsanweisung aus einem Speichersubsystem, wobei die Doppelmultiplikationsanweisung drei Quellenoperandenwerte aufweist; eine Decodiereinheit zum Decodieren der Doppelmultiplikationsanweisung, um mindestens eine uop zu erzeugen; und eine Ausführungseinheit zum Ausführen der uop ein erstes Mal, um einen ersten und einen zweiten der drei Quellenoperandenwerte zu multiplizieren, um ein erstes Zwischenergebnis zu erzeugen, und zum Ausführen der uop ein zweites Mal, um das Zwischenergebnis mit einem dritten der drei Quellenoperandenwerte zu multiplizieren, um ein Endergebnis zu erzeugen.



Beschreibung

HINTERGRUND

Technisches Gebiet

[0001] Die vorliegende Erfindung betrifft allgemein das Gebiet der Computerprozessoren. Insbesondere betrifft die Erfindung ein Verfahren und eine Vorrichtung zum Ausführen mehrerer Multiplikationsoperationen.

Stand der Technik

[0002] Ein Anweisungssatz oder eine Anweisungssatzarchitektur (ISA) ist der Teil der Computerarchitektur, der Programmierung betrifft, darunter die nativen Datentypen, Anweisungen, Registerarchitektur, Adressierungsarten, Speicherarchitektur, Interrupt- und Programmfehlerabwicklung und externe Eingabe und Ausgabe (E/A). Es sollte beachtet werden, dass sich der Ausdruck ”Anweisung” hier im Allgemeinen auf Makro-Anweisungen bezieht, das heißt, Anweisungen, die dem Prozessor zur Ausführung zugeführt werden, im Gegensatz zu Mikro-Anweisungen oder Mikro-Ops, die das Ergebnis davon sind, dass der Decoder eines Prozessors Makro-Anweisungen decodiert.

[0003] Die ISA unterscheidet sich von der Mikroarchitektur, die eine Menge von Prozessorentwurfstechniken ist, die zur Implementierung des Anweisungssatzes verwendet wird. Prozessoren mit verschiedenen Mikroarchitekturen können sich einen gemeinsamen Anweisungssatz teilen. Zum Beispiel implementieren Intel®-Pentium-4-Prozessoren, Intel®-Core™-Prozessoren und Prozessoren von Advanced Mikro Devices, Inc. in Sunnyvale CA, nahezu identische Versionen des x86-Anweisungssatzes (mit bestimmten Erweiterungen, die bei neueren Versionen hinzugefügt wurden), weisen aber verschiedene interne Entwürfe auf. Zum Beispiel kann dieselbe Registerarchitektur der ISA in verschiedenen Mikroarchitekturen implementiert werden, wobei wohlbekannte Techniken verwendet werden, darunter dedizierte physische Register, ein oder mehrere dynamisch zugeteilte physische Register, die einen Registerumbenennungsmechanismus verwenden (z. B. Verwendung einer RAT (Register Alias Table), eines ROB (Reorder Buffer) und eines Ausscheidungsregisterfile). Sofern es nicht anders spezifiziert wird, beziehen sich die hier verwendeten Ausdrücke Registerarchitektur, Registerfile und Register auf das der Software/dem Programmierer Sichtbare und die Weise, wie Anweisungen Register spezifizieren. Wenn eine Unterscheidung notwendig ist, werden die Adjektive ”logisch”, ”architekturell” oder ”softwaresichtbar” verwendet, um Register/Files in der Registerarchitektur anzuzeigen, während Register in einer gegebenen Mikroarchitektur (z. B. physisches Register, Umordnungspuffer, Ausscheidungsregister, Registerpool) mit anderen Adjektiven bezeichnet werden.

[0004] Ein Anweisungssatz umfasst ein oder mehrere Anweisungsformate. Ein gegebenes Anweisungsformat definiert verschiedene Felder (Anzahl von Bit, Ort von Bit), um unter anderem die auszuführende Operation und den bzw. die Operanden, woran diese Operation auszuführen ist, zu spezifizieren. Bestimmte Anweisungsformate werden durch die Definition von Anweisungsvorlagen (oder Subformaten) weiter zerteilt. Zum Beispiel können die Anweisungsvorlagen eines gegebenen Anweisungsformats so definiert werden, dass sie verschiedene Teilmengen der Felder des Anweisungsformats aufweisen (die eingeschlossenen Felder liegen typischerweise in derselben Reihenfolge vor, aber mindestens bestimmte Weisen verschiedene Bitpositionen auf, weil weniger Felder eingeschlossen sind) und/oder so definiert, dass bei ihnen ein gegebenes Feld anders interpretiert wird. Eine gegebene Anweisung wird unter Verwendung eines gegebenen Anweisungsformats ausgedrückt (und, falls definiert, in einer gegebenen der Anweisungsvorlagen dieses Anweisungsformats) und spezifiziert die Operation und die Operanden. Ein Anweisungsstrom ist eine spezifische Sequenz von Anweisungen, wobei jede Anweisung in der Sequenz das Auftreten einer Anweisung in einem Anweisungsformat (und, falls definiert, einer gegebenen der Anweisungsvorlagen dieses Anweisungsformats) ist.

[0005] Wissenschaftliche, finanzielle, autovektorisierte Vielzweck-, RMS- (Erkennung, Mining und Synthese) und visuelle und Multimediaanwendungen (z. B. 2D-3G-Grafik, Bildverarbeitung, Videokomprimierung/-dekomprimierung, Spracherkennungsalgorithmen und Audiomanipulation) erfordern oft das Ausführen derselben Operation an einer großen Anzahl von Datenposten (was hier als Datenparallelismus bezeichnet wird). SIMD (Single Instruction Multiple Data) bezieht sich auf eine Art von Anweisung, die bewirkt, dass ein Prozessor eine Operation an mehreren Datenposten ausführt. SIMD-Technologie eignet sich besonders für Prozessoren, die die Bit in einem Register logisch in eine Anzahl von Datenelemente fester Größe aufteilen können, die jeweils einen separaten Wert repräsentieren. Zum Beispiel können die Bit in einem 64-Bit-Register als Quellenoperand spezifiziert werden, woran als vier separate 16-Bit-Datenelemente zu operieren ist, die jeweils einen separaten 16-Bit-Wert repräsentieren. Diese Art von Daten wird als gepackter Datentyp oder Vektordatentyp bezeichnet,

und Operanden dieses Datentyps werden als gepackte Datenoperanden oder Vektoroperanden bezeichnet. Anders ausgedrückt bezieht sich ein gepackter Datenposten oder Vektor auf eine Sequenz von gepackten Datenelementen; und ein gepackter Datenoperand oder ein Vektoroperand ist ein Quellen- oder Zieloperand einer SIMD-Anweisung (auch als gepackte Datenanweisung oder Vektoranweisung bekannt).

[0006] Beispielsweise spezifiziert eine Art von SIMD-Anweisung das Ausführen einer einzigen Vektoroperation an zwei Quellenvektoroperanden auf vertikale Weise, um einen Zielvektoroperanden (der auch als Ergebnisvektoroperand bezeichnet wird) derselben Größe zu erzeugen, mit derselben Anzahl von Datenelementen und in derselben Datenelementreihenfolge. Die Datenelemente in den Quellenvektoroperanden werden als Quellendatenelemente bezeichnet, während die Datenelemente in dem Zielvektoroperanden als Ziel- oder Ergebnisdatenelemente bezeichnet werden. Diese Quellenvektoroperanden weisen dieselbe Größe auf und enthalten Datenelemente derselben Breite, und somit enthalten sie dieselbe Anzahl von Datenelementen. Die Quellendatenelemente an denselben Bitpositionen in den zwei Quellenvektoroperanden bilden Paare von Datenelementen (die auch als entsprechende Datenelemente bezeichnet werden). Die durch diese SIMD-Anweisung spezifizierte Operation wird an jedem dieser Paare von Quellendatenelementen separat ausgeführt, um eine übereinstimmende Anzahl von Ergebnisdatenelementen zu erzeugen, und somit weist jedes Paar von Quellendatenelementen ein entsprechendes Ergebnisdatenelement auf. Da die Operation vertikal ist und der Ergebnisvektoroperand dieselbe Größe aufweist, dieselbe Anzahl von Datenelementen aufweist und die Ergebnisdatenelemente in derselben Datenelementreihenfolge wie die Quellenvektoroperanden gespeichert werden, befinden sich die Ergebnisdatenelemente an denselben Bitpositionen des Ergebnisvektoroperanden wie ihr entsprechendes Paar von Quellendatenoperanden in den Quellenvektoroperanden. Zusätzlich zu dieser beispielhaften Art von SIMD-Anweisung gibt es vielfältige andere Arten von SIMD-Anweisungen (die z. B. nur einen oder mehr als zwei Vektoroperanden aufweist; die auf horizontale Weise operieren; die einen Ergebnisvektoroperanden erzeugt, der eine andere Größe aufweist, der Datenelemente verschiedener Größe aufweist und/oder der eine andere Datenelementreihenfolge aufweist). Es versteht sich, dass der Ausdruck Zielvektoroperand (oder Zieloperand) als das direkte Ergebnis des Ausführens der durch eine Anweisung spezifizierten Operation definiert ist, einschließlich der Speicherung dieses Zieloperanden an einer Speicherstelle (sei es ein Register oder in einer durch diese Anweisung spezifizierten Speicheradresse), so dass durch eine andere Anweisung (durch Spezifikation dieser selben Speicherstelle durch die andere Anweisung) als Quellenoperand auf ihn zugegriffen werden kann.

[0007] Die SIMD-Technologie, wie etwa die von den Intel®-Core™-Prozessoren verwendete, die einen Anweisungssatz aufweisen, die Anweisungen des Typs x86, MMX™, Streaming SIMD Extensions (SSE), SSE2, SSE3, SSE4.1 und SSE4.2 umfasst, hat signifikante Verbesserungen bei der Anweisungsleistungsfähigkeit ermöglicht (Core™ und MMX™ sind eingetragene Warenzeichen oder Warenzeichen der Intel Corporation in Santa Clara, Kalifornien). Außerdem wurde eine zusätzliche Menge von SIMD-Erweiterungen entworfen und publiziert, die als AVX (Advanced Vector Extensions) bezeichnet werden und das VEX-Codierungsschema verwenden.

[0008] Eine Anweisung von besonderer Relevanz für die vorliegende Anmeldung ist die Multiplikationsanweisung. Mehrere Algorithmen bei hochleistungsfähigen Datenverarbeitungsplattformen multiplizieren mehrere berechnete Werte. Im Allgemeinen erfordert die Multiplikationsoperation die Ausführung einer Anweisung.

KURZE BESCHREIBUNG DER ZEICHNUNGEN

[0009] Aus der folgenden ausführlichen Beschreibung in Verbindung mit den folgenden Zeichnungen kann ein besseres Verständnis der vorliegenden Erfindung erhalten werden. Es zeigen:

[0010] Fig. 1A ein Blockschaltbild sowohl einer beispielhaften In-Reihenfolge-Abruf-, Decodier-, Ausscheidungs-Pipeline als auch einer beispielhaften Registerumbenennungs-, Außerreihenfolge-Ausgabe/-Ausführungs-Pipeline gemäß Ausführungsformen der Erfindung;

[0011] Fig. 1B ein Blockschaltbild sowohl einer beispielhaften Ausführungsform eines In-Reihenfolge-Abruf-, Decodier-, Ausscheidungskerns als auch eines beispielhaften Registerumbenennungs-, Außerreihenfolge-Ausgabe-/Ausfahrungsarchitekturkerns zur Aufnahme in einen Prozessor gemäß Ausführungsformen der Erfindung;

[0012] Fig. 2 ein Blockschaltbild eines Einzelkernprozessors und eines Mehrkernprozessors mit integrierter Speichersteuerung und Grafik gemäß Ausführungsformen der Erfindung;

[0013] **Fig.** 3 ein Blockschaltbild eines Systems gemäß einer Ausführungsform der vorliegenden Erfindung;

[0014] **Fig.** 4 ein Blockschaltbild eines zweiten Systems gemäß einer Ausführungsform der vorliegenden Erfindung;

[0015] **Fig.** 5 ein Blockschaltbild eines dritten Systems gemäß einer Ausführungsform der vorliegenden Erfindung;

[0016] **Fig.** 6 ein Blockschaltbild eines SoC (System on a Chip) gemäß einer Ausführungsform der vorliegenden Erfindung;

[0017] **Fig.** 7 ein Blockschaltbild zur Kontrastierung der Verwendung eines Softwareanweisungsumsetzers zum Umsetzen von binären Anweisungen in einem Quellenanweisungssatz in binäre Anweisungen in einem Zielanweisungssatz;

[0018] **Fig.** 8 eine Ausführungsform einer Prozessorarchitektur, worauf Ausführungsformen der Erfindung verwendet werden können;

[0019] **Fig.** 9A eine Ausführungsform einer Architektur zum Ausführen mehrerer Multiplikationsoperationen;

[0020] **Fig.** 9B eine andere Ausführungsform einer Architektur zum Ausführen mehrerer Multiplikationsoperationen;

[0021] **Fig.** 10 eine Ausführungsform eines Verfahrens zum Ausführen mehrerer Multiplikationsoperationen;

[0022] **Fig.** 11a–b Blockschaltbilder von einem generischen vektorfreundlichen Anweisungsformat und Anweisungsvorlagen davon gemäß Ausführungsformen der Erfindung;

[0023] **Fig.** 12a–d ein Blockschaltbild eines beispielhaften spezifischen vektorfreundlichen Anweisungsformats gemäß Ausführungsformen der Erfindung; und

[0024] **Fig.** 13 ein Blockschaltbild einer Registerarchitektur gemäß einer Ausführungsform der Erfindung.

AUSFÜHRLICHE BESCHREIBUNG

[0025] In der folgenden Beschreibung werden zur Erläuterung zahlreise spezifische Einzelheiten dargelegt, um ein umfassendes Verständnis der Ausführungsformen der nachfolgend beschriebenen Erfindung zu gewährleisten. Für Fachleute ist jedoch ersichtlich, dass die Ausführungsformen der Erfindung ohne bestimmte dieser spezifischen Einzelheiten ausgeübt werden können. In anderen Fällen werden wohlbekannte Strukturen und Vorrichtungen in Blockschaltbildform gezeigt, um eine Verschleierung der zugrundeliegenden Prinzipien der Ausführungsformen der Erfindung zu vermeiden.

Beispielhafte Prozessorarchitekturen und Datentypen

[0026] **Fig.** 1A ist ein Blockschaltbild sowohl einer beispielhaften In-Reihenfolge-Abruf-, Decodier-, Ausscheidungs-Pipeline als auch einer beispielhaften Registerumbenennungs-, Außerreihenfolge-Ausgabe/-Ausführungs-Pipeline gemäß Ausführungsformen der Erfindung. **Fig.** 1B ist ein Blockschaltbild sowohl einer beispielhaften Ausführungsform eines In-Reihenfolge-Abruf-, Decodier-, Ausscheidungskerns als auch eines beispielhaften Registerumbenennungs-, Außerreihenfolge-Ausgabe-/Ausführungsarchitekturkerns zur Aufnahme in einen Prozessor gemäß Ausführungsformen der Erfindung. Die durchgezogen gezeichneten Kästen in **Fig.** 1A–B zeigen die In-Reihenfolge-Teile der Pipeline und des Kerns, während der optionale Zusatz der gestrichelt gezeichneten Kästen Pipeline und Kern mit Registerumbenennung und Außerreihenfolge-Ausgabe/Ausführung zeigen.

[0027] In **Fig.** 1A umfasst eine Prozessorpipeline **100** eine Abrufstufe **102**, eine Längendecodierstufe **104**, eine Decodierstufe **106**, eine Zuteilungsstufe **108**, eine Umbenennungsstufe **110**, eine Scheduling-(auch Dispatch- oder Ausgabe-)Stufe **112**, eine Registerlese-/Speicherlesestufe **114**, eine Ausführungsstufe **116**, eine Rückschreib-/Speicherschreibstufe **118**, eine Programmfehlerbehandlungsstufe **122** und eine Commit-Stufe **124**.

[0028] **Fig.** 1B zeigt den Prozessorkern **190** mit einer Frontend-Einheit **130**, die mit einer Ausführungs-Engine-Einheit **150** gekoppelt ist, und beide sind mit einer Speichereinheit **170** gekoppelt. Der Kern **190** kann ein RISC-Kern (Reduced Instruction Set Computing), ein SISC-Kern (Complex Instruction Set Computing), ein VLIW-Kern (Very Long Instruction Word) oder ein hybrider oder alternativer Kerntyp sein. Als weitere Möglichkeit kann der Kern **190** ein Spezialkern sein, wie zum Beispiel ein Netzwerk- oder Kommunikationskern, eine Kompressions-Engine, ein Koprozessor-Kern, ein GPGPU-Kern (Vielzweck-Datenverarbeitungs-Grafikverarbeitungseinheit), ein Grafikkern oder dergleichen.

[0029] Die Frontend-Einheit **130** umfasst eine Zweigprädiktionseinheit **132**, die mit einer Anweisungscache-Einheit **134** gekoppelt ist, die mit einem Anweisungsübersetzungs-Lookaside-Puffer (TLB) **136** gekoppelt ist, der mit einer Anweisungsabrufeinheit **138** gekoppelt ist, die mit einer Decodiereinheit **140** gekoppelt ist. Die Decodiereinheit **140** (bzw. Decodierer) kann Anweisungen decodieren und erzeugt als Ausgabe eine oder mehrere Mikro-Operationen, Mikro-Code-Eintrittspunkte, Mikroanweisungen, andere Anweisungen oder andere Steuersignale, die aus den ursprünglichen Anweisungen decodiert werden oder die anderweitig diese widerspiegeln oder aus ihnen abgeleitet werden. Die Decodiereinheit **140** kann unter Verwendung verschiedener unterschiedlicher Mechanismen implementiert werden. Beispiele für geeignete Mechanismen wären, aber ohne Beschränkung darauf, Nachschlagetabellen, Hardwareimplementierungen, programmierbare Logikarrays (PLA), Mikrocode-Nurlesespeicher (ROM) usw. Bei einer Ausführungsform umfasst der Kern **190** einen Mikrocode-ROM oder ein anderes Medium, das Mikrocode für bestimmte Makroanweisungen speichert (z. B. in der Decodiereinheit **140** oder ansonsten in der Frontend-Einheit **130**). Die Decodiereinheit **140** ist mit einer Umbenennungs-/Zuteilereinheit **152** in der Ausführungs-Engine-Einheit **150** gekoppelt.

[0030] Die Ausführungs-Engine-Einheit **150** umfasst die Umbenennungs-/Zuteilereinheit **152**, die mit einer Ausscheideeinheit **154** und einer Menge aus einer oder mehreren Scheduler-Einheiten **156** gekoppelt ist. Die Scheduler-Einheit(en) **156** repräsentiert eine beliebige Anzahl verschiedener Scheduler, darunter Reservierungsstationen, Zentral-Anweisungsfenster usw. Die Scheduler-Einheit(en) **156** ist mit den physischen Registerfile-Einheit(en) **158** gekoppelt. Jede der physischen Registerfile-Einheiten **158** repräsentiert ein oder mehrere physische Registerfiles, von denen verschiedene einen oder mehrere verschiedene Datentypen speichern, wie etwa Skalar-Integer, Skalar-Gleitkomma, Integer-gepackt, Gleitkomma-gepackt, Vektor-Integer, Vektor-Gleitkomma, Status (z. B. ein Anweisungszeiger, der die Adresse der nächsten auszuführenden Anweisung ist) usw. Bei einer Ausführungsform umfasst die physische Registerfile-Einheit **158** eine Vektorregistereinheit, eine Schreibmaskenregistereinheit und eine Skalarregistereinheit. Diese Registereinheiten können Architektur-Vektorregister, Vektormaskenregister und Vielzweckregister bereitstellen. Die physische Registerfile-Einheit (en) **158** wird durch die Ausscheidungseinheit **154** überlappt, um verschiedene Weisen darzustellen, auf die Registerumbenennung und Außerreihenfolgeausführung implementiert werden kann (z. B. unter Verwendung eines oder mehrerer Umordnungspuffer und eines oder mehrerer Ausscheidungs-Registerfiles; Verwendung von Zukunft-Files, Geschichte-Puffer und Ausscheidungsregister-Files; Verwendung von Registerabbildungen und eines Pools von Registern; usw.). Die Ausscheidungseinheit **154** und die physische Registerfile-Einheit (en) **158** sind mit dem Ausführungs-Cluster **160** gekoppelt. Das Ausführungs-Cluster **160** umfasst eine Menge aus einer oder mehreren Ausführungseinheiten **162** und eine Menge aus einer oder mehreren Speicherzugriffseinheiten **164**. Die Ausführungseinheiten **162** können verschiedene Operationen (z. B. Verschiebungen, Addition, Subtraktion, Multiplikation) ausführen, und an verschiedenen Arten von Daten (z. B. Skalar-Gleitkomma, Integer-Gepackt, Gleitkomma-Gepackt, Vektor-Integer, Vektor-Gleitkomma). Obwohl bestimmte Ausführungsformen eine Anzahl von Ausführungseinheiten umfassen können, die spezifischen Funktionen oder Mengen von Funktionen gewidmet sind, können andere Ausführungsformen nur eine Ausführungseinheit oder mehrere Ausführungseinheiten umfassen, die alle die gesamten Funktionen ausführen. Die Scheduler-Einheiten **156**, physischen Registerfile-Einheiten **158** und Ausführungs-Cluster **160** sind als möglicherweise mehrzahlig gezeigt, weil bestimmte Ausführungsformen für bestimmte Arten von Daten/Operationen getrennte Pipelines erzeugen (z. B. eine Skalar-Integer-Pipeline, eine Skalar-Gleitkomma-/Integer-Gepackt-/Gleitkomma-Gepackt-/Vektor-Integer-/Vektor-Gleitkomma-Pipeline und/oder eine Speicherzugriffspipeline, die jeweils ihre eigene Scheduler-Einheit, physische Registerfile-Einheit und/oder Ausführungs-Cluster aufweisen – und im Fall einer getrennten Speicherzugriffs-Pipeline werden bestimmte Ausführungsformen implementiert, bei denen nur das Ausführungs-Cluster dieser Pipeline die Speicherzugriffseinheit(en) **164** aufweist). Es versteht sich auch, dass, wo getrennte Pipelines verwendet werden, eine oder mehrere dieser Pipelines Außerreihenfolge-Ausgabe/-Ausführung und der Rest In-Reihenfolge sein können.

[0031] Die Menge von Speicherzugriffseinheiten **164** ist mit der Speichereinheit **170** gekoppelt, die eine Daten-TLB-Einheit **172** umfasst, die mit einer Daten-Cache-Einheit **174** gekoppelt ist, die mit einer Level-2-(L2-) Cache-Einheit **176** gekoppelt ist. Bei einer beispielhaften Ausführungsform können die Speicherzugriffseinheiten **164** eine Ladeeinheit, eine Speicheradresseneinheit und eine Speicherdateneinheit umfassen, die jeweils

mit der Daten-TLB-Einheit **172** in der Speichereinheit **170** gekoppelt sind. Die Anweisungs-Cache-Einheit **134** ist ferner mit einer Level-2-(L2-)Cache-Einheit **176** in der Speichereinheit **170** gekoppelt. Die L2-Cache-Einheit **176** ist mit einer oder mehreren anderen Ebenen von Cache und letztendlich mit einem Hauptspeicher gekoppelt.

[0032] Beispielsweise kann die beispielhafte Registerumbenennungs-Außerreihenfolge-Ausgabe-/-Ausführungs-Kern-Architektur die Pipeline **100** folgendermaßen implementieren: 1) der Anweisungs-Abruf **138** führt die Abruf- und Längendecodierungsstufen **102** und **104** aus; 2) die Decodiereinheit **140** führt die Decodierstufe **106** aus; 3) die Umbenennungs-/Zuteilereinheit **152** führt die Zuteilungsstufe **108** und die Umbenennungsstufe **110** aus; 4) die Scheduler-Einheit(en) **156** führt die Schedule-Stufe **112** aus; 5) die physischen Registerfile-Einheit(en) **158** und die Speichereinheit **170** führen die Registerlese-/Speicherlesestufe **114** aus; das Ausführungs-Cluster **160** führt die Ausführungsstufe **116** aus; 6) die Speichereinheit **170** und die physischen Registerfile-Einheit(en) **158** führen die Rückschreib-/Speicherschreibstufe **118** aus; 7) verschiedene Einheiten können an der Programmfehlerabwicklungsstufe **122** beteiligt sein; und 8) die Ausscheidungseinheit **154** und die physischen Registerfile-Einheit(en) **158** führen die Commit-Stufe **124** aus.

[0033] Der Kern **190** kann einen oder mehrere Anweisungssätze unterstützen (z. B. den x86-Anweisungssatz (mit bestimmten Erweiterungen, die bei neueren Versionen hinzugefügt wurden); den MIPS-Anweisungssatz der MIPS Technologies in Sunnyvale, CA; den ARM-Anweisungssatz (mit optionalen zusätzlichen Erweiterungen wie NEON) der ARM Holdings in Sunnyvale, CA), einschließlich der Anweisung(en), die hier beschrieben werden. Bei einer Ausführungsform umfasst der Kern **190** Logik zur Unterstützung einer Gepackte-Daten-Anweisungssatz-Erweiterung (z. B. AVX1, AVX2, und/oder eine bestimmte Form des generischen vektorfreundlichen Anweisungsformats (U=0 und/oder U=1) nachfolgend beschrieben), um dadurch eine Ausführung der von vielen Multimediaanwendungen verwendeten Operationen unter Verwendung gepackter Daten zu erlauben.

