(19) Deutsches Patent- und Markenamt

(10) **DE 10 2018 130 226 A1** 2019.07.04

(12) **Offenlegungsschrift**

(21) Aktenzeichen: **10 2018 130 226.6**
(22) Anmeldetag: **29.11.2018**
(43) Offenlegungstag: **04.07.2019**

(51) Int Cl.: ***G06F 9/30*** (2018.01)

(30) Unionspriorität:
**15/858,278** **29.12.2017** **US**

(71) Anmelder:
**INTEL CORPORATION, Santa Clara, Calif., US**

(74) Vertreter:
**Samson & Partner Patentanwälte mbB, 80538 München, DE**

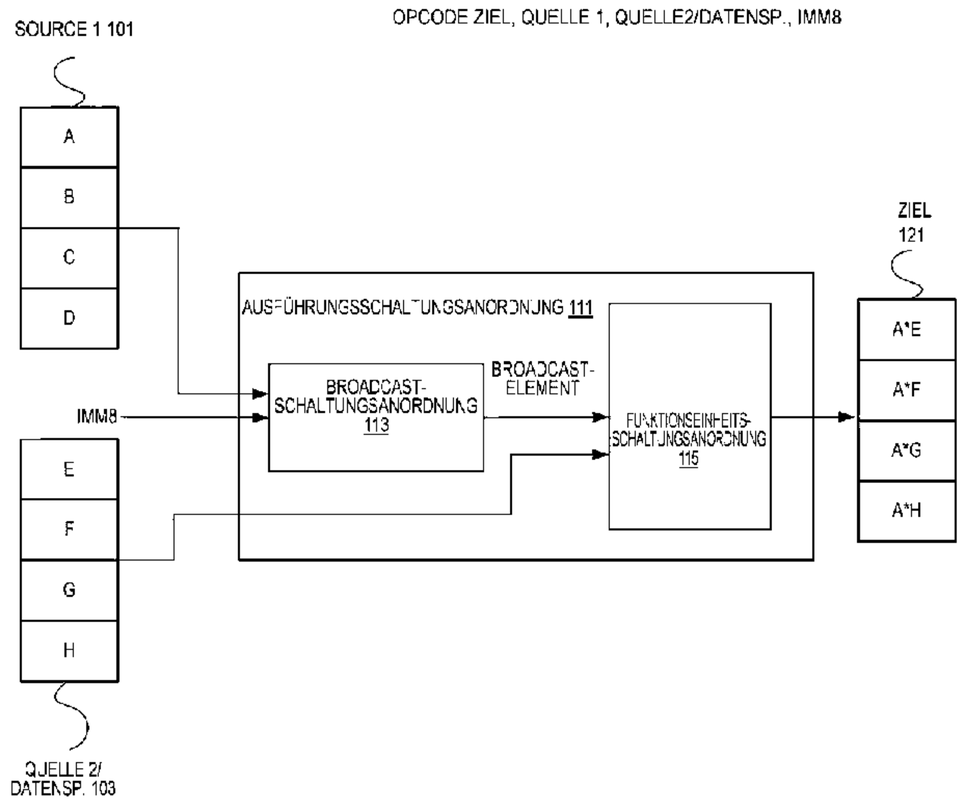
(72) Erfinder:
**Urbanski, Maciej, Gdansk, PL; Ould-Ahmed-Vall, ElMoustapha, Chandler, Ariz., US**

Prüfungsantrag gemäß § 44 PatG ist gestellt.

**Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen.**

(54) Bezeichnung: **Systeme, Verfahren und Einrichtungen für Vektor-Broadcast**

(57) Zusammenfassung: Es werden Systeme, Verfahren und Einrichtungen zum Broadcast eines ausgewählten Datenelements und Ausführen einer Operation in Reaktion auf eine einzelne Anweisung beschrieben. Beispielsweise wird ein Prozessor beschrieben, der Decodierschaltungsanordnung zum Decodieren einer Anweisung, die Felder aufweist für einen Opcode, wenigstens zwei Bezeichner von Quelloperanden für gepackte Daten, einen Bezeichner eines Zieloperanden für gepackte Daten und ein Immediate, und Ausführungsschaltungsanordnung umfasst, um die decodierte Anweisung auszuführen zum: Broadcast eines gepackten Datenelements aus dem identifizierten ersten Quelloperanden für gepackte Daten, wobei die Position des gepackten Datenelements, das durch Broadcast übertragen werden soll, basierend auf einem Wert des Immediate ausgewählt wird, Ausführen von Operationen gemäß dem Opcode auf dem durch Broadcast übertragenen gepackten Datenelement aus dem identifizierten ersten Quelloperanden für gepackte Daten und gepackten Datenelementen des identifizierten zweiten Quelloperanden für gepackte Daten.



## Beschreibung

### HINTERGRUND

**[0001]** Viele Anwendungen zum Maschinenlernen und auf anderen Gebieten betreffen die Verwendung vieler Konstanten. Beispielsweise können bei ML Gewichte der Faltung und andere Algorithmen als Konstanten betrachtet werden, die bei Vektoroperationen aus Gründen der Leistungsfähigkeit verwendet werden.

### Figurenliste

**[0002]** Verschiedene Ausführungsformen in Übereinstimmung mit der vorliegenden Offenbarung werden mit Bezug auf die Zeichnungen beschrieben; es zeigen:

**Fig. 1** eine Ausführungsform eines ausgewählten Abschnitts eines Prozessors zum Verarbeiten einer einzelnen Broadcast- und Berechnungsanweisung;

**Fig. 2** eine Ausführungsform eines ausgewählten Abschnitts eines Prozessors zum Verarbeiten einer einzelnen Broadcast- und Berechnungsanweisung;

**Fig. 3** eine Ausführungsform eines Verfahrens, das durch einen Prozessor ausgeführt wird, um eine einzelne Broadcast- und Berechnungsanweisung zu verarbeiten;

**Fig. 4** eine Ausführungsform eines Verfahrens, das durch einen Prozessor ausgeführt wird, um eine einzelne Broadcast- und Berechnungsanweisung zu verarbeiten;

**Fig. 5** eine Ausführungsform von Hardware zum Verarbeiten einer Anweisung wie z. B. einer Broadcast- und Berechnungsanweisung;

**Fig. 6A** ein Blockdiagramm, das ein generisches vektorfreundliches Anweisungsformat und Klasse-A-Anweisungsvorlagen davon gemäß Ausführungsformen der Erfindung darstellt;

**Fig. 6B** ein Blockdiagramm, das das generische vektorfreundliche Anweisungsformat und Klasse-B-Anweisungsvorlagen davon gemäß Ausführungsformen der Erfindung darstellt;

**Fig. 7A** ein Blockdiagramm, das ein beispielhaftes spezifisches vektorfreundliches Anweisungsformat gemäß Ausführungsformen der Erfindung darstellt;

**Fig. 7B** ein Blockdiagramm, das die Felder des spezifischen vektorfreundlichen Anweisungsformats **700**, die das vollständige Opcode-Feld **674** bilden, gemäß einer Ausführungsformen der Erfindung darstellt;

**Fig. 7C** ein Blockdiagramm, das die Felder des spezifischen vektorfreundlichen Anweisungsformats **700**, die das Registerindexfeld **644** bilden, gemäß einer Ausführungsformen der Erfindung darstellt;

7D ein Blockdiagramm, das die Felder des spezifischen vektorfreundlichen Anweisungsformat **700**, die das Erweiterungsoperationsfeld **650** bilden, gemäß einer Ausführungsformen der Erfindung darstellt;

**Fig. 8** ein Blockdiagramm einer Registerarchitektur **800** gemäß einer Ausführungsform der Erfindung;

**Fig. 9A** ein Blockdiagramm, das sowohl eine beispielhafte In-der-Reihenfolge-Pipeline als auch eine beispielhafte registerumbenennende Außerhalb-der-Reihenfolge-Ausgabe/Ausführungs-Pipeline gemäß Ausführungsformen der Erfindung darstellt;

**Fig. 9B** ein Blockdiagramm, das sowohl eine beispielhafte Ausführungsform eines In-der-Reihenfolge-Architekturkerns als auch eines beispielhaften registerumbenennenden Außerhalb-der-Reihenfolge-Ausgabe/Ausführungs-Architekturkerns, der in einem Prozessor aufgenommen werden soll, gemäß Ausführungsformen der Erfindung darstellt;

**Fig. 10A-B** ein Blockdiagramm einer spezifischeren beispielhaften In-der-Reihenfolge-Kern-Architektur, wobei der Kern einer von mehreren Logikblöcken (die andere Kerne des gleichen Typs und/oder unterschiedlicher Typen enthalten) in einem Chip wäre;

**Fig. 11** ein Blockdiagramm eines Prozessors **1100**, der mehr als einen Kern aufweisen kann, eine integrierte Datenspeichersteuereinheit aufweisen kann und integrierte Grafik aufweisen kann, gemäß Ausführungsformen der Erfindung;

**Fig. 12** ein Blockdiagramm eines Systems in Übereinstimmung mit einer Ausführungsform der vorliegenden Erfindung;

**Fig. 13** ein Blockdiagramm eines ersten spezifischeren beispielhaften Systems in Übereinstimmung mit einer Ausführungsform der vorliegenden Erfindung;

**Fig. 14** ein Blockdiagramm eines zweiten spezifischeren beispielhaften Systems in Übereinstimmung mit einer Ausführungsform der vorliegenden Erfindung;

**Fig. 15** ein Blockdiagramm eines SoC in Übereinstimmung mit einer Ausführungsform der vorliegenden Erfindung; und

**Fig. 16** ein Blockdiagramm, das die Verwendung eines Software-Befehlsumsetzers zum Umsetzen binärer Anweisungen in einer Quellanweisungsmenge in binäre Anweisungen in einer Zielanweisungsmenge gemäß Ausführungsformen der Erfindung gegenüberstellt.

## AUSFÜHRLICHE BESCHREIBUNG

**[0003]** In der folgenden Beschreibung sind zahlreiche spezifische Einzelheiten dargelegt. Es ist jedoch zu verstehen, dass Ausführungsformen der Erfindung ohne diese spezifischen Einzelheiten praktiziert werden können. In anderen Fällen sind bekannte Schaltungen, Strukturen und Techniken nicht im Einzelnen gezeigt worden, um das Verstehen dieser Beschreibung nicht zu verdecken.

**[0004]** Bezugnahme in der Spezifikation auf „eine Ausführungsform“, „eine Beispielausführungsform“ usw. geben an, dass die beschriebene Ausführungsform ein/e spezielle/s Merkmal, Struktur oder Eigenschaft enthalten kann, jedoch muss nicht jede Ausführungsform das/die spezielle Merkmal, Struktur oder Eigenschaft notwendigerweise enthalten. Außerdem beziehen sich solche Ausdrücke nicht notwendigerweise auf dieselbe Ausführungsform. Ferner ist, wenn ein/e spezielle/s Merkmal Struktur oder Eigenschaft in Verbindung mit einer Ausführungsform beschrieben ist, damit übermittelt, dass es innerhalb der Kenntnisse eines Fachmanns ist, ein/e solche/s Merkmal, Struktur oder Eigenschaft in Verbindung mit anderen Ausführungsformen zu beeinflussen, unabhängig davon, ob es ausdrücklich beschrieben ist.

**[0005]** Es sind hier im Einzelnen Ausführungsformen von Anweisungen und deren Unterstützung dargestellt, die bei einem Auftreten einer einzelnen Anweisung einen Broadcast eines einzelnen identifizierten gepackten Datenelements (Vektorelements oder Elements mit einer Anweisung und mehreren Dateneinheiten) und dann Berechnungs- (wie z. B. ALU-) Operationen, die unter Verwendung des durch Broadcast übertragenen einzelnen identifizierten gepackten Datenelements ausgeführt werden, verursachen. Diese Anweisungen ermöglichen einem Programmierer, häufig verwendete Konstanten in Register gepackter Daten (auch als Vektor- oder SIMD-Register bekannt) zu packen und dann einen eingebetteten Broadcast zu einer speziellen Konstanten zu verwenden, wie es in einer Berechnungsanweisung notwendig ist. Das führt typischerweise zu einer Reduktion des Cache-Drucks, der Cache/Datenspeicher-Bandbreite und außerdem des Registerdrucks.

**[0006]** **Fig. 1** stellt eine Ausführungsform eines ausgewählten Abschnitts eines Prozessors zum Verarbeiten einer einzelnen Broadcast- und Berechnungsanweisung dar. Die Anweisung weist einen Opcode, wenigstens zwei Quelloperanden (entweder Register oder Datenspeicher), ein Immediate und einen Zieloperanden auf. Das Format dieser Anweisung enthält Felder für den Opcode, die die Berechnungsoperation, die ausgeführt werden soll, definieren, einen Bezeichner eines ersten Quelloperanden für gepackte Daten (als „QUELLE 1“ gezeigt), einen Bezeichner eines zweiten Quelloperanden für gepackte Daten (als „QUELLE 2/DATENSP.“ gezeigt), ein Immediate (als „IMM8“ gezeigt - ein 8-Bit-Immediate) und einen Bezeichner eines Zieloperanden für gepackte Daten (als „ZIEL“ gezeigt).

**[0007]** Die Quelloperandenfelder für gepackte Daten repräsentieren entweder einen Registerort für gepackte Daten oder einen Datenspeicherort für gepackte Daten, die mehrere gepackte Datenelemente aufweisen.

**[0008]** Das Feld für den Zieloperanden für gepackte Daten repräsentiert einen Registerort für gepackte Daten, in dem die Ergebnisse der Berechnungsoperationen der Anweisungen gespeichert werden sollen.

**[0009]** In dem dargestellten Beispiel weist der identifizierte erste Quelloperand **101** („QUELLE 1“) mehrere gepackte Datenelemente auf. In Position 0 des identifizierten ersten Quelloperanden **101** für gepackte Daten ist ein „A“-Wert, in Position 1 ist ein „B“-Wert, usw. Der identifizierte zweite Quelloperand **103** für gepackte Daten („QUELLE 2“) weist ebenfalls mehrere gepackte Datenelemente auf. In Position **0** des identifizierten zweiten Quelloperanden **103** für gepackte Daten ist ein „E“-Wert, in Position 1 ist ein „F“-Wert, usw.

**[0010]** In diesem Beispiel wird der Immediate-Wert („IMM8“) der Anweisung für eine Broadcast-Schaltungsanordnung **113** bereitgestellt, die ein Teil der Ausführungsschaltungsanordnung **111** ist, die verwendet wird,

um die Anweisung auszuführen, sobald sie decodiert worden ist. In einigen Ausführungsformen findet der Broadcast vor der Ausführungsschaltungsanordnung **111** in der Pipeline des Prozessors statt. Die Broadcast-Schaltungsanordnung **113** verwendet den Wert des Immediate-Werts, um zu bestimmen, welche Position in dem identifizierten ersten Quelloperanden **101** für gepackte Daten durch Broadcast übertragen werden soll. In diesem Beispiel weist das Immediate einen Dezimalwert von „0“ auf (der auch binär gleich ist). Somit wird das „A“ in dem identifizierten ersten Quelloperanden **101** für gepackte Daten als Broadcast-Element ausgewählt, das für die Funktionseinheitsschaltungsanordnung **115** der bereitgestellt wird. „A“ wird jetzt in allen Berechnungen verwendet, die durch die Funktionseinheitsschaltungsanordnung **115** ausgeführt werden.

**[0011]** Die gepackten Datenelemente des identifizierten zweiten Quelloperanden **103** für gepackte Daten werden für die Funktionseinheitsschaltungsanordnung **115** bereitgestellt. In diesem Beispiel ist die Berechnung eine Multiplikation. Somit multipliziert die Funktionseinheitsschaltungsanordnung **115** „A“ mit den gepackten Datenelementen des identifizierten zweiten Quelloperanden **103** für gepackte Daten und speichert das Ergebnis jeder Multiplikation in Positionen für gepackte Datenelemente des identifizierten Zieloperanden **121** für gepackte Daten, die den Positionen der gepackten Datenelemente des identifizierten zweiten Quelloperanden **103** für gepackte Daten, die in der Operation verwendet wurden, entsprechen.

**[0012]** **Fig. 2** stellt eine Ausführungsform eines ausgewählten Abschnitts eines Prozessors zum Verarbeiten einer einzelnen Broadcast- und Berechnungsanweisung dar. Die Anweisung weist einen Opcode, wenigstens zwei Quelloperanden (entweder Register oder Datenspeicher), ein Präfix und einen Zieloperanden auf. Das Format dieser Anweisung enthält Felder für das Präfix, den Opcode, die die Berechnungsoperation, die ausgeführt werden soll, definiert, einen Bezeichner eines ersten Quelloperanden für gepackte Daten (als „QUELLE 1“ gezeigt), einen Bezeichner eines zweiten Quelloperanden für gepackte Daten (als „QUELLE 2/DATENSP.“ gezeigt) und einen Bezeichner eines Zieloperanden für gepackte Daten (als „ZIEL“ gezeigt).

**[0013]** Die Quelloperandenfelder für gepackte Daten repräsentieren entweder einen Registerort für gepackte Daten oder einen Datenspeicherort für gepackte Daten, die mehrere gepackte Datenelemente aufweisen.

**[0014]** Das Feld für den Zieloperanden für gepackte Daten repräsentiert einen Registerort für gepackte Daten, in dem die Ergebnisse der Berechnungsoperationen der Anweisungen gespeichert werden sollen.

**[0015]** In dem dargestellten Beispiel weist der identifizierte erste Quelloperand **201** („QUELLE **1**“) mehrere gepackte Datenelemente auf. In Position 0 des identifizierten ersten Quelloperanden **201** für gepackte Daten ist ein „A“-Wert, in Position 1 ist ein „B“-Wert, usw. Der identifizierte zweite Quelloperanden **203** für gepackte Daten („QUELLE **2**“) weist ebenfalls mehrere gepackte Datenelemente auf. In Position **0** des identifizierten zweiten Quelloperanden **203** für gepackte Daten ist ein „E“-Wert, in Position **1** ist ein „F“-Wert, usw.

**[0016]** In diesem Beispiel enthält das Präfix der Anweisung einen Bezeichner des Datenelements, das durch Broadcast übertragen werden soll, und dieser Bezeichnet wird für die Broadcast-Schaltungsanordnung **213** bereitgestellt, die ein Teil der Ausführungsschaltungsanordnung **211** ist, die verwendet wird, um die Anweisung auszuführen, sobald sie decodiert worden ist. In einigen Ausführungsformen findet das Broadcast vor der Ausführungsschaltungsanordnung **211** in der Pipeline des Prozessors statt. Die Broadcast-Schaltungsanordnung **213** verwendet die Werte des Bezeichners des Datenelements, das durch Broadcast übertragen werden soll, um zu bestimmen, welche Position in dem identifizierten ersten Quelloperanden **201** für gepackte Daten übertragen werden soll. In diesem Beispiel weist der Bezeichner des Datenelements, das übertragen werden soll, einen Dezimalwert von „0“ auf (der auch binär gleich ist). Somit wird das „A“ in dem identifizierten ersten Quelloperanden **201** für gepackte Daten als Broadcast-Element ausgewählt, das für die Funktionseinheitsschaltungsanordnung **215** der bereitgestellt wird. „A“ wird jetzt in allen Berechnungen verwendet, die durch die Funktionseinheitsschaltungsanordnung **215** ausgeführt werden.

**[0017]** Die gepackten Datenelemente des identifizierten zweiten Quelloperanden **203** für gepackte Daten werden für die Funktionseinheitsschaltungsanordnung **215** bereitgestellt. In diesem Beispiel ist die Berechnung eine Multiplikation. Somit multipliziert die Funktionseinheitsschaltungsanordnung **215** „A“ mit den gepackten Datenelementen des identifizierten zweiten Quelloperanden **203** für gepackte Daten und speichert das Ergebnis jeder Multiplikation in Positionen für gepackte Datenelemente des identifizierten Zieloperanden **221** für gepackte Daten, die den Positionen der gepackten Datenelemente des identifizierten zweiten Quelloperanden **203** für gepackte Daten, die in der Operation verwendet wurden, entsprechen.

**[0018]** **Fig. 3** stellt eine Ausführungsform eines Verfahrens dar, das durch einen Prozessor ausgeführt wird, um eine einzelne Broadcast- und Berechnungsanweisung zu verarbeiten.

**[0019]** Bei **301** wird eine Anweisung abgeholt. In einigen Ausführungsformen wird die Anweisung aus einem Anweisungs-Cache abgeholt. Die Anweisung weist einen Opcode, ein Immediate und Bezeichner von wenigstens zwei Quelloperanden (entweder Register oder Datenspeicher) und eines Zieloperanden auf. Das Format dieser Anweisung enthält Felder für einen Opcode, der die Berechnungsoperation definiert, die ausgeführt werden soll, einen Bezeichner eines ersten Quelloperanden für gepackte Daten, einen Bezeichner eines zweiten Quelloperanden für gepackte Daten und einen Bezeichner eines Zieloperanden für gepackte Daten.

**[0020]** Die abgeholte Anweisung wird bei **303** decodiert. Beispielsweise wird die abgeholte Anweisung zum Umwandeln von Gleitkomma zu Festkomma durch die Decodierungsschaltungsanordnung decodiert, wie hier im Einzelnen beschrieben ist.

**[0021]** Datenwerte, die den identifizierten Quelloperanden der decodierten Anweisung zugeordnet sind, werden bei **305** abgerufen, und die decodierte Anweisung wird geplant (bei Bedarf). Beispielsweise werden, wenn ein identifizierter Quelloperand ein Datenspeicheroperand ist, die Daten aus dem angegeben Datenspeicherort abgerufen.

**[0022]** Bei **307** wird die decodierte Anweisung durch die Ausführungsschaltungsanordnung (Hardware) ausgeführt, wie hier im Einzelnen beschrieben ist. Die Ausführung der Anweisung enthält: 1) Broadcast eines gepackten Datenelements aus dem identifizierten ersten Quelloperanden für gepackte Daten, wobei die Position des gepackten Datenelements, das durch Broadcast übertragen werden soll, basierend auf dem Wert des Immediate ausgewählt wird; 2) Ausführen von Operationen gemäß dem Opcode auf dem durch Broadcast übertragenen gepackten Datenelement aus dem identifizierten ersten Quelloperanden für gepackte Daten und den gepackte Datenelementen eines identifizierten zweiten Quelloperanden für gepackte Daten; und 3) Speichern der Ergebnisse der Operationen in dem identifizierten Zieloperanden für gepackte Daten in Positionen, die den Positionen der gepackten Datenelement des identifizierten zweiten Quelloperanden für gepackte Daten entsprechen.

**[0023]** In einigen Ausführungsformen wird bei **309** die Anweisung übergeben oder zurückgezogen.

**[0024]** **Fig. 4** stellt eine Ausführungsform eines Verfahrens dar, das durch einen Prozessor ausgeführt wird, um eine einzelne Broadcast- und Berechnungsanweisung zu verarbeiten.

**[0025]** Bei **401** wird eine Anweisung abgeholt. In einigen Ausführungsformen wird die Anweisung aus einem Anweisungs-Cache abgeholt. Die Anweisung weist ein Präfix, einen Opcode und Bezeichner von wenigstens zwei Quelloperanden (entweder Register oder Datenspeicher) und eines Zieloperanden auf. Das Format dieser Anweisung enthält Felder für das Präfix, das ein Kennzeichen einer Datenelementposition zum Übertragen durch Broadcast enthält, den Opcode, der die auszuführende Berechnungsoperation definiert, einen Bezeichner eines ersten Quelloperanden für gepackte Daten, einen Bezeichner eines zweiten Quelloperanden für gepackte Daten und einen Bezeichner eines Zieloperanden für gepackte Daten.

**[0026]** Die abgeholte Anweisung wird bei **403** decodiert. Beispielsweise wird die abgeholte Anweisung zum Umwandeln von Gleitkomma zu Festkomma durch die Decodierungsschaltungsanordnung decodiert, wie hier im Einzelnen beschrieben ist.

**[0027]** Datenwerte, die den identifizierten Quelloperanden für die decodierte Anweisung zugeordnet sind, werden bei **405** abgerufen, und die decodierte Anweisung wird geplant (bei Bedarf). Beispielsweise wenn ein identifizierter Quelloperand ein Datenspeicheroperand ist, werden die Daten aus dem angegeben Datenspeicherort abgerufen.

**[0028]** Bei **407** wird die decodierte Anweisung durch die Ausführungsschaltungsanordnung (Hardware) ausgeführt, wie hier im Einzelnen beschrieben ist. Die Ausführung der Anweisung enthält: 1) Broadcast eines gepackten Datenelements aus dem identifizierten ersten Quelloperanden für gepackte Daten, wobei das gepackte Datenelement, das durch Broadcast übertragen werden soll, basierend auf einem Kennzeichen der Datenelementposition des Präfix, die durch Broadcast übertragen werden soll, ausgewählt wird; 2) Ausführen von Operationen gemäß dem Opcode auf dem durch Broadcast übertragenen gepackten Datenelements aus dem identifizierten ersten Quelloperanden für gepackte Daten und den gepackten Datenelementen eines identifizierten zweiten Quelloperanden für gepackte Daten; und 3) Speichern der Ergebnisse der Operationen in dem identifizierten Zieloperanden für gepackte Daten in Positionen, die den Positionen der gepackten Datenelement des identifizierten zweiten Quelloperanden für gepackte Daten entsprechen.

**[0029]** In einigen Ausführungsformen wird bei **409** die Anweisung übergeben oder zurückgezogen.

**[0030]** **Fig. 5** stellt eine Ausführungsform von Hardware zum Verarbeiten einer Anweisung wie z. B. einer Broadcast- und Berechnungsanweisung dar. Wie dargestellt ist speichert der Speicher **503** eine Broadcast- und Berechnungsanweisung **501**, die ausgeführt werden soll.

**[0031]** Die Anweisung **501** wird durch die Decodierungsschaltungsanordnung **505** empfangen. Beispielsweise empfängt die Decodierungsschaltungsanordnung **505** diese Anweisung von der Abhollogik/Schaltungsanordnung. In einigen Ausführungsformen enthält die Anweisung Felder für einen Opcode, Bezeichner für Quelloperanden, ein Immediate und einen Zielbezeichner. In einigen Ausführungsformen enthält die Anweisung Felder für ein Präfix, einen Opcode, Bezeichner für Quelloperanden und einen Zielbezeichner. In einigen Ausführungsformen sind die identifizierten Quell- und Zieloperanden Register, und in anderen Ausführungsformen sind einer oder mehrere Datenspeicherorte.

**[0032]** Genauer dargestellte Ausführungsformen wenigstens eines Anweisungsformats werden später im Einzelnen beschrieben. Die Decodierungsschaltungsanordnung **505** decodiert die Anweisung in eine oder mehrere Operationen. In einigen Ausführungsformen enthält dieses Decodieren das Erzeugen von mehreren Mikrooperationen, die durch die Ausführungsschaltungsanordnung (wie z. B. die Ausführungsschaltungsanordnung **509**) ausgeführt werden sollen. Die Decodierungsschaltungsanordnung **505** decodiert außerdem Anweisungspräfixe.