[0034] Es versteht sich, dass der Kern Mehrfach-Threading (Ausführen von zwei oder mehr parallelen Mengen von Operationen oder Threads) unterstützen kann und zwar auf vielfältige Weisen, einschließlich Zeitscheiben-Mehrfach-Threading, Simultan-Mehrfach-Threading (wobei ein einziger physischer Kern einen logischen Kern für jeden der Threads bereitstellt, die der physische Kern simultan im Mehrfach-Threading behandelt), oder eine Kombination davon (z. B. Zeitschalt-Abruf und -Decodierung und simultanes Mehrfach-Threading danach, wie etwa bei der Hyperthreading-Technologie von Intel®).

[0035] Obwohl Registerumbenennung im Kontext der Außerreihenfolge-Ausführung beschrieben wird, versteht sich, dass Registerumbenennung in einer In-Reihenfolge-Architektur verwendet werden kann. Obwohl die dargestellte Ausführungsform des Prozessors auch getrennte Anweisungs- und Daten-Cache-Einheiten **134**/**174** und eine gemeinsam benutzte L2-Cache-Einheit **176** umfasst, können alternative Ausführungsformen einen einzigen internen Cache sowohl für Anweisungen als auch für Daten aufweisen, wie etwa zum Beispiel einen internen Level-1-(L1-)Cache oder mehrere Ebenen von internem Cache. Bei bestimmten Ausführungsformen kann das System eine Kombination aus einem internen Cache und einem externen Cache, der sich außerhalb des Kerns und/oder des Prozessors befindet, umfassen. Als Alternative kann sich der gesamte Cache außerhalb des Kerns und/oder des Prozessors befinden.

[0036] **Fig.** 2 ist ein Blockschaltbild eines Prozessors **200**, der mehr als einen Kern aufweisen kann, eine integrierte Speichersteuerung aufweisen kann und integrierte Grafik aufweisen kann, gemäß Ausführungsformen der Erfindung. Die durchgezogen gezeichneten Kästen in **Fig.** 2 zeigen einen Prozessor **200** mit einem einzigen Kern **202A**, einem Systemagenten **210**, einer Menge von einer oder mehreren Bussteuerungseinheiten **216**, während der optionale Zusatz der gestrichelt gezeichneten Kästen einen alternativen Prozessor **200** mit mehreren Kernen **202A**–N, einer Menge von einer oder mehreren integrierten Speichersteuerungseinheiten **214** in der Systemagenteinheit **210** und Speziallogik **208** zeigt.

[0037] Somit können verschiedene Implementierungen des Prozessors **200** Folgendes umfassen: 1) eine CPU mit der Speziallogik **208**, die integrierte Grafik ist, und/oder wissenschaftliche (Durchsatz-)Logik (die einen oder mehrere Kerne umfassen kann) und wobei die Kerne **202A**–N ein oder mehrere Vielzweckkerne sein können (z. B. Vielzweck-In-Reihenfolge-Kerne, Vielzweck-Außerreihenfolge-Kerne, eine Kombination der beiden); 2) einen Koprozessor, wobei die Kerne **202A**–N eine große Anzahl von Spezialkernen sind, die hauptsächlich für Grafik und/oder Wissenschaft (Durchsatz) bestimmt sind; und 3) einen Koprozessor mit den Kernen **202A**–N, die eine große Anzahl von Vielzweck-In-Reihenfolge-Kernen sind. Somit kann der Prozessor **200** ein Vielzweckprozessor, ein Koprozessor oder Spezialprozessor sein, wie zum Beispiel ein Netzwerk- oder Kommunikationsprozessor, eine Kompressions-Engine, ein Grafikprozessor, eine GPGPU (Vielzweck-Grafikverarbeitungseinheit), ein Hochdurchsatz-MIC-Koprozessor (Many Integrated Core) (mit 30 oder mehr

Kernen), ein eingebetteter Prozessor oder dergleichen. Der Prozessor kann auf einem oder mehreren Chips implementiert sein. Der Prozessor **200** kann Teil von einem oder mehreren Substraten und/oder darauf implementiert sein, die beliebige einer Anzahl von Prozessortechnologien verwenden, wie zum Beispiel BiCMOS, CMOS oder NMOS.

[0038] Die Speicherhierarchie umfasst eine oder mehrere Ebenen von Cache in den Kernen, eine Menge von einer oder mehreren gemeinsam benutzten Cache-Einheiten **206** und externen Speicher (nicht gezeigt), der mit der Menge von integrierten Speichersteuerungseinheiten **214** gekoppelt ist. Die Menge gemeinsam benutzter Cache-Einheiten **206** kann einen oder mehrere Mitebenen-Caches umfassen, wie etwa Level 2 (L2), Level 3 (L3), Level 4 (L4) oder andere Ebenen von Cache, einen Last-Level-Cache (LLC) und/oder Kombinationen davon. Obwohl bei einer Ausführungsform eine Verbindungseinheit **212** auf Ringbasis die integrierte Grafiklogik **208**, die Menge von gemeinsam benutzten Cache-Einheiten **206** und die Systemagenteinheit **210**/integrierten Speichersteuerungseinheit(en) **214** verbindet, können alternative Ausführungsformen eine beliebige Anzahl wohlbekannter Techniken zur Verbindung solcher Einheiten verwenden. Bei einer Ausführungsform wird Kohärenz zwischen einer oder mehreren Cache-Einheiten **206** und Kernen **202A**–N aufrechterhalten.

[0039] Bei bestimmten Ausführungsformen sind ein oder mehrere der Kerne **202A**–N zu Mehrfach-Threading fähig. Der Systemagent **210** umfasst die Komponenten, die die Kerne **202A**–N koordinieren und betreiben. Die Systemagenteinheit **210** kann zum Beispiel eine Leistungssteuereinheit (PCU) und eine Anzeigeeinheit umfassen. Die PCU kann Logik und Komponenten sein oder umfassen, die zum Regeln des Leistungszustands der Kerne **202A**–N und der integrierten Grafiklogik **208** benötigt werden. Die Anzeigeeinheit dient zum Ansteuern einer oder mehrerer extern verbundener Anzeigen.

[0040] Die Kerne **202A**–N können im Hinblick auf Architekturanweisungssatz homogen oder heterogen sein; das heißt, zwei oder mehr der Kerne **202A**–N können in der Lage sein, denselben Anweisungssatz auszuführen, während andere in der Lage ein können, nur eine Teilmenge dieses Anweisungssatzes oder einen anderen Anweisungssatz auszuführen. Bei einer Ausführungsform sind die Kerne **202A**–N heterogen und umfassen sowohl die ”kleinen” Kerne als auch die ”großen” Kerne, die nachfolgend beschrieben werden.

[0041] **Fig.** 3–**Fig.** 6 sind Blockschaltbilder beispielhafter Computerarchitekturen. Es sind auch andere Systementwürfe und Konfigurationen, die in der Technik für Laptops, Desktops, in der Hand gehaltene PCs, Personal Digital Assistants, technische Workstations, Server, Netzwerkvorrichtungen, Netzwerk-Hubs, Switches, eingebettete Prozessoren, digitale Signalprozessoren (DSP), Grafikvorrichtungen, Videospielvorrichtungen, Set-Top-Boxes, Mikrosteuerungen, Mobiltelefone, tragbare Medienplayer, in der Hand gehaltene Vorrichtungen und verschiedene andere elektronische Vorrichtungen bekannt sind, geeignet. Im Allgemeinen ist eine enorme Vielfalt von Systemen oder elektronischen Vorrichtungen mit der Fähigkeit, einen Prozessor und/oder andere Ausführungslogik wie hier offenbart zu enthalten, geeignet.

[0042] Nunmehr mit Bezug auf **Fig.** 3 ist ein Blockschaltbild eines Systems **300** gemäß einer Ausführungsform der vorliegenden Erfindung gezeigt. Das System **300** kann einen oder mehrere Prozessoren **310**, **315** umfassen, die mit einem Steuerungs-Hub **320** gekoppelt sind. Bei einer Ausführungsform umfasst der Steuerungs-Hub **320** einen Grafikspeichersteuerungs-Hub (GMCH) **390** und einen Eingabe/Ausgabe-Hub (IOH) **350** (die sich auf getrennten Chips befinden können); der GMCH **390** umfasst Speicher- und Grafiksteuerungen, mit denen Speicher **340** und ein Koprozessor **345** gekoppelt sind; der IOH **350** koppelt Eingabe/Ausgabe-(E/A-) Vorrichtungen **360** mit dem GMCH **390**. Als Alternative sind eine oder beide der Speicher- und Grafiksteuerungen in den Prozessor (wie hier beschrieben) integriert, der Speicher **340** und der Koprozessor **345** sind direkt mit dem Prozessor **310** gekoppelt und der Steuerungs-Hub **320** in einem einzigen Chip mit dem IOH **350**.

[0043] Die optionale Beschaffenheit der zusätzlichen Prozessoren **315** wird in **Fig.** 3 mit gestrichelten Linien bezeichnet. Jeder Prozessor **310**, **315** kann einen oder mehrere der hier beschriebenen Prozessorkerne umfassen und kann eine bestimmte Version des Prozessors **300** sein.

[0044] Der Speicher **340** kann zum Beispiel dynamischer Direktzugriffsspeicher (DRAM), Phasenänderungsspeicher (PCM) oder eine Kombination der beiden sein. Für mindestens eine Ausführungsform kommuniziert der Steuerungs-Hub **320** über einen Mehrfachabkopplungsbus, wie etwa einen Frontside-Bus (FSB), eine Punkt-zu-Punkt-Schnittstelle wie eine QuickPath-Verbindung (QPI) oder eine ähnliche Verbindung **395** mit den Prozessoren **310**, **315**.

[0045] Bei einer Ausführungsform ist der Koprozessor **345** ein Spezialprozessor, wie zum Beispiel ein Hochdurchsatz-MIC-Prozessor, ein Netzwerk- oder Kommunikationsprozessor, eine Kompressions-Engine, ein

Grafikprozessor, eine GPGPU, ein eingebetteter Prozessor oder dergleichen. Bei einer Ausführungsform kann der Steuerungs-Hub **320** einen integrierten Grafikbeschleuniger umfassen.

[0046] Es kann vielfältige Unterschiede zwischen den physischen Betriebsmitteln **310**, **315** im Hinblick auf ein Spektrum von Nutzmetriken geben, wie zum Beispiel architekturelle, mikroarchitekturelle, thermische, Stromverbrauchseigenschaften und dergleichen.

[0047] Bei einer Ausführungsform führt der Prozessor **310** Anweisungen aus, die Datenverarbeitungsoperationen eines allgemeinen Typs steuern. In die Anweisungen können Koprozessoranweisungen eingebettet sein. Der Prozessor **310** erkennt diese Koprozessoranweisungen als von einem Typ, der durch den angeschlossenen Koprozessor **345** ausgeführt werden soll. Dementsprechend gibt der Prozessor **310** diese Koprozessoranweisungen (oder Koprozessoranweisungen repräsentierende Steuersignale) auf einem Koprozessorbus oder einer anderen Verbindung an den Koprozessor **345** aus. Koprozessor(en) **345** nehmen die empfangenen Koprozessoranweisungen an und führen sie aus.

[0048] Nunmehr mit Bezug auf **Fig.** 4 ist ein Blockschaltbild eines ersten spezifischeren beispielhaften Systems **400** gemäß einer Ausführungsform der vorliegenden Erfindung gezeigt.

[0049] Wie in **Fig.** 4 gezeigt, ist das Mehrprozessorsystem **400** ein Punkt-zu-Punkt-Verbindungssystem und umfasst einen ersten Prozessor **470** und einen zweiten Prozessor **480**, die über eine Punkt-zu-Punkt-Verbindung **450** gekoppelt sind. Jeder der Prozessoren **470** und **480** kann eine bestimmte Version des Prozessors **200** sein. Bei einer Ausführungsform der Erfindung sind die Prozessoren **470** und **480** Prozessoren **310** bzw. **315**, während der Koprozessor **438** der Koprozessor **345** ist. Bei einer anderen Ausführungsform sind die Prozessoren **470** und **480** Prozessor **310** bzw. Koprozessor **345**.

[0050] Die Prozessoren **470** und **480** sind als IMC-Einheiten **472** bzw. **482** (integrierte Speichersteuerung) enthaltend gezeigt. Der Prozessor **470** umfasst als Teil seiner Bussteuerungseinheiten außerdem Punkt-zu-Punkt-(P-P-)Schnittstellen **476** und **478**; ähnlich umfasst der zweite Prozessor **480** P-P-Schnittstellen **486** und **488**. Die Prozessoren **470**, **480** können über eine Punkt-zu-Punkt-(P-P-)Schnittstelle **450** unter Verwendung von P-P-Schnittstellenschaltungen **478**, **488** Informationen austauschen. Wie in **Fig.** 4 gezeigt, koppeln IMC **472** und **482** die Prozessoren mit jeweiligen Speichern, nämlich einem Speicher **432** und einem Speicher **434**, die Teile von Hauptspeicher sein können, der lokal an die jeweiligen Prozessoren angeschlossen ist.

[0051] Die Prozessoren **470**, **480** können jeweils über einzelne P-P-Schnittstellen **452**, **454** unter Verwendung von Punkt-zu-Punkt-Schnittstellenschaltungen **476**, **494**, **486**, **498** Informationen mit einem Chipsatz **490** austauschen. Der Chipsatz **490** kann gegebenenfalls über eine Hochleistungsschnittstelle **439** Informationen mit dem Koprozessor **438** austauschen. Bei einer Ausführungsform ist der Koprozessor **438** ein Spezialprozessor, wie zum Beispiel ein Hochdurchsatz-MIC-Prozessor, ein Netzwerk- oder Kommunikationsprozessor, eine Kompressions-Engine, ein Grafikprozessor, eine GPGPU, ein eingebetteter Prozessor oder dergleichen.

[0052] Ein (nicht gezeigter) gemeinsam benutzter Cache kann in jedem Prozessor oder außerhalb beider Prozessoren enthalten sein und dennoch über P-P-Verbindungen mit den Prozessoren verbunden sein, so dass lokale Cache-Informationen von einem oder beiden der Prozessoren in dem gemeinsam benutzten Cache gespeichert werden können, wenn ein Prozessor in einen Low-Power-Modus versetzt wird.

[0053] Der Chipsatz **490** kann über eine Schnittstelle **496** mit einem ersten Bus **416** gekoppelt sein. Bei einer Ausführungsform kann der erste Bus **416** ein PCI-Bus (Peripheral Component Interconnect) oder ein Bus wie etwa ein PCI-Express-Bus oder ein anderer E/A-Verbindungsbus dritter Generation sein, obwohl der Schutzumfang der vorliegenden Erfindung nicht darauf beschränkt ist.

[0054] Wie in **Fig.** 4 gezeigt, können verschiedene E/A-Vorrichtungen **414** mit dem ersten Bus **416** zusammen mit einer Busbrücke **418**, die den ersten Bus **416** mit einem zweiten Bus **420** koppelt, gekoppelt sein. Bei einer Ausführungsform sind ein oder mehrere zusätzliche Prozessoren **415** wie Koprozessoren, Hochdurchsatz-MIC-Prozessoren, GPGPUs, Beschleuniger (wie zum Beispiel Grafikbeschleuniger oder Digitalsignalverarbeitungs-(DSP-)Einheiten), am Einsatzort programmierbare Gatearrays oder ein beliebiger anderer Prozessor mit dem ersten Bus **416** gekoppelt. Bei einer Ausführungsform kann der zweite Bus **420** ein LPC-Bus (Low Pin Count) sein. Es können bei einer Ausführungsform verschiedene Vorrichtungen mit einem zweiten Bus **420** gekoppelt sein, darunter zum Beispiel eine Tastatur und/oder eine Maus **422**, Kommunikationsvorrichtungen **427** und eine Speichereinheit **428**, wie etwa eine Festplatte oder eine andere Massenspeichervorrichtung, die Anweisungen/Code und Daten **430** umfassen können. Ferner kann eine Audio-E/A **424** mit dem zweiten Bus

420 gekoppelt sein. Man beachte, dass andere Architekturen möglich sind. Zum Beispiel kann ein System anstelle der Punkt-zu-Punkt-Architektur von **Fig.** 4 einen Mehrfachabkopplungsbus oder eine andere solche Architektur implementieren.

[0055] Nunmehr mit Bezug auf **Fig.** 5 ist ein Blockschaltbild eines zweiten spezifischeren beispielhaften Systems **500** gemäß einer Ausführungsform der vorliegenden Erfindung gezeigt. Gleiche Elemente in **Fig.** 4 und **Fig.** 5 tragen gleiche Bezugszahlen und bestimmte Aspekte von **Fig.** 4 wurden aus **Fig.** 5 weggelassen, um eine Verschleierung anderer Aspekte von **Fig.** 5 zu vermeiden.

[0056] **Fig.** 5 zeigt, dass die Prozessoren **470**, **480** integrierte Speicher- und E/A-Steuerlogik (CL) **472** bzw. **482** umfassen können. Somit umfassen CL **472**, **482** integrierte Speichersteuerungseinheiten und umfassen E/A-Steuerlogik. **Fig.** 5 zeigt, dass nicht nur die Speicher **432**, **434** mit der CL **472**, **482** gekoppelt sind, sondern auch dass E/A-Vorrichtungen **514** auch mit der Steuerlogik **472**, **482** gekoppelt sind. Veraltete E/A-Vorrichtungen **515** werden mit dem Chipsatz **490** gekoppelt.

[0057] Nunmehr mit Bezug auf **Fig.** 6 ist ein Blockschaltbild eines SoC **600** gemäß einer Ausführungsform der vorliegenden Erfindung gezeigt. Ähnliche Elemente in **Fig.** 2 tragen gleiche Bezugszahlen. Außerdem sind gestrichelt gezeichnete Kästen optionale Merkmale auf fortschrittlicheren SoC. In **Fig.** 6 ist eine Verbindungseinheit(en) **602** mit Folgendem gekoppelt: einem Anwendungsprozessor **610**, der eine Menge von einem oder mehreren Kernen **202A**–N und gemeinsam benutzte Cache-Einheit(en) **206** umfasst; einer Systemagenteinheit **210**; einer Bussteuerungseinheit(en) **216**; einer integrierten Speichersteuerungseinheit(en) **214**; einer Menge von einem oder mehreren Koprozessoren **620**, die integrierte Grafiklogik, einen Bildprozessor, einen Audioprozessor und einen Videoprozessor umfassen können; einer SRAM-Einheit **630** (statischer Direktzugriffsspeicher); einer DMA-Einheit **632** (Direct Memory Access); und einer Anzeigeeinheit **640** zur Kopplung mit einer oder mehreren externen Anzeigen. Bei einer Ausführungsform umfassen die Koprozessor(en) **620** einen Spezialprozessor, wie zum Beispiel einen Netzwerk- oder Kommunikationsprozessor, eine Kompressions-Engine, eine GPGPU, einen Hochdurchsatz-MIC-Prozessor, einen eingebetteten Prozessor oder dergleichen.

[0058] Ausführungsformen der hier offenbarten Mechanismen können in Hardware, Software, Firmware oder einer Kombination solcher Implementierungsansätze implementiert werden. Ausführungsformen der Erfindung können als Computerprogramme oder Programmcode zur Ausführung auf programmierbaren Systemen implementiert werden, die mindestens einen Prozessor, ein Speichersystem (mit flüchtigem und nichtflüchtigem Speicher und/oder Speicherelementen), mindestens eine Eingabevorrichtung und mindestens eine Ausgabevorrichtung umfassen.

[0059] Programmcode, wie etwa der in **Fig.** 4 dargestellte Code **430**, kann auf Eingabeanweisungen angewandt werden, um die hier beschriebenen Funktionen auszuführen und Ausgabeinformationen zu erzeugen. Die Ausgabeinformationen können auf bekannte Weise an eine oder mehrere Ausgabevorrichtungen angelegt werden. Für die Zwecke der vorliegenden Anmeldung umfasst ein Verarbeitungssystem ein beliebiges System mit einem Prozessor, wie etwa einem Digitalsignalprozessor (DSP), einer Mikrosteuerung, einer anwendungsspezifischen integrierten Schaltung (ASIC) oder einem Mikroprozessor.

[0060] Der Programmcode kann in einer höheren prozeduralen oder objektorientierten Programmiersprache zur Kommunikation mit einem Verarbeitungssystem implementiert werden. Der Programmcode kann gegebenenfalls auch in Assembler- oder Maschinensprache implementiert werden. Tatsächlich sind die hier beschriebenen Mechanismen bezüglich des Umfangs nicht auf irgendeine konkrete Programmiersprache beschränkt. Auf jeden Fall kann die Sprache eine kompilierte oder interpretierte Sprache sein.

[0061] Ein oder mehrere Aspekte mindestens einer Ausführungsform können durch repräsentative Anweisungen implementiert werden, die auf einem maschinenlesbaren Medium gespeichert werden, wodurch verschiedenartige Logik in dem Prozessor repräsentiert wird, die, wenn sie durch eine Maschine gelesen wird, bewirkt, dass die Maschine Logik zum Ausführen der hier beschriebenen Techniken generiert. Solche Repräsentationen, die als „IP-Kerne” bekannt sind, können auf einem greifbaren maschinenlesbaren Medium gespeichert und verschiedenen Kunden oder Herstellungseinrichtungen geliefert werden, um in die Herstellungsmaschinen geladen zu werden, die die Logik oder den Prozessor tatsächlich herstellen.

[0062] Solche maschinenlesbare Speichermedien waren zum Beispiel und ohne Beschränkung nichtflüchtige greifbare Anordnungen von Artikeln, die durch eine Maschine oder Vorrichtung hergestellt oder gebildet werden, darunter Speichermedien wie Festplatten, eine beliebige Art von Datenträger wie Disketten, optische Datenträger, CD-ROM (Compact Disk Read-Only Memories), CD-RW (Compact Disk Rewritable's) und ma-

gnetooptische Datenträger, Halbleitervorrichtungen wie Nurlesespeicher (ROM), Direktzugriffsspeicher (RAM) wie dynamische Direktzugriffsspeicher (DRAM), statische Direktzugriffsspeicher (SRAM), löschbare programmierbare Nurlesespeicher (EPROM), Flash-Speicher, elektrisch löschbare programmierbare Nurlesespeicher (EEPROM), Phasenänderungsspeicher (PCM), magnetische oder optische Karten oder eine beliebige andere Art von Medien, die zum Speichern elektronischer Anweisungen geeignet sind.

[0063] Ausführungsformen der Erfindung umfassen dementsprechend auch nichtflüchtige greifbare maschinenlesbare Medien, die Anweisungen enthalten oder Entwurfsdaten enthalten, wie etwa HDL (Hardware Description Language), wodurch Strukturen, Schaltungen, Vorrichtungen, Prozessoren und/oder Systemmerkmale definiert werden, die hier beschrieben werden. Solche Ausführungsformen können auch als Programmprodukte bezeichnet werden.

[0064] In bestimmten Fällen kann ein Anweisungsumsetzer verwendet werden, um eine Anweisung aus einem Quellenanweisungssatz in einen Zielanweisungssatz umzusetzen. Zum Beispiel kann der Anweisungsumsetzer (z. B. unter Verwendung statischer binärer Übersetzung, dynamischer binärer Übersetzung einschließlich dynamischer Kompilierung) eine Anweisung in eine oder mehrere andere Anweisungen, die durch den Kern zu verarbeiten sind, übersetzen, umformen, emulieren oder anderweitig umsetzen. Der Anweisungsumsetzer kann in Software, Hardware, Firmware oder einer Kombination davon implementiert werden. Der Anweisungsumsetzer kann auf dem Prozessor, außerhalb des Prozessors oder teilweise auf dem Prozessor und teilweise außerhalb des Prozessors sein.