**[0033]** In einigen Ausführungsformen stellt die Registerumbenennungs-, Registerzuweisungs- und/oder Planungsschaltungsanordnung **507** die Funktionalität für eines oder mehrere aus dem Folgenden bereit: 1) Umbenennen logischer Operandenwerte in physikalische Operandenwerte (in einigen Ausführungsformen z. B. eine Registeraliastabelle), 2) Zuweisen von Status-Bits und Flags zu der decodierten Anweisung und 3) Planen der decodierten Anweisung zur Ausführung auf der Ausführungsschaltungsanordnung aus einem Anweisungs-Pool (in einigen Ausführungsformen z. B. unter Verwendung einer Reservierungsstation).

**[0034]** Register (Registerdatei) und/oder Datenspeicher **508** speichern Daten als Operanden der Anweisung, auf denen durch die Ausführungsschaltungsanordnung **509** gearbeitet werden soll. Beispielhafte Registertypen enthalten Register für gepackte Daten, Allzweckregister und Gleitkommaregister.

**[0035]** Die Ausführungsschaltungsanordnung **509** führt die decodierte Anweisung aus. In einigen Ausführungsformen umfasst die Ausführung **1**) Broadcast eines gepackten Datenelements aus dem identifizierten ersten Quelloperanden für gepackte Daten, wobei die Position des gepackten Datenelements, das durch Broadcast übertragen werden soll, basierend auf dem Wert des Immediate ausgewählt wird; 2) Ausführen von Operationen gemäß dem Opcode auf dem durch Broadcast übertragenen gepackten Datenelement aus dem identifizierten ersten Quelloperanden für gepackte Daten und den gepackten Datenelementen eines identifizierten zweiten Quelloperanden für gepackte Daten; und 3) Speichern der Ergebnisse der Operationen in dem identifizierten Zieloperanden für gepackte Daten in Positionen, die den Positionen der gepackten Datenelemente des identifizierten zweiten Quelloperanden für gepackte Daten entsprechen.

**[0036]** In anderen Ausführungsformen umfasst die Ausführung **1**) Broadcast eines gepackten Datenelements aus dem identifizierten ersten Quelloperanden für gepackte Daten, wobei die Position des gepackten Datenelements, das durch Broadcast übertragen werden soll, basierend auf dem Wert des Präfix ausgewählt wird; 2) Ausführen von Operationen gemäß dem Opcode auf dem durch Broadcast übertragenen gepackten Datenelement aus dem identifizierten ersten Quelloperanden für gepackte Daten und den gepackte Datenelementen eines identifizierten zweiten Quelloperanden für gepackte Daten; und 3) Speichern der Ergebnisse der Operationen in dem identifizierten Zieloperanden für gepackte Daten in Positionen, die den Positionen der gepackten Datenelemente des identifizierten zweiten Quelloperanden für gepackte Daten entsprechen.

**[0037]** In einigen Ausführungsformen übergibt die Rückzug/Rückschreibschaltungsanordnung **511** das Zielregister architektonisch in die Register oder den Datenspeicher **508** und zieht die Anweisung zurück.

**[0038]** Eine Ausführungsform eines Formats für das Broadcast und die Berechnungsanweisung ist OPCODE DST, SRC1, SRC2/MEM, IMM8. OPCODE ist der Opcode der Anweisung. DST ist ein Feld, das einen Zieloperanden identifiziert. SCR1 und SCR2 sind Felder für Bezeichner von Quelloperanden wie z. B. ein Register und/oder ein Datenspeicherort. In einigen Ausführungsformen sind die Operandenfelder unter Verwendung des VVVV-Felds **720**, MOD R/M 740 und/oder SIB **750** codiert. In einigen Ausführungsformen ist IMM8 das Feld **672**.

**[0039]** Eine weitere Ausführungsform eines Formats für das Broadcast und die Berechnungsanweisung ist PREFIX OPCODE DST, SRC1, SRC2/MEM. PREFIX ist das Präfix für die Anweisung. OPCODE ist der Opcode der Anweisung. DST ist ein Feld, das einen Zieloperanden identifiziert. SCR1 und SCR2 sind Felder für Bezeichner von Quelloperanden wie z. B. ein Register und/oder ein Datenspeicherort. In einigen Ausführungsformen sind die Operandenfelder unter Verwendung des VVVV-Felds **720**, MOD R/M **740** und/oder SIB **750** codiert. In einigen Ausführungsformen ist das Präfix das Feld **702**.

**[0040]** In einigen Ausführungsformen sind einer der Quelloperanden und der Zieloperand gleich.

**[0041]** In Ausführungsformen enthalten die Codierungen der Anweisung einen Datenspeicheradressierungsoperanden vom Skalierungs-Index-Basis- (SIB-) Typ, der mehrere indexierte Zielorte im Datenspeicher indirekt identifiziert (z. B. Feld **750**). In einer Ausführungsform kann ein Datenspeicheroperand vom SIB-Typ eine Codierung enthalten, die ein Basisadressenregister identifiziert. Der Inhalt des Basisadressenregisters kann eine Basisadresse im Datenspeicher repräsentieren, aus der die Adressen der speziellen Zielorte im Datenspeicher berechnet werden. Beispielsweise kann die Basisadresse die Adresse des ersten Orts in einem Block potentieller Zielorte für eine erweiterte Vektoranweisung sein. In einer Ausführungsform kann ein Datenspeicheroperand vom SIB-Typ eine Codierung enthalten, die ein Indexregister identifiziert. Jedes Element des Indexregisters kann einen Index oder Versatzwert spezifizieren, der verwendbar ist, um aus der Basisadresse eine Adresse eines jeweiligen Zielorts innerhalb eines Blocks potentieller Zielorte zu berechnen. In einer Ausführungsform kann ein Datenspeicheroperand vom SIB-Typ eine Codierung enthalten, die einen Skalierungsfaktor spezifiziert, der auf jeden Indexwert angewandt werden soll, wenn eine jeweilige Zieladresse berechnet wird. Beispielsweise falls ein Skalierungsfaktorwert von vier in dem Datenspeicheroperanden vom SIB-Typ codiert ist, kann jeder Indexwert, der aus einem Element des Indexregisters erhalten wird, mit vier multipliziert und dann zu der Basisadresse addiert werden, um eine Zieladresse zu berechnen.

**[0042]** In einer Ausführungsform kann ein Datenspeicheroperand vom SIB-Typ der Form vm32{x,y,z} ein Vektorfeld von Datenspeicheroperanden definieren, die unter Verwendung der Datenspeicheradressierung vom SIB-Typ spezifiziert sind. In diesem Beispiel ist das Feld von Datenspeicheradressen unter Verwendung eines allgemeinen Basisregisters, eines konstanten Skalierungsfaktors und eines Vektorindexregisters, das individuelle Elemente enthält, von denen jedes ein 32-Bit-Indexwert ist, spezifiziert. Das Vektorindexregister kann ein 128-Bit-Register- (z. B. XMM-) Register (vm32x), ein 256-Bit- (z. B. YMM-) Register (vm32y) oder ein 512-Bit- (z. B. ZMM-) Register (vm32z) sein. In einer weiteren Ausführungsform kann ein Datenspeicheroperand vom SIB-Typ der Form vm64{x,y,z} ein Vektorfeld von Datenspeicheroperanden definieren, die unter Verwendung der Datenspeicheradressierung vom SIB-Typ spezifiziert sind. In diesem Beispiel ist das Feld von Datenspeicheradressen unter Verwendung eines allgemeinen Basisregisters, eines konstanten Skalierungsfaktors und eines Vektorindexregisters, das individuelle Elemente enthält, von denen jedes ein 64-Bit-Indexwert ist, spezifiziert. Das Vektorindexregister kann ein 128-Bit-Register- (z. B. XMM-) Register (vm64x), ein 256-Bit- (z. B. YMM-) Register (vm64y) oder ein 512-Bit- (z. B. ZMM-) Register (vm64z) sein.

**[0043]** Nachstehend sind beispielhafte Anweisungsformate, Architekturen und Systeme im Einzelnen beschrieben, die für die vorstehend im Einzelnen beschriebenen Anweisungen benutzt werden können. Beispielsweise ist eine beispielhafte Pipeline, die die Anweisungen unterstützt, im Einzelnen beschrieben, die eine Schaltungsanordnung enthält, um die hier im Einzelnen beschriebenen Verfahren auszuführen.

## Befehlssätze

**[0044]** Ein Befehlssatz kann ein oder mehrere Anweisungsformate enthalten. Ein gegebenes Anweisungsformat kann verschiedene Felder definieren (z. B. die Anzahl von Bits, den Ort der Bits), um unter anderem die Operation, die ausgeführt werden soll, (z. B. den Opcode) und den/die Operand(en), auf denen diese Operation ausgeführt werden soll, und/oder andere Datenfeld(er) (z. B. eine Maske) zu spezifizieren. Einige Anweisungsformate sind durch die Definition von Anweisungsvorlagen (oder Unterformate) weiter heruntergebrochen. Beispielsweise können die Anweisungsvorlagen eines gegebenen Anweisungsformats so definiert sein, dass sie unterschiedliche Teilmengen der Felder des Anweisungsformats aufweisen (die enthaltenen Felder sind typischerweise in der gleichen Reihenfolge, wenigstens einige weisen jedoch unterschiedliche Bit-Positionen auf, weil weniger Felder enthalten sind), und/oder so definiert sein, dass sie ein gegebenes Feld unterschiedlich interpretieren lassen. Somit wird jede Anweisung eines ISA unter Verwendung eines gegebenen Anweisungsformats (und, falls definiert, in einer gegebenen aus den Anweisungsvorlagen dieses Anweisungsformats) ausgedrückt und enthält Felder zum Spezifizieren der Operation und der Operanden. Beispielsweise weist eine beispielhafte ADD-Anweisung einen spezifischen Opcode und ein Anweisungsformat auf, das ein Opcodefeld, um diesen Opcode zu spezifizieren, und Operandenfelder, um Operanden (Quelle1/Ziel und Quelle2) auszu-

wählen, enthält; und ein Auftreten dieser ADD-Anweisung in einem Anweisungsstrom wird spezifischen Inhalt in den Operandenfeldern aufweisen, die spezifische Operanden auswählen. Eine Menge von SIMD-Erweiterungen, die als die weiterentwickelten Vektorerweiterungen (AVX) (AVX1 und AVX2) bezeichnet sind und das Vektorerweiterungs- (VEX-) Codierungsschema verwenden, ist freigegeben und/oder veröffentlich worden (z. B. siehe „Intel® 64 and IA-32 Architectures Software Developer's Manual“, September 2014; und siehe „Intel® Advanced Vector Extensions Programming Reference“, Oktober 2014).

## Beispielhafte Anweisungsformate

**[0045]** Ausführungsformen der Anweisung(en), die hier beschrieben sind, können in unterschiedlichen Formaten verwirklicht sein. Zusätzlich sind beispielhafte Systeme, Architekturen und Pipelines nachstehend im Einzelnen beschrieben. Ausführungsformen der Anweisung(en) können auf solchen Systemen, Architekturen und Pipelines ausgeführt werden, sind jedoch nicht auf die im Einzelnen beschriebenen eingeschränkt.

### Generisches vektorfreundliches Anweisungsformat

**[0046]** Ein vektorfreundliches Anweisungsformat ist ein Anweisungsformat, das für Vektoranweisungen geeignet ist (es sind z. B. spezielle Felder vorhanden, die für Vektoroperationen spezifisch sind). Obwohl Ausführungsformen beschrieben sind, in denen sowohl Vektor- als auch Skalar-Operationen durch das vektorfreundliche Anweisungsformat unterstützt werden, verwenden alternative Ausführungsformen nur Vektoroperationen in dem vektorfreundlichen Anweisungsformat.

**[0047]** **Fig. 6A-6B** sind Blockdiagramme, die ein generisches vektorfreundliches Anweisungsformat und Anweisungsvorlagen davon gemäß Ausführungsformen der Erfindung darstellen. **Fig. 6A** ist ein Blockdiagramm, das ein generisches vektorfreundliches Anweisungsformat und Klasse-A-Anweisungsvorlagen davon gemäß Ausführungsformen der Erfindung darstellt; während **Fig. 6B** ein Blockdiagramm ist, das das generische vektorfreundliche Anweisungsformat und Klasse-B-Anweisungsvorlagen davon gemäß Ausführungsformen der Erfindung darstellt. Insbesondere ein generisches vektorfreundliches Anweisungsformat **600**, für das Klasse-A- und Klasse-B-Anweisungsvorlagen definiert sind, von denen beide Kein-Datenspeicherzugriff- **605** Anweisungsvorlagen und Datenspeicherzugriff- **620** Anweisungsvorlagen enthalten. Der Begriff generisch in dem Kontext des vektorfreundlichen Anweisungsformats bezieht sich darauf, dass das Anweisungsformat nicht an einen spezifischen Befehlssatz gebunden ist.

**[0048]** Obwohl Ausführungsformen der Erfindung beschrieben werden, in denen das vektorfreundliche Anweisungsformat das Folgende unterstützt: eine 64-Byte-Vektoroperandenlänge (oder -größe) mit 32 Bit (4 Byte) oder 64 Bit (8 Byte) Datenelementbreiten (oder -größen) (und somit besteht ein 64-Byte-Vektor aus entweder 16 Elementen von Doppelwortgröße oder alternativ 8 Elementen von Quadwortgröße); eine 64-Byte-Operandenlänge (oder -größe) mit 16 Bit (2 Byte) oder 8 Bit (1 Byte) Datenelementbreiten (oder -größen); eine 32-Bit-Vektoroperandenlänge (oder -größe) mit 32 Bit (4 Byte), 64 Bit (8 Byte), 16 Bit (2 Byte) oder 8 Bit (1 Byte) Datenelementbreiten (oder -größen); und eine 16-Bit-Vektoroperandenlänge (oder -größe) mit 32 Bit (4 Byte), 64 Bit (8 Byte), 16 Bit (2 Byte) oder 8 Bit (1 Byte) Datenelementbreiten (oder -größen); können alternative Ausführungsformen größere, kleinere und/oder unterschiedliche Vektoroperandengrößen (z. B. 256 Byte-Vektoroperanden) mit größeren, kleineren oder unterschiedlichen Datenelementbreiten (z. B. 128 Bit (16 Byte) Datenelementbreiten) unterstützen.

**[0049]** Die Klasse-A-Anweisungsvorlagen in **Fig. 6A** enthalten: 1) innerhalb der Kein-Datenspeicherzugriff- **605** Anweisungsvorlagen sind eine Anweisungsvorlage für Kein-Datenspeicherzugriff-, Operation vom Vollrundungs-Steuertyp **610** und eine Anweisungsvorlage für Kein-Datenspeicherzugriff, Operation vom Datentransformationstyp **615** gezeigt. und 2) innerhalb der Datenspeicherzugriff- **620** Anweisungsvorlage ist eine Anweisungsvorlage für Datenspeicherzugriff, temporär **625** und eine Anweisungsvorlage für Datenspeicherzugriff, nicht temporär **630** gezeigt. Die Klasse-B-Anweisungsvorlagen in **Fig. 6B** enthalten: 1) innerhalb der Kein-Datenspeicherzugriff- **605** Anweisungsvorlagen sind eine Anweisungsvorlage für Kein-Datenspeicherzugriff-, Schreibmaskensteuerung, Operation vom Teilrundungs-Steuertyp **612** und eine Anweisungsvorlage für Kein-Datenspeicherzugriff, Schreibmaskensteuerung, Operation vom vsize-Typ **617** gezeigt. und 2) innerhalb der Datenspeicherzugriff- **620** Anweisungsvorlage ist eine Anweisungsvorlage für Datenspeicherzugriff, Schreibmaskensteuerung **627** gezeigt.

**[0050]** Das generische vektorfreundliche Anweisungsformat **600** enthält die folgenden Felder, die nachstehend in der in den **Fig. 6A-6B** dargestellten Reihenfolge aufgelistet sind.

Formatfeld 640 - ein spezifischer Wert (ein

**[0051]** Anweisungsformatbezeichnerwert) in diesem Feld identifiziert das vektorfreundliche Anweisungsformat eindeutig, und somit das Auftreten der Anweisungen in dem vektorfreundlichen Anweisungsformat in Anweisungsströmen. Somit ist dieses Feld optional in dem Sinn, dass es nicht benötigt wird für einen Befehlssatz, der nur das generische vektorfreundliche Anweisungsformat aufweist.

**[0052]** Basisoperationsfeld **642** - sein Inhalt unterscheidet unterschiedliche Basisoperationen.

**[0053]** Registerindexfeld **644** - sein Inhalt spezifiziert direkt oder durch Adressenerzeugung die Orte der Quell- und Zieloperanden, seien sie in Registern oder im Datenspeicher. Diese enthalten eine ausreichende Anzahl von Bits, um N Register aus einer PxQ- (z. B. 32x512-, 16x128-, 32x1024-, 64x1024-) Registerdatei auszuwählen. Während in einer Ausführungsformen N bis zu drei Quell-und ein Zielregister sein kann, können alternative Ausführungsformen mehr oder weniger Quell- und Zielregister unterstützen (können z. B. bis zu zwei Quellen unterstützen, wobei eine dieser Quellen auch als das Ziel agiert, können bis zu drei Quellen unterstützen, wobei eine dieser Quellen auch als das Ziel agiert, können bis zu zwei Quellen und ein Ziel unterstützen).

**[0054]** Modifiziererfeld **646** - sein Inhalt unterscheidet das Auftreten von Anweisungen in dem generischen vektorfreundlichen Anweisungsformat, die Datenspeicherzugriff spezifizieren, von denen, die das nicht tun; das heißt, zwischen Anweisungsvorlagen für Kein-Datenspeicherzugriff **605** und Anweisungsvorlagen für Datenspeicherzugriff **620**. Datenspeicherzugriffsoperationen lesen und/oder schreiben in die Datenspeicherhierarchie (die in einigen Fällen die Quell- und/oder Zieladressen unter Verwendung von Werten in Registern spezifizieren), während Kein-Datenspeicherzugriffsoperationen das nicht tun (z. B. die Quelle und die Ziele sind Register). Während in einer Ausführungsform dieses Feld auch unter drei unterschiedlichen Arten zum Ausführen von Datenspeicheradressenberechnungen auswählt, können alternative Ausführungsformen mehr, weniger oder unterschiedliche Arten zum Ausführen von Datenspeicheradressenberechnungen unterstützen.

**[0055]** Erweiterungsoperationsfeld **650** - sein Inhalt unterscheidet, welche aus einer Vielzahl von unterschiedlichen Operationen zusätzlich zu der Basisoperationen ausgeführt werden soll. Dieses Feld ist kontextspezifisch. In einer Ausführungsform der Erfindung ist dieses Feld in ein Klassenfeld **668**, ein Alphafeld **652** und ein Betafeld **654** unterteilt. Das Erweiterungsoperationsfeld **650** ermöglicht, dass gewöhnliche Gruppen von Operationen in einer einzelnen Anweisung anstatt in 2, 3 oder 4 Anweisungen ausgeführt werden.

**[0056]** Skalierungsfeld **660** - sein Inhalt erlaubt die Skalierung des Inhalts des Indexfelds zur Datenspeicheradressenerzeugung (z. B. zur Adressenerzeugung, die $2^{scale}$ * Index + Basis verwendet).

**[0057]** Verlagerungsfeld **662A** - sein Inhalt wird als Teil der Datenspeicheradressenerzeugung verwendet (z. B. zur Adressenerzeugung, die $2^{scale}$ * Index + Basis + Verlagerung verwendet).

**[0058]** Verlagerungsfaktorfeld **662B** (es wird darauf hingewiesen, dass die Nebeneinanderstellung des Verlagerungsfelds **662A** direkt über dem Verlagerungsfaktorfeld **662B** angibt, das eines oder das andere verwendet wird) - sein Inhalt wird als Teil der Adressenerzeugung verwendet; es spezifiziert einen Verlagerungsfaktor, der durch die Größe eines Datenspeicherzugriffs (N) skaliert werden soll - wobei N die Anzahl von Bytes in dem Datenspeicherzugriff ist (z. B. zur Adressenerzeugung, die $2^{scale}$ * Index + Basis + skalierte Verlagerung verwendet). Redundante niederwertige Bits werden ignoriert, und somit wird der Inhalt des Verlagerungsfaktorfelds mit der Gesamtgröße (N) der Datenspeicheroperanden multipliziert, um die endgültige Verlagerung zu erzeugen, die beim Berechnen einer effektiven Adresse verwendet werden soll. Der Wert von N wird durch die Prozessorhardware zur Laufzeit basierend auf dem vollständigen Opcodefeld **674** (hier später beschrieben) und dem Datenmanipulationsfeld **654C** bestimmt. Das Verlagerungsfeld **662A** und das Verlagerungsfaktorfeld **662B** sind in dem Sinn optional, dass sie nicht für die Anweisungsvorlagen für Kein-Datenspeicherzugriff **605** verwendet werden und/oder andere Ausführungsformen nur eines oder keines der beiden implementieren können.

**[0059]** Datenelementbreitefeld **664** - sein Inhalt unterscheidet, welche aus einer Anzahl von Datenelementbreiten verwendet werden soll (in einigen Ausführungsformen für alle Anweisungen; in anderen Ausführungsformen nur für einige der Anweisungen). Dieses Feld ist in dem Sinn optional, dass es nicht benötigt wird, falls nur eine Datenelementbreite unterstützt wird und/oder Datenelementbreiten unterstützt werden, die einen Aspekt der Opcodesverwenden.

**[0060]** Schreibmaskenfeld **670** - sein Inhalt steuert auf einer Basis pro Datenelementposition, ob diese Datenelementposition in dem Zielvektoroperanden das Ergebnis der Basisoperation und der Erweiterungsoperation widerspiegelt. Klasse-A-Anweisungsvorlagen unterstützen Zusammenfassen-Schreibmaskieren, während Klasse-B-Anweisungsvorlagen sowohl Zusammenfassen- als auch Nullsetzen-Schreibmaskieren unterstützen. Beim Zusammenfassen ermöglichen Vektormasken, dass irgendeine Menge von Elementen in dem Ziel gegen Aktualisierungen während der Ausführung irgendeiner Operation (spezifiziert durch die Basisoperation und die Erweiterungsoperation) geschützt sind; in einer anderen Ausführungsform Beibehalten des alten Werts jedes Elements des Ziels, wo das entsprechende Masken-Bit eine 0 aufweist. Im Gegensatz dazu ermöglichen beim Nullsetzen Vektormasken, dass irgendeine Menge von Elementen in dem Ziel während der Ausführung irgendeiner Operation (spezifiziert durch die Basisoperation und die Erweiterungsoperation) auf Null eingestellt wird; in einer Ausführungsform wird ein Element des Ziels auf 0 eingestellt, wenn das entsprechende Masken-Bit einen 0-Wert aufweist. Eine Teilmenge dieser Funktionalität ist die Fähigkeit, die Vektorlänge der ausgeführten Operation (das heißt die Spanne von Elementen, die modifiziert werden, von dem ersten zum letzten) zu steuern; es ist jedoch nicht notwendig, dass die Elemente, die modifiziert werden, fortlaufend sind. Somit erlaubt das Schreibmaskenfeld **670** partielle Vektoroperationen, die Laden, Speichern, Arithmetik, Logik usw. enthalten. Obwohl Ausführungsformen der Erfindung beschrieben sind, in denen der Inhalt des Schreibmaskenfelds **670** eines aus einer Anzahl von Schreibmaskenregistern auswählt, das die Schreibmaske enthält, die verwendet werden soll (und somit der Inhalt des Schreibmaskenfelds **670** indirekt identifiziert, dass Maskieren ausgeführt werden soll), ermöglichen es alternative Ausführungsformen stattdessen oder zusätzlich, dass der Inhalt des Schreibmaskenfelds **670** direkt spezifiziert, dass Maskieren ausgeführt werden soll.

**[0061]** Immediate-Feld **672** - sein Inhalt ermöglicht die Spezifikation eines Immediate. Dieses Feld ist in dem Sinn optional, dass es in einer Implementierung eines generischen vektorfreundlichen Formats nicht vorhanden ist, das Immediate nicht unterstützt, und es in Anweisungen nicht enthalten ist, die ein Immediate nicht verwenden.

**[0062]** Klassenfeld **668** - sein Inhalt unterscheidet zwischen unterschiedlichen Klassen von Anweisungen. Mit Bezug auf die **Fig. 6A-B** wählt der Inhalt dieses Felds zwischen Klasse-A- und Klasse-B-Anweisungen aus. In den **Fig. 6A-B** sind Rechtecke mit gerundeten Ecken verwendet, um anzugeben, dass ein spezifischer Wert in einem Feld vorhanden ist (z. B. Klasse A 668A und Klasse B 668B für das Klassenfeld **668** jeweils in den **Fig. 6A-B**).

Anweisungsvorlagen der Klasse A

**[0063]** In dem Fall der Anweisungsvorlagen für Kein-Datenspeicherzugriff **605** der Klasse A wird das Alphafeld **652** als ein RS-Feld **652A** interpretiert, dessen Inhalt unterscheidet, welcher aus den unterschiedlichen Erweiterungsoperationstypen ausgeführt werden soll (z. B. sind Runden **652A**.1 und Datentransformation **652A**.2 jeweils für den Kein-Datenspeicherzugriff, die Rundungstyp-Operation **610** und die Kein-Datenspeicherzugriff-, Datentransformationstypoperation- **615** Anweisungsvorlagen spezifiziert), während das Betafeld **654** unterscheidet, welche von den Operationen des spezifizierten Typs ausgeführt werden soll. In den Anweisungsvorlagen für Kein-Datenspeicherzugriff **605** sind das Skalierungsfeld **660**, das Verlagerungsfeld **662A** und das Verlagerungsskalierungsfeld **662B** nicht vorhanden.

Kein-Datenspeicherzugriff-Anweisungsvorlagen - Operation vom Vollrundungs-Steuertyp

**[0064]** In der Anweisungsvorlage für den Kein-Datenspeicherzugriff für Operationen vom Vollrundungs-Steuertyp **610** wird das Betafeld **654** als ein Rundungssteuertypfeld **654A** interpretiert, dessen Inhalt(e) statisches Runden bereitstellen. Obwohl in den beschriebenen Ausführungsformen der Erfindung das Rundungssteuerfeld **654A** ein Feld **656** zum Unterdrücken aller Gleitkommaausnahmen (SAE-Feld) und ein Rundungsoperationssteuerfeld **658** enthält, können alternative Ausführungsformen das Codieren dieser beiden Konzepte in das gleiche Feld unterstützen oder können nur eines oder das andere dieser Konzepte/Felder aufweisen (z. B. können nur das Rundungsoperationssteuerfeld **658** aufweisen).

**[0065]** SAE-Feld **656** - sein Inhalt unterscheidet, ob das Ausnahmeereignisberichten deaktiviert werden soll oder nicht; wenn der Inhalt des SAE-Felds **656** angibt, dass Unterdrückung aktiviert ist, meldet eine gegebene Anweisung keine Art von Gleitkommaausnahme-Flag und ruft keinen Gleitkommaausnahme-Handler auf.

**[0066]** Rundungsoperationssteuerfeld **658** - sein Inhalt unterscheidet, welche aus einer Gruppe von Rundungsoperationen ausgeführt werden soll (z. B. Aufrunden, Abrunden, Runden auf Null und Runden auf die nächstgelegene Ganzzahl). Somit ermöglicht das Rundungsoperationssteuerfeld **658** das Ändern des Run-

dungsmodus auf einer Basis pro Anweisung. In einer Ausführungsform der Erfindung, in der ein Prozessor ein Steuerregister zum Spezifizieren von Steuerungsmodi enthält, überschreibt der Inhalt des Rundungsoperationssteuerfelds **650** diesen Registerwert.