[0065] **Fig.** 7 ist ein Blockschaltbild, das die Verwendung eines Softwareanweisungsumsetzers zum Umsetzen binärer Anweisungen in einem Quellenanweisungssatz in binäre Anweisungen in einem Zielanweisungssatz gemäß Ausführungsformen der Erfindung kontrastiert. Bei der dargestellten Ausführungsform ist der Anweisungsumsetzer ein Softwareanweisungsumsetzer, obwohl der Anweisungsumsetzer als Alternative in Software, Firmware, Hardware oder verschiedenen Kombinationen davon implementiert werden kann. **Fig.** 7 zeigt ein Programm in einer höheren Sprache **702**, das unter Verwendung eines x86-Compilers **704** kompiliert werden kann, um x86-Binärcode **706** zu erzeugen, der nativ durch einen Prozessor mit mindestens einem x86-Anweisungssatzkern **716** ausgeführt werden kann. Der Prozessor mit dem mindestens einen x86-Anweisungssatzkern **716** repräsentiert einen beliebigen Prozessor, der im Wesentlichen dieselben Funktionen wie ein Intel-Prozessor mit mindestens einem x86-Anweisungssatzkern ausführen kann, indem Folgendes kompatibel ausgeführt oder anderweitig verarbeitet wird: (1) ein wesentlicher Teil des Anweisungssatzes des Intel-x86-Anweisungssatzkerns oder (2) Objektcodeversionen von Anwendungen oder anderer Software, die dafür bestimmt sind, auf einem Intel-Prozessor mit mindestens einem x86-Anweisungssatzkern ausgeführt zu werden, um im Wesentlichen dasselbe Ergebnis wie ein Intel-Prozessor mit mindestens einem x86-Anweisungssatzkern zu erzielen. Der x86-Compiler **704** repräsentiert einen Compiler, der betreibbar ist, um x86-Binärcode **706** (z. B. Objektcode) zu erzeugen, der mit oder ohne zusätzliche Verknüpfungsverarbeitung auf dem Prozessor mit dem mindestens einen x86-Anweisungssatzkern **716** ausgeführt werden kann. Ähnlich zeigt **Fig.** 7, dass das Programm in der höheren Sprache **702** unter Verwendung eines Alternativ-Anweisungssatz-Compilers **708** kompiliert werden kann, um Alternativ-Anweisungssatz-Binärcode **710** zu erzeugen, der nativ durch einen Prozessor ohne mindestens einen x86-Anweisungssatzkern **714** ausgeführt werden kann (z. B. einen Prozessor mit Kernen, die den MIPS-Anweisungssatz der MIPS Technologies in Sunnyvale, CA, ausführen und/oder die den ARM-Anweisungssatz der ARM Holdings in Sunnyvale, CA, ausführen). Der Anweisungsumsetzer **712** dient zum Umsetzen des x86-Binärcodes **706** in Code, der nativ durch den Prozessor ohne einen x86-Anweisungssatzkern **714** ausgeführt werden kann. Dieser umgesetzte Code ist wahrscheinlich nicht derselbe wie der Alternativ-Anweisungssatz-Binärcode **710**, da ein Anweisungssatzumsetzer, der hierzu fähig ist, schwierig herzustellen ist; der umgesetzte Code erreicht jedoch die allgemeine Operation und kann aus Anweisungen aus dem Alternativ-Anweisungssatz bestehen. Somit repräsentiert der Anweisungsumsetzer **712** Software, Firmware, Hardware oder eine Kombination davon, die durch Emulation, Simulation oder einen beliebigen anderen Prozess einem Prozessor oder einer anderen elektronischen Vorrichtung ohne einen x86-Anweisungssatzprozessor oder -kern erlaubt, den x86-Binärcode **706** auszuführen.

VERFAHREN UND VORRICHTUNG ZUM AUSFÜHREN MEHRERER MULTIPLIKATIONSOPERATIONEN

[0066] Die nachfolgend beschriebenen Ausführungsformen der Erfindung stellen architekturelle Erweiterungen für eine Familie von Multiplikationsanweisungen bereit, die zwei Multiplikationen in einer einzigen Anweisung ausführen. Bei einer Ausführungsform werden die architekturellen Erweiterungen der Intel-Architektur (IA) bereitgestellt, aber die zugrundeliegenden Prinzipien der Erfindung sind nicht auf irgendeine konkrete ISA beschränkt.

[0067] Bei existierenden Prozessorarchitekturen führt jede Multiplikationsanweisung eine einzige Multiplikationsoperation aus. Zum Beispiel multiplizieren in der Intel-Architektur VMULSS und VMULPS zwei Einzelgenauigkeits-Gleitkommawerte und VMULSD und VMULPD multiplizieren zwei Doppelgenauigkeits-Gleitkommawerte. Im Gegensatz dazu führt die hier beschriebene Familie von Doppelmultiplikationsanweisungen (bei einer Ausführungsform als VMUL3-Anweisungen bezeichnet) zwei Multiplikationen in einer einzigen Anweisung aus, um dadurch die Leistung zu verringern und Decodierungsschlitze für andere Anweisungen zu befreien. Bei einer Ausführungsform werden die zwei Multiplikationen an drei Quellenoperanden ausgeführt: der zweite und dritte Quellenoperand können zuerst multipliziert werden, um ein Zwischenergebnis zu erzeugen, das dann mit dem ersten Quellenoperand multipliziert wird.

[0068] Wie in **Fig.** 8 dargestellt, umfasst ein beispielhafter Prozessor **855**, auf dem Ausführungsformen der Erfindung implementiert werden können, eine Ausführungseinheit **840** mit VMUL3-Ausführungslogik **841** zum Ausführen der hier beschriebenen VMUL3-Anweisungen. Eine Registermenge **805** stellt Registerspeicherung für Operanden, Steuerdaten und andere Arten von Daten bereit, während die Ausführungseinheit **840** den Anweisungsstrom ausführt.

[0069] In **Fig.** 8 sind der Einfachheit halber die Einzelheiten eines einzigen Prozessorkerns (”Kern 0”) dargestellt. Es versteht sich jedoch, dass jeder in **Fig.** 8 gezeigte Kern dieselbe Menge von Logik wie Kern 0 aufweisen kann. Wie dargestellt, kann jeder Kern einen dedizierten Level-1-(L1)-Cache **812** und Level-2-(L2)-Cache **811** zum Cache-Speichern von Anweisungen und Daten gemäß einer spezifizierten Cache-Verwaltungsrichtlinie umfassen. Der L1-Cache **811** umfasst einen separaten Anweisungs-Cache **120** zum Speichern von Anweisungen und einen separaten Daten-Cache **121** zum Speichern von Daten. Die in den verschiedenen Prozessor-Caches gespeicherten Anweisungen und Daten werden auf der Granularität von Cache-Leitungen verwaltet, die eine feste Größe aufweisen können (z. B. von 64, 128, 512 Byte Länge). Jeder Kern dieser beispielhaften Ausführungsform besitzt eine Anweisungs-Abrufeinheit **810** zum Abrufen von Anweisungen aus dem Hauptspeicher **800** und/oder einen gemeinsam benutzten Level-3-(L3)-Cache **816**; eine Decodiereinheit **820** zum Decodieren der Anweisungen (z. B. Decodieren von Programmanweisungen in Mikro-Operationen oder ”uops”); eine Ausführungseinheit **840** zum Ausführen der Anweisungen (z. B. der hier beschriebenen VMUL3-Anweisungen); und eine Rückschreibeinheit **850** zum Ausscheiden der Anweisungen und Rückschreiben der Ergebnisse.

[0070] Die Anweisungsabrufeinheit **810** umfasst verschiedene wohlbekannte Komponenten, darunter ein Nächste-Anweisung-Zeiger **803** zum Speichern der Adresse der nächsten aus dem Speicher **800** (oder einem der Caches) abzurufenden Anweisung; einen Anweisungsübersetzungs-Look-Aside-Puffer (ITLB) **804** zum Speichern einer Map vor kurzem verwendeter virtuell-zu-physisch-Anweisungsadressen zur Verbesserung der Geschwindigkeit der Adressenübersetzung; eine Zweigvorhersageeinheit **802** zum spekulativen Vorhersagen von Anwendungszweigadressen; und Zweigzielpuffer (BTB) **801** zum Speichern von Zweigadressen und Zieladressen. Nach dem Abruf werden Anweisungen dann zu den verbleibenden Stufen der Anweisungspipeline, einschließlich der Decodiereinheit **830**, der Ausführungseinheit **840** und der Rückschreibeinheit **850**, gestreamt. Die Struktur und Funktion jeder dieser Einheiten ist Durchschnittsfachleuten wohlbekannt und wird hier nicht im Einzelnen beschrieben, um eine Verschleierung der relevanten Aspekte der verschiedenen Ausführungsformen der Erfindung zu vermeiden.

[0071] Bei einer Ausführungsform der Erfindung führt die VMUL3-Ausführungslogik **841** die folgende Familie von Anweisungen aus:
VMUL3SS xmm1 {k1} {z}, xmm2, xmm3/mV {er}
VMUL3PS zmm1 {k1} {z}, zmm2, zmm3/B32(mV) {er}
VMUL3SD xmm1 {k1} {z}, xmm2, xmm3/mV {er}
VMUL3PD zmm1 {k1} {z}, zmm2, zmm3/B64(mV) {er}
wobei xmm1–3 und zmm1–3 Register in der Registermenge **805** sind, die gepackte oder skalare Gleitkommawerte entweder im Einzelgenauigkeits-(32-Bit-) oder Doppelgenauigkeits-(64-Bit-)Gleitkommaformat speichern.

[0072] Insbesondere multipliziert bei einer Ausführungsform VMUL3SS drei skalare Einzelgenauigkeits-Gleitkommawerte, die in xmm1, xmm2 und xmm3 gespeichert sind. Im Betrieb kann der zweite Operand (aus xmm2) mit dem dritten Operanden (aus xmm3) multipliziert werden, und die Ergebnisse können (mit Zwischenrundung) mit dem ersten Operanden (aus xmm1) multipliziert und in einem Zielregister gespeichert werden. Bei einer Ausführungsform ist das Zielregister dasselbe Register, das zum Speichern des ersten Operanden verwendet wird (z. B. xmm1).

[0073] Bei einer Ausführungsform multipliziert VMUL3PS drei gepackte Einzelgenauigkeits-Gleitkommawerte, die in zmm1, zmm2 und zmm3 gespeichert sind. Im Betrieb kann der zweite Operand (aus zmm2) mit dem dritten Operanden (aus zmm3) multipliziert werden, und die Ergebnisse können (mit Zwischenrundung) mit dem ersten Operanden (aus zmm1) multipliziert und in einem Zielregister gespeichert werden. Bei einer Ausführungsform ist das Zielregister dasselbe Register, das zum Speichern des ersten Operanden verwendet wird (z. B. zmm1).

[0074] Bei einer Ausführungsform multipliziert VMUL3SD drei skalare Doppelgenauigkeits-Gleitkommawerte, die in xmm1, xmm2 und xmm3 gespeichert sind. Im Betrieb kann der zweite Operand (aus xmm2) mit dem dritten Operanden (aus xmm3) multipliziert werden, und die Ergebnisse können (mit Zwischenrundung) mit dem ersten Operanden (aus xmm1) multipliziert und in einem Zielregister gespeichert werden. Bei einer Ausführungsform ist das Zielregister dasselbe Register, das zum Speichern des ersten Operanden verwendet wird (z. B. xmm1).

[0075] Schließlich multipliziert bei einer Ausführungsform VMUL3PD drei gepackte Doppelgenauigkeits-Gleitkommawerte, die in zmm1, zmm2 und zmm3 gespeichert sind. Im Betrieb kann der zweite Operand (aus zmm2) mit dem dritten Operanden (aus zmm3) multipliziert werden, und die Ergebnisse können (mit Zwischenrundung) mit dem ersten Operanden (aus zmm1) multipliziert und in einem Zielregister gespeichert werden. Bei einer Ausführungsform ist das Zielregister dasselbe Register, das zum Speichern des ersten Operanden verwendet wird (z. B. zmm1).

[0076] Bei einer Ausführungsform werden drei Zwischenbit [2:0] jeder der VMUL3-Anweisungen zur Steuerung des Vorzeichens der Multiplikationen verwendet. Zum Beispiel kann der Wert von Bit 0 des Zwischenelements das Vorzeichen des ersten Operanden steuern (z. B. 1 = negativ und 0 = positiv oder umgekehrt); der Wert von Bit 1 des Zwischenelements kann das Vorzeichen des zweiten Operanden steuern; und der Wert von Bit 2 des Zwischenelements kann das Vorzeichen des dritten Operanden steuern.

[0077] Bei einer Ausführungsform werden der erste und zweite Operand aus SIMD-Registern (Single Instruction Multiple Data) gelesen, während der dritte Operand aus einem SIMD-Register oder einer Speicherstelle gelesen werden kann.

[0078] **Fig.** 9A zeigt zusätzliche Einzelheiten, die einer Ausführungsform der VMUL3-Ausführungslogik **841** zugeordnet sind, einschließlich eines Zuteilers **940** zum Zuteilen von Betriebsmitteln für jede VMUL3-uop und einer Reservierungsstation **902** zum Einteilten von VMUL3-uops zur Ausführung durch Funktionseinheiten **912**. Im Betrieb transferiert der Anweisungsdecoder **806** nach der Decodierstufe **830**, in der jede VMUL3-Anweisung zu uops decodiert wird, die uops zu einer Zuteilereinheit **940**, die eine Registeraliastabelle (RAT) **941** umfasst. In einer Außerreihenfolge-Pipeline weist die Zuteilereinheit **940** jede ankommende uop einer Speicherstelle in einem Umordnungspuffer (ROB) **950** zu, um dadurch die logische Zieladresse der uop auf eine entsprechende physische Zieladresse im ROB **950** abzubilden. Die RAT **941** hält diese Abbildung.

[0079] Die Inhalte eines ROB **905** können letztendlich in Speicherstellen in einem realen Registerfile (RRF) **951** ausgeschieden werden. Die RAT **941** kann auch ein Reales-Registerfile-Gültig-Bit speichern, das angibt, ob der durch die logische Adresse angegebene Wert an der physischen Adresse im ROB **950** oder im RRF **951** nach dem Ausscheiden zu finden ist. Wenn er im RRF gefunden wird, wird der Wert als Teil des aktuellen Prozessorarchitekturzustands betrachtet. Auf der Basis dieser Abbildung ordnet die RAT **941** auch jede logische Quellenadresse einer entsprechenden Speicherstelle im ROB **950** oder RRF **951** zu.

[0080] Jede ankommende uop wird außerdem durch den Zuteiler **940** in einen Eintrag in der Reservierungsstation (RS) **902** zugewiesen und geschrieben. Die Reservierungsstation **902** stellt die Ausführung durch die Funktionseinheit **912** erwartenden VMUL3-uops zusammen. Im vorliegenden Fall führen zwei FMA-Funktionseinheiten (Fused Multiply and Add) FMA0 **910** und FMA1 **911** Multiplikationsoperationen wie nachfolgend beschrieben aus, um die VMUL3-Anweisungen auszuführen. Wenn es notwendig ist, können Ergebnisse über einen Rückschreibbus wieder in RS **902** geschrieben werden.

[0081] Bei einer Ausführungsform werden Reservierungsstationseinträge logisch in Gruppen unterteilt, um die Anzahl der Lese- und Schreibports zu verringern, die zum Lesen bzw. Schreiben der Einträge erforderlich sind. Bei der in **Fig.** 9A gezeigten Ausführungsform teilen zwei Reservierungsstationsgruppen RS0 **900** und RS1 **901** die Ausführung der VMUL3-uops durch die Funktionseinheiten FMA0 **900** und FMA1 **901** über die Ports 0 bzw. 1 ein.

[0082] Bei einer Ausführungsform können beliebige der VMUL3-Anweisungen mittels der Pipeline als eine einzige uop ausgeführt werden. Insbesondere wird die uop zuerst durch FMA0 **900** (über RS0 **900**) ausgeführt, die die erste Multiplikation des zweiten und dritten Operanden (z. B. aus xmm2/xmm3 oder zmm2/zmm3 wie oben besprochen) ausführt, um ein Zwischenergebnis zu produzieren. Die uop wird in der Puffereinheit **905** verzögert und dann durch FMA1 **911** (über RS1 **901**) ein zweites Mal ausgeführt, um das Zwischenergebnis und den ersten Operanden (z. B. aus xxm1/zmm1) zu multiplizieren. Wie zuvor erwähnt, kann das Endergebnis in xmm1/zmm1 gespeichert werden. Außerdem kann wie erwähnt der unmittelbare Wert der VMUL3-Anweisung das Vorzeichen für jeden der drei Quellenoperanden spezifizieren. Bei einer Ausführungsform wird das zweite Ausgeben der uop (über den Puffer **905**) genau für die FMA-Latenz (z. B. 5 Taktzyklen) vor dem Neuausgeben der Anweisung zum Warten gezwungen.

[0083] Es können verschiedene existierende Datenumgehungen verwendet werden, um das Zwischenergebnis der FMA1 **911** auf Port 1 zuzuführen. Bei einer Ausführungsform wird das Zwischenergebnis vorübergehend im ROB **950** oder an einer beliebigen anderen Speicherstelle gespeichert, woraus es durch FMA1 **911** gelesen und verwendet werden kann. Bei einer Ausführungsform kann der Rückschreibbus verwendet werden, um das Zwischenergebnis RS1 **910** zuzuführen, die dann das Zwischenergebnis über Port 1 FMA1 **911** zur Verfügung stellt. Die zugrundeliegenden Prinzipien der Erfindung sind jedoch nicht auf irgendeine konkrete Weise beschränkt, das Zwischenergebnis FMA1 **911** zuzuführen. Obwohl in **Fig.** 9A ein ROB **950** dargestellt ist, versteht sich außerdem, dass bei bestimmten Prozessorimplementierungen (z. B. In-Reihenfolge-Pipelines) kein ROB **950** verwendet wird und eine andere Form von Speicherung verwendet werden kann, um das Zwischenergebnis und das Endergebnis nach Ausführung zu speichern.

[0084] Wie in **Fig.** 9B dargestellt, sind nicht zwei Funktionseinheiten zur Implementierung der zugrundeliegenden Prinzipien der Erfindung erforderlich. Speziell führt bei dieser Ausführungsform dieselbe Funktionseinheit – FMA0 **910** – die VMUL3-uop zweimal in der Folge aus, um das Endergebnis zu erzeugen. Das heißt, FMA0 **910** führt die erste Multiplikation zwischen den zweiten und dritten Operanden aus und rezirkuliert das Zwischenergebnis und die uop durch sich selbst hindurch, um die zweite Multiplikation auszuführen (die nach dem Abschluss durch den Rest der Pipeline geleitet wird). Obwohl die zweite Iteration der uop als durch die Reservierungsstation **902** verlaufend gezeigt ist, wird die Rezirkulation bei einer Ausführungsform einfach in der Funktionseinheitsstufe **912** durchgeführt (d. h. direkt von 1 MA0 **910** zu sich selbst unter Verwendung von temporärer Pufferspeicherung in der Funktionseinheitsstufe **921**). Außerdem führt bei einer anderen Implementierung eine neue dedizierte Funktionseinheit in der Menge von Funktionseinheiten **912** die VMUL3-Anweisung unabhängig aus (d. h. ohne Verwendung einer fusionierten Multiplizier- und Addierfunktionseinheit).

[0085] Die oben beschriebene Ausführungsform stellt verbesserten Stromverbrauch gegenüber der Verwendung von zwei VMUL-Anweisungen bereit, da nur eine Anweisung decodiert wird. Außerdem ist garantiert, dass die zeitliche Quelle mittels Umgehungen gelesen wird, so dass keine Daten aus dem Registerfile gelesen werden müssen.

[0086] Bei Anwendungen, bei denen mehrere Elemente miteinander multipliziert werden, kann die Anzahl der Multiplikationsanweisungen unter Benutzung der hier beschriebenen VMUL3-Anweisungen durch 2 dividiert werden. Beispielsweise kann für eine lange Schleife, die vektorisiert werden kann, wobei Gleitkommawerte multipliziert werden, ein VMUL3 verwendet werden, um den Anweisungszählwert praktisch um 2 zu verringern.

[0087] Eine Ausführungsform eines Verfahrens zum Ausführen mehrerer Multiplikationsoperationen ist in **Fig.** 10 dargestellt. Bei **1001** wird eine einzelne VMUL3-Anweisung aus dem Speichersubsystem abgerufen. Wie erwähnt, umfasst die VMUL3-Anweisung einen ersten, zweiten und dritten Quellenoperanden, einen Zieloperanden und einen Unmittelbarwert. Bei **1002** wird die VMUL3-Anweisung zu uops decodiert. Wie bereits erwähnt, kann bei einer Ausführungsform eine einzige Multiplizier-uop erzeugt werden (und zweimal ausgeführt werden für die zwei zum Abschluss der VMUL3-Anweisung erforderlichen Multiplikationsoperationen).

[0088] Bei **1003** werden die Quellenoperandenwerte als Vorbereitung zur Ausführung durch die Funktionseinheiten abgerufen. Diese Operation kann zum Beispiel durch die Reservierungsstation **902** und/oder die Zuteilereinheit **940** ausgeführt werden.

[0089] Bei **1004** wird die VMUL3-Anweisung ausgeführt. Bei einer Ausführungsform wird die Multiplizier-uop einmal unter Verwendung des zweiten und dritten Operanden zur Erzeugung eines Zwischenergebnisses ausgeführt. Die uop wird dann unter Verwendung des Zwischenergebnisses und des ersten Operanden ein zweites Mal ausgeführt, um das Endergebnis (d. h. die Multiplikation des ersten, zweiten und dritten Quellenoperanden)

zu erzeugen. Wie erwähnt, kann das Vorzeichen jedes der Quellenoperanden als ein Drei-Bit-Zwischenwert bereitgestellt werden.

[0090] Bei **1005** wird das Ergebnis der VMUL3-Anweisung in einer Zieloperandenspeicherstelle (z. B. einem Register) gespeichert, woraus es für eine oder mehrere nachfolgende Operationen gelesen werden kann.

BEISPIELHAFTE ANWEISUNGSFORMATE

[0091] Ausführungsformen der hier beschriebenen Anweisung(en) können in verschiedenen Formaten realisiert werden. Zusätzlich sind nachfolgend beispielhafte Systeme, Architekturen und Pipelines aufgeführt. Ausführungsformen der Anweisung(en) können auf solchen Systemen, Architekturen und Pipelines ausgeführt werden, sind aber nicht auf die aufgeführten beschränkt.

[0092] Ein vektorfreundliches Anweisungsformat ist ein Anweisungsformat, das für Vektoranweisungen geeignet ist (z. B. gibt es bestimmte Felder, die für Vektoroperationen spezifisch sind). Obwohl Ausführungsformen beschrieben werden, bei denen sowohl Vektor- als auch Skalaroperationen durch das vektorfreundliche Anweisungsformat unterstützt werden, verwenden alternative Ausführungsform mit nur Vektoroperationen das vektorfreundliche Anweisungsformat.

[0093] **Fig.** 11A–**Fig.** 11B sind Blockschaltbilder eines generischen vektorfreundlichen Anweisungsformats und der Anweisungsvorlagen davon gemäß Ausführungsformen der Erfindung. **Fig.** 11A ist ein Blockschaltbild, das ein generisches vektorfreundliches Anweisungsformat und Klasse-A-Anweisungsvorlagen davon gemäß Ausführungsformen der Erfindung zeigt; dagegen ist **Fig.** 11B ein Blockschaltbild, das das generische vektorfreundliche Anweisungsformat und Klasse-B-Anweisungsvorlagen davon gemäß Ausführungsformen der Erfindung zeigt. Speziell ein generisches vektorfreundliches Anweisungsformat **1500**, für das Anweisungsvorlagen der Klasse A und Klasse B definiert sind, die beide Anweisungsvorlagen ohne Speicherzugriff **1505** und Anweisungsvorlagen mit Speicherzugriff **1520** umfassen. Der Ausdruck generisch im Kontext des vektorfreundlichen Anweisungsformats bezieht sich darauf, dass das Anweisungsformat nicht an irgendeinen spezifischen Anweisungssatz gebunden ist.

[0094] Obwohl Ausführungsformen der Erfindung beschrieben werden, bei denen das vektorfreundliche Anweisungsformat Folgendes unterstützt: eine 64-Byte-Vektoroperandenlänge (oder -größe) mit Datenelementbreiten (oder -größen) von 32 Bit (4 Byte) oder 64 Bit (8 Byte) (und somit besteht ein 64-Byte-Vektor entweder aus 16 Elementen mit Doppelwortgröße oder als Alternative 8 Elementen mit Quadwortgröße); eine 64-Byte-Vektoroperandenlänge (oder -größe) mit Datenelementbreiten (oder -größen) von 16 Bit (2 Byte) oder 8 Bit (1 Byte); eine 32-Byte-Vektoroperandenlänge (oder -größe) mit Datenelementbreiten (oder -größen) von 32 Bit (4 Byte), 64 Bit (8 Byte), 16 Bit (2 Byte) oder 8 Bit (1 Byte); und eine 16-Byte-Vektoroperandenlänge (oder -größe) mit Datenelementbreiten (oder -größen) von 32 Bit (4 Byte), 64 Bit (8 Byte), 16 Bit (2 Byte) oder 8 Bit (1 Byte); können alternative Ausführungsformen mehr, weniger und/oder andere Vektoroperandengrößen (z. B. 256-Byte-Vektoroperanden) mit mehr, weniger oder anderen Datenelementbreiten (z. B. Datenelementbreiten von 168 Bit (16 Byte)) unterstützen.

[0095] Zu den Klasse-A-Anweisungsvorlagen in **Fig.** 11A gehört Folgendes: 1) innerhalb der Anweisungsvorlagen ohne Speicherzugriff **1505** ist eine Anweisungsvorlage ohne Speicherzugriff und Operation des Vollrundungssteuertyps **1510** und eine Anweisungsvorlage ohne Speicherzugriff und Datentransformationstypoperation **1515** gezeigt; und 2) innerhalb der Anweisungsvorlagen mit Speicherzugriff **1520** sind eine Anweisungsvorlage mit Speicherzugriff, zeitlich **1525** und eine Anweisungsvorlage mit Speicherzugriff, nicht zeitlich **1530** gezeigt. Zu den Klasse-B-Anweisungsvorlagen in **Fig.** 11B gehört Folgendes: 1) innerhalb der Anweisungsvorlagen ohne Speicherzugriff **1505** sind eine Anweisungsvorlage ohne Speicherzugriff und mit Schreibmaskensteuerungs-Partialrundungssteuertypoperation **1516** und eine Anweisungsvorlage ohne Speicherzugriff mit Schreibmaskensteuerungs-vsize-Typoperation **1517** gezeigt; und 2) innerhalb der Anweisungsvorlagen mit Speicherzugriff **1520** ist eine Anweisungsvorlage mit Speicherzugriff und Schreibmaskensteuerung **1527** gezeigt.

[0096] Das generische Vektorfreundliche Anweisungsformat **1500** umfasst die folgenden nachfolgend aufgelisteten Felder in der in den **Fig.** 11A–**Fig.** 11B dargestellten Reihenfolge.

[0097] Das Formatfeld **1540** – ein spezifischer Wert (ein Anweisungsformat-Kennungswert) in diesem Feld identifiziert eindeutig das vektorfreundliche Anweisungsformat und somit das Auftreten von Anweisungen in dem vektorfreundlichen Anweisungsformat in Anweisungsströmen. Dementsprechend ist dieses Feld in dem

Sinne optional, als dass es für einen Anweisungssatz, der nur das generische vektorfreundliche Anweisungsformat aufweist, nicht benötigt wird.

[0098] Das Basisoperationsfeld **1542** – sein Inhalt unterscheidet verschiedene Basisoperationen.

[0099] Das Registerindexfeld **1544** – sein Inhalt spezifiziert direkt oder mittels Adressenerzeugung die Speicherstellen der Quellen- und Zieloperanden, seien sie in Registern oder in Speicher. Zu diesen gehört eine ausreichende Anzahl von Bit zur Auswahl von N Registern aus einem Registerfile von PxQ (z. B. 32x516, 16x168, 32x1024, 64x1024). Obwohl bei einer Ausführungsform N bis zu drei Quellen und ein Zielregister sein kann, können alternative Ausführungsformen mehr oder weniger Quellen und Zielregister unterstützen (können z. B. bis zu zwei Quellen unterstützen, wobei eine dieser Quellen auch als das Ziel wirkt, können bis zu drei Quellen unterstützen, wobei eine dieser Quellen auch als das Ziel wirkt, können bis zu zwei Quellen und ein Ziel unterstützen).

[0100] Das Modifiziererfeld **1546** – sein Inhalt unterscheidet das Auftreten von Anweisungen in dem generischen Vektoranweisungsformat, die Speicherzugriff spezifizieren, von denjenigen, die es nicht tun; das heißt zwischen Anweisungsvorlagen ohne Speicherzugriff **1505** und Anweisungsvorlagen mit Speicherzugriff **1520**. Speicherzugriffsoperationen Lesen und/oder Schreiben in die Speicherhierarchie (wobei in bestimmten Fällen die Quellen- und/oder Zieladressen unter Verwendung von Werten in Registern spezifiziert werden), während Nicht-Speicherzugriffsoperationen es nicht tun (z. B. sind die Quelle und Ziele Register). Obwohl bei einer Ausführungsform dieses Feld auch zwischen drei verschiedenen Weisen zum Durchführen von Speicheradressenberechnungen auswählt, können alternative Ausführungsformen mehr, weniger oder andere Weisen zum Durchführen von Speicheradressenberechnungen unterstützen.

[0101] Das Ergänzungsoperationsfeld **1550** – sein Inhalt unterscheidet, welche von vielfältigen verschiedenen Operationen zusätzlich zu der Basisoperation auszuführen ist. Dieses Feld ist kontextspezifisch. Bei einer Ausführungsform der Erfindung ist dieses Feld in ein Klassenfeld **1568**, ein Alphafeld **1552** und ein Betafeld **1554** aufgeteilt. Das Ergänzungsoperationsfeld **1550** erlaubt das Ausführen gemeinsamer Gruppen von Operationen in einer einzigen Anweisung, statt 2, 3 oder 4 Anweisungen.

[0102] Das Skalenfeld **1560** – sein Inhalt erlaubt die Skalierung des Inhalts des Indexfelds zur Speicheradressenerzeugung (z. B. zur Adressenerzeugung, die 2^{Skala}·Index + Basis verwendet).

[0103] Das Verschiebungsfeld **1562A** – sein Inhalt wird als Teil der Speicheradressenerzeugung verwendet (z. B. zur Adressenerzeugung, die 2^{Skala}·Index + Basis + Verschiebung verwendet).

[0104] Das Verschiebungsfaktorfeld **1562B** (man beachte, dass die Überlagerung des Verschiebungsfelds **1562A** direkt über das Verschiebungsfaktorfeld **1562B** angibt, dass eines oder das andere verwendet wird) – sein Inhalt wird als Teil der Adressenerzeugung verwendet; er spezifiziert einen Verschiebungsfaktor, der um die Größe eines Speicherzugriffs (N) zu skalieren ist – wobei N die Anzahl der Byte in der Speicheradresse ist (z. B. zur Adressenerzeugung, die 2^{Skala}·Index + Basis + skalierte Verschiebung verwendet). Redundante Bit niedriger Ordnung werden ignoriert, und daher wird der Inhalt des Verschiebungsfaktorfelds mit der Speicheroperanden-Gesamtgröße (N) multipliziert, um die letztendliche bei der Berechnung einer effektiven Adresse zu verwendende Verschiebung zu erzeugen. Der Wert von N wird durch die Prozessorhardware zur Laufzeit auf der Basis des vollen Opcode-Felds **1574** (hier beschrieben) und des Datenmanipulationsfelds **1554C** bestimmt. Das Verschiebungsfeld **1562A** und das Verschiebungsfaktorfeld **1562B** sind in dem Sinne optional, dass sie nicht für die Anweisungsvorlagen ohne Speicherzugriff **1505** verwendet werden, und/oder andere Ausführungsformen nur eine oder keine der beiden implementieren können.

[0105] Das Datenelementbreitenfeld **1564** – sein Inhalt unterscheidet, welche einer Anzahl von Datenelementbreiten zu verwenden ist (bei bestimmten Ausführungsformen für alle Anweisungen; bei anderen Ausführungsformen nur für bestimmte der Anweisungen). Dieses Feld ist in dem Sinne optional, dass es nicht benötigt wird, wenn nur eine Datenelementbreite unterstützt wird und/oder Datenelementbreiten unterstützt werden, die einen bestimmten Aspekt der Opcodes verwenden.

[0106] Das Schreibmaskenfeld **1570** – sein Inhalt steuert datenelementpositionsweise, ob diese Datenelementposition im Zielvektoroperanden das Ergebnis der Basisoperation und Ergänzungsoperation widerspiegelt. Klasse-A-Anweisungsvorlagen unterstützen Zusammenlegungs-Schreibmaskierung, während Klasse-B-Anweisungsvorlagen sowohl Zusammenlegungs- als auch Nullungs-Schreibmaskierung unterstützen. Beim Zusammenlegen erlauben Vektormasken einen Schutz einer beliebigen Menge von Elementen im Ziel vor

Aktualisierungen während der Ausführung einer beliebigen Operation (spezifiziert durch die Basisoperation und die Ergänzungsoperation); und bei einer anderen Ausführungsform Bewahren des alten Werts jedes Elements des Ziels, wenn das entsprechende Maskenbit eine 0 aufweist. Beim Nullen erlauben Vektormasken dagegen das Nullen einer beliebigen Menge von Elementen im Ziel während der Ausführung einer beliebigen Operation (spezifiziert durch die Basisoperation und die Ergänzungsoperation); bei einer Ausführungsform wird ein Element des Ziels auf 0 gesetzt, wenn das entsprechende Maskenbit einen 0-Wert aufweist. Eine Teilmenge dieser Funktionalität ist die Möglichkeit, die Vektorlänge der ausgeführten Operation zu steuern (d. h., der Spanne von modifizierten Elementen vom ersten bis zum letzen); es ist jedoch nicht notwendig, dass die Elemente, die modifiziert werden, aufeinanderfolgend sind. Das Schreibmaskenfeld **1570** erlaubt somit teilweise Vektoroperationen, darunter Ladevorgänge, Speichervorgänge, Arithmetik, Logik usw. Obwohl Ausführungsformen der Erfindung beschrieben werden, bei denen der Inhalt des Schreibmaskenfelds **1570** eines von mehreren Schreibmaskenregistern auswählt, das die zu verwendende Schreibmaske enthält (und somit der Inhalt des Schreibmaskenfelds **1570** indirekt die auszuführende Maskierung identifiziert), erlauben alternative Ausführungsformen stattdessen oder zusätzlich, dass Inhalt des der Maskenschreibfelds **1570** die auszuführende Maskierung direkt spezifiziert.

[0107] Das Unmittelbar-Feld **1572** – sein Inhalt erlaubt die Spezifikation eines Unmittelbaren. Dieses Feld ist in dem Sinne optional, dass es bei einer Implementierung des generischen vektorfreundlichen Formats, das Unmittelbares nicht unterstützt, nicht anwesend ist und in Anweisungen, die kein Unmittelbares verwenden, nicht anwesend ist.

[0108] Das Klassenfeld **1568** – sein Inhalt unterscheidet zwischen verschiedenen Klassen von Anweisungen. Mit Bezug auf **Fig.** 11A–B wählen die Inhalte dieses Felds zwischen Anweisungen der Klasse A und Klasse B aus. In **Fig.** 11A–B werden Quadrate mit abgerundeten Ecken verwendet, um anzuzeigen, dass ein spezifischer Wert in einem Feld anwesend ist (z. B. Klasse A **1568A** und Klasse B **1568B** für das Klassenfeld **1568** jeweils in **Fig.** 11A–B).

Anweisungsvorlagen der Klasse A

[0109] Im Fall der Anweisungsvorlagen ohne Speicherzugriff **1505** der Klasse A wird das Alphafeld **1552** als RS-Feld **1552A** interpretiert, dessen Inhalt unterscheidet, welcher der verschiedenen Ergänzungsoperationstypen auszuführen ist (z. B. werden Runden **1552A.1** bzw. Datentransformation **1552A.2** für die Anweisungsvorlagen ohne Speicherzugriff mit Rundungstypoperation **1510** und ohne Speicherzugriff mit Datentransformationstypoperation **1515** spezifiziert), während das Betafeld **1554** unterscheidet, welche der Operationen des spezifizierten Typs auszuführen ist. In den Anweisungsvorlagen ohne Speicherzugriff **1505** sind das Skalenfeld **1560**, das Verschiebungsfeld **1562A** und das Verschiebungsskalenfeld **1562B** nicht anwesend.

Anweisungsvorlagen ohne Speicherzugriff – Vollrundungssteuertypoperation

[0110] In der Anweisungsvorlage ohne Speicherzugriff mit Vollrundungssteuertypoperation **1510** wird das Betafeld **1554** als ein Rundungssteuerfeld **1554A** interpretiert, dessen Inhalte) statische Rundung bereitstellt bzw. bereitstellen. Obwohl bei den beschriebenen Ausführungsformen der Erfindung das Rundungssteuerfeld **1554A** ein SAE-Feld **1556** (Suppress all Floating Point Exceptions) und ein Rundungsoperationssteuerfeld **1558** umfasst, können alternative Ausführungsformen beide diese Konzepte im selben Feld unterstützen/codieren oder nur eines oder das andere dieser Konzepte/Felder aufweisen (können z. B. nur das Rundungsoperationssteuerfeld **1558** aufweisen).

[0111] Das SAE-Feld **1556** – sein Inhalt unterscheidet, ob die Programmfehlerereignismeldung zu deaktivieren ist oder nicht; wenn der Inhalt des SAE-Felds **1556** angibt, dass Unterdrückung aktiviert ist, meldet eine gegebene Anweisung keinerlei Art von Gleitkommaprogrammfehlerflag und erhebt keinerlei Gleitkomma-Programmfehlerhandler.

[0112] Das Rundungsoperations-Steuerfeld **1558** – sein Inhalt unterscheidet, welche einer Gruppe von Rundungsoperationen auszuführen ist (z. B. Aufrunden, Abrunden, Runden in Richtung von null und Runden zum Nächsten). Das Rundungsoperations-Steuerfeld **1558** erlaubt das anweisungsweise Ändern des Rundungsmodus. Bei einer Ausführungsform der Erfindung, bei der ein Prozessor ein Steuerregister zum Spezifizieren von Rundungsmodi umfasst, übersteuert der Inhalt des Rundungsoperations-Steuerfelds **1550** diesen Registerwert.

Anweisungsvorlagen ohne Speicherzugriff Datentransformationstypoperation

[0113] In der Anweisungsvorlage ohne Speicherzugriff mit Datentransformationstypoperation **1515** wird das Betafeld **1554** als ein Datentransformationsfeld **1554B** interpretiert, dessen Inhalt unterscheidet, welche einer Anzahl von Datentransformationen auszuführen ist (z. B. keine Datentransformation, Swizzle, Broadcast).

[0114] Im Fall einer Anweisungsvorlage mit Speicherzugriff **1520** der Klasse A wird das Alphafeld **1552** als ein Ausräumhinweisfeld **1552B** interpretiert, dessen Inhalt unterscheidet, welcher der Ausräumhinweise zu verwenden ist (in **Fig.** 12A werden zeitlich **1552B.1** bzw. nicht zeitlich **1552B.2** für die Anweisungsvorlage mit Speicherzugriff, zeitlich **1525** und die Anweisungsvorlage mit Speicherzugriff, nicht zeitlich **1530** spezifiziert), während das Betafeld **1554** als ein Datenmanipulationsfeld **1554C** interpretiert wird, dessen Inhalt unterscheidet, welche einer Anzahl von Datenmanipulationsoperationen (die auch als Primitiven bekannt sind) auszuführen ist (z. B. keine Manipulation; Broadcast; Aufwärtsumsetzung einer Quelle; und Abwärtsumsetzung eines Ziels). Die Anweisungsvorlagen mit Speicherzugriff **1520** umfassen das Skalenfeld **1560** und gegebenenfalls das Verschiebungsfeld **1562A** oder das Verschiebungsskalenfeld **1562B**.

[0115] Vektorspeicheranweisungen führen Vektorladevorgänge aus und Vektorspeichervorgänge in Speicher mit Umsetzungsunterstützung durch. Wie bei regulären Vektoranweisungen transferieren Vektorspeicheranweisungen Daten aus/in Speicher auf datenelementweise Art, wobei die Elemente, die tatsächlich transferiert werden, durch die Inhalte der Vektormaske vorgeschrieben werden, die als die Schreibmaske ausgewählt wird.

Anweisungsvorlagen mit Speicherzugriff – zeitlich

[0116] Zeitliche Daten sind Daten, die wahrscheinlich bald genug wiederverwendet werden, um aus Cache-Speicherung Nutzen zu ziehen. Dies ist jedoch ein Hinweis, und verschiedene Prozessoren können es auf verschiedene Weisen implementieren, einschließlich völliges Ignorieren des Hinweises.

Anweisungsvorlagen mit Speicherzugriff – nicht zeitlich

[0117] Nicht zeitliche Daten werden nur wenig wahrscheinlich bald genug wiederverwendet, um aus Cache Speicherung im Cache der 1. Ebene Nutzen zu ziehen, und sollten Priorität für Ausräumung erhalten. Dies ist jedoch ein Hinweis, und verschiedene Prozessoren können es auf verschiedene Weisen implementieren, einschließlich völliges Ignorieren des Hinweises.

Anweisungsvorlagen der Klasse B

[0118] Im Fall von Anweisungsvorlagen der Klasse B wird das Alphafeld **1552** als Feld **1552C** zur Schreibmaskensteuerung (Z) interpretiert, dessen Inhalt unterscheidet, ob die durch das Schreibmaskenfeld **1570** gesteuerte Schreibmaskierung eine Zusammenlegung oder eine Nullung sein soll.

[0119] Im Fall der Anweisungsvorlagen ohne Speicherzugriff **1505** der Klasse B wird ein Teil des Betafelds **1554** als ein RL-Feld **1557A** interpretiert, dessen Inhalt unterscheidet, welcher der verschiedenen Ergänzungsoperationstypen auszuführen ist (z. B. werden Rundung **1557A.1** und Vektorlänge (VSIZE) **1557A.2** für die Anweisungsvorlage ohne Speicherzugriff mit Schreibmaskensteuerung-Partialrundungssteuertypoperation **1516** bzw. die Anweisungsvorlage ohne Speicherzugriff mit Schreibmaskensteuerung-VSIZE-Typ-Operation **1517** spezifiziert), während der Rest des Betafelds **1554** unterscheidet, welche der Operationen des spezifizierten Typs auszuführen ist. In den Anweisungsvorlagen ohne Speicherzugriff **1505** sind das Skalenfeld **1560**, das Verschiebungsfeld **1562A** und das Verschiebungsskalenfeld **1552B** nicht anwesend.

[0120] In der Anweisungsvorlage ohne Speicherzugriff mit Schreibmaskensteuerung-Partialrundungssteuertypoperation **1510** wird der Rest des Betafelds **1554** als ein Rundungsoperationsfeld **1559A** interpretiert und Programmfehlerereignismeldung deaktiviert (eine gegebene Anweisung meldet keinerlei Art von Gleitkomma-Programmfehlerflag und ruft keinerlei Gleitkomma-Programmfehlerhandler hervor).

[0121] Das Rundungsoperationssteuerfeld **1559A** – wie beim Rundungsoperationssteuerfeld **1558** unterscheidet sein Inhalt, welche einer Gruppe von Rundungsoperationen auszuführen ist (z. B. Aufrunden, Abrunden, Runden nach null und Runden zum Nächsten). Das Rundungsoperationssteuerfeld **1559A** erlaubt somit das anweisungsweise Ändern des Rundungsmodus. Bei einer Ausführungsform der Erfindung, bei der ein Prozessor ein Steuerregister zum Spezifizieren von Rundungsmodi umfasst, übersteuert der Inhalt des Rundungsoperationssteuerfelds **1550** diesen Registerwert.

[0122] In der Anweisungsvorlage ohne Speicherzugriff mit Schreibmaskensteuerung-VSIZE-Typ-Operation **1517** wird des Rest des Betafelds **1554** als ein Vektorlängenfeld **1559B** interpretiert, dessen Inhalt unterscheidet, an welcher einer Anzahl von Datenvektorlängen auszuführen ist (z. B. 168, 256 oder 516 Byte).

[0123] Im Fall einer Anweisungsvorlage mit Speicherzugriff **1520** der Klasse B wird ein Teil des Betafelds **1554** als ein Broadcast-Feld **1557B** interpretiert, dessen Inhalt unterscheidet, ob die Broadcast-Typ-Datenmanipulationsoperation auszuführen ist, während der Rest des Betafelds **1554** als Vektorlängenfeld **1559B** interpretiert wird. Die Anweisungsvorlagen mit Speicherzugriff **1520** umfassen das Skalenfeld **1560** und gegebenenfalls das Verschiebungsfeld **1562A** oder das Verschiebungsskalenfeld **1562B**.

[0124] Hinsichtlich des generischen vektorfreundlichen Anweisungsformats **1500** ist ein Voll-Opcode-Feld **1574** gezeigt, das das Formatfeld **1540**, das Basisoperationsfeld **1542** und das Datenelementbreitenfeld **1564** umfasst. Obwohl eine Ausführungsform gezeigt ist, bei der das Voll-Opcode-Feld **1574** alle diese Felder umfasst, umfasst das Voll-Opcode-Feld **1574** bei Ausführungsformen, die diese nicht alle unterstützen, weniger als alle diese Felder. Das Voll-Opcode-Feld **1574** stellt den Operationscode (Opcode) bereit.

[0125] Das Ergänzungsoperationsfeld **1550**, das Datenelementbreitenfeld **1564** und das Schreibmaskenfeld **1570** erlauben ein anweisungsweises Spezifizieren dieser Merkmale in dem generischen vektorfreundlichen Anweisungsformat.

[0126] Die Kombination von Schreibmaskenfeld und Datenelementbreitenfeld erzeugt insofern typisierte Anweisungen, als sie das Anwenden der Maske an verschiedenen Datenelementbreiten erlauben.

[0127] Die verschiedenen in Klasse A und Klasse B anzutreffenden Anweisungsvorlagen sind in verschiedenen Situationen nützlich. Bei bestimmten Ausführungsformen der Erfindung können verschiedene Prozessoren oder verschiedene Kerne in einem Prozessor nur Klasse A, nur Klasse B oder beide Klassen unterstützen. Zum Beispiel kann ein hochleistungsfähiger Vielzweck-Außerreihenfolgekern, der für Vielzweckdatenverarbeitung bestimmt ist, nur Klasse B unterstützen, ein hauptsächlich für Grafik- und/oder wissenschaftliche (Durchsatz-)Datenverarbeitung gedachter Kern kann nur Klasse A unterstützen, und ein für beides bestimmter Kern kann beides unterstützen (natürlich liegt ein Kern mit einer bestimmten Mischung von Vorlagen und Anweisungen aus beiden Klassen, aber nicht allen Vorlagen und Anweisungen aus beiden Klassen, im Umfang der Erfindung). Außerdem kann ein einzelner Prozessor mehrere Kerne umfassen, die alle dieselbe Klasse unterstützen oder in denen verschiedene Kerne verschiedene Klasse unterstützen. Zum Beispiel kann in einem Prozessor mit separaten Grafik- und Vielzweckkernen einer der hauptsächlich für Grafik- und/oder wissenschaftliche Datenverarbeitung bestimmten Grafikkerne nur Klasse A unterstützen, während einer oder mehrere der Vielzweckkerne hochleistungsfähige Vielzweckkerne mit Außerreihenfolge-Ausführung und Registerumbenennung sein können, die für Vielzweckdatenverarbeitung bestimmt sind, die nur Klasse B unterstützen. Ein anderer Prozessor, der nicht über einen separaten Grafikkern verfügt, kann einen oder mehrere Vielzweck-In-Reihenfolge- oder -Außerreihenfolgekerne umfassen, die sowohl Klasse A als auch Klasse B unterstützen. Merkmale aus einer Klasse können natürlich bei verschiedenen Ausführungsformen der Erfindung auch in der anderen Klasse implementiert werden. In einer höheren Sprache geschriebene Programme würden in vielfältige verschiedene ausführbare Formen gelegt (z. B. Just-in-Time-kompiliert oder statisch kompiliert), darunter 1) eine Form, die nur Anweisungen der Klasse(n), die durch den Zielprozessor für Ausführung unterstützt wird, aufweist; oder 2) eine Form, die alternative Routinen aufweist, die unter Verwendung verschiedener Kombinationen der Anweisungen aller Klassen geschrieben werden und Steuerflusscode aufweisen, der die auszuführenden Routinen auf der Basis der Anweisungen auswählt, die durch den Prozessor unterstützt werden, der den Code gerade ausführt.

[0128] **Fig.** 12A–D ist ein Blockschaltbild eines beispielhaften spezifischen vektorfreundlichen Anweisungsformats gemäß Ausführungsformen der Erfindung. **Fig.** 12A–D zeigt ein spezifisches vektorfreundliches Anweisungsformat **1600**, das in dem Sinne spezifisch ist, dass es Ort, Größe, Interpretation und Reihenfolge der Felder sowie Werte für bestimmte dieser Felder spezifiziert. Das spezifische vektorfreundliche Anweisungsformat **1600** kann verwendet werden, um den x86-Anweisungssatz zu erweitern, und somit sind bestimmte der Felder ähnlich oder dieselben wie die in dem existierenden x86-Anweisungssatz und seinen Erweiterungen (z. B. AVX) verwendeten. Dieses Format bleibt mit dem Präfix-Codierungsfeld, Real-Opcode-Bytefeld, MOD-R/M-Feld, SIB-Feld, Verschiebungsfeld und Unmittelbar-Feldern des existierenden x86-Anweisungssatzes mit Erweiterungen vereinbar. Es sind die Felder aus **Fig.** 11 dargestellt, auf die Felder von **Fig.** 12 abgebildet werden.

[0129] Es versteht sich, dass zu Anschauungszwecken, obwohl Ausführungsformen der Erfindung mit Bezug auf das spezifische vektorfreundliche Anweisungsformat **1600** im Kontext des generischen vektorfreund-

lichen Anweisungsformats **1500** beschrieben werden, die Erfindung nicht auf das spezifische vektorfreundliche Anweisungsformat **1600** beschränkt ist, außer wenn es beansprucht wird. Zum Beispiel zieht das generische vektorfreundliche Anweisungsformat **1500** vielfältige mögliche Größen für die verschiedenen Felder in Betracht, während das spezifische vektorfreundliche Anweisungsformat **1600** als Felder spezifischer Größen aufweisend gezeigt ist. Als spezifisches Beispiel ist, obwohl das Datenelementbreitenfeld **1564** in dem spezifischen vektorfreundlichen Anweisungsformat **1600** als ein Ein-Bit-Feld dargestellt ist, die Erfindung nicht darauf beschränkt (das heißt, das generische vektorfreundliche Anweisungsformat **1500** zieht andere Größen des Datenelementbreitenfelds **1564** in Betracht).

[0130] Das generische vektorfreundliche Anweisungsformat **1500** umfasst die Folgenden, nachfolgend in der in **Fig.** 12A dargestellten Reihenfolge aufgelisteten Felder.

[0131] Der EVEX Präfix (Byte 0-3) **1602** – wird in Vier-Byte-Form codiert.

[0132] Das Formatfeld **1640** (EVEX Byte 0, Bit [7:0]) – das erste Byte (EVEX Byte 0) ist das Formatfeld **1640**, und es enthält 0x62 (den einzigartigen Wert, mit dem bei einer Ausführungsform der Erfindung das vektorfreundliche Anweisungsformat unterschieden wird).

[0133] Das zweite bis vierte Byte (EVEX Byte 1-3) umfassen eine Anzahl von Bitfeldern, die spezifische Fähigkeiten bereitstellen.