Anweisungsvorlagen für Kein-Datenspeicherzugriff - Operation vom Datentransformationstyp

**[0067]** In der Anweisungsvorlage für Kein-Datenspeicherzugriff für Operationen vom Datentransformationstyp **615** wird das Betafeld **654** als ein Datentransformationsfeld **654B** interpretiert, dessen Inhalt unterscheidet, welche aus einer Anzahl von Datentransformationen ausgeführt werden soll (z. B. keine Datentransformation, Swizzle, Broadcast).

**[0068]** In dem Fall einer Anweisungsvorlage für Datenspeicherzugriff **620** von Klasse A wird das Alphafeld **652** als ein Räumungshinweisfeld **652B** interpretiert, dessen Inhalt unterscheidet, welcher aus den Räumungshinweisen verwendet werden soll (in **Fig. 6A** temporär **652B.1** und nicht-temporär **652B.2** sind jeweils für die Anweisungsvorlage für Datenspeicherzugriff zeitlich **625** bzw. die Anweisungsvorlage für Datenspeicherzugriff, nicht-temporär **630** spezifiziert), während das Betafeld **654** als ein Datenmanipulationsfeld **654C** interpretiert wird, dessen Inhalt unterscheidet, welche aus einer Anzahl von Datenmanipulationsoperationen (auch als Grundelemente bekannt) ausgeführt werden soll (z. B. keine Manipulation; Broadcast; Aufwärtsumsetzung einer Quelle; und Abwärtsumsetzung eines Ziels). Die Anweisungsvorlagen für Datenspeicherzugriff **620** enthalten das Skalierungsfeld **660** und optional das Verlagerungsfeld **662A** oder das Verlagerungsskalenfeld **662B**.

**[0069]** Vektordatenspeicheranweisungen führen Vektorladen aus dem und Vektorspeichern in den Datenspeicher aus, mit Umsetzungsunterstützung. Wie bei regulären Vektoranweisungen übertragen Vektordatenspeicheranweisungen Daten aus dem/in den Datenspeicher datenelementweise, wobei die Elemente, die tatsächlich übertragen werden, durch die Inhalte der Vektormaske angeordnet sind, die als die Schreibmaske ausgewählt ist.

Anweisungsvorlagen für Datenspeicherzugriff - temporär

**[0070]** Temporäre Daten sind Daten, für die es wahrscheinlich ist, dass sie ausreichend bald wiederverwendet werden, um von Cachen zu profitieren. Das ist jedoch ein Hinweis, und unterschiedliche Prozessoren können ihn auf unterschiedliche Arten implementieren, was das vollständige Ignorieren des Hinweises einschließt.

Anweisungsvorlagen für Datenspeicherzugriff - nicht temporär

**[0071]** Nicht-temporäre Daten sind Daten, für die es unwahrscheinlich ist, dass sie ausreichend bald wiederverwendet werden, um vom Cachen im Cache 1. Ebene zu profitieren und denen Priorität zum Ausräumen gegeben werden sollte. Das ist jedoch in Hinweis, und unterschiedliche Prozessoren können ihn auf unterschiedliche Arten implementieren, was das vollständige Ignorieren des Hinweises einschließt.

Anweisungsvorlagen der Klasse B

**[0072]** In dem Fall der Anweisungsvorlagen der Klasse B wird das Alphafeld **652** als ein Schreibmaskensteuerungsfeld (Z-Feld) **652C** interpretiert, dessen Inhalt unterscheidet, ob das Schreibmaskieren, das durch das Schreibmaskenfeld **670** gesteuert ist, ein Zusammenfassen oder ein Nullsetzen sein sollte.

**[0073]** In dem Fall der Anweisungsvorlagen für Kein-Datenspeicherzugriff **605** der Klasse B wird ein Teil des Betafelds **654** als ein RL-Feld **657A** interpretiert, dessen Inhalt unterscheidet, welcher aus den unterschiedlichen Erweiterungsoperationstypen ausgeführt werden soll (z. B. Runden **657A**.1 und Vektorlänge (VSIZE) **657A**.2 sind jeweils für die Kein-Datenspeicherzugriff, Schreibmaskensteuerung, Teilrundungs-Steuertypoperation- **612** Anweisungsvorlage und die kein-Datenspeicherzugriff-, Schreibmaskensteuerung-, VSIZE-Typ-Operation- **617** Anweisungsvorlage spezifiziert), während der Rest des Betafelds **654** unterscheidet, welche aus den Operationen des spezifizierten Typs ausgeführt werden soll. In den Anweisungsvorlagen für Kein-Datenspeicherzugriff **605** sind das Skalierungsfeld **660**, das Verlagerungsfeld **662A** und das Verlagerungsskalierungsfeld **662B** nicht vorhanden.

**[0074]** In der Kein-Datenspeicherzugriff-, Schreibmaskensteuerung-, Teilrundungssteuertypoperation- **610** Anweisungsvorlage wird der Rest des Betafelds **654** als ein Rundungsoperationsfeld **659A** interpretiert, und das Ausnahmeereignisberichten ist deaktiviert (eine gegebene Anweisung berichtet keine Art von Gleitkommaausnahme-Flag und ruft keinen Gleitkommaausnahme-Handler auf).

**[0075]** Rundungsoperationssteuerfeld **659A** - genau wie das Rundungsoperationssteuerfeld **658** unterscheidet sein Inhalt, welche aus einer Gruppe von Rundungsoperationen ausgeführt werden soll (z. B. Aufrunden, Abrunden, Runden auf Null und Runden auf die nächstgelegene Ganzzahl). Somit ermöglicht das Rundungsoperationssteuerfeld **659A** das Ändern des Rundungsmodus auf einer Basis pro Anweisung. In einer Ausführungsform der Erfindung, in der ein Prozessor ein Steuerregister zum Spezifizieren von Steuerungsmodi enthält, überschreibt der Inhalt des Rundungsoperationssteuerfelds **650** diesen Registerwert.

**[0076]** In der Kein-Datenspeicherzugriff-, Schreibmaskensteuerung-, VSIZE-Typ-Operation- **617** Anweisungsvorlage wird der Rest des Betafelds **654** als ein Vektorlängenfeld **659B** interpretiert, dessen Inhalt unterscheidet, auf welcher aus einer Anzahl von Datenvektorlängen gearbeitet werden soll (z. B. 128, 256 oder 512 Byte).

**[0077]** In dem Fall einer Anweisungsvorlage für Datenspeicherzugriff **620** der Klasse B wird ein Teil des Betafelds **654** als ein Broadcast-Feld **657B** interpretiert, dessen Inhalt unterscheidet, ob die Manipulationsoperation vom Broadcast-Typ ausgeführt werden soll oder nicht, während der Rest des Betafelds **654** als das Vektorlängenfeld **659B** interpretiert wird. Die Anweisungsvorlagen für Datenspeicherzugriff **620** enthalten das Skalierungsfeld **660** und optional das Verlagerungsfeld **662A** oder das Verlagerungsskalierungsfeld **662B**.

**[0078]** In Bezug auf das generische vektorfreundliche Anweisungsformat **600** ist ein vollständiges Opcodefeld **674** gezeigt, das das Formatfeld **640**, das Basisoperationsfeld **642** und das Datenelementbreitenfeld **664** enthält. Während eine Ausführungsform gezeigt ist, in der das vollständige Opcodefeld **674** alle diese Felder enthält, enthält das vollständige Opcodefeld **674** weniger als alle diese Felder in Ausführungsformen, die sie nicht alle unterstützen. Das vollständige Opcodefeld **674** stellt den Operationscode (Opcode) bereit.

**[0079]** Das Erweiterungsoperationsfeld **650**, das Datenelementbreitenfeld **664** und das Schreibmaskenfeld **670** ermöglichen, dass diese Merkmale in dem generischen vektorfreundlichen Anweisungsformat pro Anweisung spezifiziert werden.

**[0080]** Die Kombination des Schreibmaskenfelds und des Datenelementbreitenfelds erzeugt typisierte Anweisungen, in denen sie ermöglichen, dass eine Maske basierend auf unterschiedlichen Datenelementbreiten angewandt wird.

**[0081]** Die verschiedenen Anweisungsvorlagen, die innerhalb der Klasse A und Klasse B zu finden sind, sind in unterschiedlichen Situationen nützlich. In einigen Ausführungsformen der Erfindung können unterschiedliche Prozessoren oder unterschiedliche Kerne innerhalb eines Prozessors nur Klasse A, nur Klasse B oder beide Klassen unterstützen. Beispielsweise kann ein Außerhalb-der-Reihenfolge-Allzweck-Hochleistungskern, der für Allzweckberechnung vorgesehen ist, nur Klasse B unterstützen, ein Kern, der primär für Grafik und/oder wissenschaftliche Berechnung (mit hohem Durchsatz) vorgesehen ist, kann nur Klasse A unterstützen, und ein Kern, der für beides vorgesehen ist, kann beides unterstützen (selbstverständlich ist ein Kern, der eine Mischung von Vorlagen und Anweisungen aus beiden Klassen jedoch nicht alle Vorlagen und Anweisungen aus beiden Klassen aufweist, innerhalb des Geltungsbereichs der Erfindung). Außerdem kann ein einzelner Prozessor mehrere Kerne enthalten, von denen alle dieselbe Klasse unterstützen, oder in dem unterschiedliche Kerne unterschiedliche Klassen unterstützen. Beispielsweise in einem Prozessor mit separaten Grafik- und Allzweck-Kernen kann einer der Grafik-Kerne, der primär für Grafik und/oder wissenschaftliche Berechnung vorgesehen ist, nur Klasse A unterstützen, während einer oder mehrere der Allzweck-Kerne Hochleistungs-Allzweck-Kerne mit Außerhalb-der-Reihenfolge-Ausführung und Registerumbenennung sein können, die für Allzweck-Berechnung vorgesehen sind, die nur Klasse B unterstützen. Ein weiterer Prozessor der keinen separaten Grafik-Kern aufweist, kann einen oder mehrere In-der-Reihenfolge- oder Außerhalb-der-Reihenfolge-Allzweck-Kerne aufweisen, die sowohl Klasse A als auch Klasse B unterstützen. Selbstverständlich können Merkmale aus einer Klasse in unterschiedlichen Ausführungsformen der Erfindung auch in der anderen Klasse implementiert sein. Programme, die in einer Hochsprache geschrieben sind, würden (z. B. „just in time“-kompiliert oder statisch kompiliert) in eine Vielzahl unterschiedlicher ausführbarer Formen gebracht werden, die enthalten: 1) eine Form, die nur Anweisungen der/den Klasse(n) aufweisen, die durch den Zielprozessor zur Ausführung unterstützt werden; oder 2) eine Form, die alternative Routinen aufweist, die unter Verwendung unterschiedlicher Kombinationen der Anweisungen aller Klassen geschrieben sind und die Steuerflusscode aufweisen, der die Routinen zum Ausführen basierend auf den Anweisungen auswählt, die von dem Prozessor unterstützt werden, der derzeit den Code ausführt.

Beispielhaftes spezifisches vektorfreundliches Anweisungsformat

**[0082]** **Fig. 7A** ist ein Blockdiagramm, das ein beispielhaftes spezifisches vektorfreundliches Anweisungsformat gemäß Ausführungsformen der Erfindung darstellt. **Fig. 7A** zeigt ein spezifisches vektorfreundliches Anweisungsformat **700**, das in dem Sinn spezifisch ist, dass es sowohl den Ort, die Größe, die Interpretation und die Reihenfolge als auch Werte für einige dieser Felder spezifiziert. Das spezifische vektorfreundliche Anweisungsformat **700** kann verwendet werden, um den x86-Befehlssatz zu erweitern, und somit sind einige der Felder ähnlich oder gleich denjenigen, die in dem existierenden x86-Befehlssatz und seinen Erweiterungen (z. B. AVX) verwendet werden. Dieses Format bleibt mit dem Präfix-Codierungsfeld, dem Real-Opcode-Byte-Feld, MOD R/M-Feld, SIB-Feld, Verlagerungsfeld und Immediate-Feldern des existierenden x86-Befehlssatzes mit Erweiterungen konsistent. Die Felder aus **Fig. 6**, in die die Felder von **Fig. 7A** abbilden, sind dargestellt.

**[0083]** Es ist zu verstehen, dass, obwohl Ausführungsformen der Erfindung mit Bezug auf das spezifische vektorfreundliche Anweisungsformat **700** in dem Kontext des generischen vektorfreundlichen Anweisungsformat **600** zu anschaulichen Zwecken beschrieben sind, die Erfindung nicht auf das spezifische vektorfreundliche Anweisungsformat **700** beschränkt ist, außer wo es beansprucht ist. Beispielsweise berücksichtigt das generische vektorfreundliche Anweisungsformat **600** eine Vielzahl möglicher Größen für die verschiedenen Felder, während das spezifische vektorfreundliche Anweisungsformat **700** so gezeigt ist, dass es Felder mit spezifischen Größen aufweist. Als ein spezifisches Beispiel ist, obwohl das Datenelementbreitenfeld **664** als ein Ein-Bit-Feld in dem spezifischen vektorfreundlichen Anweisungsformat **700** dargestellt ist, die Erfindung nicht so eingeschränkt (das heißt, das generische vektorfreundliche Anweisungsformat **600** berücksichtigt andere Größen des Datenelementbreitenfelds **664**).

**[0084]** Das generische vektorfreundliche Anweisungsformat **600** enthält die folgenden Felder, die nachstehend in der in **Fig. 7A** dargestellten Reihenfolge aufgelistet sind.

EVEX-Präfix (Bytes 0-3) 702 - ist in einer Vier-Byte-Form codiert.

**[0085]** Formatfeld **640** (EVEX Byte 0, Bits [7:0]) - das erste Byte (EVEX Byte 0) ist das Formatfeld **640**, und es enthält 0x62 (den eindeutigen Wert, der zum Unterscheiden des vektorfreundlichen Anweisungsformat in einer Ausführungsform der Erfindung).

**[0086]** Die zweiten - vierten Bytes (EVEX-Bytes 1-3) enthalten eine Anzahl von Bitfeldern, die spezifische Fähigkeiten bereitstellen.

**[0087]** REX-Feld **705** (EVEX-Byte-1, Bits [7-5]) -besteht aus einem EVEX.R-Bitfeld (EVEX-Byte 1, Bit [7] - R), EVEX.X-Bitfeld (EVEX-Byte 1, Bit [6] - X) und 657BEX-Byte 1, Bit[5] - B). Die EVEX.R-, EVEX.X- und EVEX.B-Bitfelder stellen die gleiche Funktionalität bereit wie die entsprechenden VEX-Bitfelder und sind unter Verwendung von 1s-Komplementform codiert, z. B. ZMM0 ist als 1111B codiert, ZMM15 ist als 0000B codiert. Andere Felder der Anweisungen codieren die niederwertigen Bits der Registerindizes wie im Stand der Technik (rrr, xxx und bbb), so das Rrrr, Xxxx und Bbbb durch Hinzufügen von EVEX.R, EVEX.X und EVEX.B gebildet werden können.

**[0088]** REX'-Feld **610** - das ist der erste Teil des REX'-Felds **610** und ist das EVEX.R'-Bitfeld (EVEX-Byte 1, Bit [4] - R'), das verwendet wird, um entweder die höheren 16 oder niedrigeren 16 der erweiterten 32-Registermenge zu codieren. In einer Ausführungsform der Erfindung ist dieses Bit, zusammen mit anderen, wie nachstehend angegeben, im Bitinvertierten Format gespeichert, um es (in der bekannten x86-32-Bit-Betriebsart) von der BOUND-Anweisung zu unterscheiden, deren Real-Opcode-Byte 62 ist, das jedoch in dem MOD R/M-Feld (nachstehend beschrieben) den Wert **11** in dem MOD-Feld nicht annimmt; alternative Ausführungsformen der Erfindung speichern dieses und die anderen angegebenen nachstehenden Bits nicht in dem invertierten Format. Ein Wert 1 ist verwendet, um die niedrigeren 16 Register zu codieren. Mit andere Worten wird R'Rrrr durch Kombinieren von EVEX.R', EVEX.R und den anderen RRR aus anderen Feldern gebildet.

**[0089]** Opcode-Abbildungsfeld **715** (EVEX-Byte 1, Bits [3:0] - mmmm) - sein Inhalt codiert ein impliziertes führendes Opcode-Byte (0F, 0F 38 oder 0F 3).

**[0090]** Datenelementbreitefeld **664** (EVEX-Byte 2, Bit [7] - W) - ist durch die Notation EVEX.W repräsentiert. EVEX.W wird verwendet, um die Granularität (Größe) des Datentyps zu definieren (entweder 32-Bit-Datenelemente oder 64-Bit-Datenelemente).

**[0091]** EVEX.vvvv **720** (EVEX-Byte 2, Bits [6:3]-vwv)- die Rolle von EVEX.wvv kann das Folgende enthalten: 1) EVEX.vvvv codiert den ersten Quellregisteroperanden, der in invertierter (1s-Komplement-) Form spezifiziert ist und für Anweisungen mit 2 oder mehr Quell-Operanden gültig ist; 2) EVEX.ww codiert den Zielregisteroperanden, spezifiziert in 1s-Komplementform für spezielle Vektorverschiebungen; oder 3) EVEX.vvvv codiert keinen Operanden, das Feld ist reserviert und sollte **1111b** enthalten. Somit codiert das EVEX.vvw-Feld **720** die 4 niederwertigen Bits des ersten Quellregister-Spezifizierers, die in intervierter (1s-Komplement-) Form gespeichert sind. Abhängig von der Anweisung wird ein zusätzliches unterschiedliches EVEX-Bitfeld verwendet, um die Spezifizierergröße auf 32 Register zu erweitern.

**[0092]** EVEX.U **668** Klassenfeld (EVEX-Byte 2, Bit [2]-U) - Falls EVEX.U = 0, gibt es Klasse A oder EVEX.U0 an; falls EVEX.U = 1, gibt es Klasse B oder EVEX.U1 an.

**[0093]** Präfix-Codierungsfeld **725** (EVEX-Byte 2, Bits [1:0]-pp) - stellt zusätzliche Bits für das Basisoperationsfeld bereit. Zusätzlich zum Bereitstellen von Unterstützung für alte SSE-Anweisungen in dem EVEX-Präfixformat hat das auch den Vorteil, das SIMD-Präfix kompakt zu machen (anstatt ein Byte zu benötigen, um das SIMD-Präfix auszudrücken, erfordert das EVEX-Präfix nur 2 Bits). In einer Ausführungsform sind, um alte SSE-Anweisungen, die ein SIMD-Präfix (66H, F2H, F3H) verwenden, sowohl im alten Format als auch in dem EVEX-Präfix-Format zu unterstützen, diese SIMD-Präfixe in das SIMD-Präfix-Codierungsfeld codiert; und werden zur Laufzeit in das alte SIMD-Präfix erweitert, bevor sie für den PLA des Decodierers bereitgestellt werden, (somit kann der PLA sowohl das alte als auch das EVEX-Format dieser alten Anweisungen ohne Modifikation ausführen). Obwohl neuere Anweisungen den Inhalt des EVEX-Präfix-Codierungsfelds direkt als eine Opcode-Erweiterung verwenden könnten, erweitern spezielle Ausführungsformen auf ähnliche Art aus Gründen der Konsistenz, ermöglichen jedoch, dass unterschiedliche Bedeutungen durch diese alten SIMD-Präfixe spezifiziert sind. Eine alternative Ausführungsform kann den PLA neu designen, um die 2-Bit-SIMD-Präfix-Codierungen zu unterstützen, und erfordert somit die Erweiterung nicht.

**[0094]** Alphafeld **652** (EVEX-Byte 3, Bit [7] - EH; auch als EVEX.EH, EVEX.rs, EVEX.RL, EVEX.Schreibmaskensteuerung und EVEX.N bekannt; auch mit α dargestellt), - wie vorstehend beschrieben, ist dieses Feld kontextspezifisch.

**[0095]** Betafeld **654** (EVEX-Byte 3, Bits [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 vorstehend beschrieben, ist dieses Feld kontextspezifisch.

**[0096]** REX'-Feld **610** - das ist der Rest des REX'-Felds und ist das EVEX.V'-Bitfeld (EVEX-Byte 3, Bit [3] - V'), das verwendet werden kann, um entweder die höheren 16 oder niedrigeren 16 der erweiterten 32-Registermenge zu codieren. Dieses Bit ist in dem bitinvertierten Format gespeichert. Ein Wert 1 ist verwendet, um die niedrigeren 16 Register zu codieren. Mit anderen Worten ist V'VVVV durch Kombinieren von EVEX.V', EVEX.vvvv gebildet.

**[0097]** Schreibmaskenfeld **670** (EVEX-Byte 3, Bits [2:0]-kkk) - sein Inhalt spezifiziert den Index eines Registers in den Schreibmaskenregistern, wie vorstehend beschrieben. In einer Ausführungsform der Erfindung weist der spezifische Wert EVEX.kkk=000 ein spezielles Verhalten auf, das impliziert, dass keine Schreibmaske für die spezielle Anweisung verwendet wird (das kann in einer Vielzahl von Arten implementiert sein, die das Verwendungen einer Schreibmaske, die für alle fest verdrahtet ist, oder Hardware, die die Maskierungs-Hardware umgeht, enthalten).

**[0098]** Das Real-Opcodefeld **730** (Byte 4) ist auch als das Opcode-Byte bekannt. Ein Teil des Opcode ist in diesem Feld spezifiziert.

**[0099]** Das MOD R/M-Feld **740** (Byte 5) enthält das MOD-Feld **742**, das Reg-Feld **744** und das R/M-Feld **746**. Wie vorstehend beschrieben unterscheidet der Inhalt des MOD-Felds **742** zwischen Datenspeicherzugriffs- und Kein-Datenspeicherzugriffs-Operationen. Die Rolle des Reg-Felds **744** kann auf zwei Situationen zusammengefasst werden: Codieren entweder des Zielregisteroperanden oder eines Quellregisteroperanden, oder als eine Opcode-Erweiterung behandelt und nicht verwendet zu werden, um irgendeinen Anweisungsoperanden zu codieren. Die Rolle des R/M-Felds **746** kann das Folgende enthalten: Codieren des Anweisungsoperanden, der eine Datenspeicheradresse referenziert, oder Codieren entweder des Zielregisteroperanden oder eines Quellregisteroperanden.

**[0100]** Skalierung-, Index-, Basis- (SIB) Byte (Byte 6) - Wie vorstehend beschrieben wird der Inhalt des Skalierungsfelds **650** zur Datenspeicheradressenerzeugung verwendet. SIB.xxx 754 und SIB.bbb **756** - auf Inhalte dieser Felder ist vorstehend mit Bezug auf die Registerindizes Xxxx und Bbbb Bezug genommen worden.

**[0101]** Verlagerungsfeld **662A** (Bytes 7-10) - wenn das MOD-Feld **742** 10 enthält, sind die Bytes 7-10 das Verlagerungsfeld **662A**, und es arbeitet gleich wie die alte 32-Bit-Verlagerung (disp32) und arbeitet mit einer Byte-Granularität.

**[0102]** Verlagerungsfaktorfeld **662B** (Byte **7**) - wenn das MOD-Feld **742 01** enthält, ist das Byte **7** das Verlagerungsfaktorfeld **662B**. Der Ort dieses Felds ist derselbe wie der der alten x86-Befehlssatz-8-Bit-Verlagerung (disp8), die mit Byte-Granularität arbeitet. Da disp8 um ein Vorzeichen erweitert ist, kann es nur zwischen -128- und 127-Byte-Versatze adressieren; hinsichtlich 64-Byte-Cache-Leitungen verwendet disp8 8 Bits, die auf nur vier wirklich nützliche Werte -128, -64, 0 und 64 eingestellt werden können; da häufig ein größerer Bereich benötigt wird, wird disp32 verwendet; disp32 erfordert jedoch 4 Bytes. Im Gegensatz zu disp8 und disp32 ist das Verlagerungsfaktorfeld **662B** eine Neuinterpretation von disp8; wenn das Verlagerungsfaktorfeld **662B** verwendet wird, wird die tatsächliche Verlagerung durch den Inhalt des Verlagerungsfaktorfelds multipliziert mit der Größe des Datenspeicheroperandenzugriffs (N) bestimmt. Dieser Typ von Verlagerung ist als disp8*N bezeichnet. Das reduziert die mittlere Anweisungslänge (ein einzelnes Byte wird für diese Verlagerung verwendet, jedoch mit einem viel größeren Bereich). Eine solche komprimierte Verlagerung basiert auf der Annahme, dass die effektive Verlagerung ein Vielfaches der Granularität des Datenspeicherzugriffs ist, und somit müssen die redundanten niederwertigen Bits des Adressenversatzes nicht codiert werden. Mit anderen Worten ersetzt das Verlagerungsfaktorfeld **662B** die alte x86-Befehlssatz-8-Bit-Verlagerung. Somit ist das Verlagerungsfaktorfeld **662B** auf die gleiche Weise codiert wie eine x86-Befehlssatz-8-Bit-Verlagerung (somit keine Änderungen in den ModRM/SIB-Codierungsregeln) mit der einzigen Ausnahme, dass disp8 auf disp8*N überladen ist. Mit anderen Worten gibt es keine Änderungen an den Codierungsregeln oder Codierungslängen, sondern nur in der Interpretation des Verlagerungswerts durch die Hardware (die die Verlagerung mit der Größe des Datenspeicheroperanden skalieren muss, um einen byteweisen Adressenversatz zu erhalten). Das Immediate-Feld **672** arbeitet wie vorstehend beschrieben.

Vollständiges Opcodefeld

**[0103]** **Fig. 7B** ist ein Blockdiagramm, das die Felder des spezifischen vektorfreundlichen Anweisungsformats **700**, die das vollständige Opcode-Feld **674** bilden, gemäß einer Ausführungsform der Erfindung darstellt. Insbesondere enthält das vollständige Opcodefeld **674** das Formatfeld **640**, das Basisoperationsfeld **642** und das Datenelementbreiten- (W-) Feld **664**. Das Basisoperationsfeld **642** enthält das Präfixcodierungsfeld **725**, das Opcode-Abbildungsfeld **715** und das Real-Opcodefeld **730**.

Registerindexfeld

**[0104]** **Fig. 7C** ist ein Blockdiagramm, das die Felder des spezifischen vektorfreundlichen Anweisungsformats **700**, die das Registerindexfeld **644** bilden, gemäß einer Ausführungsform der Erfindung darstellt. Insbesondere enthält das Registerindexfeld **644** das REX-Feld **705**, das REX'-Feld **710**, das MODR/M.reg-Feld **744**, das MODR/M.r/m-Feld **746**, das VVVV-Feld **720**, xxx-Feld **754** und das bbb-Feld **756**.