[0134] Das REX-Feld **1605** (EVEX Byte 1, Bit [7-5]) – besteht aus einem EVEX.R-Bitfeld (EVEX Byte 1, Bit [7] – R), einem EVEX.X-Bitfeld (EVEX Byte 1, Bit [6] – X) und dem **1557** BEX Byte 1, Bit [5] – B). Die Bitfelder EVEX.R, EVEX.X und EVEX.B stellen dieselbe Funktionalität wie die entsprechenden VEX-Bitfelder bereit und werden unter Verwendung der 1er-Komplementform codiert, d. h. ZMM0 wird als **1611B** codiert, ZMM15 wird als 0000B codiert. Andere Felder der Anweisungen codieren die unteren drei Bit der Registerindizes wie in der Technik bekannt (rrr, xxx und bbb), so dass Rrrr, Xxxx und Bbbb durch Addieren von EVEX.R, EVEX.X und EVEX.B gebildet werden können.

[0135] Das REX'-Feld **1605** – dies ist der erste Teil des REX'-Felds **1510** und ist das EVEX.R'-Bitfeld (EVEX Byte 1, Bit [4] – R') mit dem entweder die oberen 16 oder unteren 16 des erweiterten 32-Registersatzes codiert werden. Bei einer Ausführungsform der Erfindung wird dieses Bit zusammen mit anderen wie nachfolgend angegeben im bitinvertierten Format gespeichert, um (im wohlbekannten x86-32-Bitmodus) von der BOUND-Anweisung zu unterscheiden, deren Real-Opcode-Byte 62 ist, akzeptiert aber nicht in dem MOD-R/M-Feld (später beschrieben) den Wert von 11 im MOD-Feld; alternative Ausführungsformen der Erfindung speichern dies und die anderen angegebenen Bit nachfolgend in dem invertierten Format nicht. Ein Wert von 1 wird zur Codierung der unteren 16 Register verwendet. Anders ausgedrückt, wird R'Rrrr durch Kombinieren von EVEX.R', EVEX.R und des anderen RRR aus anderen Feldern gebildet.

[0136] Das Opcode-Map-Feld **1615** (EVEX Byte 1, Bit [3:0] – mmmm) – sein Inhalt codiert ein impliziertes vorderes Opcode-Byte (0F, 0F 38 oder 0F 3).

[0137] Das Datenelementbreitenfeld **1664** (EVEX Byte 2, Bit [7] – W) – wird durch die Notation EVEX.W repräsentiert. EVEX.W dient zur Definition der Granularität (Größe) des Datentyps (entweder 32-Bit-Datenelemente oder 64-Bit-Datenelemente).

[0138] EVEX.vvvv **1620** (EVEX Byte 2, Bit [6:3]-vvvv)- die Rolle von EVEX.vvvv kann Folgendes umfassen: 1) EVEX.vvvv codiert den ersten Quellenregisteroperanden, spezifiziert in invertierter (1er-Komplement-)Form und ist für Anweisungen mit zwei oder mehr Quellenoperanden gültig; 2) EVEX.vvvv codiert den Zielregisteroperanden, spezifiziert in 1er-Komplementform für bestimmte Vektorverschiebungen; oder 3) EVEX.vvvv codiert keinerlei Operanden, das Feld ist reserviert. Somit codiert das EVEX.vvvv-Feld **1620** die vier Bit niedriger Ordnung der ersten Quellenregisterkennung, die in invertierter (1er-Komplement-)Form gespeichert wird. Abhängig von der Anweisung wird ein zusätzliches unterschiedliches EVEX-Bitfeld verwendet, um die Kennungsgröße auf 32 Register zu erweitern.

[0139] Das EVEX.0 **1668**-Klassenfeld (EVEX Byte 2, Bit [2]-U) – Im Fall EVEX.U = 0 gibt dies Klasse A oder EVEX.U0 an; im Fall EVEX.0 = 1 gibt dies Klasse B oder EVEX.U1 an.

[0140] Das Präfix-Codierungsfeld **1625** (EVEX Byte 2, Bit [1:0]-pp) – stellt zusätzliche Bit für das Basisoperationsfeld bereit. Zusätzlich zu der Bereitstellung von Unterstützung der veralteten SSE-Anweisungen im EVEX-

Präfixformat hat dies auch den Vorteil des Kompaktierens des SIMD-Präfix (statt ein Byte zum Ausdrücken des SIMD-Präfix zu erfordern, erfordert das EVEX-Präfix nur 2 Bit). Bei einer Ausführungsform werden zur Unterstützung von veralteten SSE-Anweisungen, die ein SIMD-Präfix (66H, F2H, F3H) sowohl im veralteten Format als auch im EVEX-Präfixformat verwenden, diese veralteten SIMD-Präfixe in das SIMD-Präfixcodierungsfeld codiert; und werden zur Laufzeit in das veraltete SIMD-Präfix expandiert, bevor sie dem PLA des Decoders zugeführt werden (so dass der PLA sowohl das veraltete als auch das EVEX-Format dieser veralteten Anweisungen ohne Modifikation ausführen kann). Obwohl neuere Anweisungen den Inhalt des EVEX-Präfixcodierungsfelds direkt als Opcode-Erweiterung verwenden könnten, expandieren bestimmte Ausführungsformen auf ähnliche Weise für Einheitlichkeit, erlauben aber das Spezifizieren verschiedener Bedeutungen durch diese veralteten SIMD-Präfixe. Eine alternative Ausführungsform kann den PLA umentwerfen, um die 2-Bit-SIMD-Präfixcodierungen zu unterstützen, und erfordern somit die Expansion nicht.

[0141] Das Alphafeld **1652** (EVEX Byte 3, Bit [7] – EH; auch als EVEX.EH, EVEX.rs, EVEX.RL, EVEX.write mask control und EVEX.N bekannt; auch mit α dargestellt) – wie zuvor beschrieben ist dieses Feld kontextspezifisch.

[0142] Das Betafeld **1654** (EVEX Byte 3, Bit[6:4]-SSS, auch als EVEX.s_{2-0}, EVEX.r_{2-0}, EVEX.rr1, EVEX.LL0, EVEX.LLB bekannt; auch mit βββ dargestellt) – wie zuvor beschrieben ist dieses Feld kontextspezifisch.

[0143] Das REX'-Feld **1610** – dies ist der Rest des REX'-Felds und ist das EVEX.V'-Bitfeld (EVEX Byte 3, Bit [3] - V') mit dem entweder die oberen 16 oder unteren 16 des erweiterten 32-Registersatzes codiert werden können. Dieses Bit wird in bitinvertiertem Format gespeichert. Ein Wert von 1 wird zur Codierung der unteren 16 Register verwendet. Anders ausgedrückt, wird V'VVVV durch Kombinieren von EVEX.V', EVEX.vvvv gebildet.

[0144] Das Schreibmaskenfeld **1670** (EVEX Byte 3, Bit [2:0]-kkk) – sein Inhalt spezifiziert den Index eines Registers in den Schreibmaskenregistern wie zuvor beschrieben. Bei einer Ausführungsform der Erfindung hat der spezifische Wert EVEX.kkk = 000 ein spezielles Verhalten, das impliziert, dass keine Schreibmaske für die konkrete Anweisung verwendet wird (dies kann auf vielfältige Weisen implementiert werden, einschließlich Verwendung einer auf nur Einsen fest verdrahteten Schreibmaske oder von Hardware, die die Maskierungshardware umgeht).

[0145] Das Real-Opcode-Feld **1630** (Byte 4) ist auch als das Opcode-Byte bekannt. In diesem Feld wird ein Teil des Opcodes spezifiziert.

[0146] Das MOD-R/M-Feld **1640** (Byte 5) umfasst das MOD-Feld **1642**, das Reg-Feld **1644** und das R/M-Feld **1646**. Wie zuvor beschrieben unterscheidet der Inhalt des MOD-Felds **1642** zwischen Speicherzugriffs- und Nicht-Speicherzugriffsoperationen. Die Rolle des Reg-Felds **1644** kann in zwei Situationen zusammengefasst werden: codieren entweder des Zielregisteroperanden oder eines Quellenregisteroperanden oder Behandlung als Opcode-Erweiterung und nicht zur Codierung irgendeines Anweisungsoperanden verwendet. Die Rolle des R/M-Felds **1646** kann Folgendes umfassen: Codieren des Anweisungsoperanden, der auf eine Speicheradresse verweist, oder Codieren entweder des Zielregisteroperanden oder eines Quellenregisteroperanden.

[0147] Das Byte für Skala, Index, Basis (SIB) (Byte 6) – Wie zuvor beschrieben wird der Inhalt des Skalenfelds **1650** zur Speicheradressenerzeugung verwendet. SIB.xxx **1654** und SIB.bbb **1656** – auf die Inhalte dieser Felder wurde zuvor mit Bezug auf die Registerindizes Xxxx und Bbbb verwiesen.

[0148] Das Verschiebungsfeld **1662A** (Byte 7–10) – wenn das MOD-Feld **1642** 10 enthält, sind die Byte 7–10 das Verschiebungsfeld **1662A**, und es funktioniert genauso wie die veraltete 32-Bit-Verschiebung (disp32) und funktioniert auf Bytegranularität.

[0149] Das Verschiebungsfaktorfeld **1662B** (Byte 7) – wenn das MOD-Feld **1642** 01 enthält, ist Byte 7 das Verschiebungsfaktorfeld **1662B**. Die Speicherstelle dieses Felds ist dieselbe wie die der veralteten x86-Anweisungssatz-8-Bit-Verschiebung (disp8), die auf Bytegranularität funktioniert. Da disp8 vorzeichenerweitert ist, kann es nur zwischen –168 und 167 Byte Offsets adressieren; im Hinblick auf 64-Byte-Cacheleitungen verwendet disp8 8 Bit, die auf nur vier wirklich nützliche Werte –168, –64, 0 und 64 gesetzt werden können; da oft ein größerer Bereich notwendig ist, wird disp32 verwendet; disp32 erfordert jedoch 4 Byte. Im Gegensatz zu disp8 und disp32 ist das Verschiebungsfaktorfeld **1662B** eine Uminterpretation von disp8; bei Verwendung des Verschiebungsfaktorfelds **1662B** wird die tatsächliche Verschiebung durch den Inhalt des Verschiebungsfaktorfelds, multipliziert mit der Größe des Speicheroperandenzugriffs (N), bestimmt. Diese Art von Verschiebung wird als disp8*N bezeichnet. Dies verringert die mittlere Anweisungsmenge (es wird ein einziges Byte für die

Verschiebung verwendet, aber mit viel größerem Bereich). Eine solche komprimierte Verschiebung basiert auf der Annahme, dass die effektive Verschiebung ein Vielfaches der Granularität des Speicherzugriffs ist und daher die redundanten Bit niedriger Ordnung des Adressenoffsets nicht codiert werden müssen. Anders ausgedrückt, ersetzt das Verschiebungsfaktorfeld **1662B** die veraltete x86-Anweisungssatz-8-Bit-Verschiebung. Das Verschiebungsfaktorfeld **1662B** wird somit auf dieselbe Weise wie eine x86-Anweisungssatz-8-Bit-Verschiebung codiert (also keine Änderungen der ModRM/SIB-Codierungsregeln), mit der einzigen Ausnahme, dass disp8 auf disp8*N überladen wird. Anders ausgedrückt, gibt es keine Änderungen der Codierungsregeln oder Codierungslängen, sondern nur der Interpretation des Verschiebungswerts durch Hardware (die die Verschiebung um die Größe des Speicheroperanden skalieren muss, um ein byteweises Adressenoffset zu erhalten). Das Unmittelbar-Feld **1672** operiert wie zuvor beschrieben.

Das Voll-Opcode-Feld

[0150] Fig. 12B ist ein Blockschaltbild der Felder des spezifischen vektorfreundlichen Anweisungsformats **1600**, aus denen das Voll-Opcode-Feld **1674** besteht, gemäß einer Ausführungsform der Erfindung. Speziell umfasst das Voll-Opcode-Feld **1674** das Formatfeld **1640**, das Basisoperationsfeld **1642** und das Feld **1664** für die Datenelementbreite (W). Das Basisoperationsfeld **1642** umfasst das Präfixcodierungsfeld **1625**, das Opcode-Map-Feld **1615** und das Real-Opcode-Feld **1630**.

Das Registerindexfeld

[0151] Fig. 12C ist ein Blockschaltbild der Felder des spezifischen vektorfreundlichen Anweisungsformats **1600**, aus denen das Registerindexfeld **1644** besteht, gemäß einer Ausführungsform der Erfindung. Speziell umfasst das Registerindexfeld **1644** das REX-Feld **1605**, das REX'-Feld **1610**, das MODR/M.reg-Feld **1644**, das MODR/M.r/m-Feld **1646**, das VVVV-Feld **1620**, das xxx-Feld **1654** und das bbb-Feld **1656**.

Das Ergänzungsoperationsfeld

[0152] Fig. 12D ist ein Blockschaltbild der Felder des spezifischen vektorfreundlichen Anweisungsformats **1600**, aus denen das Ergänzungsoperationsfeld **1650** besteht, gemäß einer Ausführungsform der Erfindung. Wenn das Klassen-(U-)-Feld **1668** 0 enthält, bedeutet dies EVEX.U0 (Klasse A **1668A**); wenn es 1 enthält, bedeutet dies EVEX.U1 (Klasse B **1668B**). Wenn U=0 und das MOD-Feld **1642** 11 enthält (was eine Nicht-Speicherzugriffsoperation bedeutet), wird das Alphafeld **1652** (EVEX Byte 3, Bit [7] – EH) als das rs-Feld **1652A** interpretiert. Wenn das rs-Feld **1652A** eine 1 enthält (Rundung **1652A.1**), wird das Betafeld **1654** (EVEX Byte 3, Bit [6:4]- SSS) als das Rundungssteuerfeld **1654A** interpretiert. Das Rundungssteuerfeld **1654A** umfasst ein Ein-Bit-SAE-Feld **1656** und ein Zwei-Bit-Rundungsoperationsfeld **1658**. Wenn das rs-Feld **1652A** eine 0 enthält (Datentransformation **1652A.2**), wird das Betafeld **1654** (EVEX Byte 3, Bit [6:4]- SSS) als ein Drei-Bit-Datentransformationsfeld **1654B** interpretiert. Wenn U=0 ist und das MOD-Feld **1642** 00, 01 oder 10 enthält (was eine Speicherzugriffsoperation bedeutet), wird das Alphafeld **1652** (EVEX Byte 3, Bit [7] – EH) als das Ausräumhinweis-(EH)Feld **1652B** interpretiert und das Betafeld **1654** (EVEX Byte 3, Bit [6:4]- SSS) wird als ein Drei-Bit-Datenmanipulationsfeld **1654C** interpretiert.

[0153] Wenn U=1 ist, wird das Alphafeld **1652** (EVEX Byte 3, Bit [7] – EH) als das Feld **1652C** für Schreibmaskensteuerung (Z) interpretiert. Wenn U=1 ist und das MOD-Feld **1642** 11 enthält (was eine Nicht-Speicherzugriffsoperation bedeutet), wird ein Teil des Betafelds **1654** (EVEX Byte 3, Bit [4]- So) als das RL-Feld **1657A** interpretiert; wenn es eine 1 enthält (Rundung **1657A.1**), wird der Rest des Betafelds **1654** (EVEX Byte 3, Bit [6-5]- S_{2-1}) als das Rundungsoperationsfeld **1659A** interpretiert, während, wenn das RL-Feld **1657A** eine 0 enthält (VSIZE **1657.A2**) der Rest des Betafelds **1654** (EVEX Byte 3, Bit [6-5]- S_{2-1}) als das Vektorlängenfeld **1659B** (EVEX Byte 3, Bit [6-5]- L_{1-0}) interpretiert wird. Wenn U=1 ist und das MOD-Feld **1642** 00, 01 oder 10 enthält (was eine Speicherzugriffsoperation bedeutet), wird das Betafeld **1654** (EVEX Byte 3, Bit [6:4]- SSS) als das Vektorlängenfeld **1659B** (EVEX Byte 3, Bit [6-5]- L_{1-0}) und das Broadcast-Feld **1657B** (EVEX Byte 3, Bit [4]- B) interpretiert.

[0154] Fig. 13 ist ein Blockschaltbild einer Registerarchitektur **1700** gemäß einer Ausführungsform der Erfindung. Bei der dargestellten Ausführungsform gibt es 32 Vektorregister **1710** mit einer Breite von 516 Bit; diese Register werden als zmm0 bis zmm31 bezeichnet. Die 256 Bit niedrigerer Ordnung der unteren 16 zmm-Register werden den Registern ymm0-16 überlagert. Die 168 Bit niedrigerer Ordnung der niedrigeren 16 zmm-Register (die 168 Bit niedrigerer Ordnung der ymm-Register) werden den Registern xmm0-15 überlagert. Das spezifische vektorfreundliche Anweisungsformat **1600** operiert an diesem überlagerten Registerfile wie in den nachfolgenden Tabellen dargestellt.

Einstellbare Vektorlänge	Klasse	Operationen	Register
Anweisungsvorlagen, die das Vektorlängenfeld **1559B** nicht umfassen	A (Fig. 16A; U=0)	**1510**, **1515**, **1525**, **1530**	zmm-Register (die Vektorlänge ist 64 Byte)
	B (Fig. 16B; U=1)	**1516**	zmm-Register (die Vektorlänge ist 64 Byte)
Anweisungsvorlagen, die das Vektorlängenfeld **1559B** umfassen	B (Fig. 16B; U=1)	**1517**, **1527**	Register zmm, ymm oder xmm (die Vektorlänge ist 64 Byte, 32 Byte oder 16 Byte) abhängig vom Vektorlängenfeld **1559B**

[0155] Anders ausgedrückt wählt das Vektorlangenfeld **1559B** zwischen einer Maximallänge und einer oder mehreren anderen kürzeren Langen aus, wobei jede solche kürzere Länge die Hälfte der Länge der vorausgehenden Länge beträgt; und Anweisungsvorlagen ohne das Vektorlängenfeld **1559B** operieren an der Maximalvektorlänge. Ferner operieren bei einer Ausführungsform die Klasse-B-Anweisungsvorlagen des spezifischen vektorfreundlichen Anweisungsformats **1600** an gepackten oder skalaren Einzel-Doppelgenauigkeits-Gleitkommadaten und gepackten oder skalaren Integer-Daten. Skalare Operationen sind Operationen, die an der Datenelementposition niedrigster Ordnung in einem zmm/ymm/xmm-Register ausgeführt werden; die Datenelementpositionen höherer Ordnung werden abhängig von der Ausführungsform entweder so gelassen, wie sie vor der Anweisung waren, oder genullt.

[0156] Die Schreibmaskenregister **1715** – bei der dargestellten Ausführungsform gibt es 8 Schreibmaskenregister (k0 bis k7) jeweils mit einer Größe von 64 Bit. Bei einer alternativen Ausführungsform weisen die Schreibmaskenregister **1715** eine Größe von 16 Bit auf. Wie zuvor beschrieben, kann bei einer Ausführungsform der Erfindung das Vektormaskenregister k0 nicht als Schreibmaske verwendet werden; wenn die Codierung, die normalerweise k0 angeben würde, für eine Schreibmaske verwendet wird, wählt sie eine festverdrahtete Schreibmaske von 0xFFFF, wodurch effektiv die Schreibmaskierung für diese Anweisung deaktiviert wird.

[0157] Die Vielzweckregister **1725** – bei der dargestellten Ausführungsform gibt es 16 64-Bit-Vielzweckregister, die zusammen mit den existierenden x86-Adressierungsmodi zum Adressieren von Speicheroperanden verwendet werden. Diese Register werden mit den Namen RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP und R8 bis R15 bezeichnet.

[0158] Das Skalar-Gleitkomma-Stack-Registerfile (x87 Stack) **1745**, worauf das MMX-Packed-Integer-Flat-Registerfile **1750** zugeordnet wird – bei der dargestellten Ausführungsform ist der x87-Stapel ein Acht-Element-Stapel, mit dem skalare Gleitkommaoperationen an 32/64/80-Bit Gleitkommadaten unter Verwendung der x87-Anweisungssatzerweiterung ausgeführt werden, dagegen dienen die MMX-Register zum Ausführen von Operationen an 64-Bit-Packed-Integer-Daten sowie zum Halten von Operanden für bestimmte zwischen den MMX- und XMM-Registern ausgeführte Operationen.

[0159] Alternative Ausführungsformen der Erfindung können breitere oder schmälere Register verwenden. Zusätzlich können alternative Ausführungsformen der Erfindung mehr, weniger oder andere Registerfiles und Register benutzen.

[0160] In der obigen Beschreibung wurde die Erfindung mit Bezug auf spezifische beispielhafte Ausführungsformen davon beschrieben. Es ist jedoch offensichtlich, dass verschiedene Modifikationen und Änderungen daran vorgenommen werden können, ohne von dem allgemeineren Gedanken und Schutzumfang der Erfindung, so wie er in den angefügten Ansprüchen dargelegt wird, abzuweichen. Die Beschreibung und Zeichnungen sind dementsprechend im veranschaulichenden und nicht im einschränkenden Sinne zu betrachten.

[0161] Ausführungsformen der Erfindung können verschiedene Schritte umfassen, die oben beschrieben wurden. Die Schritte können in maschinenausführbaren Anweisungen realisiert werden, die benutzt werden können, um zu bewirken, dass ein Vielzweck- oder Spezialprozessor die Schritte ausführt. Als Alternative können die Schritte durch spezifische Hardwarekomponenten ausgeführt werden, die festverdrahtete Logik zum Ausführen der Schritte enthalten, oder durch eine beliebige Kombination von programmierten Computerkomponenten und kundenspezifischen Hardwarekomponenten.

[0162] Wie hier beschrieben, können sich Anweisungen auf spezifische Konfigurationen von Hardware, wie etwa anwendungsspezifische integrierte Schaltungen (ASIC), die dafür ausgelegt sind bestimmte Operationen auszuführen, oder die eine vorbestimmte Funktionalität aufweisen, oder auf Softwareanweisungen, die in Speicher, der in einem nichtflüchtigen computerlesbaren Medium realisiert wird, gespeichert sind, beziehen. Die in den Figuren gezeigten Techniken können somit unter Verwendung von Code und Daten implementiert werden, die auf einer oder mehreren elektronischen Vorrichtungen (z. B. einer Endstation, einem Netzwerkelement usw.) gespeichert und ausgeführt werden. Solche elektronischen Vorrichtungen speichern und übermitteln (intern und/oder mit anderen elektronischen Vorrichtungen über ein Netzwerk) Code und Daten unter Verwendung von computermaschinenlesbaren Medien, wie etwa nichtflüchtigen computermaschinenlesbaren Speichermedien (z. B. magnetischen Datenträgern; optischen Datenträgern; Direktzugriffsspeicher, Nurlesespeicher; Flash-Speichervorrichtungen; Phasenänderungsspeichern) und flüchtigen computermaschinenlesbaren Kommunikationsmedien (z. B. elektrischen, optischen, akustischen oder andersartigen propagierten Signalen – wie etwa Trägerwellen, Infrarotsignalen, Digitalsignalen usw.). Außerdem umfassen solche elektronischen Vorrichtungen typischerweise eine Menge von einem oder mehreren Prozessoren, die mit einer oder mehreren anderen Komponenten, wie etwa einer oder mehreren Speichervorrichtungen (nichtflüchtigen maschinenlesbaren Speichermedien), Benutzereingabe-/-ausgabevorrichtungen (z. B. einer Tastatur, einem Berührungsschirm und/oder einem Display) und Netzwerkverbindungen, gekoppelt sind. Die Kopplung der Menge von Prozessoren und anderen Komponenten erfolgt typischerweise mittels eines oder mehrerer Busse und Brücken (die auch als Bussteuerungen bezeichnet werden). Die Speichervorrichtung und Signale, die den Netzwerkverkehr führen, repräsentieren jeweils ein oder mehrere maschinenlesbare Speichermedien und maschinenlesbare Kommunikationsmedien. Die Speichervorrichtung einer gegebenen elektronischen Vorrichtung speichert somit typischerweise Code und/oder Daten zur Ausführung auf der Menge von einem oder mehreren Prozessoren dieser elektronischen Vorrichtung. Natürlich können ein oder mehrere Teile einer Ausführungsform der Erfindung unter Verwendung verschiedener Kombinationen von Software, Firmware und/oder Hardware implementiert werden. Im Verlauf dieser ausführlichen Beschreibung wurden zur Erläuterung zahlreiche spezifische Einzelheiten dargelegt, um ein umfassendes Verständnis der vorliegenden Erfindung zu gewährleisten. Für Fachleute ist jedoch erkennbar, dass die Erfindung ohne bestimmte dieser spezifischen Einzelheiten ausgeübt werden kann. In bestimmten Fällen wurden wohlbekannte Strukturen und Funktionen nicht ausführlich beschrieben, um eine Verschleierung des Gegenstands der vorliegenden Erfindung zu vermeiden. Der Umfang und Gedanke der Erfindung sollte dementsprechend über die nachfolgenden Ansprüche entschieden werden.

Patentansprüche

1. Prozessor, umfassend:
eine Anweisungsabrufeinheit zum Abrufen einer Doppelmultiplikationsanweisung aus einem Speichersubsystem, wobei die Doppelmultiplikationsanweisung drei Quellenoperandenwerte aufweist;
eine Decodiereinheit zum Decodieren der Doppelmultiplikationsanweisung, um mindestens eine uop zu erzeugen; und
eine Ausführungseinheit zum Ausführen der uop ein erstes Mal, um einen ersten und einen zweiten der drei Quellenoperandenwerte zu multiplizieren, um ein erstes Zwischenergebnis zu erzeugen, und zum Ausführen des uop ein zweites Mal, um das Zwischenergebnis mit einem dritten der drei Quellenoperandenwerte zu multiplizieren, um ein Endergebnis zu erzeugen.

2. Prozessor nach Anspruch 1, wobei die Ausführungseinheit einen Verzögerungspuffer zum Verzögern der uop vor dem zweitmaligen Ausführen der uop umfasst.

3. Prozessor nach Anspruch 1 oder 2, wobei die Ausführungseinheit ferner Folgendes umfasst:
eine Reservierungsstation zum Einteilen der Doppelmultiplikationsanweisung zur Ausführung durch mindestens eine Funktionseinheit, wobei die uop von der Reservierungsstation zu einer ersten Funktionseinheit zu übertragen ist und außerdem vor der Ausführung durch eine Funktionseinheit dem Verzögerungspuffer zugeführt wird.

4. Prozessor nach Anspruch 3, wobei die Funktionseinheit eine fusionierte Multiplizier- und Addierfunktionseinheit umfasst.

5. Prozessor nach Anspruch 3 oder 4, wobei die uop ferner zu einem Zeitpunkt, an dem die erste Funktionseinheit eine erste Ausführung der uop abgeschlossen und das Zwischenergebnis erzeugt hat, von dem Verzögerungspuffer zu einer zweiten Funktionseinheit zu übertragen ist, wobei die zweite Funktionseinheit das Zwischenergebnis mit dem dritten der drei Quellenoperandenwerte multipliziert, um das Endergebnis zu erzeugen.

6. Prozessor nach Anspruch 5, wobei das Endergebnis erzeugt wird, wenn eine einzelne uop aus einer einzelnen Doppelmultiplikationsanweisung zweimal der Reihe nach ausgeführt wird.

7. Prozessor nach einem der vorhergehenden Ansprüche, wobei der erste, zweite und dritte Quellenoperand der Doppelmultiplikationsanweisung Gleitkommawerte sind.

8. Prozessor nach Anspruch 7, wobei die Gleitkommawerte Einzelgenauigkeits- oder Doppelgenauigkeits-Gleitkommawerte umfassen.

9. Prozessor nach einem der vorhergehenden Ansprüche, wobei die Doppelmultiplikationsanweisung einen Unmittelbarwert zur Angabe eines Vorzeichens jeweils für den ersten Quellenoperanden, den zweiten Quellenoperanden und den dritten Quellenoperanden umfasst.

10. Prozessor nach Anspruch 9, wobei der Unmittelbarwert einen Drei-Bit-Wert umfasst, wobei der Wert jedes Bit ein Vorzeichen für den ersten Quellenoperanden, den zweiten Quellenoperanden und den dritten Quellenoperanden angibt.

11. Prozessor nach einem der Ansprüche 3 bis 10, wobei die Reservierungsstation eine erste Reservierungsstationspartition zum Einteilen der ersten Ausführung der uop über einen ersten Ausführungsport und eine zweite Reservierungsstationspartition zum Einteilen der zweiten Ausführung der uop über einen zweiten Ausführungsport umfasst.

12. Verfahren, umfassend:
Abrufen einer Doppelmultiplikationsanweisung aus einem Speichersubsystem, wobei die Doppelmultiplikationsanweisung drei Quellenoperandenwerte aufweist;
Decodieren der Doppelmultiplikationsanweisung, um mindestens eine uop zu erzeugen; und
Ausführen der uop ein erstes Mal, um einen ersten und einen zweiten der drei Quellenoperandenwerte zu multiplizieren, um ein erstes Zwischenergebnis zu erzeugen, und Ausführen der uop ein zweites Mal, um das Zwischenergebnis mit einem dritten der drei Quellenoperandenwerte zu multiplizieren, um ein Endergebnis zu erzeugen.

13. Verfahren nach Anspruch 12, das ferner Verzögern der uop in einem Verzögerungspuffer vor der zweitmaligen Ausführung der uop umfasst.

14. Verfahren nach Anspruch 12 oder 13, ferner umfassend:
Einteilen der Doppelmultiplikationsanweisung zur Ausführung durch mindestens eine Funktionseinheit, wobei die uop zu einer ersten Funktionseinheit zu übertragen ist und außerdem vor der Ausführung durch eine Funktionseinheit dem Verzögerungspuffer zugeführt wird.

15. Verfahren nach Anspruch 14, wobei die Funktionseinheit eine fusionierte Multiplizier- und Addierfunktionseinheit umfasst.

16. Verfahren nach Anspruch 14 oder 15, wobei die uop ferner zu einem Zeitpunkt, an dem die erste Funktionseinheit eine erste Ausführung der uop abgeschlossen und das Zwischenergebnis erzeugt hat, von dem Verzögerungspuffer zu einer zweiten Funktionseinheit zu übertragen ist, wobei die zweite Funktionseinheit das Zwischenergebnis mit dem dritten der drei Quellenoperandenwerte multipliziert, um das Endergebnis zu erzeugen.

17. Verfahren nach Anspruch 16, wobei das Endergebnis erzeugt wird, wenn eine einzelne uop aus einer einzelnen Doppelmultiplikationsanweisung zweimal der Reihe nach ausgeführt wird.

18. Verfahren nach einem der Ansprüche 12 bis 17, wobei der erste, zweite und dritte Quellenoperand der Doppelmultiplikationsanweisung Gleitkommawerte sind.

19. Verfahren nach Anspruch 18, wobei die Gleitkommawerte Einzelgenauigkeits- oder Doppelgenauigkeits-Gleitkommawerte umfassen.

20. Verfahren nach einem der Ansprüche 12 bis 19, wobei die Doppelmultiplikationsanweisung einen Unmittelbarwert zur Angabe eines Vorzeichens jeweils für den ersten Quellenoperanden, den zweiten Quellenoperanden und den dritten Quellenoperanden umfasst.

21. Verfahren nach Anspruch 20, wobei der Unmittelbarwert einen Drei-Bit-Wert umfasst, wobei der Wert jedes Bit ein Vorzeichen für den ersten Quellenoperanden, den zweiten Quellenoperanden und den dritten Quellenoperanden angibt.

22. Verfahren nach einem der Ansprüche 14 bis 21, wobei das Einteilen von einer Reservierungsstation ausgeführt wird, die eine erste Reservierungsstationspartition zum Einteilen der ersten Ausführung der uop über einen ersten Ausführungsport und eine zweite Reservierungsstationspartition zum Einteilen der zweiten Ausführung der uop über einen zweiten Ausführungsport umfasst.

Es folgen 16 Seiten Zeichnungen

Anhängende Zeichnungen

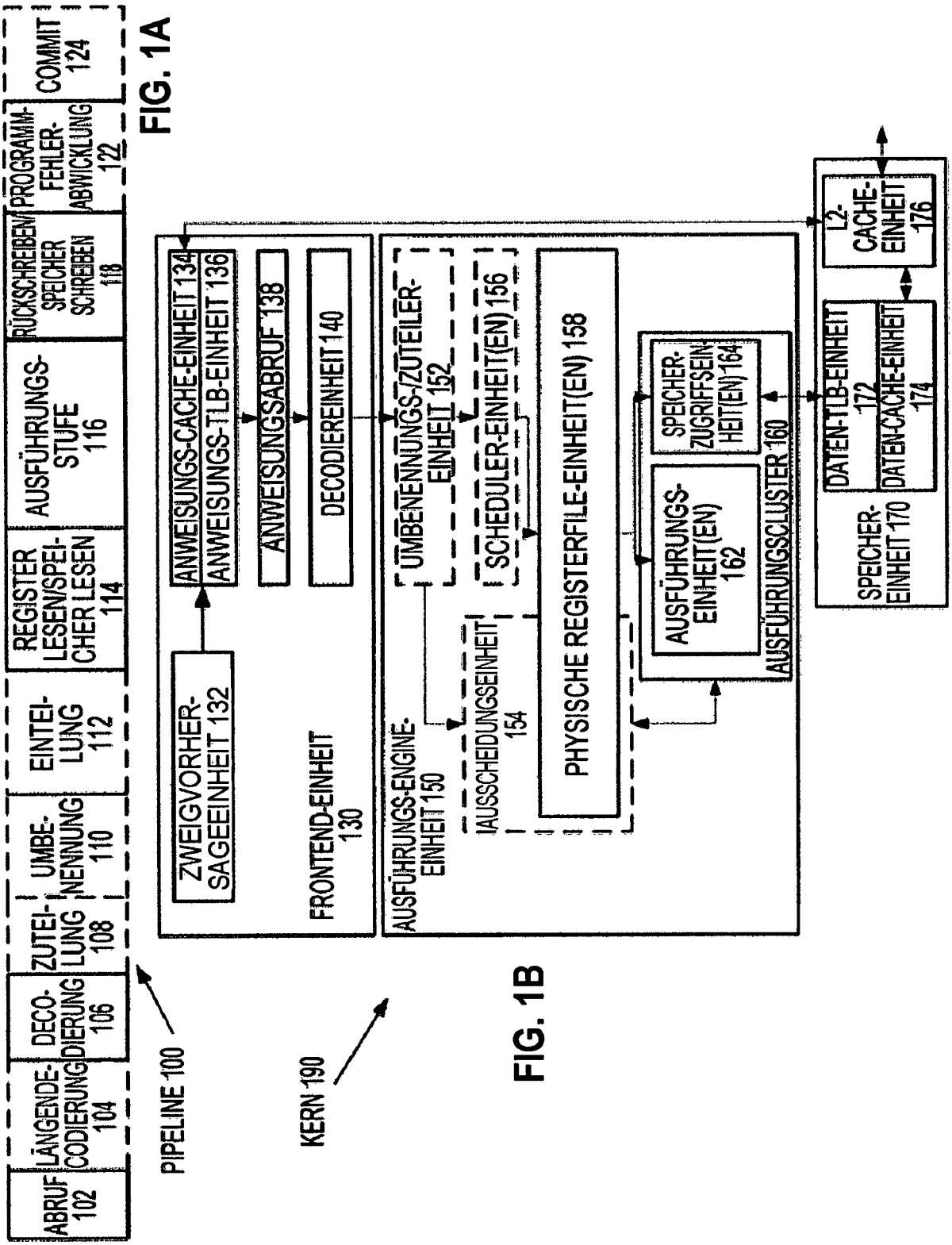


FIG. 1A

FIG. 1B

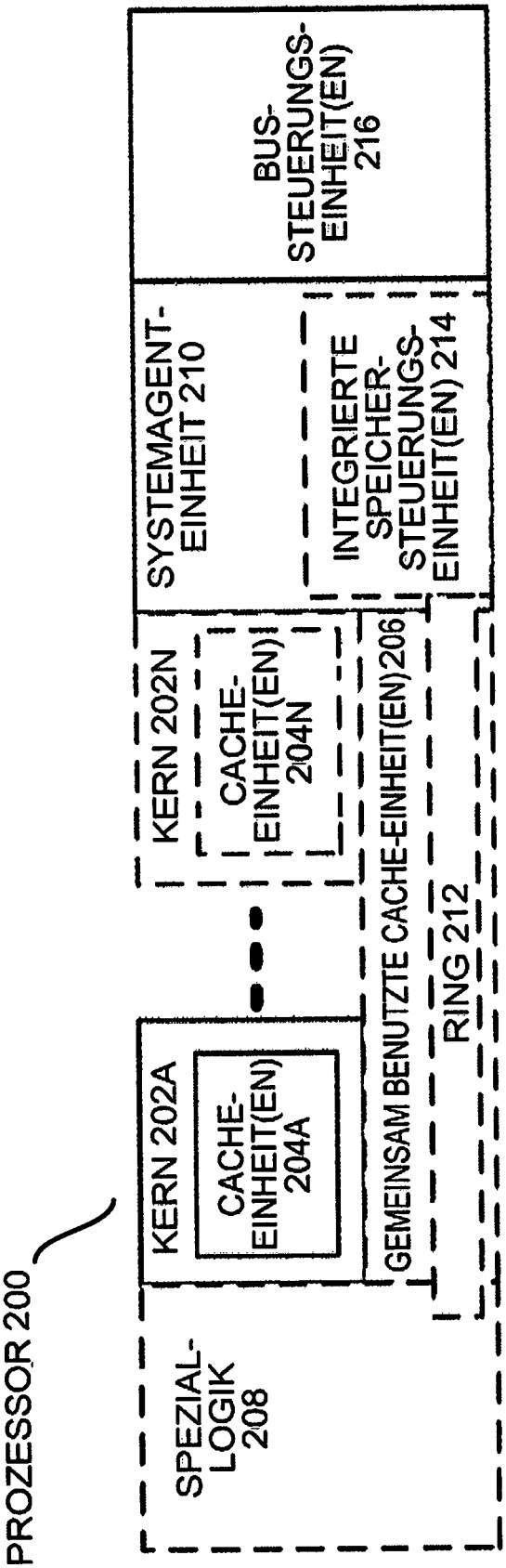

PROZESSOR 200
SPEZIAL-LOGIK 208
KERN 202A
CACHE-EINHEIT(EN) 204A
KERN 202N
CACHE-EINHEIT(EN) 204N
GEMEINSAM BENUTZTE CACHE-EINHEIT(EN) 206
RING 212
SYSTEMAGENT-EINHEIT 210
INTEGRIERTE SPEICHER-STEUERUNGS-EINHEIT(EN) 214
BUS-STEUERUNGS-EINHEIT(EN) 216

FIG. 2

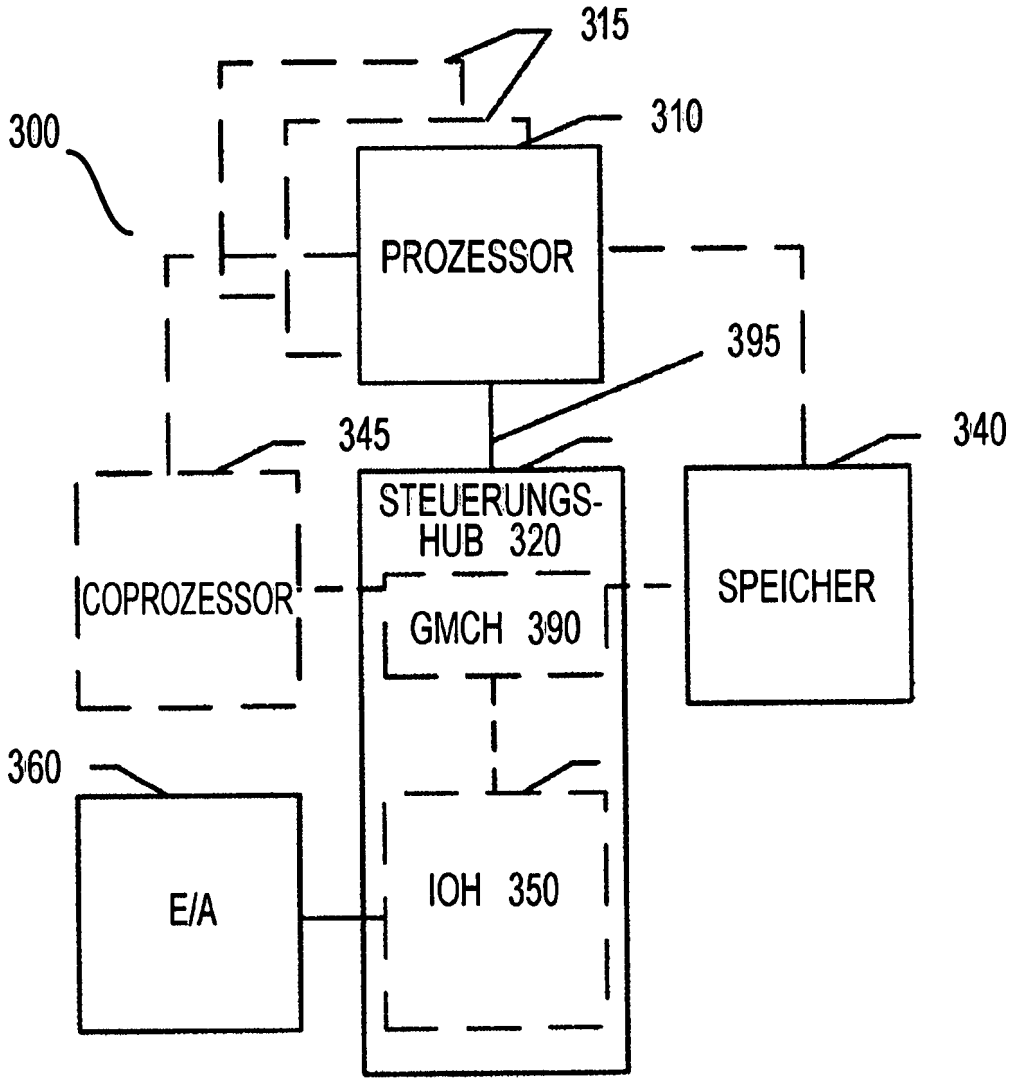

315
310
300
PROZESSOR
395
345
340
STEUERUNGS-
HUB 320
COPROZESSOR
GMCH 390
SPEICHER
360
E/A
IOH 350

FIG. 3

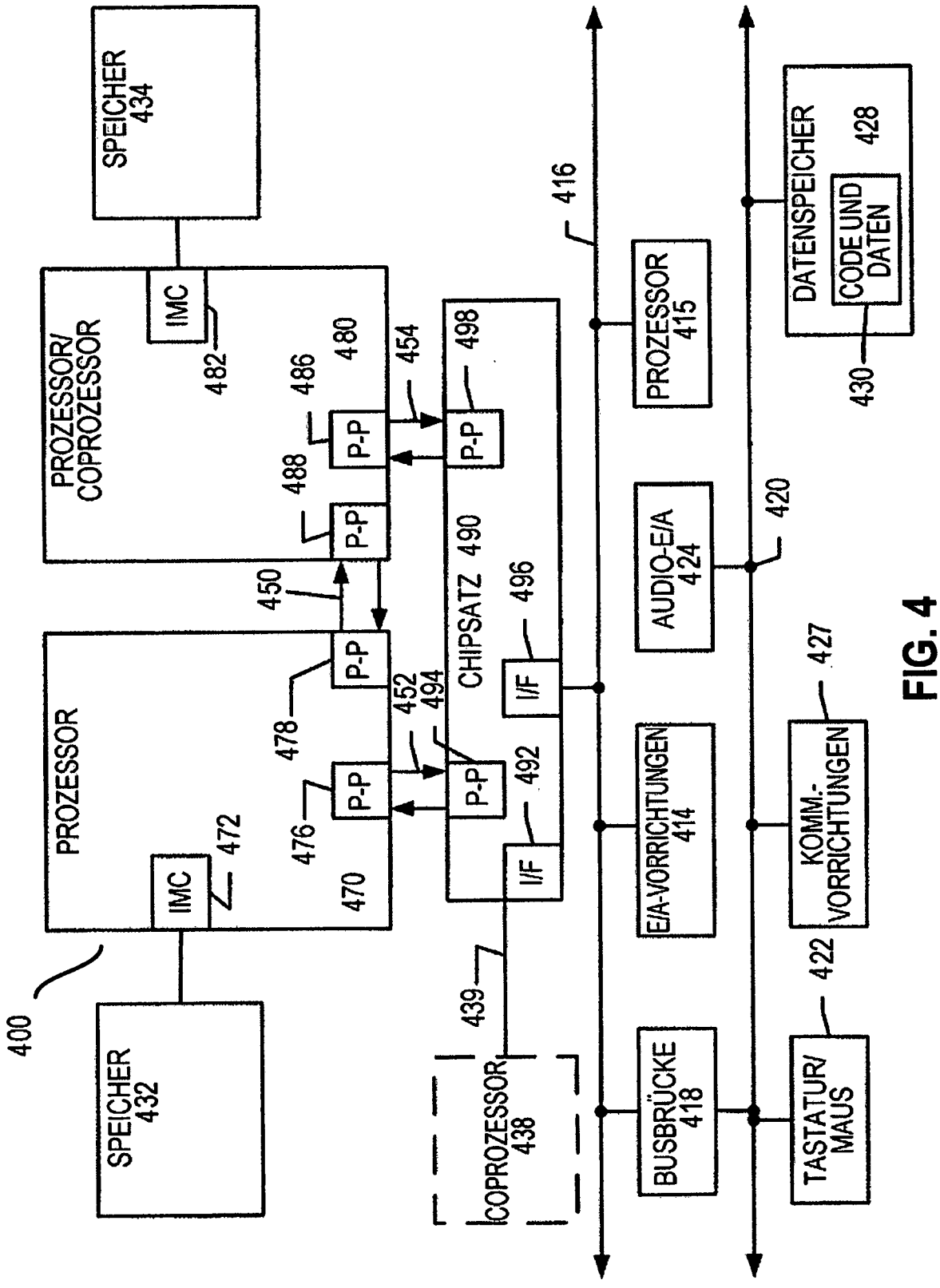

400
SPEICHER 432
PROZESSOR
IMC
472
470
476
478
P-P
452
494
P-P
492
I/F
439
COPROZESSOR 438
450
PROZESSOR/ COPROZESSOR
IMC
482
480
486
488
P-P
454
P-P
498
CHIPSATZ 490
496
I/F
SPEICHER 434
416
PROZESSOR 415
AUDIO-E/A 424
420
E/A-VORRICHTUNGEN 414
KOMM.-VORRICHTUNGEN
427
BUSBRÜCKE 418
TASTATUR/ MAUS
422
DATENSPEICHER
CODE UND DATEN
428
430

FIG. 4

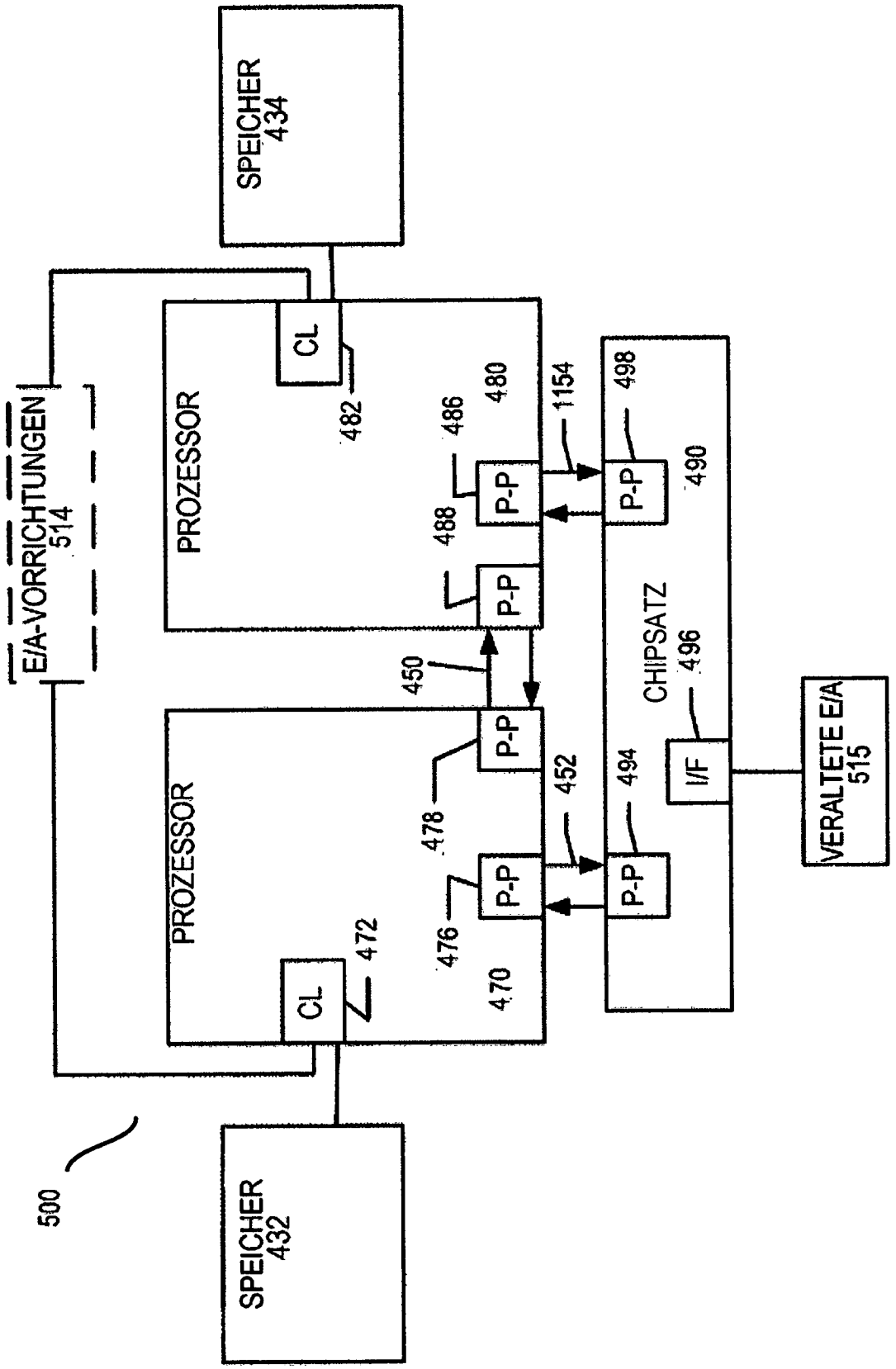

SPEICHER
434
E/A-VORRICHTUNGEN
514
PROZESSOR
CL
482
480
486
488
P-P
P-P
1154
498
P-P
490
CHIPSATZ
496
I/F
VERALTETE E/A
515
450
PROZESSOR
P-P
478
452
494
P-P
476
P-P
470
472
CL
500
SPEICHER
432