Erweiterungsoperationsfeld

**[0105]** **Fig. 7D** ist ein Blockdiagramm, das die Felder des spezifischen vektorfreundlichen Anweisungsformats **700**, die das Erweiterungsoperationsfeld **650** bilden, gemäß einer Ausführungsform der Erfindung darstellt. Wenn das Klassen- (U-) Feld **668 0** enthält, bedeutet es EVEX.U0 (Klasse A **668A**); wenn es 1 enthält, bedeutet es EVEX.U1 (Klasse B **668B**). Wenn U=0 ist und das MOD-Feld **742 11** enthält (was eine Kein-Datenspeicherzugriff-Operation bedeutet), wird das Alphafeld **652** (EVEX-Byte **3**, Bit [7] - EH) als das rs-Feld 652A interpretiert. Wenn das rs-Feld 652A eine 1 enthält (Runden **652A**.1), wird das Betafeld **654** (EVEX-Byte **3**, Bits [6:4]- SSS) als das Rundungssteuerfeld **654A** interpretiert. Das Rundungssteuerfeld **654A** enthält ein Ein-Bit-SAE-Feld **656** und ein Zwei-Bit-Rundungsoperationsfeld **658**. Wenn das rs-Feld 652A eine 0 enthält (Datentransformation **652A**.2), wird das Betafeld **654** (EVEX-Byte **3**, Bits [6:4]- SSS) als ein Drei-Bit-Datentransformationsfeld **654B** interpretiert. Wenn U=0 ist und das MOD-Feld **742 00**, **01** oder **10** enthält (was eine Datenspeicherzugriffsoperation bedeutet), wird das Alphafeld **652** (EVEX-Byte **3**, Bit [7] - EH) als das Räumungshinweis- (EH-) Feld **652B** interpretiert, und das Betafeld **654** (EVEX-Byte **3**, Bits [6:4]- SSS) wird als ein Drei-Bit-Datenmanipulationsfeld **654C** interpretiert.

**[0106]** Wenn U=1 ist, wird das Alphafeld **652** (EVEX-Byte 3, Bit [7] - EH) als das Schreibmaskensteuer- (Z-) Feld **652C** interpretiert. Wenn U=1 ist und das MOD-Feld **742** 11 enthält (was eine Kein-Datenspeicherzugriff-Operation bedeutet), wird ein Teil des Betafelds **654** (EVEX-Byte 3, Bit [4]- $\mathbf{S_0}$) als das RL-Feld **657A** interpretiert; wenn es eine 1 enthält (Runden **657A**.1) wird der Rest des Betafelds **654** (EVEX-Byte 3, Bit [6-5]- $\mathbf{S_{2-1}}$) als das Rundungsoperationsfeld **659A** interpretiert, während dann, wenn das RL-Feld **657A** eine 0 enthält (VSIZE 657.A2), der Rest des Betafelds **654** (EVEX-Byte 3, Bit [6-5]- $\mathbf{S_{2-1}}$) als das Vektorlängenfeld **659B** (EVEX-Byte 3,Bit [6-5]- $\mathbf{L_{1-0}}$) interpretiert wird). Wenn U=1 ist und das MOD-Feld **742** 00, 01 oder 10 enthält (was eine Datenspeicherzugriffsoperation bedeutet), wird das Betafeld **654** (EVEX-Byte **3**, Bits [6:4]- SSS) als das Vektorlängenfeld **659B** (EVEX-Byte 3, Bit [6-5]- $L_{1-0}$) und das Broadcast-Feld **657B** (EVEX-Byte 3, Bit [4]- B) interpretiert.

## Beispielhafte Registerarchitektur

**[0107]** **Fig. 8** ist ein Blockdiagramm einer Registerarchitektur **800** gemäß einer Ausführungsform der Erfindung. In der dargestellten Ausführungsform sind 32 Vektorregister **810** vorhanden, die 512 Bits breit sind; diese Register sind als zmm0 bis zmm31 bezeichnet. Die niederwertigen 256 Bits der niedrigen 16 zmm Register sind auf den Registern ymm0-16 überlagert. Die niederwertigen 128 Bits der niedrigen 16 zmm Register (die niederwertigen 128 Bits der ymm Register) sind auf den Registern xmm0-15 überlagert Das spezifische vektorfreundliche Anweisungsformat **700** arbeitet auf diesen überlagerten Registerdatei, wie in den nachstehenden Tabellen dargestellt ist.

| Anpassbare Vektorlänge | Klasse | Operationen | Register |
|---|---|---|---|
| Anweisungsvorlagen, die das Vektorlängenfeld 659B nicht enthalten | A (**Fig. 6A**; U=0) | 610, 615, 625, 630 | zmm Register (die Vektorlänge ist 64 Bytes) |
| | B (**Fig. 6B**; U=1) | 612 | zmm Register (die Vektorlänge ist 64 Bytes) |
| Anweisungsvorlagen, die das Vektorlängenfeld 659B enthalten | B (**Fig. 6B**; U=1) | 617, 627 | zmm-, ymm- oder xmm-Register (die Vektorlänge ist 64 Byte, 32 Byte oder 16 Byte) abhängig von dem Vektorlängenfeld 659B. |

**[0108]** Mit anderen Worten wählt das Vektorlängenfeld **659B** zwischen einer maximalen Länge und einer oder mehreren anderen kürzeren Längen aus, wobei jede solche kürzere Länge die Hälfte der Länge der vorangehenden Länge ist; und Anweisungsvorlagen ohne das Vektorlängenfeld **659B** arbeiten mit der maximalen Vektorlänge. Ferner arbeiten in einer Ausführungsform die Klasse-B-Anweisungsvorlagen des spezifischen vektorfreundlichen Anweisungsformats **700** auf gepackten oder skalaren Gleitkommadaten mit Einzel-/Doppelgenauigkeit und gepackten oder skalaren Ganzzahldaten. Skalare Operationen sind Operationen, die auf der niedrigstwertigen Datenelementposition in einem zmm/ymm/xmm-Register ausgeführt werden; die höherwertigen Datenelementpositionen werden entweder gleich gelassen, wie sie vor der Anweisung waren, oder auf Null gesetzt, abhängig von der Ausführungsform.

**[0109]** Schreibmaskenregister **815** - in der dargestellten Ausführungsform sind 8 Schreibmaskenregister (k0 bis k7) vorhanden, jedes 64 Bits groß. In einer alternativen Ausführungsform sind die Schreibmaskenregister **815** 16 Bits groß. Wie früher beschrieben kann in einer Ausführungsform der Erfindung das Vektormaskenregister k0 nicht als eine Schreibmaske verwendet werden; wenn die Codierung, die normalerweise k0 angeben würde, als eine Schreibmaske verwendet wird, wählt sie eine fest verdrahtete Schreibmaske 0xFFFF aus, was effektiv das Schreibmaskieren für diese Anweisung deaktiviert.

**[0110]** Allzweckregister **825** - in der dargestellten Ausführungsform sind sechzehn 64-Bit-Allzweckregister vorhanden, die zusammen mit den existierenden x86-Adressierungsmodi verwendet werden, um Datenspeicheroperanden zu adressieren. Diese Register sind durch die Namen RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP und R8 bis R15 bezeichnet.

**[0111]** Skalar-Gleitkommastack-Registerdatei (x87-Stack) 845, auf die die MMXpaketierte Ganzzahl-Flat-Registerdatei **850** als Alias abgebildet ist - in der dargestellten Ausführungsform ist der x87-Stack ein Acht-Elemente-Stack, der verwendet wird, um skalare Gleitkommaoperationen auf 32/64/80-Bit-Gleitkommadaten unter Verwendung der x87-Befehlssatzerweiterung auszuführen; während die MMX-Register verwendet werden,

um sowohl Operationen auf gepackten 64-Bit-Ganzzahldaten auszuführen, als auch um Operanden für einige Operationen zu halten, die zwischen den MMX- und XMM-Registern ausgeführt werden.

**[0112]** Alternative Ausführungsformen der Erfindung können breitere oder schmalere Register verwenden. Zusätzlich können alternative Ausführungsformen der Erfindung mehr, weniger oder andere Registerdateien und Register verwenden.

## Beispielhafte Kern-Architekturen, Prozessoren und Computer-Architekturen

**[0113]** Prozessorkerne können auf unterschiedliche Arten, für unterschiedliche Zwecke und in unterschiedlichen Prozessoren implementiert sein. Beispielsweise kann eine Implementierungen solcher Kerne enthalten: 1) einen In-der-Reihenfolge-Allzweck-Kern, der für Allzweck-Berechnung vorgesehen ist; 2) einen Außerhalb-der-Reihenfolge-Hochleistungs-Allzweck-Kern, der für Allzweck-Berechnung vorgesehen ist; 3) einen Allzweck-Kern, der primär für Grafik und/oder wissenschaftliche Berechnung (mit hohem Durchsatz) vorgesehen ist. Implementierungen unterschiedliche Prozessoren können enthalten: 1) eine CPU, die einen oder mehrere In-der-Reihenfolge-Allzweck-Kerne enthalten, die für Allzweck-Berechnung vorgesehen sind, und/oder einen oder mehrere Außerhalb-der-Reihenfolge-Allzweck-Kerne, die für Allzweck-Berechnung vorgesehen sind, enthält; und 2) einen Coprozessor, der einen oder mehrere Spezial-Kerne enthält, die primär für Grafik oder Wissenschaft (Durchsatz) vorgesehen sind. Solche unterschiedlichen Prozessoren führen zu unterschiedlichen Computersystem-Architekturen, die enthalten können: 1) den Coprozessor auf einem von der CPU getrennten Chip; 2) den Coprozessor auf einem separaten Baustein in derselben Baugruppe wie eine CPU; 3) den Coprozessor auf demselben Baustein wie die CPU (wobei in diesem Fall ein solcher Coprozessor manchmal als eine Speziallogik, wie z. B. integrierte Grafik- und/oder Wissenschafts- (Durchsatz-) Logik, oder als Spezialkerne bezeichnet ist); und 4) ein Einchipsystem, das auf demselben Baustein die beschriebene CPU (manchmal als der/die Anwendungskern(e) bezeichnet) oder Anwendungsprozessor(en), den vorstehend beschriebenen Coprozessor und zusätzliche Funktionalität enthalten kann. Beispielhafte Kern-Architekturen werden als Nächstes beschrieben, gefolgt von Beschreibungen der beispielhaften Prozessoren und Computer-Architekturen.

## Beispielhafte Kern-Architekturen

### Blockdiagramm des In-der-Reihenfolge- und Außerhalb-der-Reihenfolge-Kerns

**[0114]** **Fig. 9A** ist ein Blockdiagramm, das sowohl eine beispielhafte In-der-Reihenfolge-Pipeline als auch eine beispielhafte registerumbenennende Außerhalb-der-Reihenfolge-Ausgabe/Ausführungs-Pipeline gemäß Ausführungsformen der Erfindung darstellt. **Fig. 9B** ist ein Blockdiagramm, das sowohl eine beispielhafte Ausführungsform eines In-der-Reihenfolge-Architekturkerns als auch eines beispielhaften registerumbenennenden Außerhalb-der-Reihenfolge-Ausgabe/Ausführungs-Architekturkerns, der in einem Prozessor aufgenommen werden soll, gemäß Ausführungsformen der Erfindung darstellt. Die durchgezogen umrandeten Kästen in den **Fig. 9A-B** stellen die In-der-Reihenfolge-Pipeline und den In-der-Reihenfolge-Kern dar, während das optionale Hinzufügen der gestrichelten Kästen die registerumbenennende Außerhalb-der-Reihenfolge-Ausgabe/Ausführungs-Pipeline und -Kern darstellen. Unter der Annahme, dass der In-der-Reihenfolge-Aspekt eine Teilmenge der Außerhalb-der-Reihenfolge-Aspekts ist, wird der Außerhalb-der-Reihenfolge-Aspekt beschrieben.

**[0115]** In **Fig. 9A** enthält eine Prozessor-Pipeline **900** eine Abholstufe **902**, eine Längendecodierungsstufe **904**, eine Decodierungsstufe **906**, eine Zuweisungsstufe **908**, eine Umbenennungsstufe **910**, eine Planungsstufe (auch als Dispatch- oder Ausgabestufe bezeichnet) **912**, eine Registerlese/Datenspeicherlesestufe **914**, eine Ausführungsstufe **916**, eine Zurückschreib/Datenspeicherschreibstufe **918**, eine Ausnahmebehandlungsstufe **922** und eine Übergabestufe **924**.

**[0116]** **Fig. 9B** zeigt den Prozessorkern **990**, der eine Frontend-Einheit **930** gekoppelt mit einer Ausführungs-Engine-Einheit **950** enthält, und beide sind mit einer Datenspeichereinheit **970** gekoppelt. Der Kern **990** kann ein Kern zur Berechnung mit reduziertem Befehlssatz (RISC-Kern), ein Kern zur Berechnung mit einem komplexen Befehlssatz (CISC-Kern), ein Kern mit sehr langem Anweisungswort (VLIW-Kern) oder ein Hybrid- oder ein alternativer Kern-Typ sein. Als noch eine weitere Option kann der Kern **990** ein Spezialkern sein wie beispielsweise ein Netz- oder Kommunikationskern, eine Komprimierungs-Engine, ein Coprozessor-Kern, ein Kern einer Allzweckberechnungs-Grafikverarbeitungseinheit (GPGPU-Kern), ein Grafik-Kern oder dergleichen.

**[0117]** Die Frontend-Einheit **930** enthält eine Verzweigungsvorhersageeinheit **932**, die mit einer Anweisungs-Cache-Einheit **934** gekoppelt ist, die mit einem Anweisungsübersetzungspuffer (TLB) **936** gekoppelt ist, der mit einer Anweisungsabholeinheit **938** gekoppelt ist, die mit einer Decodiereinheit **940** gekoppelt ist. Die De-

codiereinheit **940** (oder der Decodierer) kann Anweisungen decodieren und als eine Ausgabe eine oder mehrere Mikrooperationen, Mikrocodeeintrittspunkte, Mikroanweisungen, andere Anweisungen oder andere Steuersignale erzeugen, die aus den ursprünglichen Anweisungen decodiert werden oder diese auf andere Weise widerspiegeln oder von ihnen abgeleitet sind. Die Decodiereinheit **940** kann unter Verwendung verschiedener unterschiedlicher Mechanismen implementiert sein. Beispiele für geeignete Mechanismen enthalten Nachschlagetabellen, Hardware-Implementierungen, programmierbare Logik-Arrays (PLAs), Mikrocode-Festwertspeicher (ROMs) usw., sind jedoch nicht darauf beschränkt. In einer Ausführungsform enthält der Kern **990** einen Mikrocode-ROM oder ein anderes Medium, das Mikrocode für spezielle Makroanweisungen speichert (z. B. in der Decodiereinheit **940** oder auf andere Weise innerhalb der Frontend-Einheit **930**). Die Decodiereinheit **940** mit einer Umbenennungs/Zuweisereinheit **952** in der Ausführungs-Engine-Einheit **950** gekoppelt.

**[0118]** Die Ausführungs-Engine-Einheit **950** enthält die Umbenennungs/Zuweisereinheit **952**, die mit einer Rückzugseinheit **954** und einer Gruppe aus einer oder mehreren Schedulereinheit(en) **956** gekoppelt ist. Die Schedulereinheit(en) **956** repräsentieren irgendeine Anzahl unterschiedlicher Scheduler, die Reservierungsstationen, zentrales Anweisungsfenster usw. enthalten. Die Schedulereinheit(en) **956** sind mit dem/den physikalischen Registerdatei(en)-Einheit(en) **958** gekoppelt. Jede der physikalischen Registerdatei(en)-Einheiten **958** repräsentiert eine oder mehrere physikalische Registerdateien, von denen unterschiedliche einen oder mehrere unterschiedliche Datentypen speichern, wie z. B. skalare Ganzzahl, skalare Gleitkommazahl, gepackte Ganzzahl, gepackte Gleitkommazahl, Vektorganzzahl, Vektorgleitkommazahl, Status (z. B. einen Anweisungszeiger, der die Adresse der nächsten auszuführenden Anweisung ist), usw. In einer Ausführungsform umfasst die physikalische Registerdatei(en)-Einheit **958** eine Vektorregistereinheit, eine Schreibmaskenregistereinheit und eine Skalarregistereinheit. Diese Registereinheiten können architektonische Vektorregister, Vektormaskenregister und Allzweckregister bereitstellen. Die physikalische Registerdatei(en)-Einheit(en) **958** sind durch die Rückzugseinheit **954** überlappt, um verschiedene Arten darzustellen, in denen Registerumbenennung und Außerhalb-der-Reihenfolge-Ausführung implementiert sein können (z. B. unter Verwendung eines Umordnungspuffers und einer Rückzugsregisterdatei(en); unter Verwendung einer Zukunftsdatei(en); unter Verwendung einer Registerabbildung und eines Pools von Registern; usw.). Die Rückzugseinheit **954** und die physikalische Registerdatei(en)-Einheit(en) **958** sind mit dem/den Ausführungscluster(n) 960 gekoppelt. Die Ausführungscluster **960** enthalten eine Gruppe aus einer oder mehreren Ausführungseinheiten **962** und eine Gruppe aus einer oder mehreren Datenspeicherzugriffseinheiten **964**. Die Ausführungseinheiten **962** können verschiedene Operationen (z. B. Verschiebungen, Addition, Subtraktion, Multiplikation) und auf verschiedenen Datentypen (z. B. Skalargleitkommazahl, gepackte Ganzzahl, gepackte Gleitkommazahl, Vektorganzzahl, Vektorgleitkommazahl) ausführen. Während einige Ausführungsformen eine Anzahl von Ausführungseinheiten enthalten können, die für spezifische Funktionen oder Gruppen von Funktionen dediziert sind, können andere Ausführungsformen nur eine Ausführungseinheit oder mehrere Ausführungseinheiten, die alle alle Funktionen ausführen, enthalten. Die Schedulereinheit(en) **956**, physikalischen Registerdatei(en)-Einheit(en) **958** und Ausführungscluster **960** sind so gezeigt, dass sie möglicherweise mehrfach vorhanden sind, weil spezielle Ausführungsformen separate Pipelines für spezielle Typen von Daten/Operationen erzeugen (z. B. eine Skalarganzzahl-Pipeline, eine Skalargleitkommazahl/gepackte Ganzzahl/gepackte Gleitkommazahl/Vektorganzzahl/Vektorgleitkommazahl-Pipeline und/oder eine Datenspeicherzugriffs-Pipeline, von denen jede ihre eigene Schedulereinheit, physikalische Registerdatei(en)-Einheit(en) und/oder Ausführungscluster aufweist - und in dem Fall einer separaten Datenspeicherzugriffs-Pipeline sind spezielle Ausführungsformen implementiert, in denen nur das Ausführungscluster dieser Pipeline die Datenspeicherzugriffseinheit(en) **964** aufweist). Es ist ebenfalls zu verstehen, dass dort, wo separate Pipelines verwendet werden, eine oder mehrere dieser Pipelines Außerhalb-der-Reihenfolge-Ausgabe/Ausführung sein kann und der Rest In-der-Reihenfolge.

**[0119]** Die Gruppe von Datenspeicherzugriffseinheiten **964** ist mit der Datenspeichereinheit **970** gekoppelt, die eine Daten-TLB-Einheit **972** enthält, die mit einer Daten-Cache-Einheit **974** gekoppelt ist, die mit einer Ebene-2- (L2-) Cache-Einheit **976** gekoppelt ist. In einer beispielhaften Ausführungsform können die Datenspeicherzugriffseinheiten **964** eine Ladeeinheit, eine Speicheradresseneinheit und eine Speicherdateneinheit enthalten, von denen jede mit der Daten-TLB-Einheit **972** in der Datenspeichereinheit **970** gekoppelt ist. Die Anweisungs-Cache-Einheit **934** ist ferner mit einer Ebene-2- (L2-) Cache-Einheit **976** in der Datenspeichereinheit **970** gekoppelt. Die **L2**-Cache-Einheit **976** ist mit einer oder mehreren anderen Cache-Ebenen und schließlich mit einem Hauptspeicher gekoppelt.

**[0120]** Als Beispiel kann die beispielhafte registerumbenennende Ausgabe/Ausführungs-Kern-Architektur die Pipeline **900** wie folgt implementieren: 1) das Anweisungsabholen **938** führt das Abholen und die Längendecodierungsstufen **902** und **904** aus; 2) die Decodiereinheit **940** führt die Decodierungsstufe **906** aus; 3) die Umbenennungs/Zuweisereinheit **952** führt die Zuweisungsstufe **908** und die Umbenennungsstufe **910** aus; 4) die Schedulereinheit(en) **956** führt/führen die Planungsstufe **912** aus; 5) die physikalischen Registerdatei(en)

-Einheit(en) **958** und die Datenspeichereinheit **970** führen die Registerlese/Datenspeicherlesestufe **914** aus; der Ausführungscluster **960** führt die Ausführungsstufe **916** aus; 6) die Datenspeichereinheit **970** und die physikalischen Registerdatei(en)-Einheit(en) **958** führen die Zurückschreib/Datenspeicherschreibstufe **918** aus; 7) verschiedene Einheiten können an der Ausnahmebehandlungsstufe **922** beteiligt sein; und 8) die Rückzugseinheit **954** und die physikalischen Registerdatei(en)-Einheit(en) **958** führen die Übergabestufe **924** aus.

**[0121]** Der Kern **990** kann einen oder mehrere Befehlssätze unterstützen (z. B. den x86-Befehlssatz (mit einigen Erweiterungen, die mit neueren Versionen hinzufügt worden sind), den MIPS-Befehlssatz von MIPS Technologies in Sunnyvale, CA; den ARM-Befehlssatz (mit optional zusätzlichen Erweiterungen wie z. B. NEON) von ARM Holdings in Sunnyvale, CA), die die hier beschriebenen Anweisung(en) enthalten. In einer Ausführungsform enthält der Kern **990** Logik zum Unterstützen einer Befehlssatzerweiterung für gepackte Daten (z. B. AVX1, AVX2), und ermöglicht dadurch, dass Operationen, die durch viele Multimedia-Anwendungen verwendet werden, unter Verwendung gepackter Daten ausgeführt werden.

**[0122]** Es ist zu verstehen, dass der Kern Multithreading unterstützen kann (Ausführen von zwei oder mehr parallelen Gruppen von Operationen oder Threads), und das auf eine Vielzahl von Arten tun kann, die Zeitscheiben-Multithreading, simultanes Multithreading (wobei ein einzelner physikalischer Kern einen logischen Kern für jeden der Threads, für die der physikalische Kern simultan Multithreading ausführt, bereitstellt) oder eine Kombination daraus (z. B. Zeitscheibenabholen und -decodieren und simultanes Multithreading danach, wie z. B. in der Intel® Hyperthreading-Technologie).

**[0123]** Obwohl die Registerumbenennung im Kontext der Außerhalb-der-Reihenfolge-Ausführung beschrieben ist, ist zu verstehen, dass Registerumbenennung in einer In-der-Reihenfolge-Architektur verwendet werden kann. Während die dargestellte Ausführungsform des Prozessors auch separate Anweisungs- und Daten-Cache-Einheiten **934/974** und eine gemeinsam verwendet L2-Cache-Einheit **976** enthält, können alternative Ausführungsformen eine einzelnen internen Cache für sowohl Anweisungen als auch Daten aufweisen, wie beispielsweise einen internen Ebene-1- (L1) Cache oder mehrere Ebenen von internem Cache. In einigen Ausführungsformen kann das System eine Kombination aus einem internen Cache und einem externen Cache, der außerhalb des Kerns und/oder des Prozessors ist, enthalten. Alternativ kann der gesamte Cache außerhalb des Kerns und/oder des Prozessors sein.

## Spezifische beispielhafte In-der-Reihenfolge-Kern-Architektur

**[0124]** **Fig. 10A-B** stellen ein Blockdiagramm einer spezifischeren beispielhaften In-der-Reihenfolge-Kern-Architektur dar, wobei der Kern einer von mehreren Logikblöcken (die andere Kerne des gleichen Typs und/oder unterschiedlicher Typen enthalten) in einem Chip sein können. Die Logikblöcke kommunizieren durch ein Zusammenschaltungsnetz mit hoher Bandbreite (z. B. ein Ringnetz) mit einiger fester Funktionslogik, Datenspeicher-I/O-Schnittstellen und anderer notwendiger I/O-Logik, abhängig von der Anwendung.

**[0125]** **Fig. 10A** ist ein Blockdiagramm eines einzelnen Prozessorkerns, zusammen mit seiner Verbindung zu dem bausteininternen Zusammenschaltungsnetz **1002** und mit seiner lokalen Teilmenge des Ebene-**2**-(L2-) Cache **1004** gemäß Ausführungsformen der Erfindung. In einer Ausführungsform unterstützt ein Anweisungsdecodierer **1000** den x86-Befehlssatz mit einer Erweiterung des Befehlssatzes für gepackte Daten. Ein **L1**-Cache **1006** ermöglicht Zugriffe mit geringer Latenz auf jeden Cache-Datenspeicher in die Skalar- und Vektoreinheiten. Während in einer Ausführungsform (um die Konstruktion zu vereinfachen) eine Skalareinheit **1008** und eine Vektoreinheit **1010** separate Registergruppen (die Skalarregister **1012** bzw. die Vektorregister **1014**) verwenden und Daten, die zwischen ihnen übertragen werden, in den Datenspeicher geschrieben und dann aus einem Ebene-1- (L1-) Cache **1006** zurück eingelesen werden, können alternative Ausführungsformen der Erfindung eine andere Herangehensweise verwenden (z. B. eine einzige Registergruppe verwenden oder einen Kommunikationspfad enthalten, der es ermöglicht, dass Daten zwischen den zwei Registerdateien übertragen werden, ohne geschrieben und zurückgelesen zu werden).

**[0126]** Die lokale Teilmenge des L2-Cache **1004** ist Teil eines globalen L2-Cache, der in separate lokale Teilmengen unterteilt ist, eine pro Prozessorkern. Jeder Prozessorkern besitzt einen direkten Zugriffspfad zu seiner eigenen lokalen Teilmenge des L2-Cache **1004**. Daten, die durch einen Prozessorkern gelesen werden, werden in seiner L2-Cache-Teilmenge **1004** gespeichert, und es kann schnell auf sie zugegriffen werden, parallel mit anderen Prozessorkernen, die auf ihre eigenen lokalen L2-Cache-Teilmengen zugreifen. Daten, die durch einen Prozessorkern geschrieben werden, werden in seiner eigenen lokalen L2-Teilmenge **1004** gespeichert und aus anderen Teilmengen entfernt, falls notwendig. Das Ringnetz stellt die Kohärenz für gemeinsam verwendete Daten sicher. Das Ringnetz ist bidirektional, um es Agenten wie z. B. Prozessorkernen **L2**-Caches

und anderen Logikblöcken zu ermöglichen, innerhalb des Chips miteinander zu kommunizieren. Jeder Ringdatenpfad ist 1012 Bits pro Richtung breit.

**[0127]** **Fig. 10B** ist eine vergrößerte Ansicht eines Teils des Prozessorkerns in **Fig. 10A** gemäß Ausführungsformen der Erfindung. **Fig. 10B** enthält sowohl einen L1-Daten-Cache-Teil 1006A des L1-Cache **1004** als auch mehr Einzelheiten bezüglich der Vektoreinheit **1010** und der Vektorregister **1014**. Insbesondere ist die Vektoreinheit **1010** eine 16-breite Vektorverarbeitungseinheit (VPU) (siehe eine 16-breite ALU **1028**), die eine oder mehrere aus Ganzzahl-, einfach genauer Gleitkommazahl- und doppelt genauer Gleitkommazahlanweisung ausführt. Die VPU unterstützt das Swizzling der Registereingaben mit der Swizzle-Einheit **1020**, die numerische Umsetzung mit den Numerikumsetzungseinheiten **1022A-B** und die Replizierung mit der Replizierungseinheit **1024** auf der Datenspeichereingabe. Schreibmaskenregister **1026** ermöglichen das Begründen resultierender Vektorschreibvorgänge.