FIG. 5

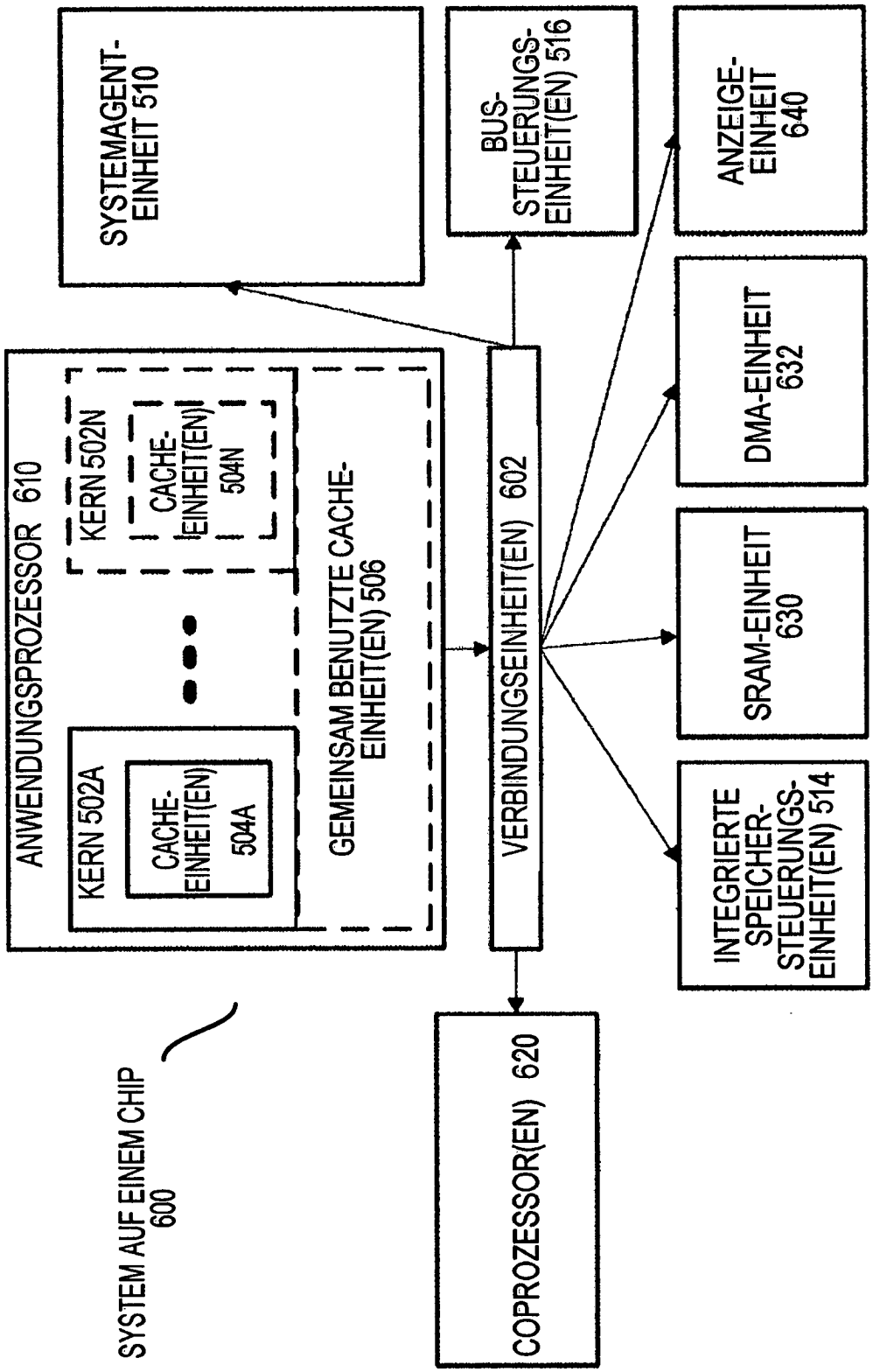

SYSTEM AUF EINEM CHIP 600
ANWENDUNGSPROZESSOR 610
KERN 502A
CACHE-EINHEIT(EN) 504A
KERN 502N
CACHE-EINHEIT(EN) 504N
GEMEINSAM BENUTZTE CACHE-EINHEIT(EN) 506
SYSTEMAGENT-EINHEIT 510
BUS-STEUERUNGS-EINHEIT(EN) 516
VERBINDUNGSEINHEIT(EN) 602
ANZEIGE-EINHEIT 640
DMA-EINHEIT 632
SRAM-EINHEIT 630
INTEGRIERTE SPEICHER-STEUERUNGS-EINHEIT(EN) 514
COPROZESSOR(EN) 620

FIG. 6

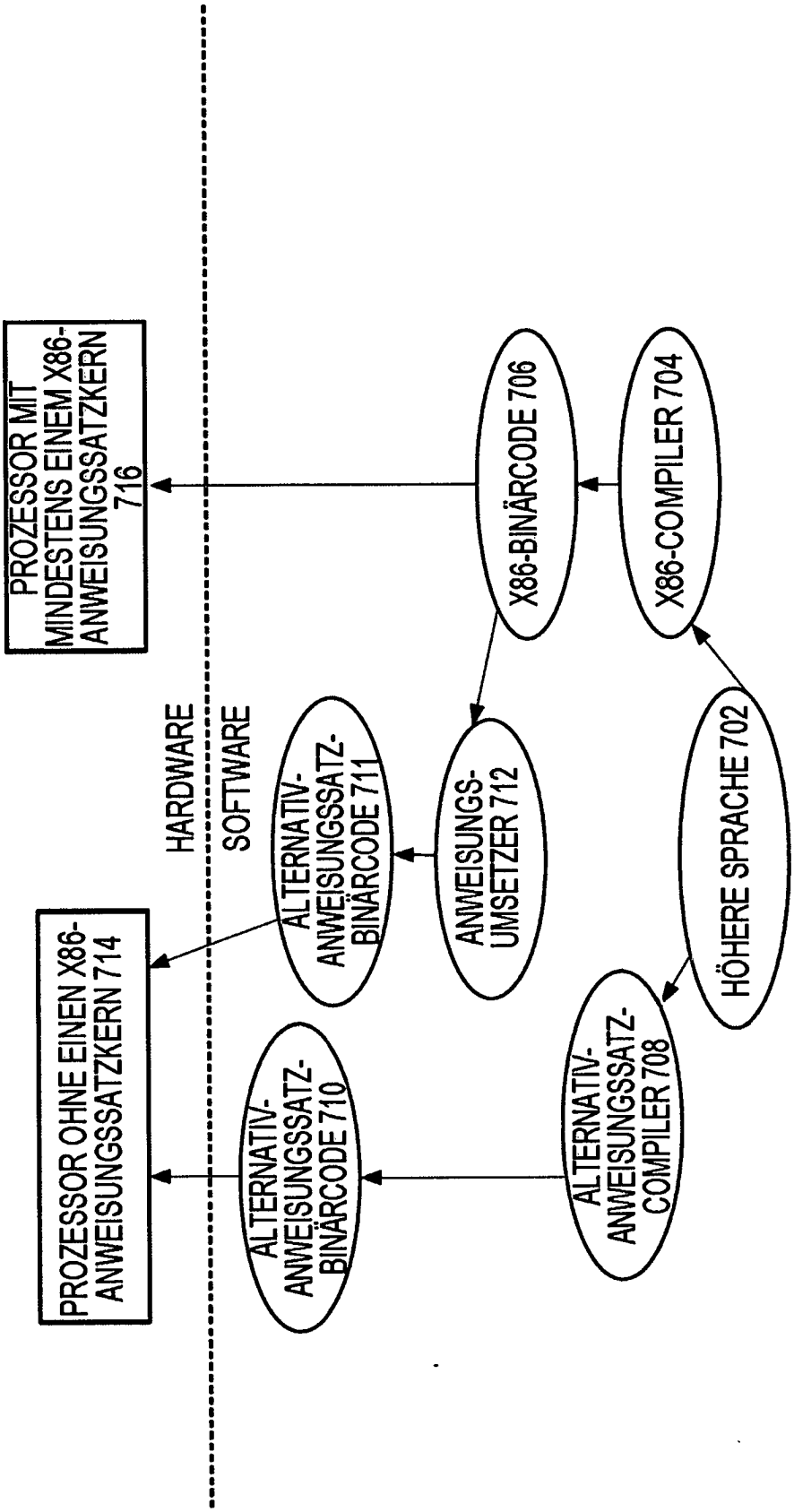

PROZESSOR MIT MINDESTENS EINEM X86-ANWEISUNGSSATZKERN 716
X86-BINÄRCODE 706
X86-COMPILER 704
HARDWARE
SOFTWARE
ALTERNATIV-ANWEISUNGSSATZ-BINÄRCODE 711
ANWEISUNGS-UMSETZER 712
HÖHERE SPRACHE 702
PROZESSOR OHNE EINEN X86-ANWEISUNGSSATZKERN 714
ALTERNATIV-ANWEISUNGSSATZ-BINÄRCODE 710
ALTERNATIV-ANWEISUNGSSATZ-COMPILER 708

FIG. 7

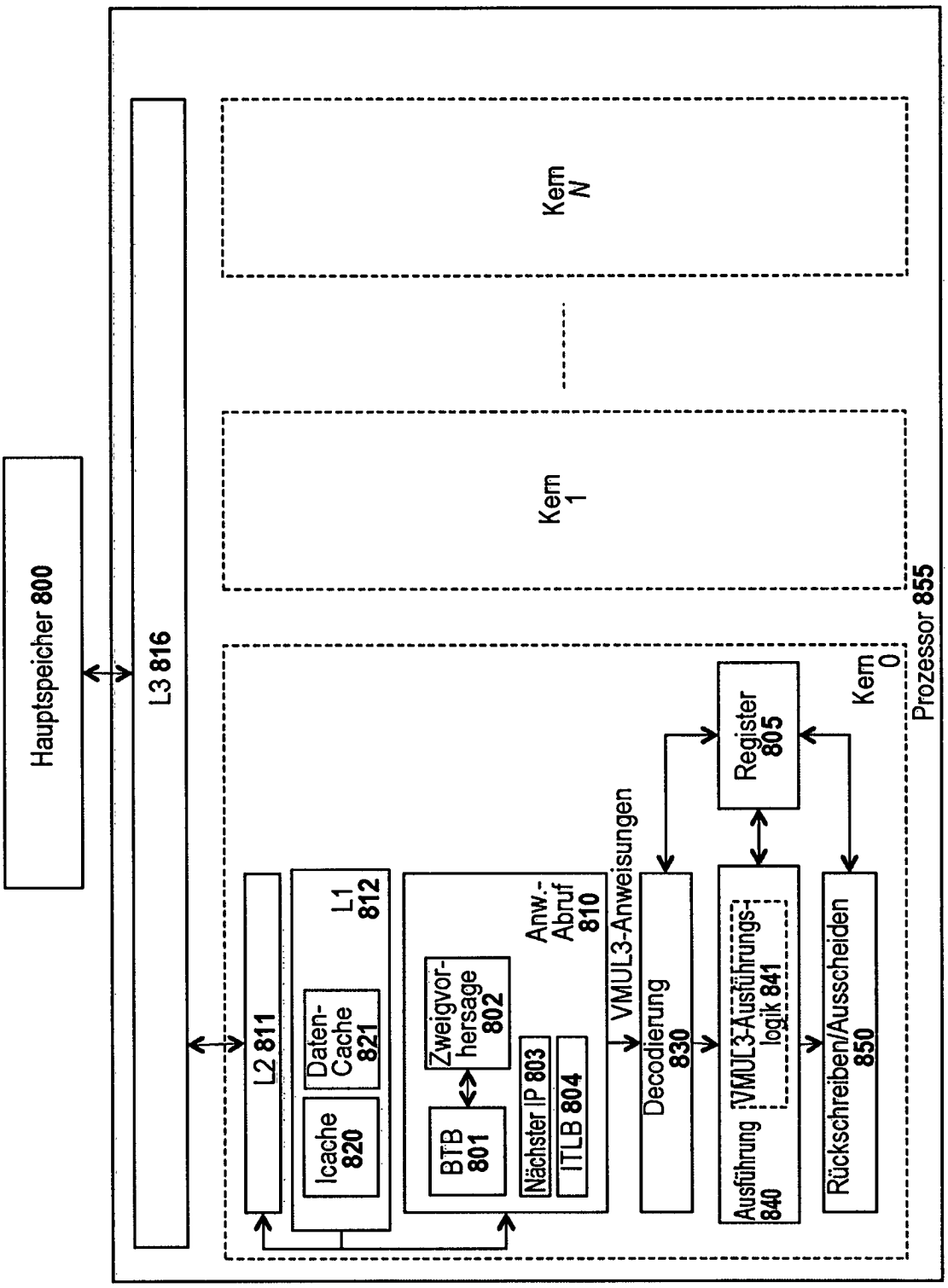

Hauptspeicher 800
L3 816
Kern N
Kern 1
Kern 0
Prozessor 855
L2 811
L1 812
Daten-Cache 821
Icache 820
Zweigvor-hersage 802
BTB 801
Nächster IP 803
ITLB 804
Anw.-Abruf 810
VMUL3-Anweisungen
Decodierung 830
Register 805
Ausführung 840
VMUL3-Ausführungs-logik 841
Rückschreiben/Ausscheiden 850

Fig. 8

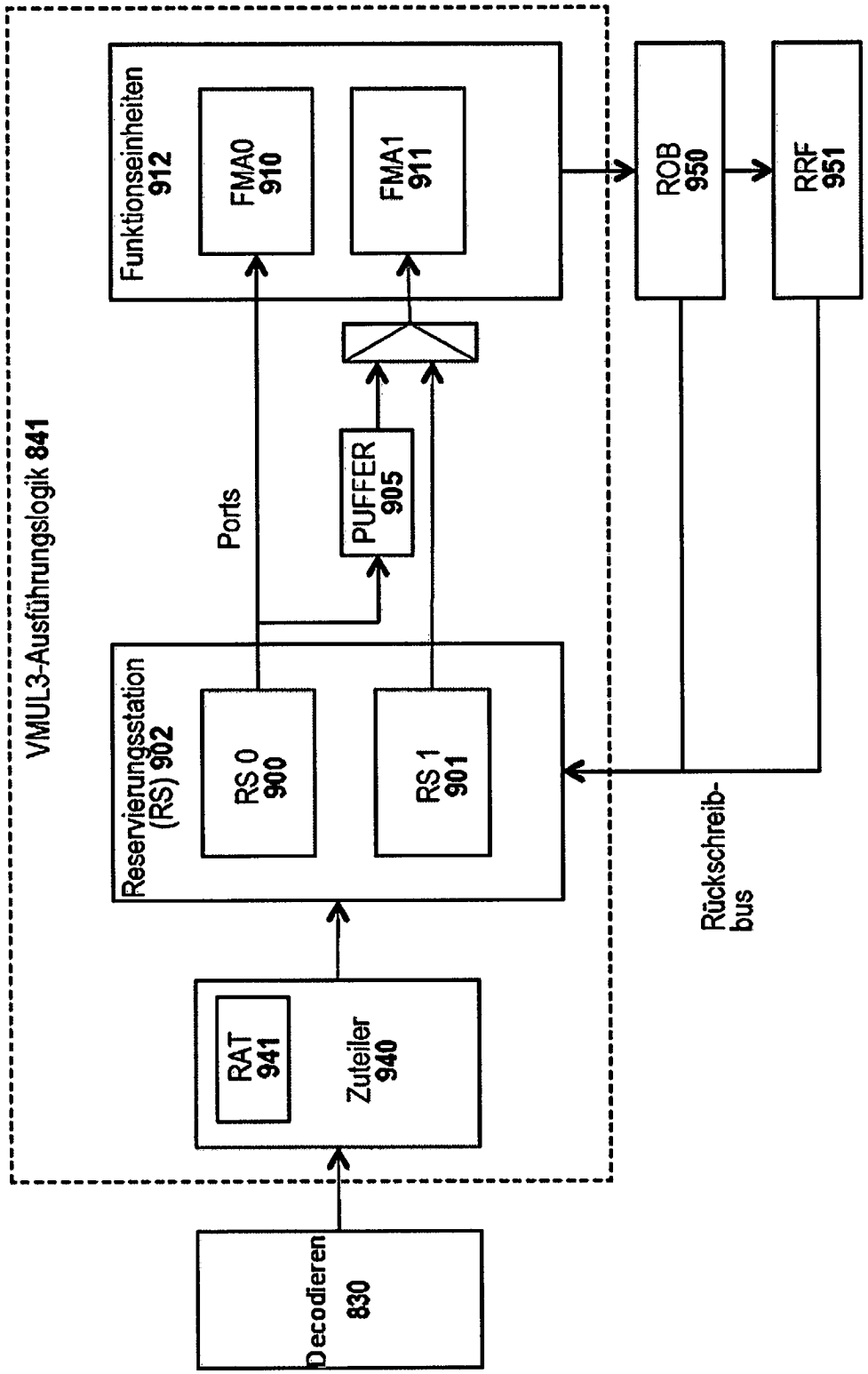

VMUL3-Ausführungslogik 841
Funktionseinheiten 912
FMA0 910
FMA1 911
ROB 950
RRF 951
Ports
PUFFER 905
Reservierungsstation (RS) 902
RS 0 900
RS 1 901
Rückschreib-bus
RAT 941
Zuteiler 940
Decodieren 830

Fig. 9A

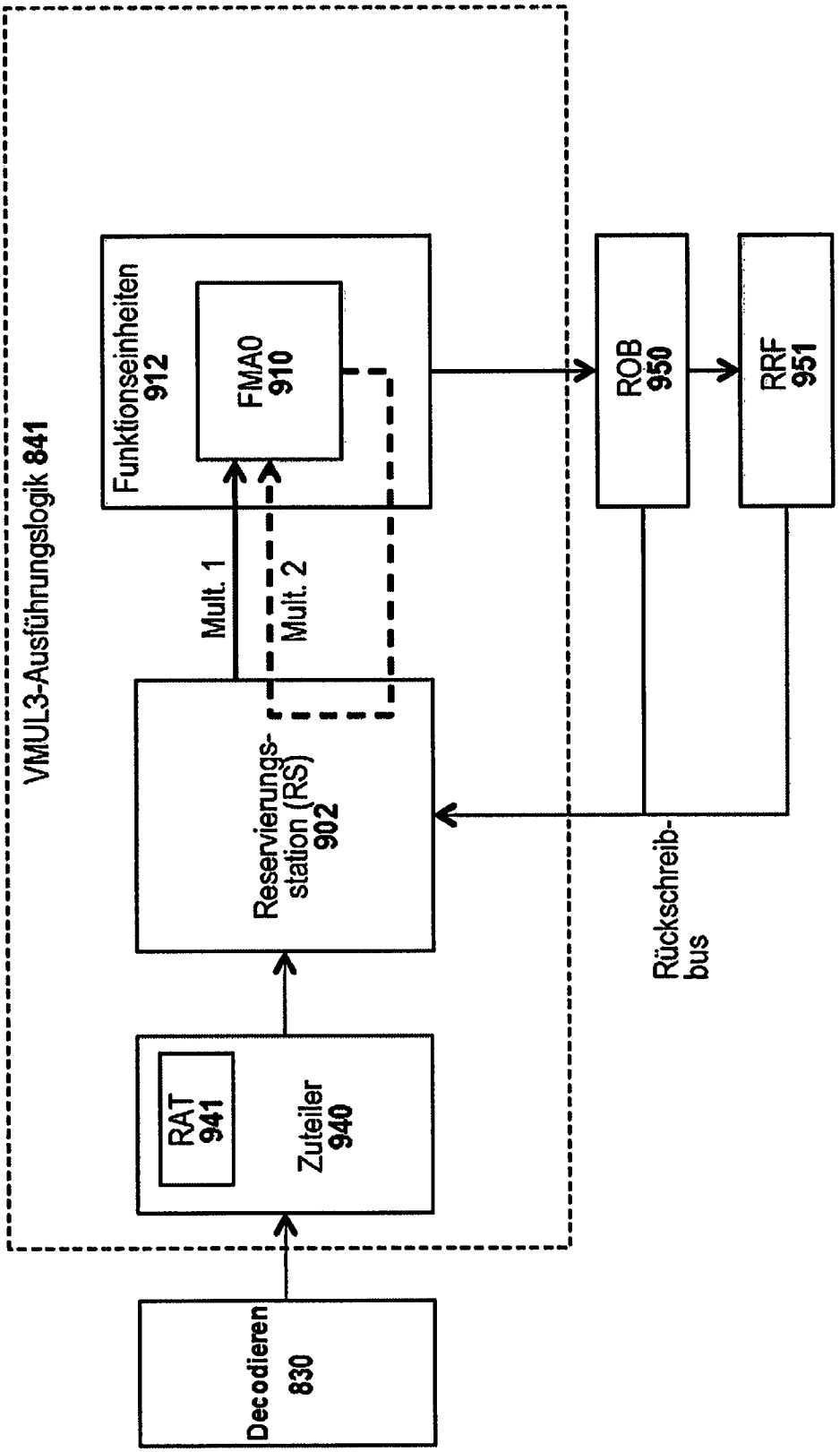

VMUL3-Ausführungslogik 841
Funktionseinheiten 912
FMA0 910
ROB 950
RRF 951
Mult. 1
Mult. 2
Reservierungs-station (RS) 902
Rückschreib-bus
RAT 941
Zuteiler 940
Decodieren 830

Fig. 9B

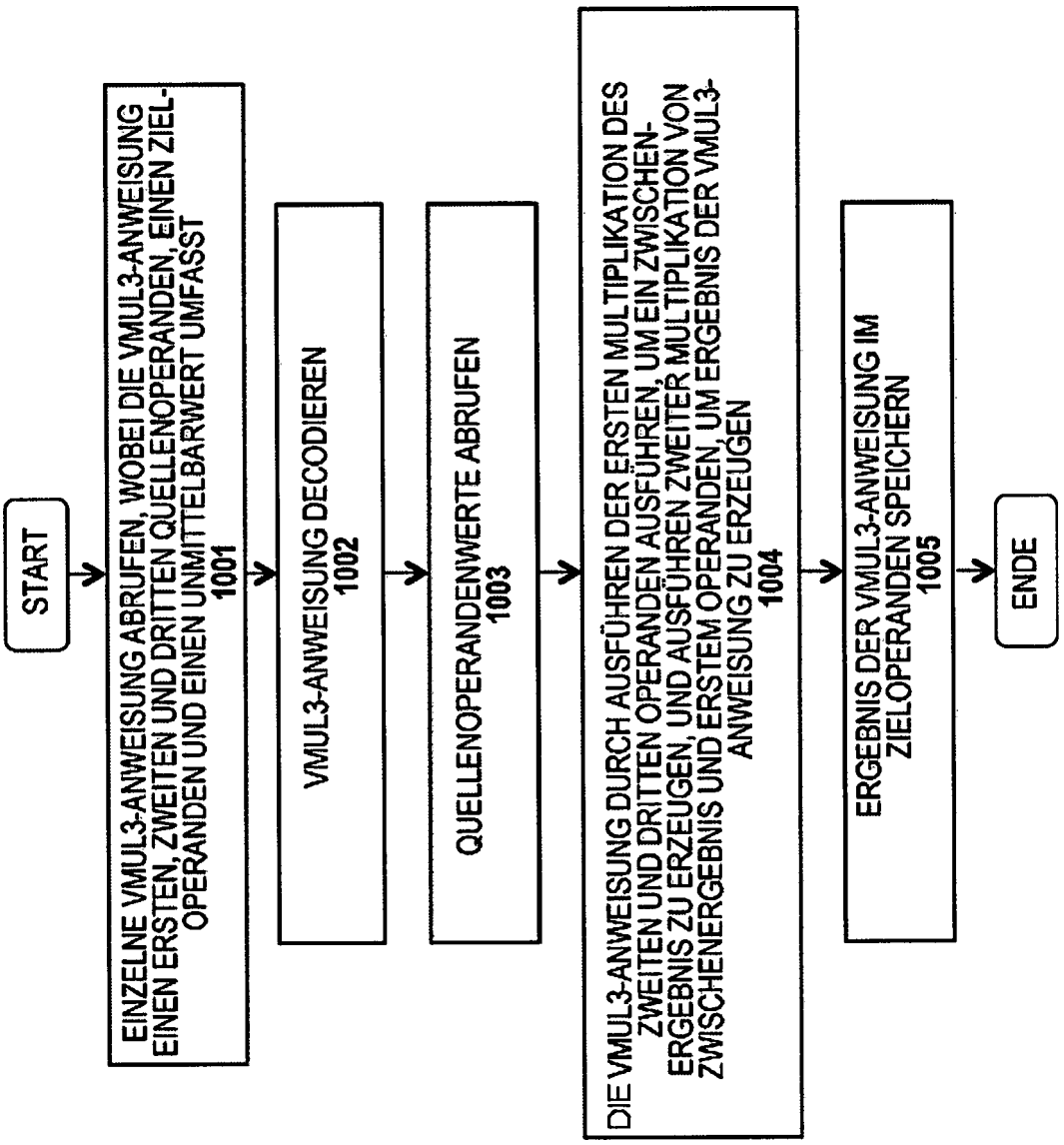

START
EINZELNE VMUL3-ANWEISUNG ABRUFEN, WOBEI DIE VMUL3-ANWEISUNG EINEN ERSTEN, ZWEITEN UND DRITTEN QUELLENOPERANDEN, EINEN ZIEL-OPERANDEN UND EINEN UNMITTELBARWERT UMFASST
1001
VMUL3-ANWEISUNG DECODIEREN
1002
QUELLENOPERANDENWERTE ABRUFEN
1003
DIE VMUL3-ANWEISUNG DURCH AUSFÜHREN DER ERSTEN MULTIPLIKATION DES ZWEITEN UND DRITTEN OPERANDEN AUSFÜHREN, UM EIN ZWISCHEN-ERGEBNIS ZU ERZEUGEN, UND AUSFÜHREN ZWEITER MULTIPLIKATION VON ZWISCHENERGEBNIS UND ERSTEM OPERANDEN, UM ERGEBNIS DER VMUL3-ANWEISUNG ZU ERZEUGEN
1004
ERGEBNIS DER VMUL3-ANWEISUNG IM ZIELOPERANDEN SPEICHERN
1005
ENDE

Fig. 10

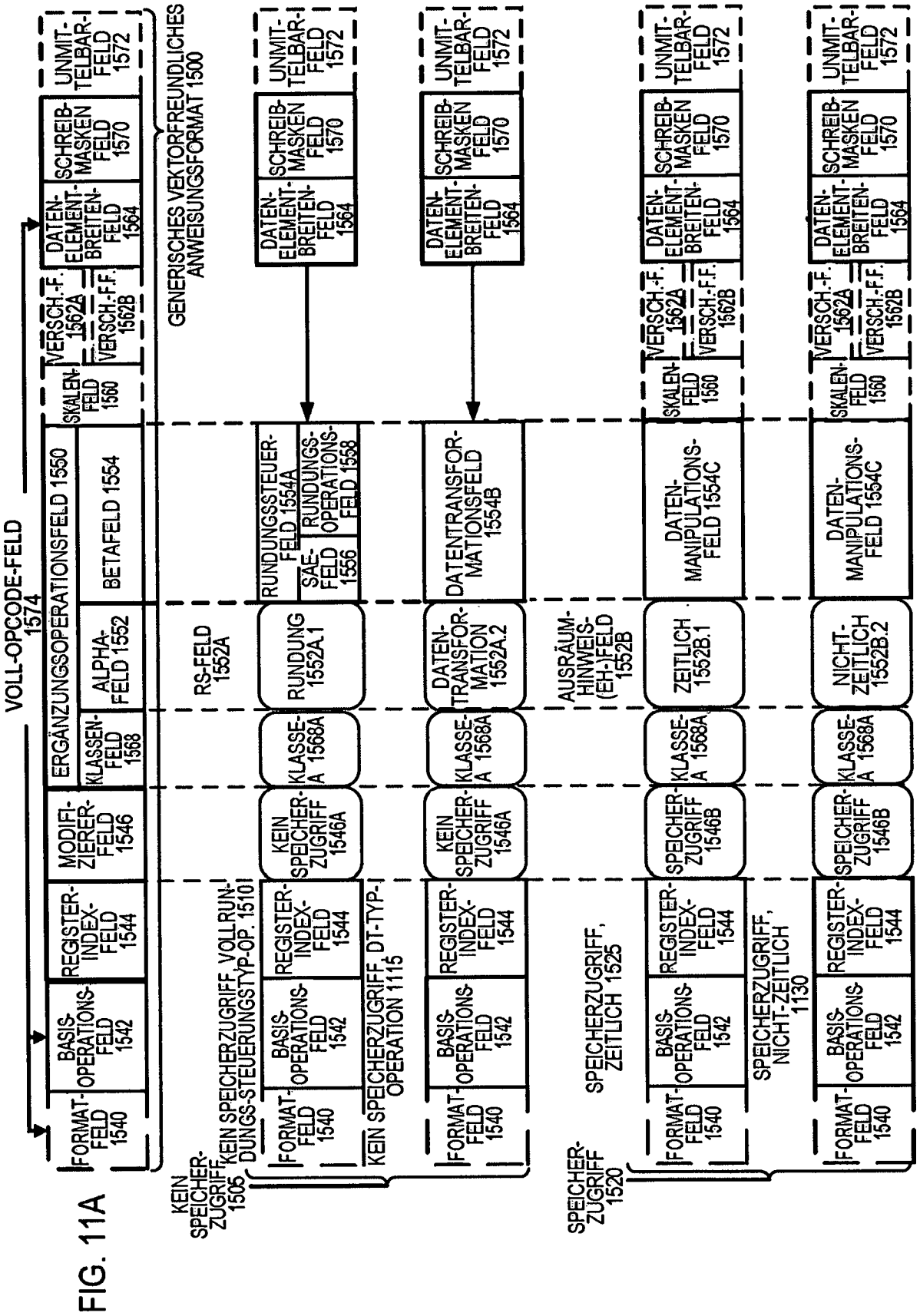

VOLL-OPCODE-FELD 1574
GENERISCHES VEKTORFREUNDLICHES ANWEISUNGSFORMAT 1500
FORMAT-FELD 1540
BASIS-OPERATIONS-FELD 1542
REGISTER-INDEX-FELD 1544
MODIFI-ZIERER-FELD 1546
ERGÄNZUNGSOPERATIONSFELD 1550
KLASSEN-FELD 1568
ALPHA-FELD 1552
BETAFELD 1554
SKALEN-FELD 1560
VERSCH.-F. 1562A
VERSCH.-F.F. 1562B
DATEN-ELEMENT-BREITEN-FELD 1564
SCHREIB-MASKEN FELD 1570
UNMIT-TELBAR-FELD 1572
RS-FELD 1552A
RUNDUNGSSTEUER-FELD 1554A
SAE-FELD 1556
RUNDUNGS-OPERATIONS-FELD 1558
RUNDUNG 1552A.1
DATEN-TRANSFOR-MATION 1552A.2
DATENTRANSFOR-MATIONSFELD 1554B
AUSRÄUM-HINWEIS-(EH-)FELD 1552B
ZEITLICH 1552B.1
NICHT-ZEITLICH 1552B.2
DATEN-MANIPULATIONS-FELD 1554C
KLASSE-A 1568A
KEIN SPEICHER-ZUGRIFF 1546A
SPEICHER-ZUGRIFF 1546B
KEIN SPEICHERZUGRIFF 1505
KEIN SPEICHERZUGRIFF, VOLLRUNDUNGS-STEUERUNGSTYP-OP. 1510
KEIN SPEICHERZUGRIFF, DT-TYP-OPERATION 1115
SPEICHERZUGRIFF 1520
SPEICHERZUGRIFF, ZEITLICH 1525
SPEICHERZUGRIFF, NICHT-ZEITLICH 1130

FIG. 11A

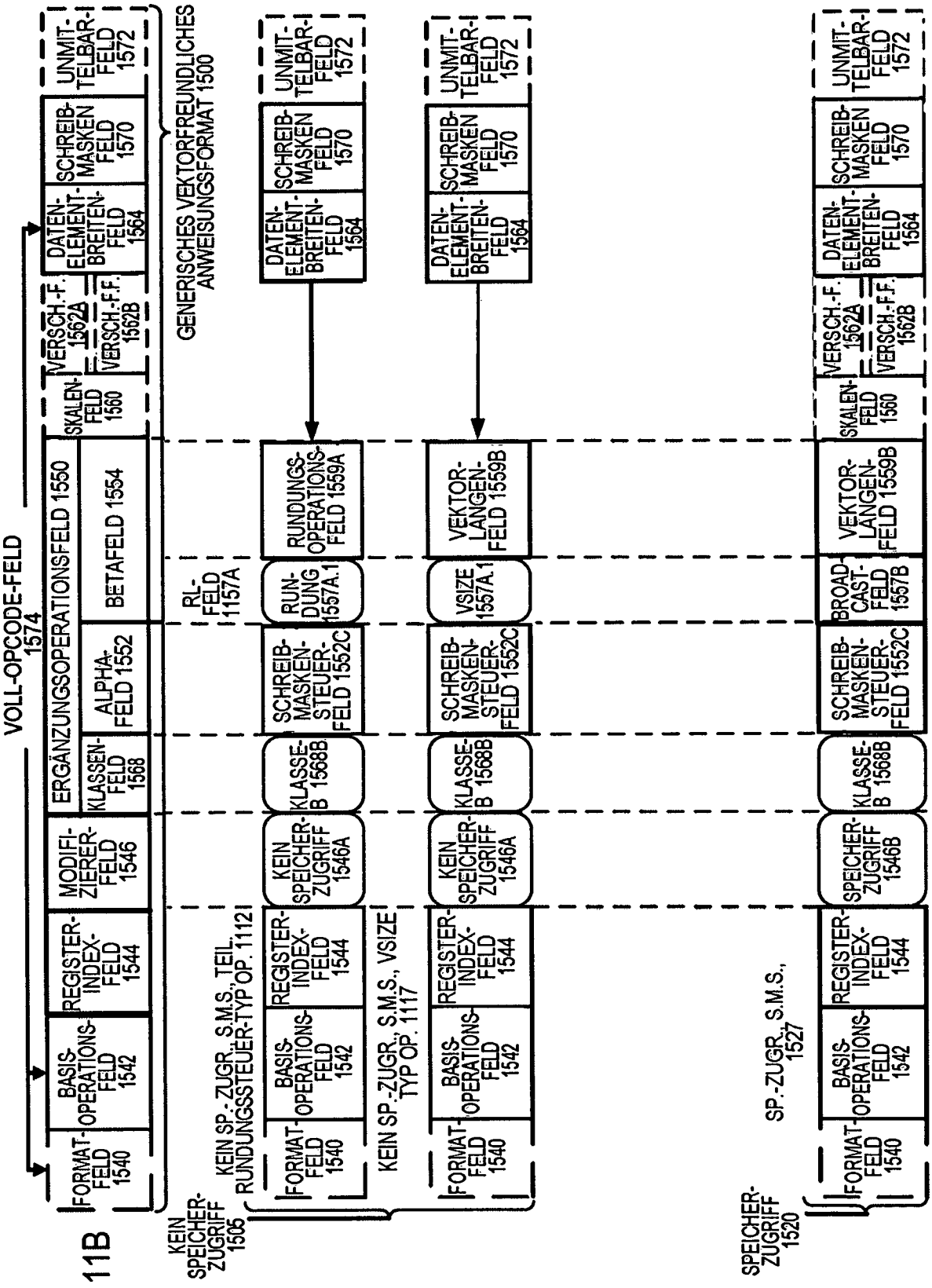

VOLL-OPCODE-FELD 1574
FORMAT-FELD 1540
BASIS-OPERATIONS-FELD 1542
REGISTER-INDEX-FELD 1544
MODIFIZIERER-FELD 1546
ERGÄNZUNGSOPERATIONSFELD 1550
KLASSEN-FELD 1568
ALPHA-FELD 1552
BETAFELD 1554
SKALEN-FELD 1560
VERSCH.-F. 1562A
VERSCH.-F.F. 1562B
DATEN-ELEMENT-BREITEN-FELD 1564
SCHREIB-MASKEN FELD 1570
UNMIT-TELBAR-FELD 1572
GENERISCHES VEKTORFREUNDLICHES ANWEISUNGSFORMAT 1500
RL-FELD 1157A
KEIN SPEICHER-ZUGRIFF 1505
KEIN SP.-ZUGR., S.M.S., TEIL. RUNDUNGSSTEUER-TYP OP. 1112
KEIN SP.-ZUGR., S.M.S., VSIZE TYP OP. 1117
KEIN SPEICHER-ZUGRIFF 1546A
KLASSE-B 1568B
SCHREIB-MASKEN-STEUER-FELD 1552C
RUN-DUNG 1557A.1
RUNDUNGS-OPERATIONS-FELD 1559A
VSIZE 1557A.1
VEKTOR-LÄNGEN-FELD 1559B
SP.-ZUGR., S.M.S., 1527
SPEICHER-ZUGRIFF 1520
SPEICHER-ZUGRIFF 1546B
BROAD-CAST-FELD 1557B

FIG. 11B

FIG. 12A

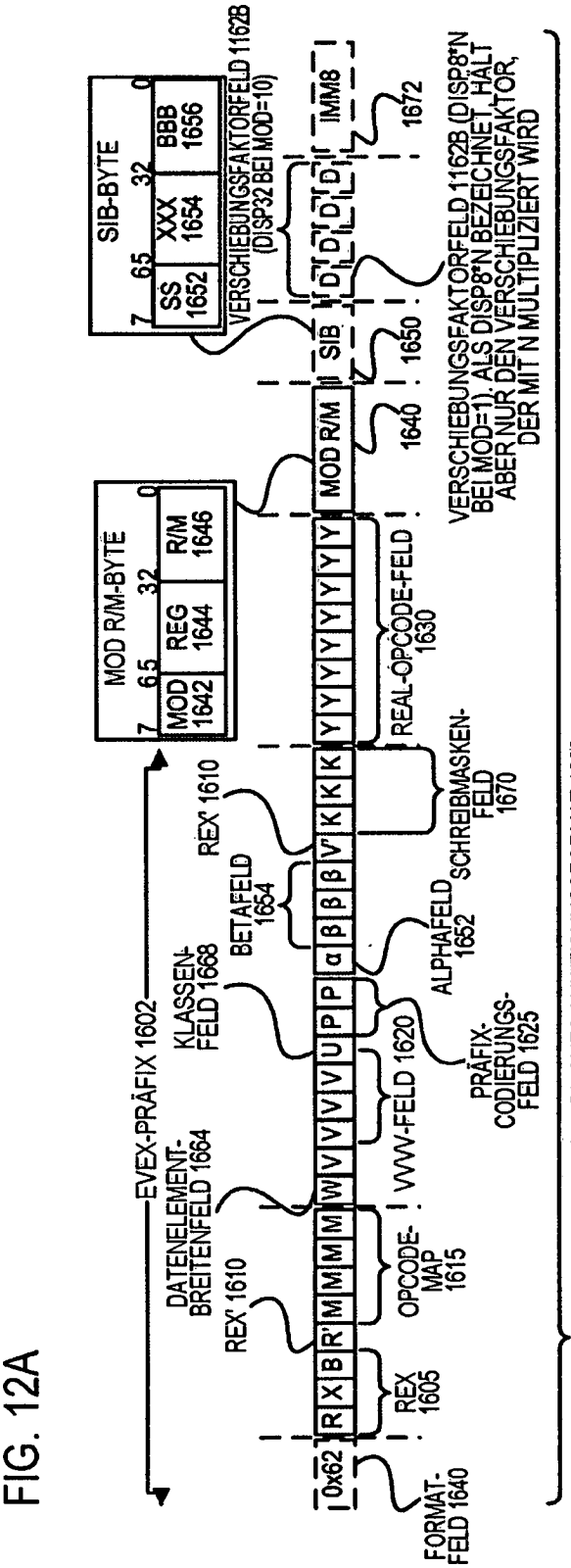


FIG. 12B

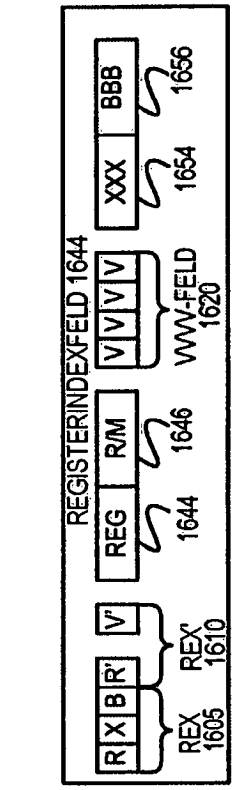


FIG. 12C

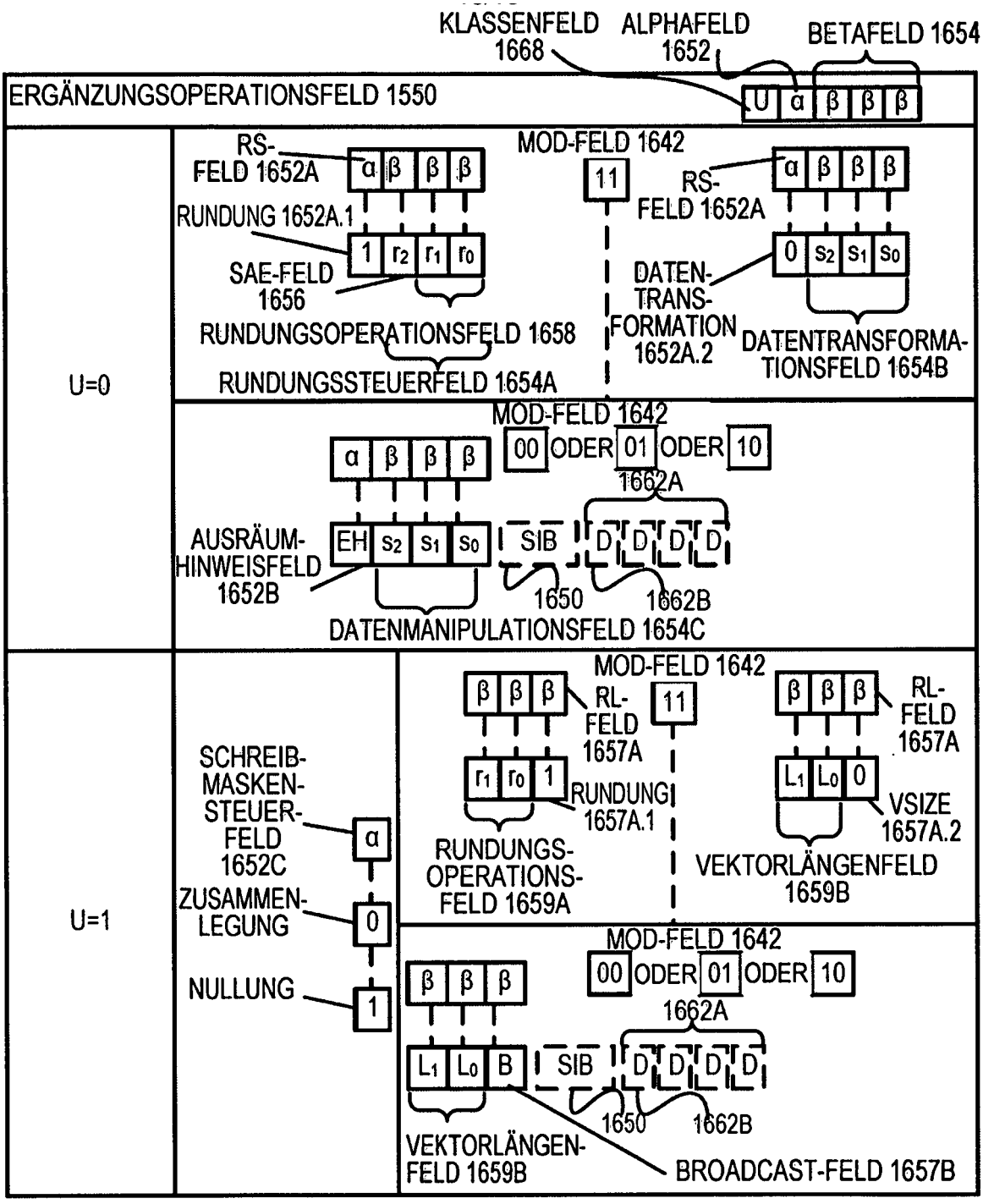

KLASSENFELD 1668
ALPHAFELD 1652
BETAFELD 1654
ERGÄNZUNGSOPERATIONSFELD 1550
U α β β β
U=0
RS-FELD 1652A
α β β β
RUNDUNG 1652A.1
1 r2 r1 r0
SAE-FELD 1656
RUNDUNGSOPERATIONSFELD 1658
RUNDUNGSSTEUERFELD 1654A
MOD-FELD 1642
11
RS-FELD 1652A
α β β β
DATEN-TRANS-FORMATION 1652A.2
0 s2 s1 s0
DATENTRANSFORMA-TIONSFELD 1654B
MOD-FELD 1642
00 ODER 01 ODER 10
1662A
α β β β
AUSRÄUM-HINWEISFELD 1652B
EH s2 s1 s0
SIB
D D D D
1650
1662B
DATENMANIPULATIONSFELD 1654C
U=1
SCHREIB-MASKEN-STEUER-FELD 1652C
α
ZUSAMMEN-LEGUNG
0
NULLUNG
1
MOD-FELD 1642
β β β
RL-FELD 1657A
11
r1 r0 1
RUNDUNG 1657A.1
RUNDUNGS-OPERATIONS-FELD 1659A
β β β
RL-FELD 1657A
L1 L0 0
VSIZE 1657A.2
VEKTORLÄNGENFELD 1659B
MOD-FELD 1642
00 ODER 01 ODER 10
1662A
β β β
L1 L0 B
SIB
D D D D
1650
1662B
VEKTORLÄNGEN-FELD 1659B
BROADCAST-FELD 1657B

FIG. 12D

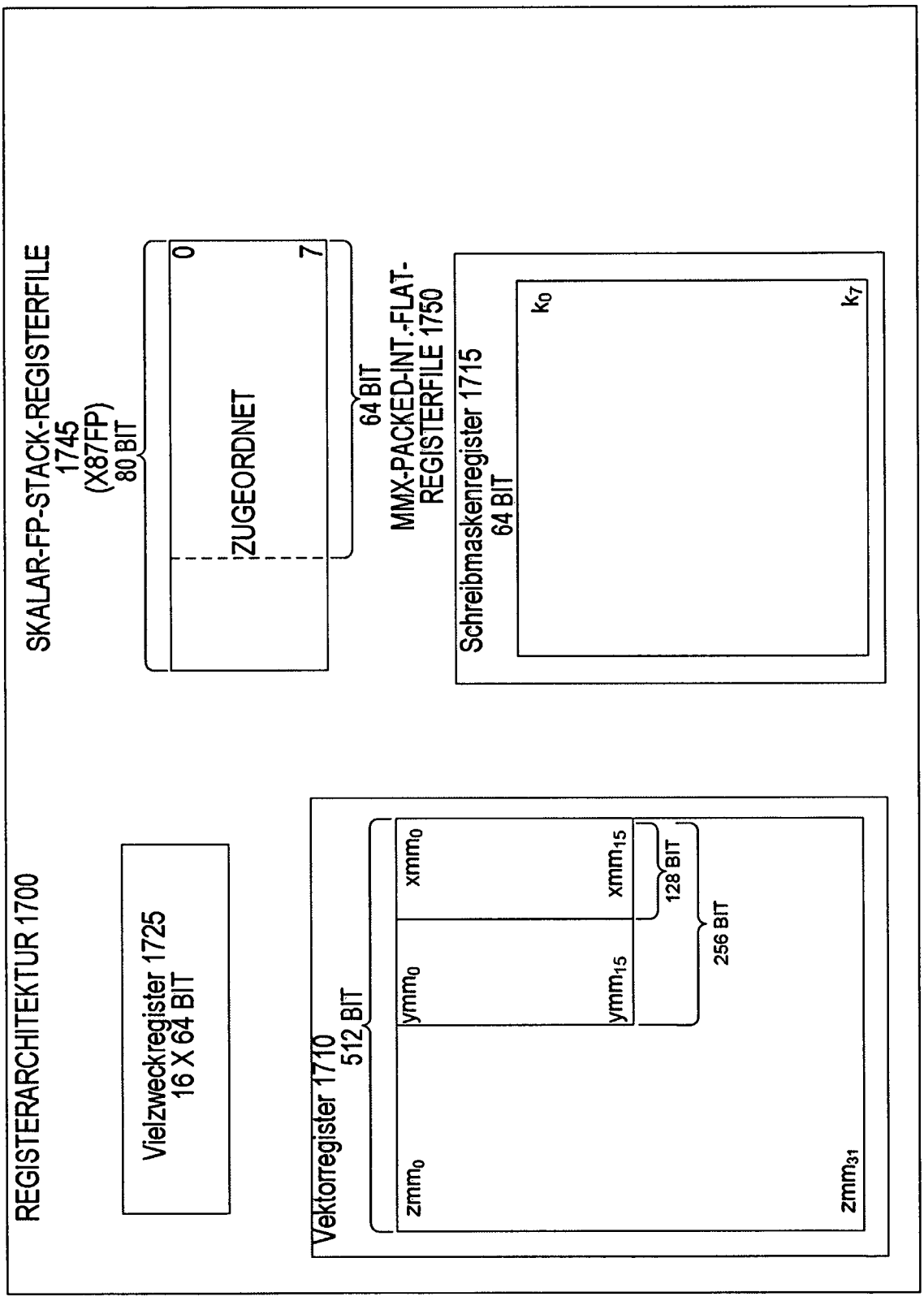

REGISTERARCHITEKTUR 1700
SKALAR-FP-STACK-REGISTERFILE
1745
(X87FP)
80 BIT
0
7
ZUGEORDNET
64 BIT
MMX-PACKED-INT.-FLAT-
REGISTERFILE 1750
Schreibmaskenregister 1715
64 BIT
k_0
k_7
Vielzweckregister 1725
16 X 64 BIT
Vektorregister 1710
512 BIT
zmm_0
zmm_{31}
xmm_0
xmm_{15}
128 BIT
ymm_0
ymm_{15}
256 BIT

FIG. 13