**[0128]** **Fig. 11** ist ein Blockdiagramm eines Prozessors **1100**, der mehr als einen Kern aufweisen kann, eine integrierte Datenspeichersteuereinheit aufweisen kann und integrierte Grafik aufweisen kann, gemäß Ausführungsformen der Erfindung. Die durchgezogen umrandeten Kästen in **Fig. 11** stellen einen Prozessor **1100** mit einem einzigen Kern **1102A**, einen Systemagenten **1110**, eine Gruppe aus einer oder mehreren Bussteuereinheiten **1116** dar, während das optionale Hinzufügen der gestrichelten Kästen einen alternativen Prozessor **1100** mit mehreren Kernen **1102A-N**, einer Gruppe aus einer oder mehreren integrierten Datenspeichersteuereinheit(en) **1114** in der Systemagenteneinheit **1110** und einer Speziallogik **1108** darstellt.

**[0129]** Somit können unterschiedliche Implementierungen des Prozessors **1100** enthalten: 1) eine CPU mit einer Speziallogik **1108**, die integrierte Grafik und/oder wissenschaftliche (Durchsatz-) Logik (die einen oder mehrere Kerne enthalten kann) ist, und wobei die Kerne **1102A-N** ein oder mehrere Allzweckkerne (z. B. In-der-Reihenfolge-Allzweck-Kerne, Außerhalb-der-Reihenfolge-Allzweck-Kerne, eine Kombination der beiden) sind; 2) einen Coprozessor, wobei die Kerne **1102A-N** eine große Anzahl von Spezialkernen sind, die primär für Grafik und/oder wissenschaftlich (Durchsatz) vorgesehen sind; und 3) einen Coprozessor, wobei die Kerne **1102A-N** eine große Anzahl von In-der-Reihenfolge-Allzweck-Kernen sind. Somit kann der Prozessor **1100** ein Allzweckprozessor, Coprozessor oder Spezialprozessor sein, wie beispielsweise ein Netz- oder Kommunikationsprozessor, eine Kompressions-Engine, ein Grafikprozessor, eine GPGPU (Allzweck-Grafikverarbeitungseinheit), ein Coprozessor mit hohem Durchsatz und vielen integrierten Kernen (MIC) (der 30 oder mehr Kerne enthält), ein eingebetteter Prozessor oder dergleichen. Der Prozessor kann auf einem oder mehreren Chips implementiert sein. Der Prozessor **1100** kann ein Teil eines und/oder kann auf einem oder mehreren Substraten unter Verwendung irgendeiner Anzahl von Prozesstechnologien, wie beispielsweise BiCMOS, CMOS oder NMOS, implementiert sein.

**[0130]** Die Datenspeicherhierarchie enthält eine oder mehrere Cache-Ebenen innerhalb der Kerne, eine Gruppe oder eine oder mehrere gemeinsam verwendete Cache-Einheiten **1106** und externen Datenspeicher (nicht gezeigt), der mit der Gruppe integrierter Datenspeichersteuereinheiten **1114** gekoppelt ist. Die Gruppen gemeinsam verwendeter Cache-Einheiten **1106** können einen oder mehrere Caches mittlerer Ebene, wie z. B. Ebene-2- (L2-), Ebene-3- (L3-), Ebene-4- (L4-) und anderer Cache-Ebenen, einen Cache der letzten Ebene (LLC) und/oder Kombinationen daraus enthalten. Während in einer Ausführungsform eine ringbasierte Zusammenschaltungseinheit **1112** die integrierte Grafiklogik **1108** (integrierte Grafiklogik **1108** ist ein Beispiel dafür und ist hier auch als Speziallogik bezeichnet), die Gruppe gemeinsam verwendeter Cache-Einheiten **1106** und die Systemagenteneinheit 1110/integrierte Datenspeichersteuereinheit(en) **1114** zusammenschaltet, können alternative Ausführungsformen irgendeine Anzahl bekannter Techniken zum Zusammenschalten solcher Einheiten verwenden. In einer Ausführungsform wird die Kohärenz zwischen einer oder mehreren Cache-Einheiten **1106** und den Kernen **1102-A-N** aufrechterhalten.

**[0131]** In einigen Ausführungsformen sind ein oder mehrere der Kerne **1102A-N** zum Multithreading fähig. Der Systemagent **1110** enthält diese Komponenten, die die Kerne **1102A-N** koordinieren und betreiben. Die Systemagenteneinheit **1110** kann beispielsweise eine Stromversorgungssteuereinheit (PCU) und eine Anzeigeeinheit enthalten. Die PCU kann Logik und Komponenten sein oder sie enthalten, die zum Regulieren des Stromversorgungszustands der Kerne **1102A-N** und der integrierten Grafiklogik **1108** benötigt werden. Die Anzeigeeinheit dient zum Ansteuern einer oder mehrere externer Anzeigevorrichtungen.

**[0132]** Die Kerne **1102A-N** können hinsichtlich des Architektur-Befehlssatzes homogen oder heterogen sein; das heißt, zwei oder mehr der Kerne **1102A-N** können zum Ausführen desselben Befehlssatzes fähig sein, während andere zum Ausführen nur einer Teilmenge dieses Befehlssatzes oder eines unterschiedlichen Befehlssatzes fähig sein können.

Beispielhafte Computer-Architekturen

**[0133]** Die **Fig. 12-15** sind Blockdiagramme beispielhafter Computer-Architekturen. Andere Systemkonstruktionen und -Konfigurationen aus dem Stand der Technik für Laptops, Desktops, tragbare PCs, persönliche digitale Assistenten, Technik-Workstations, Server, Netzvorrichtungen, Netz-Hubs, Switches, eingebettete Prozessoren, digitale Signalprozessoren (DSPs), Grafikvorrichtungen, Videospiel-Vorrichtungen, Set-Top-Boxen, Mikrosteuereinheiten, Mobiltelefone, tragbare Mediaplayer, tragbare Vorrichtungen und verschiedene andere elektronische Vorrichtungen sind ebenfalls geeignet. Im Allgemeinen ist eine große Vielzahl von Systemen oder elektronischen Vorrichtungen, die zum Integrieren eines Prozessors und/oder anderer Ausführungslogik wie sie hier offenbart ist, fähig sind, allgemein geeignet.

**[0134]** Jetzt Bezug nehmen auf **Fig. 12** ist ein Blockdiagramm eines Systems **1200** in Übereinstimmung mit einer Ausführungsform der vorliegenden Erfindung gezeigt. Das System **1200** kann einen oder mehrere Prozessoren **1210**, **1215** enthalten, die mit einem Steuereinheit-Hub **1220** gekoppelt sind. In einer Ausführungsform enthält der Steuereinheit-Hub **1220** einen Grafikdatenspeichersteuereinheit-Hub (GMCH) **1290** und einen Eingabe/Ausgabe-Hub (IOH) **1250** (die auf separaten Chips sein können); der GMCH **1290** enthält Datenspeicher- und Grafik-Steuereinheiten, an die Datenspeicher **1240** und ein Coprozessor **1245** gekoppelt sind; der IOH **1250** koppelt Eingabe/Ausgabe- (I/O-) Vorrichtungen **1260** mit dem GMCH **1290**. Alternativ sind eines oder beide aus den Datenspeicher- und den Grafik-Steuereinheiten innerhalb des Prozessors integriert (wie hier beschrieben), der Datenspeicher **1240** und der Coprozessor **1245** sind direkt mit dem Prozessor **1210** gekoppelt, und der Steuereinheit-Hub **1220** ist in einem einzigen Chip mit dem IOH **1250**.

**[0135]** Die optionale Beschaffenheit der zusätzlichen Prozessoren **1215** ist in **Fig. 12** durch gestrichelte Linien gekennzeichnet. Jeder Prozessor **1210**, **1215** kann einen oder mehrere der Verarbeitungskerne enthalten, die hier beschrieben sind, und kann eine Version des Prozessors **1100** sein.

**[0136]** Der Datenspeicher **1240** kann beispielsweise dynamischer Direktzugriffsspeicher (DRAM), Phasenwechselspeicher (PCM) oder eine Kombination aus den beiden sein. Für wenigstens eine Ausführungsform kommuniziert der Steuereinheit-Hub **1220** mit dem/den Prozessor(en) **1210**, **1215** über einen Mehrfachbus, wie z. B. einen vorderseitigen Bus (FSB), eine Punkt-zu-Punkt-Schnittstelle wie z. B. QuickPath Interconnect (QPI) oder eine ähnliche Verbindung **1295**.

**[0137]** In einer Ausführungsform ist der Coprozessor **1245** ein Spezialprozessor wie beispielsweise ein MIC-Prozessor mit hohem Durchsatz, ein Netz- oder Kommunikationsprozessor, eine Kompressions-Engine, ein Grafikprozessor, eine GPGPU, ein eingebetteter Prozessor oder dergleichen. In einer Ausführungsform kann der Steuereinheit-Hub **1220** einen integrierten Grafikbeschleuniger enthalten.

**[0138]** Es kann eine Vielzahl von Unterschieden zwischen den physikalischen Betriebsmitteln **1210**, **1215** hinsichtlich eines Spektrums von Metriken der Vorzüge vorhanden sein, die architektonische, mikroarchitektonische, thermische, Stromverbrauchs-Eigenschaften und dergleichen enthalten.

**[0139]** In einer Ausführungsform führt der Prozessor **1210** Anweisungen aus, die Datenverarbeitungsoperationen eines allgemeinen Typs steuern. In den Anweisungen können Coprozessor-Anweisungen eingebettet sein. Der Prozessor **1210** erkennt, dass diese Coprozessor-Anweisungen von einem Typ sind, der durch den angeschlossenen Coprozessor **1245** ausgeführt werden sollte. Dementsprechend gibt der Prozessor **1210** diese Coprozessor-Anweisungen (oder Steuersignale, die die Coprozessor-Anweisungen repräsentieren) auf einem Coprozessor-Bus oder einer anderen Zusammenschaltung zu dem Coprozessor **1245** aus. Der/die Coprozessor(en) **1245** nehmen die empfangenen Coprozessor-Anweisungen an und führen sie aus.

**[0140]** Jetzt Bezug nehmend auf **Fig. 13** ist ein Blockdiagramm eines ersten spezifischeren beispielhaften Systems **1300** in Übereinstimmung mit einer Ausführungsform der vorliegenden Erfindung gezeigt. Wie in **Fig. 13** gezeigt ist, ist das Mehrprozessor-System **1300** ein Punkt-zu-Punkt-Zusammenschaltungs-System und enthält einen ersten Prozessor **1370** und einen zweiten Prozessor **1380**, die über eine Punkt-zu-Punkt-Zusammenschaltung **1350** gekoppelt sind. Jeder aus den Prozessoren **1370** und **1380** kann eine Version des Prozessors **1100** sein. In einer Ausführungsform der Erfindung sind die Prozessoren **1370** und **1380** die Prozessoren **1210** bzw. **1215**, während der Coprozessor **1338** der Coprozessor **1245** ist. In einer weiteren Ausführungsform sind die Prozessoren **1370** und **1380** der Prozessor **1210** bzw. der Coprozessor **1245**.

**[0141]** Die Prozessoren **1370** und **1380** sind so gezeigt, dass sie die integrierten Datenspeichersteuereinheiten (IMC) **1372** bzw. **1382** enthalten. Der Prozessor **1370** enthält außerdem als Teil seiner Bus-Steuereinheiten

die Punkt-zu-Punkt- (P-P-) Schnittstellen **1376** und **1378**; ähnlich enthält der zweite Prozessor **1380** die P-P-Schnittstellen **1386** und **1388**. Die Prozessoren **1370**, **1380** können Informationen über eine Punkt-zu-Punkt-(P-P-) Schnittstelle **1350** unter Verwendung der P-P-Schnittstellenschaltungen **1378**, **1388** austauschen. Wie in **Fig. 13** gezeigt ist koppeln die IMCs **1372** und **1382** die Prozessoren mit jeweiligen Datenspeichern, und zwar einem Datenspeicher **1332** und einem Datenspeicher **1334**, die Abschnitte eines Hauptspeichers sein können, der lokal an die jeweiligen Prozessoren angeschlossen ist.

**[0142]** Die Prozessoren **1370**, **1380** können Informationen mit einem Chipsatz **1390** über die individuellen P-P-Schnittstellen **1352**, **1354** unter Verwendung der Punkt-zu-Punkt-Schnittstellenschaltungen **1376**, **1394**, **1386**, **1398** austauschen. Der Chipsatz **1390** kann optional Informationen mit dem Coprozessor **1338** über eine Hochleistungsschnittstelle **1392** austauschen. In einer Ausführungsform ist der Coprozessor **1338** ein Spezialprozessor wie beispielsweise ein MIC-Prozessor mit hohem Durchsatz, ein Netz- oder Kommunikationsprozessor, eine Kompressions-Engine, ein Grafikprozessor, eine GPGPU, ein eingebetteter Prozessor oder dergleichen.

**[0143]** Ein gemeinsam verwendeter Cache (nicht gezeigt) kann in einem Prozessor oder außerhalb beider Prozessoren, jedoch mit den Prozessoren über eine P-P-Zusammenschaltung verbunden, enthalten sein, so dass lokale Cache-Informationen eines Prozessors oder beider Prozessoren in dem gemeinsam verwendeten Cache gespeichert werden können, falls ein Prozessor in eine Niederleistungsbetriebsart versetzt wird.

**[0144]** Der Chipsatz **1390** kann mit einem ersten Bus **1316** über eine Schnittstelle **1396** gekoppelt sein. In einer Ausführungsform kann der erste Bus **1316** ein „Peripheral Component Interconnect“- (PCI-) Bus oder ein Bus wie z. B. ein PCI-Express-Bus oder ein anderer I/O-Zusammenschaltungs-Bus der dritten Generation sein, obwohl der Umfang der vorliegenden Erfindung nicht so eingeschränkt ist.

**[0145]** Wie in **Fig. 13** gezeigt ist, können verschiedene I/O-Vorrichtungen **1314** mit dem ersten Bus **1316** gekoppelt sein, zusammen mit einer Bus-Bridge **1318**, die den ersten Bus **1316** mit einem zweiten Bus **1320** koppelt. In einer Ausführungsform sind ein oder mehrere zusätzliche Prozessor(en) **1315**, wie z. B. Coprozessoren, Hochdurchsatz-MIC-Prozessoren, GPGPUs, Beschleuniger (wie z. B. Grafikbeschleuniger oder digitale Signalverarbeitungs- (DSP-) Einheiten), feldprogrammierbare Gate-Arrays oder irgendein anderer Prozessor mit dem ersten Bus **1316** gekoppelt. In einer Ausführungsform kann der zweite Bus **1320** ein Bus mit geringer Pin-Zahl (LPC-Bus) sein. Verschiedene Vorrichtungen können mit einem zweiten Bus **1320** gekoppelt sein, die in einer Ausführungsform beispielsweise eine Tastatur und/oder Maus **1322**, Kommunikationsvorrichtungen **1327** und eine Speichereinheit **1328** wie z. B. Plattenlaufwerk oder eine andere Massenspeichervorrichtung, die Anweisungen/Code und Daten **1330** enthalten können, enthalten. Ferner kann ein Audio-I/O **1324** mit dem zweiten Bus **1320** gekoppelt sein. Es wird darauf hingewiesen, dass andere Architekturen möglich sind. Beispielsweise kann ein System anstelle der Punkt-zu-Punkt-Architektur von **Fig. 13** einen Mehrfachbus oder eine andere solche Architektur implementieren.

**[0146]** Jetzt Bezug nehmend auf **Fig. 14** ist ein Blockdiagramm eines zweiten spezifischeren beispielhaften Systems **1400** in Übereinstimmung mit einer Ausführungsform der vorliegenden Erfindung gezeigt. Gleiche Elemente in den **Fig. 13** und **Fig. 14** tragen gleiche Bezugszeichen, und spezielle Aspekte von **Fig. 13** sind aus **Fig. 14** weggelassen worden, um das Verdecken anderer Aspekte von **Fig. 14** zu vermeiden.

**[0147]** **Fig. 14** stellt dar, dass die Prozessoren **1370**, **1380** integrierten Datenspeicher und I/O-Steuerlogik („CL“) **1372** bzw. **1382** enthalten können. Somit enthält die CL **1372**, **1382** integrierte Datenspeichersteuereinheiten und enthält I/O-Steuerlogik. **Fig. 14** stellt dar, dass nicht nur die Datenspeicher **1332**, **1334** mit der CL **1372**, **1382**, gekoppelt sind, sondern auch, dass die I/O-Vorrichtungen **1414** ebenfalls mit der Steuerlogik **1372**, **1382** gekoppelt sind. Alte I/O-Vorrichtungen **1415** sind mit dem Chipsatz **1390** gekoppelt.

**[0148]** Jetzt Bezug nehmen auf **Fig. 15** ist ein Blockdiagramm eines SoC **1500** in Übereinstimmung mit einer Ausführungsform der vorliegenden Erfindung gezeigt. Ähnliche Elemente in **Fig. 11** tragen gleiche Bezugszeichen. Außerdem sind Kästen mit gestrichelter Linie Merkmale auf weiter entwickelten SoCs. In **Fig. 15** ist eine Zusammenschaltungseinheit(en) **1502** gekoppelt mit: einem Anwendungsprozessor **1510**, der eine Gruppe aus einem oder mehreren Kernen **1102A-N** enthält, die Cache-Einheiten **1104A-N** und gemeinsam verwendete Cache-Einheit(en) **1106** enthalten; einer Systemagenteneinheit **1110**; einer Bussteuereinheit(en) **1116**; einer integrierten Datenspeichersteuereinheit(en) **1114**; einer Gruppe aus einem oder mehreren Coprozessoren **1520**, die integrierte Grafiklogik, einen Bildprozessor, einen Audioprozessor und einen Videoprozessor enthalten kann; einer statischen Direktzugriffsspeicher- (SRAM-) Einheit **1530**; einer Direktspeicherzugriffs-(DMA-) Einheit **1532**; und einer Anzeigeeinheit **1540** zum Koppeln mit einer oder mehreren externen Anzeige-

vorrichtungen. In einer Ausführungsform enthalten die Coprozessor(en) **1520** einen Spezialprozessor wie beispielsweise einen Netz- oder Kommunikationsprozessor, eine Komprimierungs-Engine, eine GPGPU, einen Hochdurchsatz-MIC-Prozessor, einen eingebetteten Prozessor oder dergleichen.

**[0149]** Ausführungsformen der Mechanismen, die hier offenbart sind, können in Hardware, Software, Firmware oder einer Kombination aus solchen Implementierungsherangehensweisen implementiert sein. Ausführungsformen der Erfindung können als Computerprogramme oder Programmcode implementiert sein, der auf programmierbaren Systemen abläuft, die wenigstens einen Prozessor, ein Speichersystem (das flüchtigen und nichtflüchtigen Datenspeicher und/oder Speicherelemente enthält), wenigstens eine Eingabevorrichtung und wenigstens eine Ausgabevorrichtung umfassen.

**[0150]** Programmcode wie z. B. der Code **1330**, der in **Fig. 13** dargestellt ist, kann auf eingegebene Anweisungen angewandt werden, um die hier beschriebenen Funktionen auszuführen und Ausgabeinformationen zu erzeugen. Die Ausgabeinformationen können auf eine oder mehrere Ausgabevorrichtungen auf bekannte Weise angewandt werden. Für die Zwecke dieser Anmeldung enthält ein Verarbeitungssystem irgendeinen System, das einen Prozessor aufweist, wie beispielsweise einen digitalen Signalprozessor (DSP), eine Mikrosteuereinheit, eine anwendungsspezifische integrierte Schaltung (ASIC) oder einen Mikroprozessor.

**[0151]** Der Programmcode kann in einer prozeduralen oder objektorientierten Programmierhochsprache implementiert sein, um mit einem Verarbeitungssystem zu kommunizieren. Der Programmcode kann auch in Assembler- oder Maschinensprache implementiert sein, falls gewünscht. Tatsächlich ist der Umfang der hier beschriebenen Mechanismen nicht auf irgendeine spezielle Programmiersprache beschränkt. In jedem Fall kann die Sprache eine kompilierte oder interpretierte Sprache sein.

**[0152]** Ein oder mehrere Aspekte wenigstens einer Ausführungsform kann durch repräsentative Anweisungen implementiert sein, die auf einem maschinenlesbaren Medium gespeichert sind, das verschiedene Logik innerhalb des Prozessors repräsentiert, die dann, wenn sie durch eine Maschine gelesen werden, bewirken, dass die Maschine Logik herstellt, um die hier beschriebenen Techniken auszuführen. Solche Repräsentationen, als „IP-Kerne“ bekannt, können auf einem greifbaren maschinenlesbaren Medium gespeichert und an verschiedene Kunden oder Produktionsanalgen geliefert werden, um sie in die Produktionsmaschinen zu laden, die die Logik oder den Prozessor tatsächlich herstellen.

**[0153]** Solche maschinenlesbare Speichermedien können ohne Einschränkung nichttransitorische greifbare Anordnungen von Gegenständen, die durch eine Maschine oder eine Vorrichtung hergestellt oder gebildet sind, enthalten, die Speichermedien wie z. B. Festplatten, irgendeinen andere Typ einer Platte, die Disketten, optische Platten, Compact Disk-Festwertspeicher (CD-ROMs) Compact Disk-wiederbeschreibbar (CD-RWs) und magneto-optische Platten, enthalten, Halbleitervorrichtungen wie z. B. Festwertspeicher (ROMs), Direktzugriffsspeicher (RAMs) wie z. B. dynamische Direktzugriffsspeicher (DRAMs), statische Direktzugriffsspeicher (SRAMs), löschbare programmierbare Festwertspeicher (EPROMs), Flash-Datenspeicher, elektrisch löschbare programmierbare Festwertspeicher (EEPROMs), Phasenwechselspeicher (PCM), magnetische oder optische Karten oder irgendeinen anderen Typ von Medien, die zum Speichern elektronischer Anweisungen geeignet sind, enthalten.

**[0154]** Dementsprechend enthalten Ausführungsformen der Erfindung auch nichttransitorische greifbare maschinenlesbare Medien, die Anweisungen beinhalten oder Konstruktionsdaten beinhalten, wie z. B. Hardware-Beschreibungssprache (HDL), die Strukturen, Schaltungen, Einrichtungen, Prozessoren und/oder Systemmerkmale, die hier beschrieben sind, definieren. Solche Ausführungsformen können auch als Programmprodukte bezeichnet sein.

Emulation (die Binärübersetzung, Code-Morphing usw. enthält)

**[0155]** In einigen Fällen kann ein Befehlsumsetzer verwendet werden, um eine Anweisung aus einem Quellbefehlssatz in einen Zielbefehlssatz umzusetzen. Beispielsweise kann der Befehlsumsetzer eine Anweisung übersetzen (z. B. unter Verwendung statischer Binärübersetzung, dynamischer Binärübersetzung, die dynamische Kompilierung enthält), morphen, emulieren oder auf andere Weise in eine oder mehrere andere Anweisungen umsetzen, die durch den Kern verarbeitet werden sollen. Der Befehlsumsetzer kann in Software, Hardware, Firmware oder einer Kombination daraus implementiert sein. Der Befehlsumsetzer kann auf einem Prozessor, außerhalb des Prozessors oder teilweise auf dem und teilweise außerhalb des Prozessors sein.

**[0156]** **Fig. 16** ist ein Blockdiagramm, das die Verwendung eines Software-Befehlsumsetzers zum Umsetzen binärer Anweisungen in einem Quellbefehlssatz in binäre Anweisungen in einem Zielbefehlssatz gemäß Ausführungsformen der Erfindung gegenüberstellt. In der dargestellten Ausführungsform ist der Befehlsumsetzer ein Software-Befehlsumsetzer, obwohl alternativ der Befehlsumsetzer in Software, Firmware, Hardware oder verschiedenen Kombinationen daraus implementiert sein kann. **Fig. 16** zeigt ein Programm in einer Hochsprache **1602**, das unter Verwendung eines x86-Compilers **1604** kompiliert werden kann, um x86-Binärcode 1606 zu erzeugen, der durch einen Prozessor mit wenigstens einem Kern **1616** mit x86-Befehlssatz nativ ausgeführt werden kann. Der Prozessor mit wenigstens einem Kern **1616** mit x86-Befehlssatz repräsentiert irgendeinen Prozessor, der im Wesentlichen die gleichen Funktionen wie in Intel-Prozessor mit wenigstens einem Kern mit x86-Befehlssatz ausführen kann durch kompatibles Ausführen oder auf andere Weise Verarbeiten (1) eines wesentlichen Abschnitts des Befehlssatzes des Kerns mit dem Intel-x86-Befehlssatz oder (2) Objektcodeversionen von Anwendungen oder anderer Software, die darauf zielen, auf einem Intel-Prozessor mit wenigstens einem Kern mit x86-Befehlssatz abzulaufen, um im Wesentlichen das gleiche Ergebnis zu erreichen wie ein Intel-Prozessor mit wenigstens einem Kern mit x86-Befehlssatz. Der x86-Compiler **1604** repräsentiert einen Compiler, der arbeitet, um x86-Binärcode **1606** (z. B. Objektcode) zu erzeugen, der mit oder ohne zusätzlicher Verknüpfungsverarbeitung, auf dem Prozessor mit wenigstens einem Kern **1616** mit x86-Befehlssatz ausgeführt werden kann. Ähnlich zeigt **Fig. 16**, dass das Programm in der Hochsprache **1602** unter Verwendung eines Compilers **1608** mit einem alternativen Befehlssatz kompiliert werden kann, um Binärcode **1610** eines alternativen Befehlssatzes zu erzeugen, der durch einen Prozessor ohne wenigstens einen Kern **1614** mit einem x86-Befehlssatz nativ ausgeführt werden kann (z. B. einen Prozessor mit Kernen, die den MIPS-Befehlssatz von MIPS Technologies aus Sunnyvale, CA, ausführen können und/oder die den ARM-Befehlssatz von ARM Holdings aus Sunnyvale, CA, ausführen können). Der Befehlsumsetzer **1612** wird verwendet, um den x86-Binärcode 1606 in Code umzusetzen, der durch den Prozessor ohne einen Kern **1614** mit einem x86-Befehlssatz nativ ausgeführt werden kann. Dieser umgesetzte Code ist wahrscheinlich nicht der gleiche wie der Binärcode **1610** des alternativen Befehlssatzes, weil ein Befehlsumsetzer, der dazu fähig ist, schwierig herzustellen ist; der umgesetzte Code kann jedoch die allgemeine Operation verwirklichen und kann aus Anweisungen aus dem alternativen Befehlssatz bestehen. Somit repräsentiert der Befehlsumsetzer **1612** Software, Firmware, Hardware oder eine Kombination daraus, die es durch Emulation, Simulation oder irgendeinen anderen Prozess einem Prozessor oder einer anderen elektronischen Vorrichtung, die keinen Prozessor oder Kern mit x86-Befehlssatz aufweist, ermöglicht, den x86-Binärcode 1606 auszuführen.

**[0157]** Beispiele von Ausführungsformen, die vorstehend im Einzelnen dargestellt sind, sind nachstehend beschrieben.

**[0158]** Beispiel 1. Ein Prozessor, der Folgendes umfasst:

Decodierschaltungsanordnung, um eine Anweisung zu decodieren, die Felder für einen Opcode, wenigstens zwei Bezeichner von Quelloperanden für gepackte Daten, einen Bezeichner eines Zieloperanden für gepackte Daten und ein Immediate aufweist; und

Ausführungsschaltungsanordnung, um die decodierte Anweisung auszuführen zum:

Broadcast eines gepackten Datenelements aus dem identifizierten ersten Quelloperanden für gepackte Daten, wobei die Position des gepackten Datenelements, das durch Broadcast übertragen werden soll, basierend auf einem Wert des Immediate ausgewählt wird, Ausführen von Operationen gemäß dem Opcode auf dem durch Broadcast übertragenen gepackten Datenelement aus dem identifizierten ersten Quelloperanden für gepackte Daten und gepackter Datenelemente des identifizierten zweiten Quelloperanden für gepackte Daten, und Speichern der Ergebnisse der Operationen in dem identifizierten Zieloperanden für gepackte Daten in Positionen, die den Positionen gepackter Datenelemente des identifizierten zweiten Quelloperanden für gepackte Daten entsprechen.

**[0159]** Beispiel 2. Prozessor nach Beispiel 1, wobei die Operationen Berechnungen sind.

**[0160]** Beispiel 3. Prozessor nach Beispiel 2, wobei das Immediate ein 8-Bit-Wert ist.

**[0161]** Beispiel 4. Prozessor nach einem der Beispiele 1-3, wobei die Exponentenkomponente in einer niedrigstwertigen Position des gepackten Datenelements des identifizierten Zieloperanden für gepackte Daten gespeichert ist.

**[0162]** Beispiel 5. Prozessor nach einem der Beispiele 1-4, wobei einer der identifizierten Quelloperanden für gepackte Daten und das Ziel gleich sind.

**[0163]** Beispiel 6. Prozessor nach einem der Beispiele 1-5, wobei der identifizierte zweite Quelloperand für gepackte Daten ein Datenspeicherort ist.

**[0164]** Beispiel 7. Verfahren, das Folgendes umfasst: Decodieren einer Anweisung, die Felder für einen Opcode, wenigstens zwei Bezeichner von Quelloperanden für gepackte Daten, einen Bezeichner eines Zieloperanden für gepackte Daten und ein Immediate aufweist; und Ausführen der decodierten Anweisung zum: Broadcast eines gepackten Datenelements aus dem identifizierten ersten Quelloperanden für gepackte Daten, wobei die Position des gepackten Datenelements, das durch Broadcast übertragen werden soll, basierend auf einem Wert des Immediate ausgewählt wird, Ausführen von Operationen gemäß dem Opcode auf dem durch Broadcast übertragenen gepackten Datenelement aus dem identifizierten ersten Quelloperanden für gepackte Daten und gepackten Datenelementen des identifizierten zweiten Quelloperanden für gepackte Daten, und Speichern der Ergebnisse der Operationen in dem identifizierten Zieloperanden für gepackte Daten in Positionen, die den Positionen gepackter Datenelemente des identifizierten zweiten Quelloperanden für gepackte Daten entsprechen.

**[0165]** Beispiel 8. Verfahren nach Beispiel 7, wobei die Operationen Berechnungen sind.

**[0166]** Beispiel 9. Verfahren nach Beispiel 8, wobei das Immediate ein 8-Bit-Wert ist.

**[0167]** Beispiel 10. Verfahren nach Beispiel 9, wobei die Exponentenkomponente in einer niedrigstwertigen Position des gepackten Datenelements des identifizierten Zieloperanden für gepackte Daten gespeichert ist.

**[0168]** Beispiel 11. Verfahren nach einem der Beispiele 7-9, wobei einer der identifizierten Quelloperanden für gepackte Daten und das Ziel gleich sind.

**[0169]** Beispiel 12. Verfahren nach einem der Beispiele 7-9, wobei der identifizierte zweite Quelloperand für gepackte Daten ein Datenspeicherort ist.

**[0170]** Beispiel 13. Nicht-transitorisches maschinenlesbares Medium, das ein Auftreten einer Anweisung speichert, wobei ein Hardwareprozessor in Reaktion auf das Auftreten der Anweisung ein Verfahren ausführen soll, das Folgendes umfasst: Decodieren der Anweisung, die Felder für einen Opcode, wenigstens zwei Bezeichner von Quelloperanden für gepackte Daten, einen Bezeichner eines Zieloperanden für gepackte Daten und ein Immediate aufweist; und Ausführen der decodierten Anweisung zum: Broadcast eines gepackten Datenelements aus dem identifizierten ersten Quelloperanden für gepackte Daten, wobei die Position des gepackten Datenelements, das durch Broadcast übertragen werden soll, basierend auf einem Wert des Immediate ausgewählt wird, Ausführen von Operationen gemäß dem Opcode auf dem durch Broadcast übertragenen gepackten Datenelement aus dem identifizierten ersten Quelloperanden für gepackte Daten und gepackten Datenelementen des identifizierten zweiten Quelloperanden für gepackte Daten, und Speichern der Ergebnisse der Operationen in dem identifizierten Zieloperanden für gepackte Daten in Positionen, die den Positionen gepackter Datenelemente des identifizierten zweiten Quelloperanden für gepackte Daten entsprechen.

**[0171]** Beispiel 14. Nicht-transitorisches maschinenlesbares Medium nach Beispiel 13, wobei die Operationen Berechnungen sind.

**[0172]** Beispiel 15. Nicht-transitorisches maschinenlesbares Medium nach Beispiel 14, wobei das Immediate ein 8-Bit-Wert ist.

**[0173]** Beispiel 16. Nicht-transitorisches maschinenlesbares Medium nach einem der Beispiele 13-15, wobei die Exponentenkomponente in einer niedrigstwertigen Position des gepackten Datenelements des identifizierten Zieloperanden für gepackte Daten gespeichert ist.

**[0174]** Beispiel 17. Nicht-transitorisches maschinenlesbares Medium nach einem der Beispiele 13-18, wobei einer der identifizierten Quelloperanden für gepackte Daten und das Ziel gleich sind.

**[0175]** Beispiel 18. Nicht-transitorisches maschinenlesbares Medium nach einem der Beispiele 13-17, wobei der identifizierte zweite Quelloperand für gepackte Daten ein Datenspeicherort ist.

## Patentansprüche

1. Prozessor, der Folgendes umfasst:

Decodiermittel zum Decodieren einer Anweisung, die Felder für einen Opcode, wenigstens zwei Bezeichner von Quelloperanden für gepackte Daten, einen Bezeichner eines Zieloperanden für gepackte Daten und ein Immediate aufweist; und
Ausführungsmittel zum Ausführen der decodierten Anweisung zum:
Broadcast eines gepackten Datenelements aus dem identifizierten ersten Quelloperanden für gepackte Daten, wobei die Position des gepackten Datenelements, das durch Broadcast übertragen werden soll, basierend auf einem Wert des Immediate ausgewählt wird,
Ausführen von Operationen gemäß dem Opcode auf dem durch Broadcast übertragenen gepackten Datenelement aus dem identifizierten ersten Quelloperanden für gepackte Daten und gepackten Datenelementen des identifizierten zweiten Quelloperanden für gepackte Daten, und
Speichern der Ergebnisse der Operationen in dem identifizierten Zieloperanden für gepackte Daten in Positionen, die den Positionen der gepackten Datenelemente des identifizierten zweiten Quelloperanden für gepackte Daten entsprechen.

2. Prozessor nach Anspruch 1, wobei die Operationen Berechnungen sind.

3. Prozessor nach Anspruch 2, wobei das Immediate ein 8-Bit-Wert ist.

4. Prozessor nach einem der Ansprüche 1-3, wobei die Exponentenkomponente in einer niedrigstwertigen Position des gepackten Datenelements des identifizierten Zieloperanden für gepackte Daten gespeichert ist.

5. Prozessor nach einem der Ansprüche 1-3, wobei einer der identifizierten Quelloperanden für gepackte Daten und das Ziel gleich sind.

6. Prozessor nach einem der Ansprüche 1-4, wobei der identifizierte zweite Quelloperand für gepackte Daten ein Datenspeicherort ist.

7. Verfahren, das Folgendes umfasst:
Decodieren einer Anweisung, die Felder für einen Opcode, wenigstens zwei Bezeichner von Quelloperanden für gepackte Daten, einen Bezeichner eines Zieloperanden für gepackte Daten und ein Immediate aufweist; und
Ausführen der decodierten Anweisung zum:
Broadcast eines gepackten Datenelements aus dem identifizierten ersten Quelloperanden für gepackte Daten, wobei die Position des gepackten Datenelements, das durch Broadcast übertragen werden soll, basierend auf einem Wert des Immediate ausgewählt wird,
Ausführen von Operationen gemäß dem Opcode auf dem durch Broadcast übertragenen gepackten Datenelements aus dem identifizierten ersten Quelloperanden für gepackte Daten und gepackten Datenelementen des identifizierten zweiten Quelloperanden für gepackte Daten, und
Speichern der Ergebnisse der Operationen in dem identifizierten Zieloperanden für gepackte Daten in Positionen, die den Positionen der gepackten Datenelemente des identifizierten zweiten Quelloperanden für gepackte Daten entsprechen.

8. Verfahren nach Anspruch 7, wobei die Operationen Berechnungen sind.

9. Verfahren nach Anspruch 8, wobei das Immediate ein 8-Bit-Wert ist.

10. Verfahren nach Anspruch 9, wobei die Exponentenkomponente in einer niedrigstwertigen Position des gepackten Datenelements des identifizierten Zieloperanden für gepackte Daten gespeichert ist.

11. Verfahren nach Anspruch 10, wobei einer der identifizierten Quelloperanden für gepackte Daten und das Ziel gleich sind.

12. Verfahren nach Anspruch 7, wobei der identifizierte zweite Quelloperand für gepackte Daten ein Datenspeicherort ist.

13. Nicht-transitorisches maschinenlesbares Medium, das ein Auftreten einer Anweisung speichert, wobei ein Hardwareprozessor in Reaktion auf das Auftreten der Anweisung ein Verfahren ausführen soll, das Folgendes umfasst:
Decodieren der Anweisung, die Felder für einen Opcode, wenigstens zwei Bezeichner von Quelloperanden für gepackte Daten, einen Bezeichner eines Zieloperanden für gepackte Daten und ein Immediate aufweist; und

Ausführen der decodierten Anweisung zum:
Broadcast eines gepackten Datenelements aus dem identifizierten ersten Quelloperanden für gepackte Daten, wobei die Position des gepackten Datenelements, das durch Broadcast übertragen werden soll, basierend auf einem Wert des Immediate ausgewählt wird,
Ausführen von Operationen gemäß dem Opcode auf dem durch Broadcast übertragenen gepackten Datenelement aus dem identifizierten ersten Quelloperanden für gepackte Daten und gepackten Datenelementen des identifizierten zweiten Quelloperanden für gepackte Daten, und
Speichern der Ergebnisse der Operationen in dem identifizierten Zieloperanden für gepackte Daten in Positionen, die den Positionen der gepackten Datenelemente des identifizierten zweiten Quelloperanden für gepackte Daten entsprechen.

14. Nicht-transitorisches maschinenlesbares Medium nach Anspruch 13, wobei die Operationen Berechnungen sind.

15. Nicht-transitorisches maschinenlesbares Medium nach Anspruch 14, wobei das Immediate ein 8-Bit-Wert ist.

16. Nicht-transitorisches maschinenlesbares Medium nach einem der Ansprüche 13-15, wobei die Exponentenkomponente in einer niedrigstwertigen Position des gepackten Datenelements des identifizierten Zieloperanden für gepackte Daten gespeichert ist.

17. Nicht-transitorisches maschinenlesbares Medium nach einem der Ansprüche 13-16, wobei einer der identifizierten Quelloperanden für gepackte Daten und das Ziel gleich sind.

18. Nicht-transitorisches maschinenlesbares Medium nach einem der Ansprüche 13-16, wobei der identifizierte zweite Quelloperand für gepackte Daten ein Datenspeicherort ist.

Es folgen 18 Seiten Zeichnungen

**Anhängende Zeichnungen**

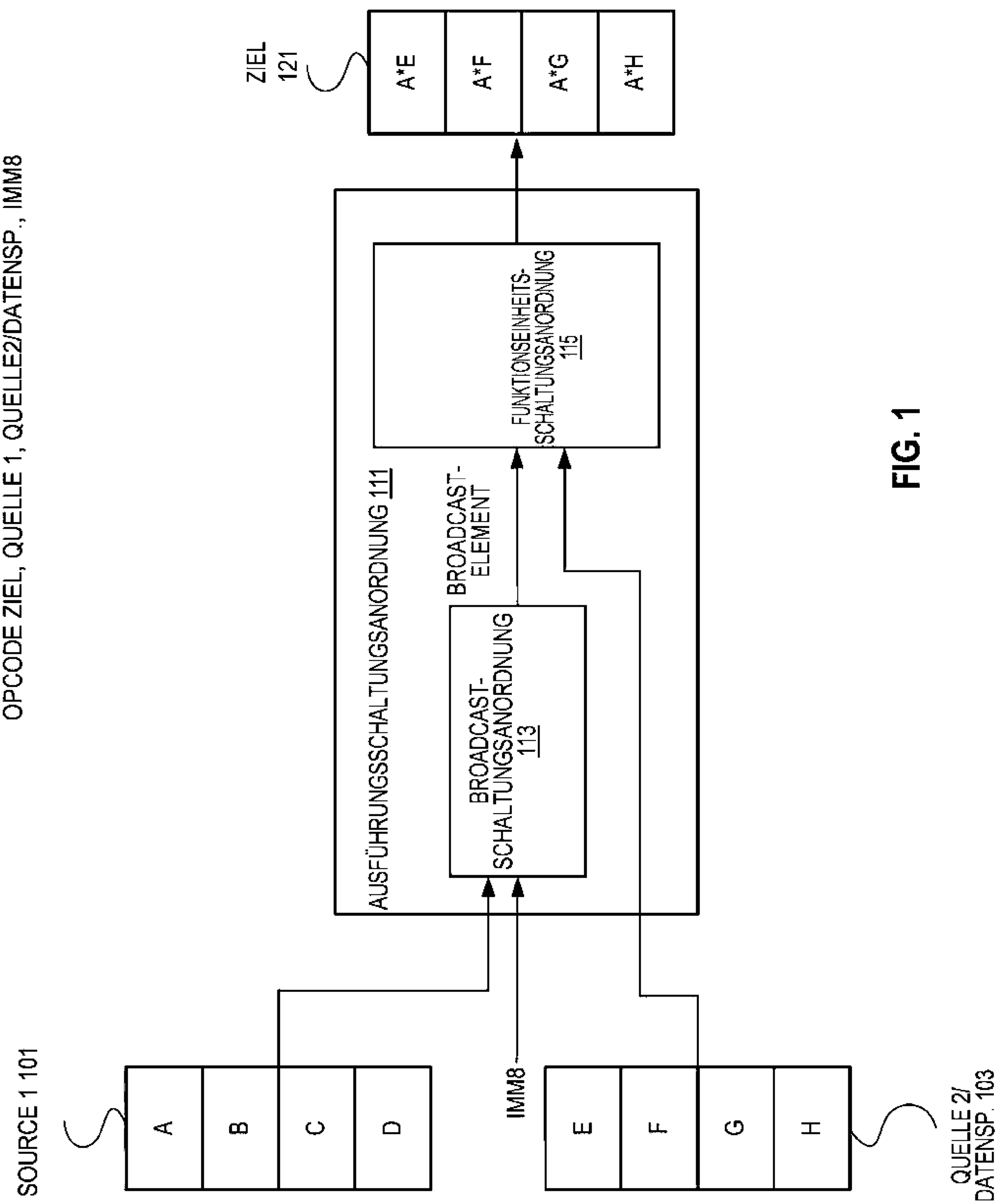



FIG. 1

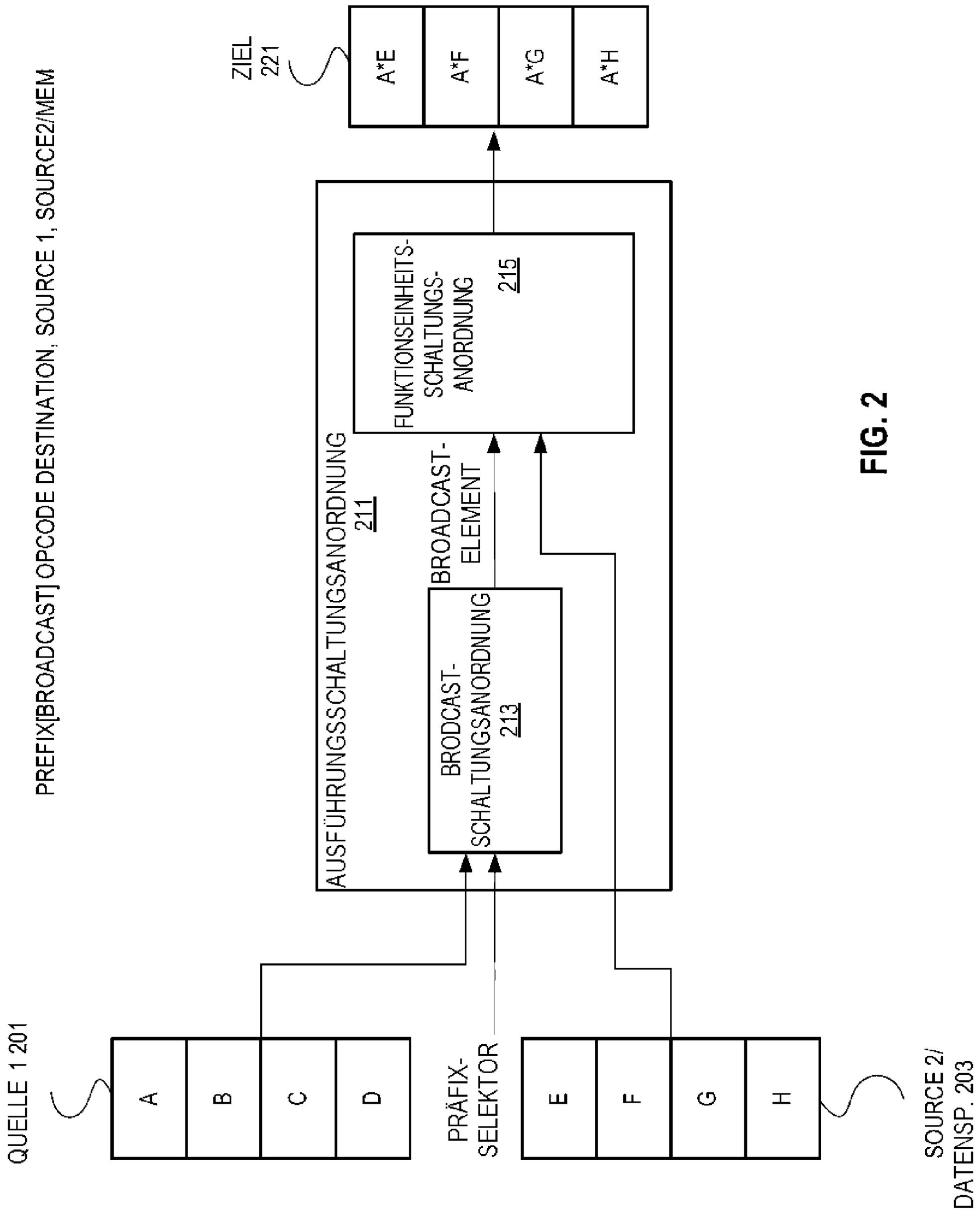

PREFIX[BROADCAST] OPCODE DESTINATION, SOURCE 1, SOURCE 2/MEM
QUELLE 1 201
A
B
C
D
PRÄFIX-SELEKTOR
E
F
G
H
SOURCE 2/ DATENSP. 203
ZIEL 221
A*E
A*F
A*G
A*H
AUSFÜHRUNGSSCHALTUNGSANORDNUNG 211
BROADCAST-ELEMENT
BRODCAST-SCHALTUNGSANORDNUNG 213
FUNKTIONSEINHEITS-SCHALTUNGS-ANORDNUNG 215

FIG. 2

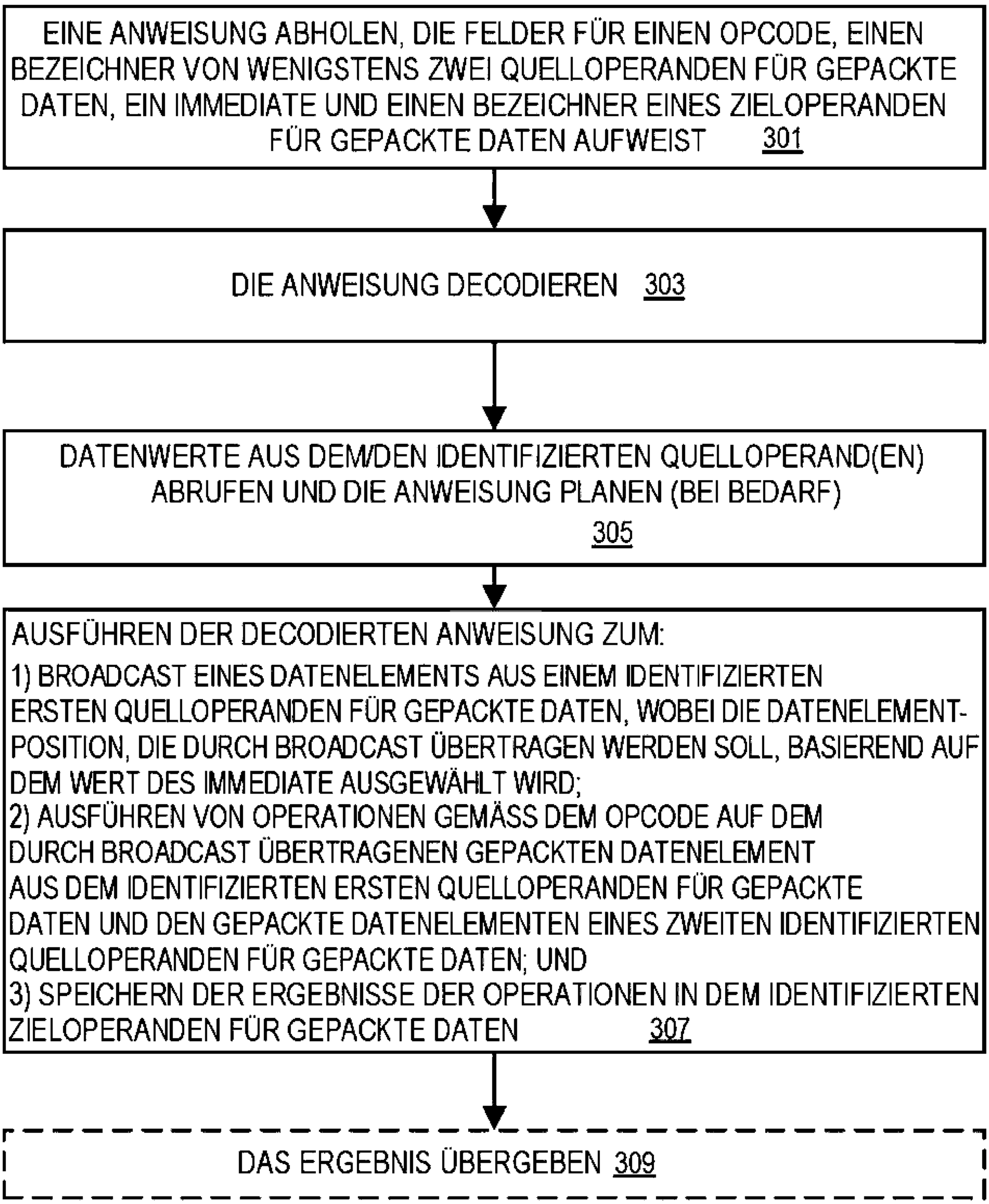

EINE ANWEISUNG ABHOLEN, DIE FELDER FÜR EINEN OPCODE, EINEN BEZEICHNER VON WENIGSTENS ZWEI QUELLOPERANDEN FÜR GEPACKTE DATEN, EIN IMMEDIATE UND EINEN BEZEICHNER EINES ZIELOPERANDEN FÜR GEPACKTE DATEN AUFWEIST 301
DIE ANWEISUNG DECODIEREN 303
DATENWERTE AUS DEM/DEN IDENTIFIZIERTEN QUELLOPERAND(EN) ABRUFEN UND DIE ANWEISUNG PLANEN (BEI BEDARF) 305
AUSFÜHREN DER DECODIERTEN ANWEISUNG ZUM:
1) BROADCAST EINES DATENELEMENTS AUS EINEM IDENTIFIZIERTEN ERSTEN QUELLOPERANDEN FÜR GEPACKTE DATEN, WOBEI DIE DATENELEMENT-POSITION, DIE DURCH BROADCAST ÜBERTRAGEN WERDEN SOLL, BASIEREND AUF DEM WERT DES IMMEDIATE AUSGEWÄHLT WIRD;
2) AUSFÜHREN VON OPERATIONEN GEMÄSS DEM OPCODE AUF DEM DURCH BROADCAST ÜBERTRAGENEN GEPACKTEN DATENELEMENT AUS DEM IDENTIFIZIERTEN ERSTEN QUELLOPERANDEN FÜR GEPACKTE DATEN UND DEN GEPACKTE DATENELEMENTEN EINES ZWEITEN IDENTIFIZIERTEN QUELLOPERANDEN FÜR GEPACKTE DATEN; UND
3) SPEICHERN DER ERGEBNISSE DER OPERATIONEN IN DEM IDENTIFIZIERTEN ZIELOPERANDEN FÜR GEPACKTE DATEN 307
DAS ERGEBNIS ÜBERGEBEN 309

FIG. 3

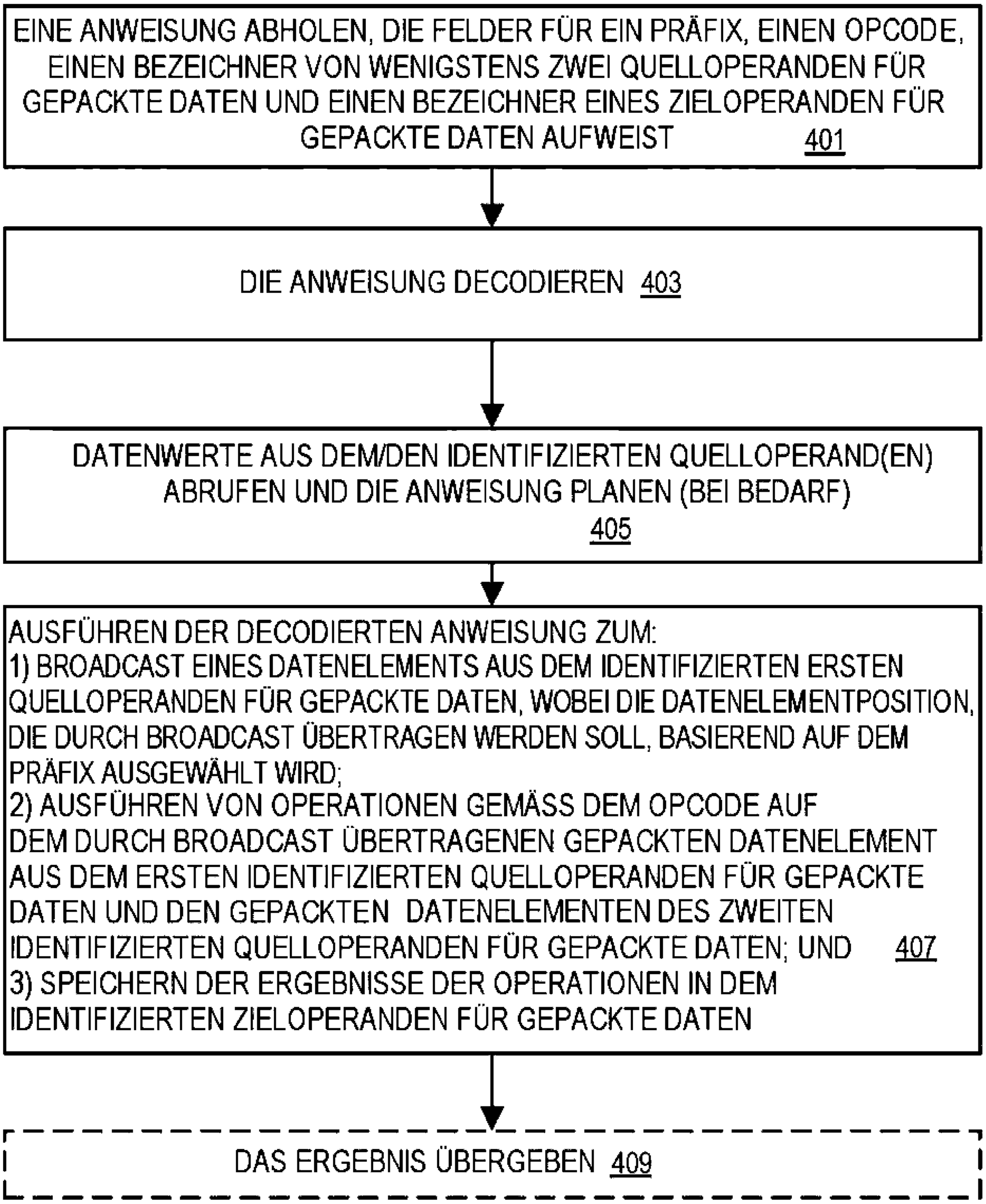

EINE ANWEISUNG ABHOLEN, DIE FELDER FÜR EIN PRÄFIX, EINEN OPCODE, EINEN BEZEICHNER VON WENIGSTENS ZWEI QUELLOPERANDEN FÜR GEPACKTE DATEN UND EINEN BEZEICHNER EINES ZIELOPERANDEN FÜR GEPACKTE DATEN AUFWEIST 401
DIE ANWEISUNG DECODIEREN 403
DATENWERTE AUS DEM/DEN IDENTIFIZIERTEN QUELLOPERAND(EN) ABRUFEN UND DIE ANWEISUNG PLANEN (BEI BEDARF) 405
AUSFÜHREN DER DECODIERTEN ANWEISUNG ZUM:
1) BROADCAST EINES DATENELEMENTS AUS DEM IDENTIFIZIERTEN ERSTEN QUELLOPERANDEN FÜR GEPACKTE DATEN, WOBEI DIE DATENELEMENTPOSITION, DIE DURCH BROADCAST ÜBERTRAGEN WERDEN SOLL, BASIEREND AUF DEM PRÄFIX AUSGEWÄHLT WIRD;
2) AUSFÜHREN VON OPERATIONEN GEMÄSS DEM OPCODE AUF DEM DURCH BROADCAST ÜBERTRAGENEN GEPACKTEN DATENELEMENT AUS DEM ERSTEN IDENTIFIZIERTEN QUELLOPERANDEN FÜR GEPACKTE DATEN UND DEN GEPACKTEN DATENELEMENTEN DES ZWEITEN IDENTIFIZIERTEN QUELLOPERANDEN FÜR GEPACKTE DATEN; UND 407
3) SPEICHERN DER ERGEBNISSE DER OPERATIONEN IN DEM IDENTIFIZIERTEN ZIELOPERANDEN FÜR GEPACKTE DATEN
DAS ERGEBNIS ÜBERGEBEN 409

FIG. 4

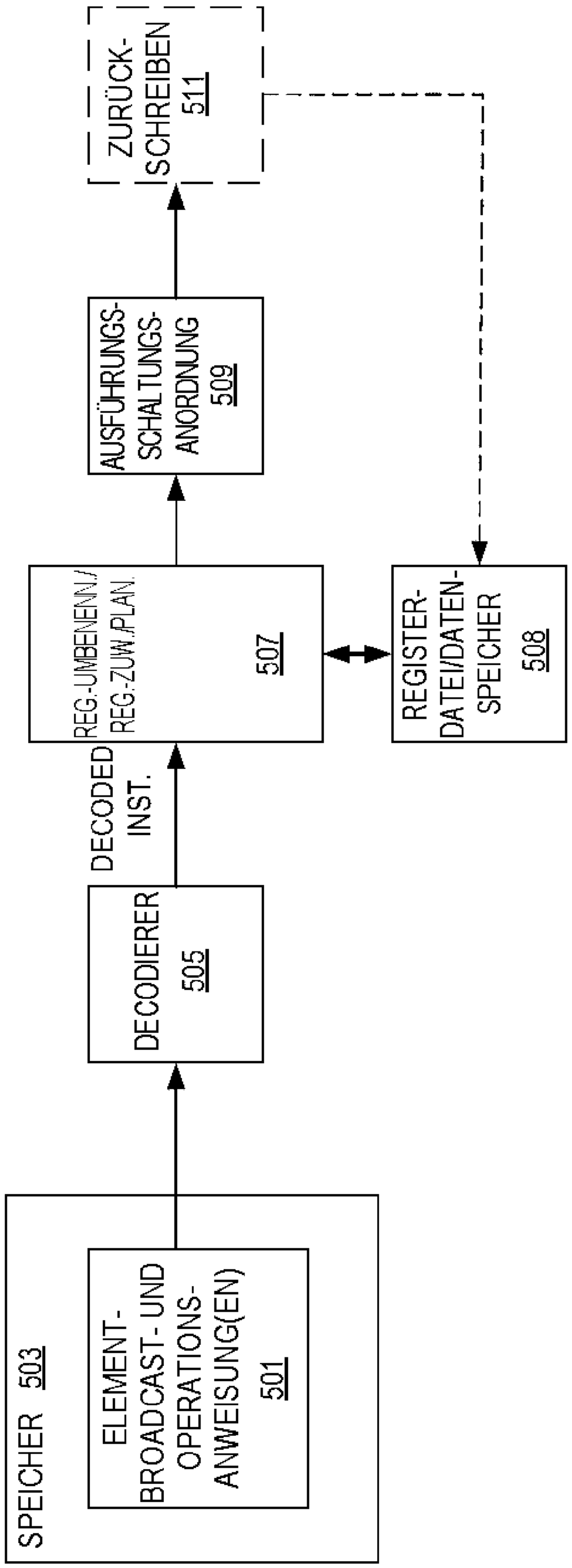
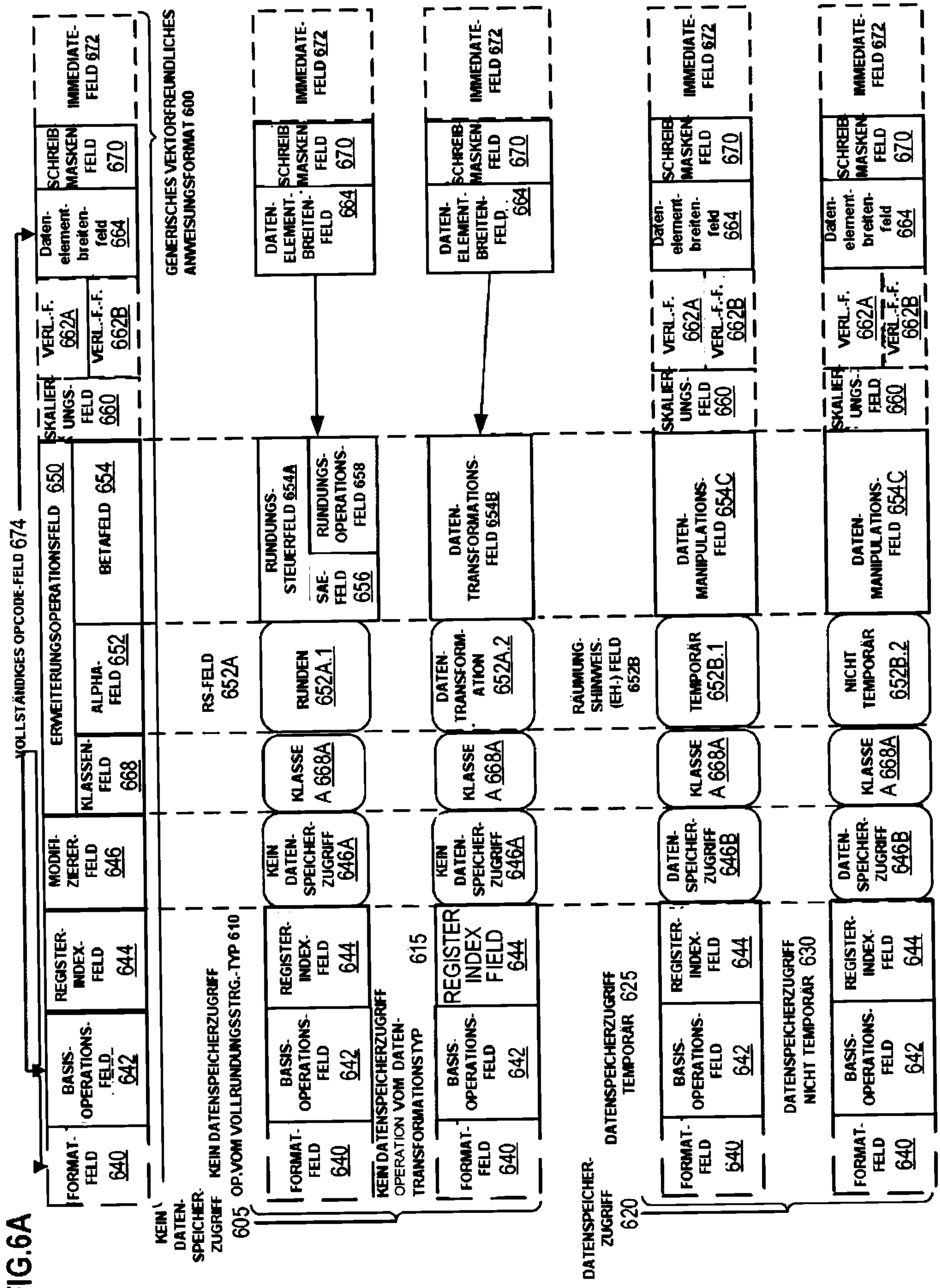

ZURÜCK-SCHREIBEN 511
AUSFÜHRUNGS-SCHALTUNGS-ANORDNUNG 509
REG.-UMBENENN./ REG.-ZUW./PLAN. 507
REGISTER-DATEI/DATEN-SPEICHER 508
DECODED INST.
DECODIERER 505
SPEICHER 503
ELEMENT-BROADCAST- UND OPERATIONS-ANWEISUNG(EN) 501

FIG. 5

FIG.6A


KEIN DATEN-SPEICHER-ZUGRIFF
605
FORMAT-FELD 640
REGISTER-INDEX-FELD 644
BASIS-OPERATIONS-FELD 642
MODIFIZIERER-FELD 646
ERWEITERUNGSOPERATION
KLASSEN-FELD 668
ALPHA-FELD 652
VOLLSTÄNDIGES OPCODE-FELD 674
SKALIERUNGS-FELD 660
VERL.-F. 662A
VERL.-F.-F. 662B
IMMEDIATE-FELD 672
GENERISCHES VEKTORFREUNDLICHES ANWEISUNGSFORMAT 600
DATENELEMENTBREITENFELD 664
SCHREIBMASKENFELD 670
BETAFELD 654
SAE-FELD 656
RUNDUNGSOPERATIONSFELD 658
RUNDUNGSSTEUERFELD 654A
DATENTRANSFORMATIONSFELD 654B
DATENMANIPULATIONSFELD 654C
KEIN DATENSPEICHERZUGRIFF VOM VOLLRUNDUNGSSTRG.-TYP 610
615
KEIN DATENSPEICHERZUGRIFF VOM DATENOPERATIONSTYP TRANSFORMATIONSTYP
DATENSPEICHERZUGRIFF TEMPORÄR 625
DATENSPEICHERZUGRIFF NICHT TEMPORÄR 630
620
KEIN DATENSPEICHERZUGRIFF
KEIN DATENSPEICHERZUGRIFF
KLASSE A 668A
DATENSPEICHERZUGRIFF 646B
RS-FELD 652A
RUNDEN 652A.1
DATENTRANSFORMATION 652A.2
RÄUMUNGSHINWEIS-(EH-)FELD 652B
TEMPORÄR 652B.1
NICHT TEMPORÄR 652B.2

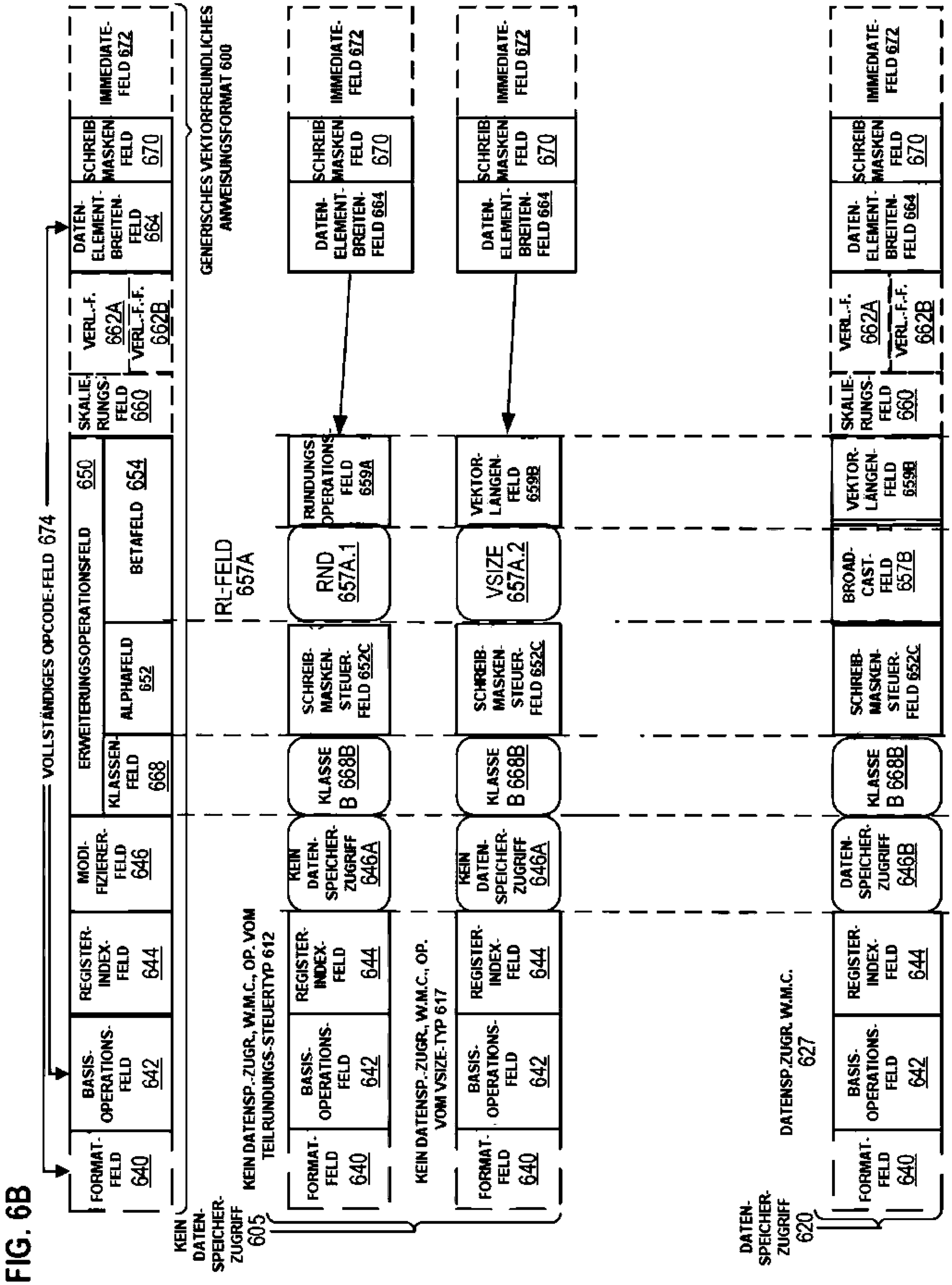

GENERISCHES VEKTORFREUNDLICHES ANWEISUNGSFORMAT 600
VOLLSTÄNDIGES OPCODE-FELD 674
FORMAT-FELD 640
BASIS-OPERATIONS-FELD 642
REGISTER-INDEX-FELD 644
MODIFIZIERER-FELD 646
ERWEITERUNGSOPERATIONSFELD 650
ALPHAFELD 652
BETAFELD 654
SCHREIBMASKEN-FELD 670
IMMEDIATE-FELD 672
DATEN-ELEMENT-BREITEN-FELD 664
SKALIERUNGS-FELD 660
VERL.-F. 662A
VERL.-F.-F. 662B
KLASSEN-FELD 668
IRL-FELD 657A
RND 657A.1
VSIZE 657A.2
RUNDUNGSOPERATIONS-FELD 659A
VEKTOR-LÄNGEN-FELD 659B
BROAD-CAST-FELD 657B
SCHREIB-MASKEN-STEUER-FELD 652C
KLASSE B 668B
KEIN DATEN-SPEICHER-ZUGRIFF 646A
DATEN-SPEICHER-ZUGRIFF 646B
KEIN DATENSP.-ZUGR., W.M.C., OP. VOM TEILRUNDUNGS-STEUERTYP 612
KEIN DATENSP.-ZUGR., W.M.C., OP. VOM VSIZE-TYP 617
DATENSP.ZUGR. W.M.C.
627
KEIN DATEN-SPEICHER-ZUGRIFF 605
DATEN-SPEICHER-ZUGRIFF 620

FIG. 6B

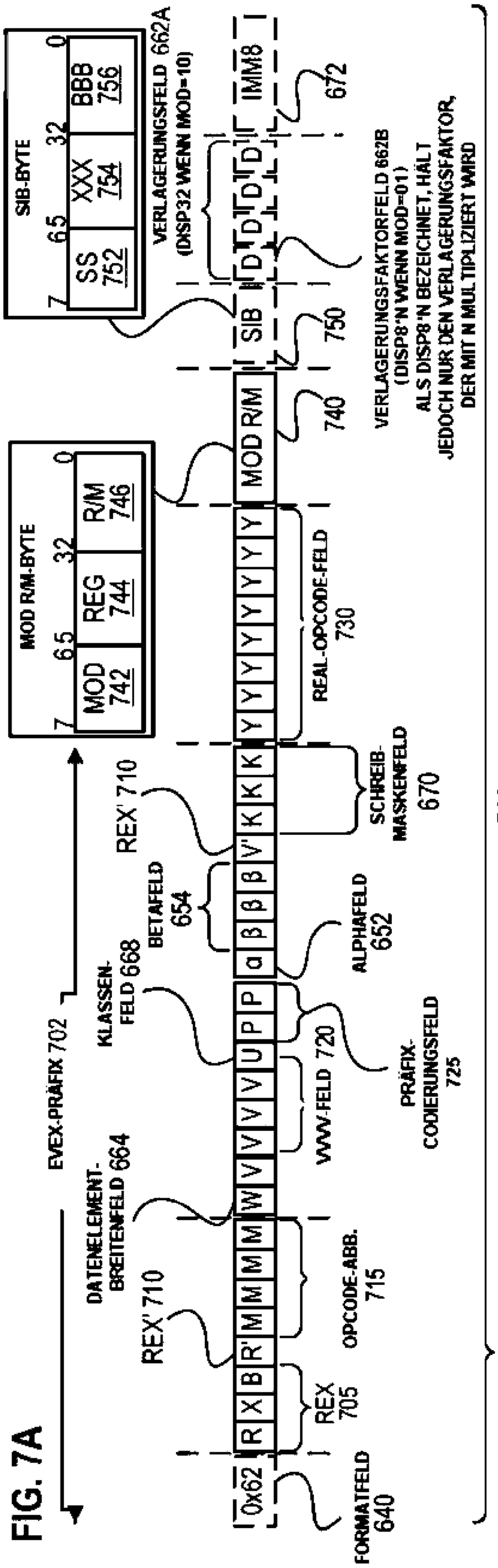


FIG. 7A
EVEX-PRÄFIX 702
FORMATFELD 640
0x62
REX 705
REX' 710
OPCODE-ABB. 715
DATENELEMENT-BREITENFELD 664
KLASSEN-FELD 668
BETAFELD 654
ALPHAFELD 652
VVV-FELD 720
PRÄFIX-CODIERUNGSFELD 725
SCHREIB-MASKENFELD 670
REAL-OPCODE-FELD 730
MOD R/M-BYTE
SIB-BYTE
VERLAGERUNGSFELD 662A
(DISP32 WENN MOD=10)
VERLAGERUNGSFAKTORFELD 662B
(DISP8*N WENN MOD=01)
ALS DISP8*N BEZEICHNET, HÄLT
JEDOCH NUR DEN VERLAGERUNGSFAKTOR,
DER MIT N MULTIPLIZIERT WIRD
IMM8
672
750
740
746
744
742
752
754
756
SS
XXX
BBB
MOD
REG
R/M
SIB
MOD R/M
REGISTERINDEXFELD 644
WWW-FELD 720
FIG. 7C
SPEZIFISCHES VEKTORFREUNDLICHES ANWEISUNGSFORMAT 700
VOLLSTÄNDIGES OPCODE-FELD 674
OPCODE-ABB. 715
DATENELEMENT-BREITENFELD 664
BASISOPERATIONSFELD 642
REAL-OPCODE-FELD 730
PRÄFIX-CODIERUNGS-FELD 725
FORMAT-FELD 640
FIG. 7B

FIG. 7D

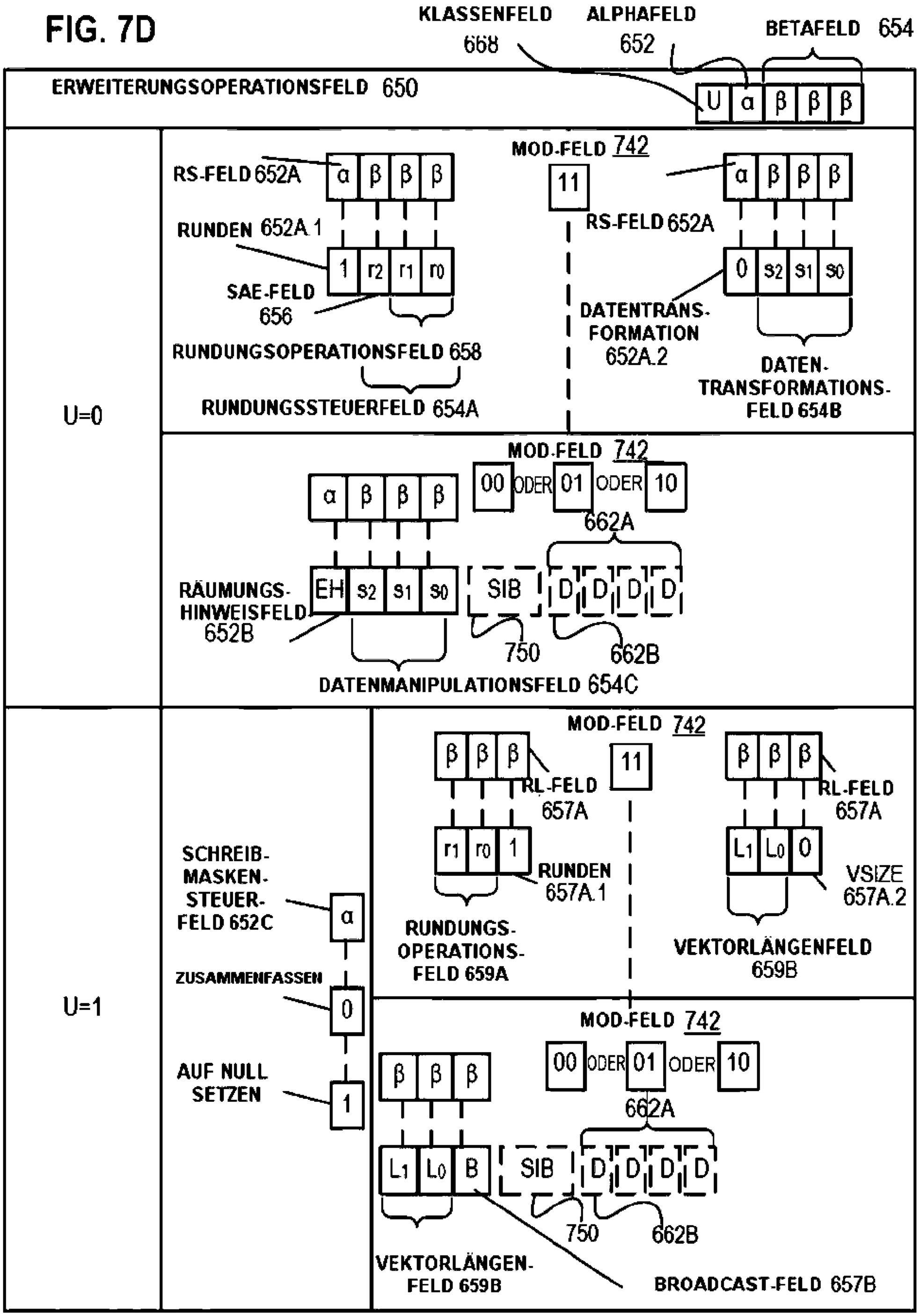
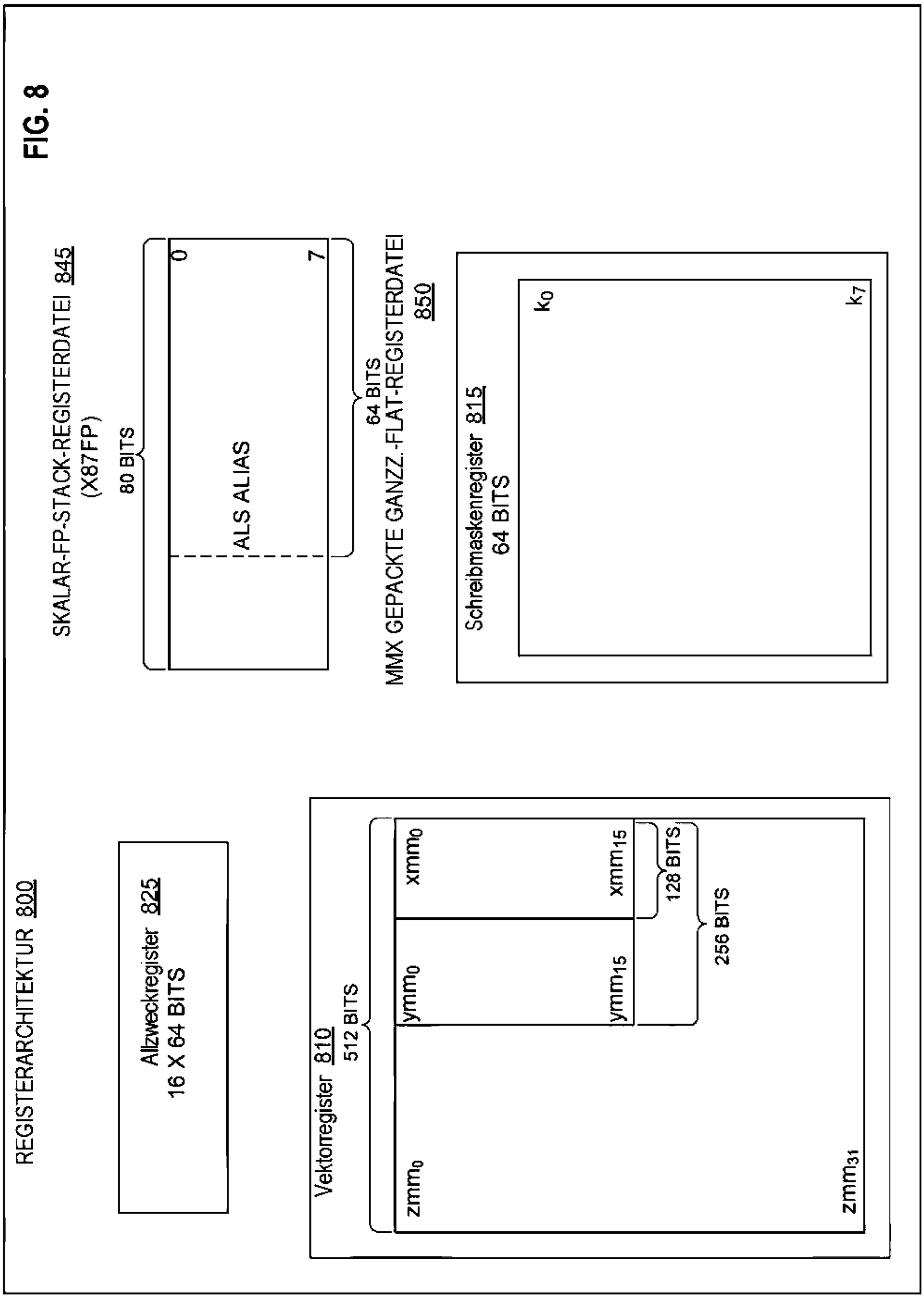

KLASSENFELD 668
ALPHAFELD 652
BETAFELD 654
ERWEITERUNGSOPERATIONSFELD 650
U α β β β
U=0
RS-FELD 652A
α β β β
RUNDEN 652A.1
1 r2 r1 r0
SAE-FELD 656
RUNDUNGSOPERATIONSFELD 658
RUNDUNGSSTEUERFELD 654A
MOD-FELD 742
11
RS-FELD 652A
α β β β
0 s2 s1 s0
DATENTRANS-FORMATION 652A.2
DATEN-TRANSFORMATIONS-FELD 654B
MOD-FELD 742
00 ODER 01 ODER 10
662A
α β β β
RÄUMUNGS-HINWEISFELD 652B
EH s2 s1 s0
SIB
D D D D
750
662B
DATENMANIPULATIONSFELD 654C
U=1
SCHREIB-MASKEN-STEUER-FELD 652C
α
ZUSAMMENFASSEN
0
AUF NULL SETZEN
1
MOD-FELD 742
β β β
11
RL-FELD 657A
r1 r0 1
RUNDEN 657A.1
RUNDUNGS-OPERATIONS-FELD 659A
β β β
RL-FELD 657A
L1 L0 0
VSIZE 657A.2
VEKTORLÄNGENFELD 659B
MOD-FELD 742
00 ODER 01 ODER 10
662A
β β β
L1 L0 B
SIB
D D D D
750
662B
VEKTORLÄNGEN-FELD 659B
BROADCAST-FELD 657B

FIG. 8


REGISTERARCHITEKTUR 800
Allzweckregister 825
16 X 64 BITS
Vektorregister 810
512 BITS
zmm0
zmm31
ymm0
ymm15
xmm0
xmm15
128 BITS
256 BITS
SKALAR-FP-STACK-REGISTERDATEI 845
(X87FP)
80 BITS
0
7
ALS ALIAS
64 BITS
MMX GEPACKTE GANZZ.-FLAT-REGISTERDATEI
850
Schreibmaskenregister 815
64 BITS
k0
k7

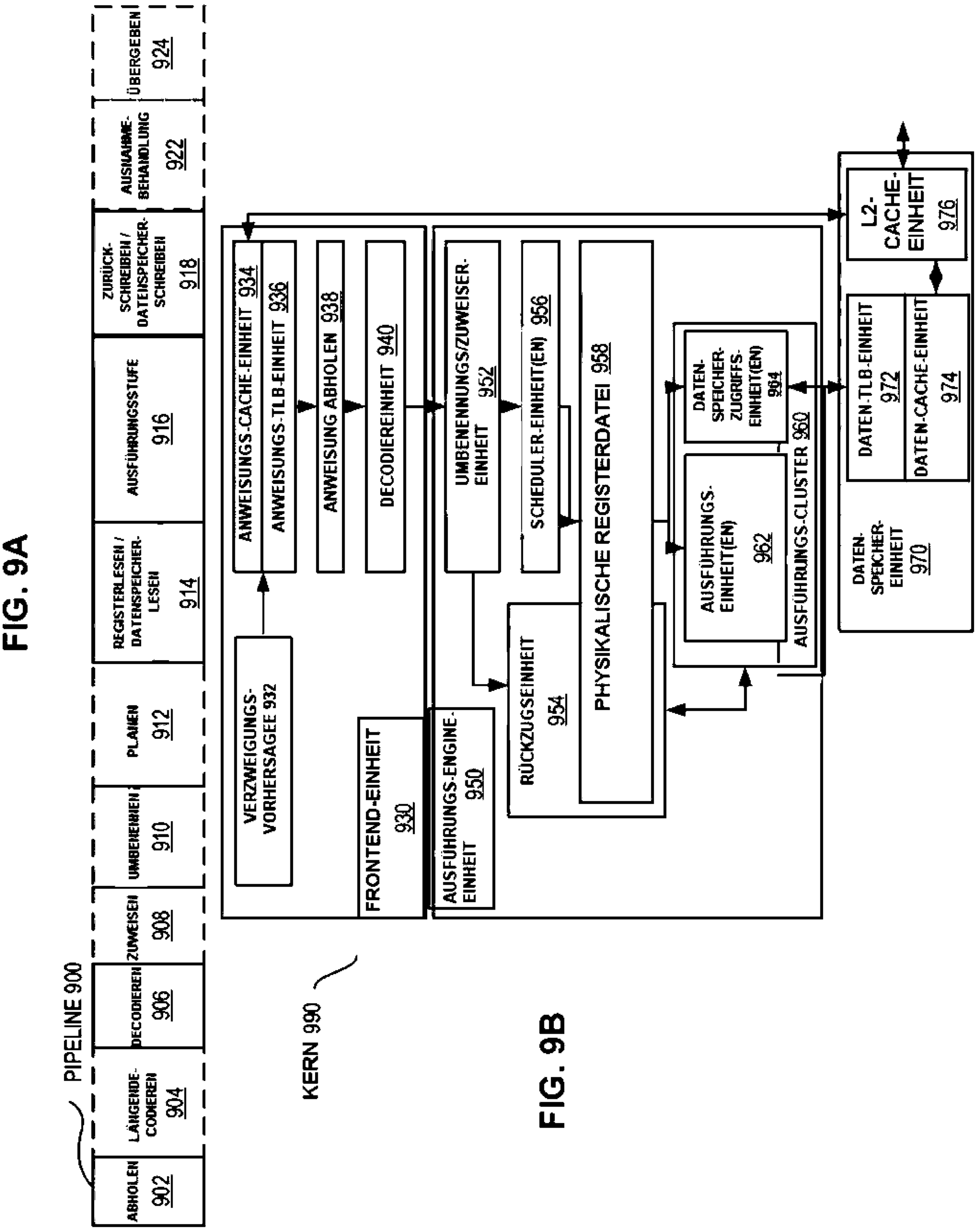

PIPELINE 900
ABHOLEN 902
LÄNGENDE-CODIEREN 904
DECODIEREN 906
ZUWEISEN 908
UMBENENNEN 910
PLANEN 912
REGISTERLESEN / DATENSPEICHER-LESEN 914
AUSFÜHRUNGSSTUFE 916
ZURÜCK-SCHREIBEN / DATEISPEICHER-SCHREIBEN 918
AUSNAHME-BEHANDLUNG 922
ÜBERGEBEN 924
KERNEL 990
VERZWEIGUNGS-VORHERSAGEE 932
FRONTEND-EINHEIT 930
ANWEISUNGS-CACHE-EINHEIT 934
ANWEISUNGS-TLB-EINHEIT 936
ANWEISUNG ABHOLEN 938
DECODIEREINHEIT 940
AUSFÜHRUNGS-ENGINE-EINHEIT 950
UMBENENNUNGS/ZUWEISER-EINHEIT 952
RÜCKZUGSEINHEIT 954
SCHEDULER-EINHEIT(EN) 956
PHYSIKALISCHE REGISTERDATEI 958
AUSFÜHRUNGS-CLUSTER 960
AUSFÜHRUNGS-EINHEIT(EN) 962
DATEN-SPEICHER-ZUGRIFFS-EINHEIT(EN) 964
DATEN-SPEICHER-EINHEIT 970
DATEN-TLB-EINHEIT 972
DATEN-CACHE-EINHEIT 974
L2-CACHE-EINHEIT 976

FIG. 9B

FIG. 10B

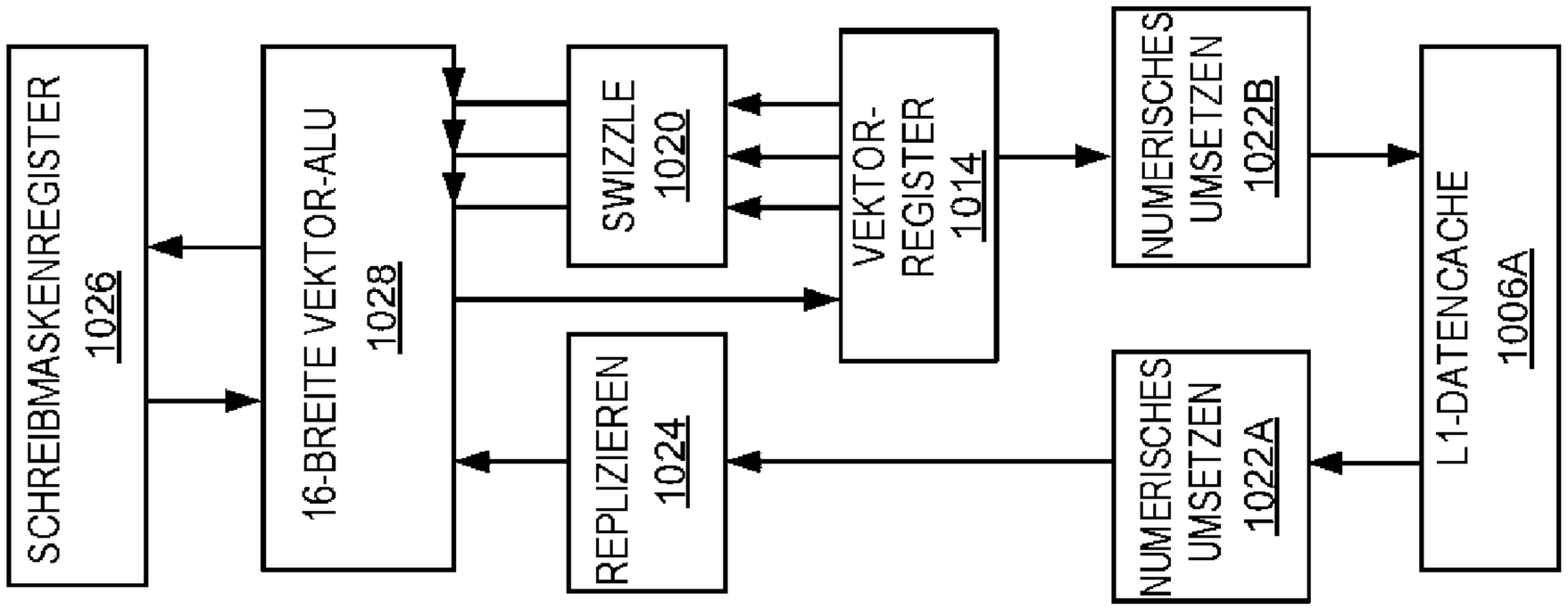
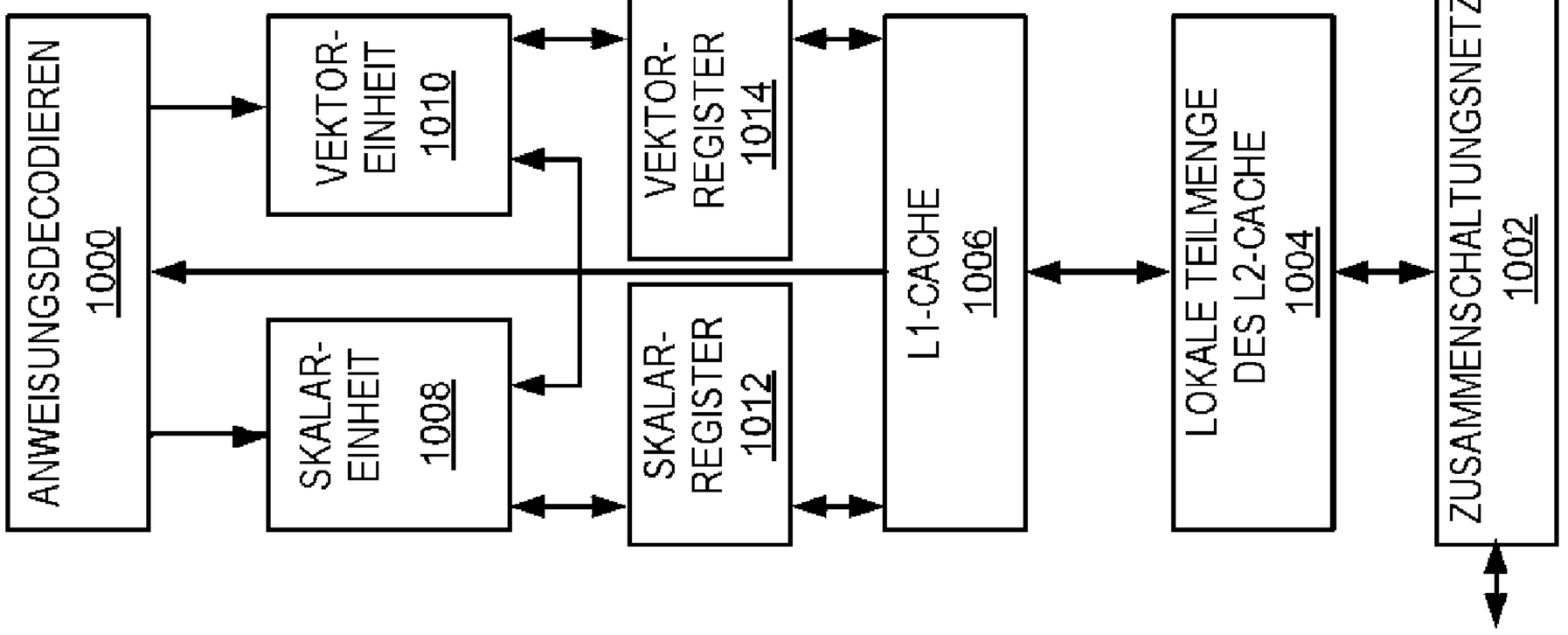



FIG. 10A



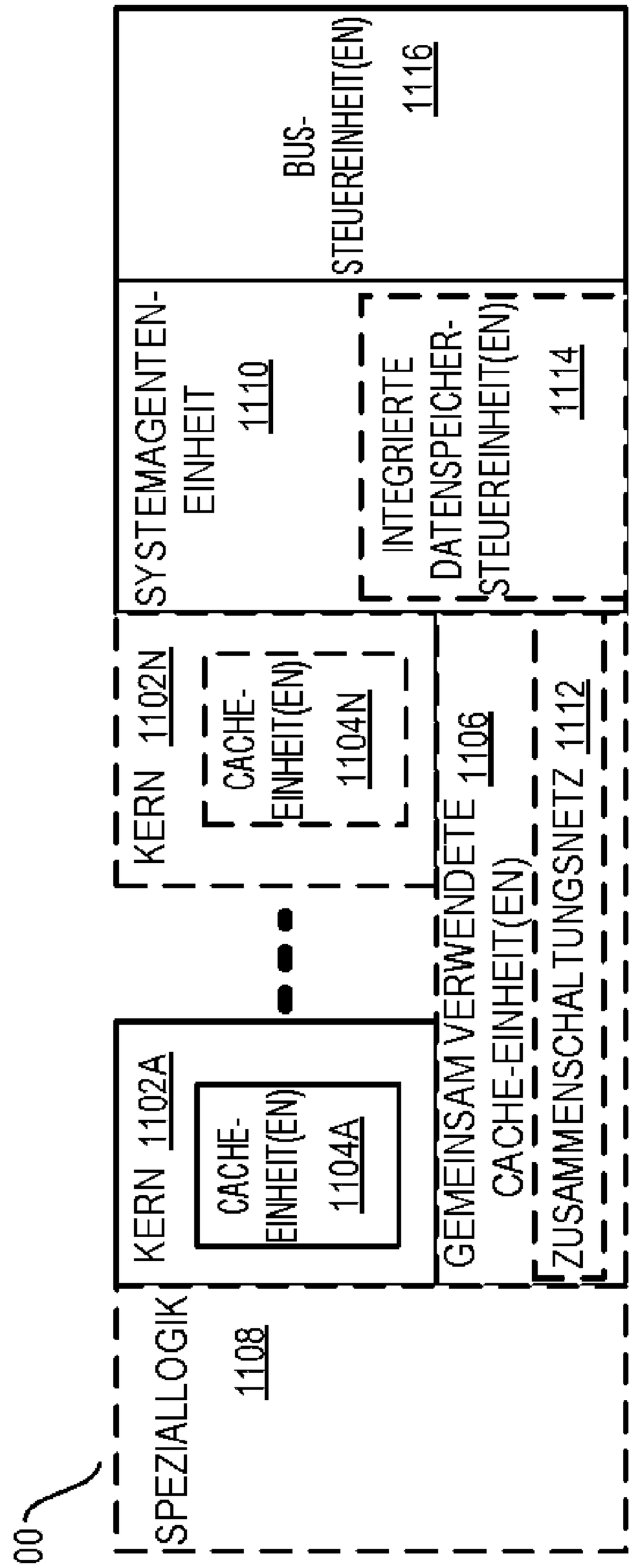

PROZESSOR 1100
SPEZIALLOGIK 1108
KERN 1102A
CACHE-EINHEIT(EN) 1104A
KERN 1102N
CACHE-EINHEIT(EN) 1104N
GEMEINSAM VERWENDETE CACHE-EINHEIT(EN) 1106
ZUSAMMENSCHALTUNGSNETZ 1112
SYSTEMAGENTEN-EINHEIT 1110
INTEGRIERTE DATENSPEICHER-STEUEREINHEIT(EN) 1114
BUS-STEUEREINHEIT(EN) 1116

FIG. 11

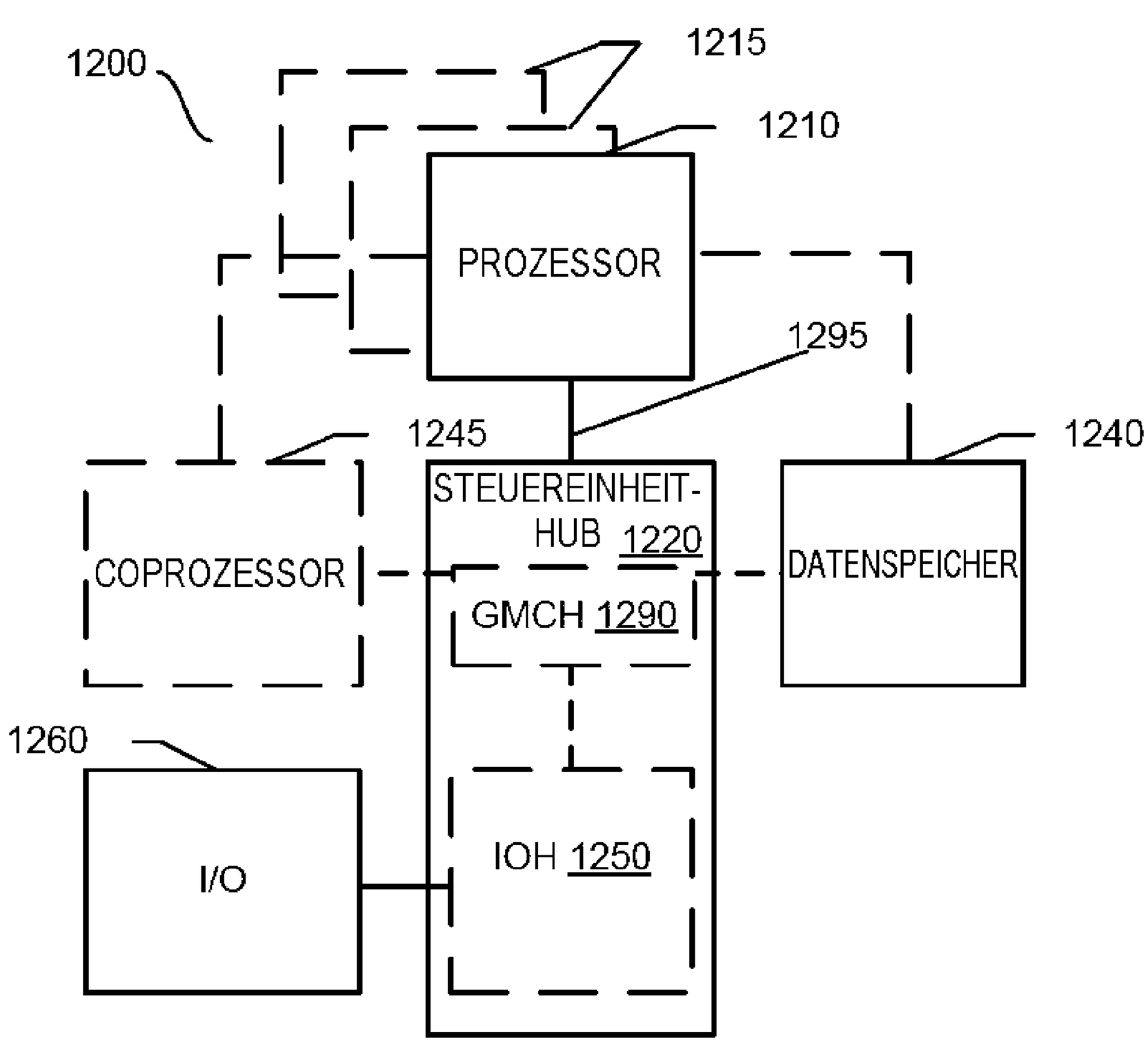

1200
1215
1210
PROZESSOR
1295
1245
1240
STEUEREINHEIT-
HUB 1220
COPROZESSOR
DATENSPEICHER
GMCH 1290
1260
I/O
IOH 1250

FIG. 12

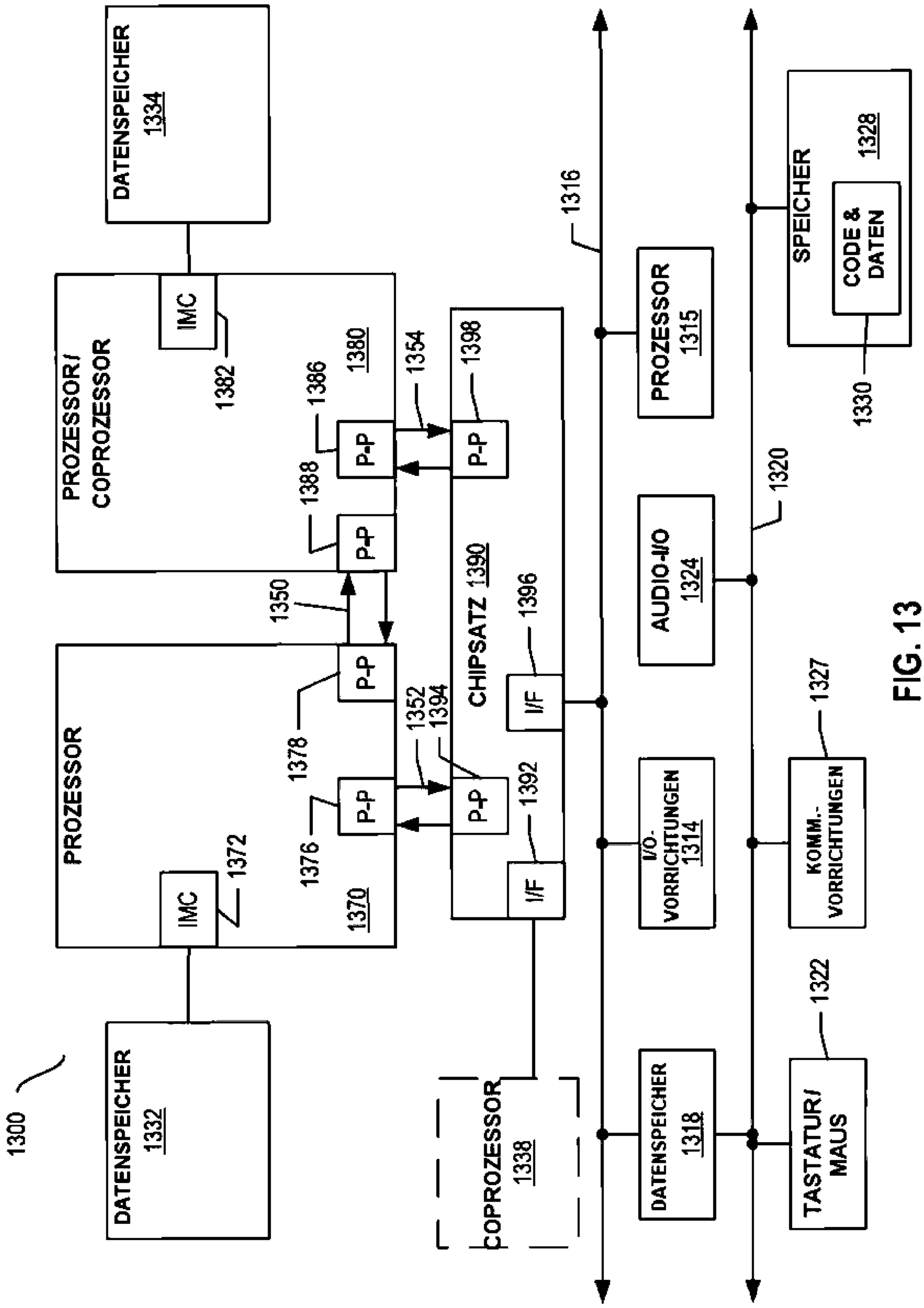

1300
DATENSPEICHER 1332
PROZESSOR
IMC
1372
1370
1376
1378
P-P
P-P
1352
1394
CHIPSATZ 1390
I/F
1392
COPROZESSOR 1338
1396
PROZESSOR/ COPROZESSOR
IMC
1382
1380
1386
1388
P-P
P-P
1350
1354
1398
DATENSPEICHER 1334
1316
PROZESSOR 1315
AUDIO-I/O 1324
I/O-VORRICHTUNGEN 1314
DATENSPEICHER 1318
1320
SPEICHER 1328
CODE & DATEN
1330
KOMM.-VORRICHTUNGEN
1327
TASTATUR/ MAUS
1322

FIG. 13

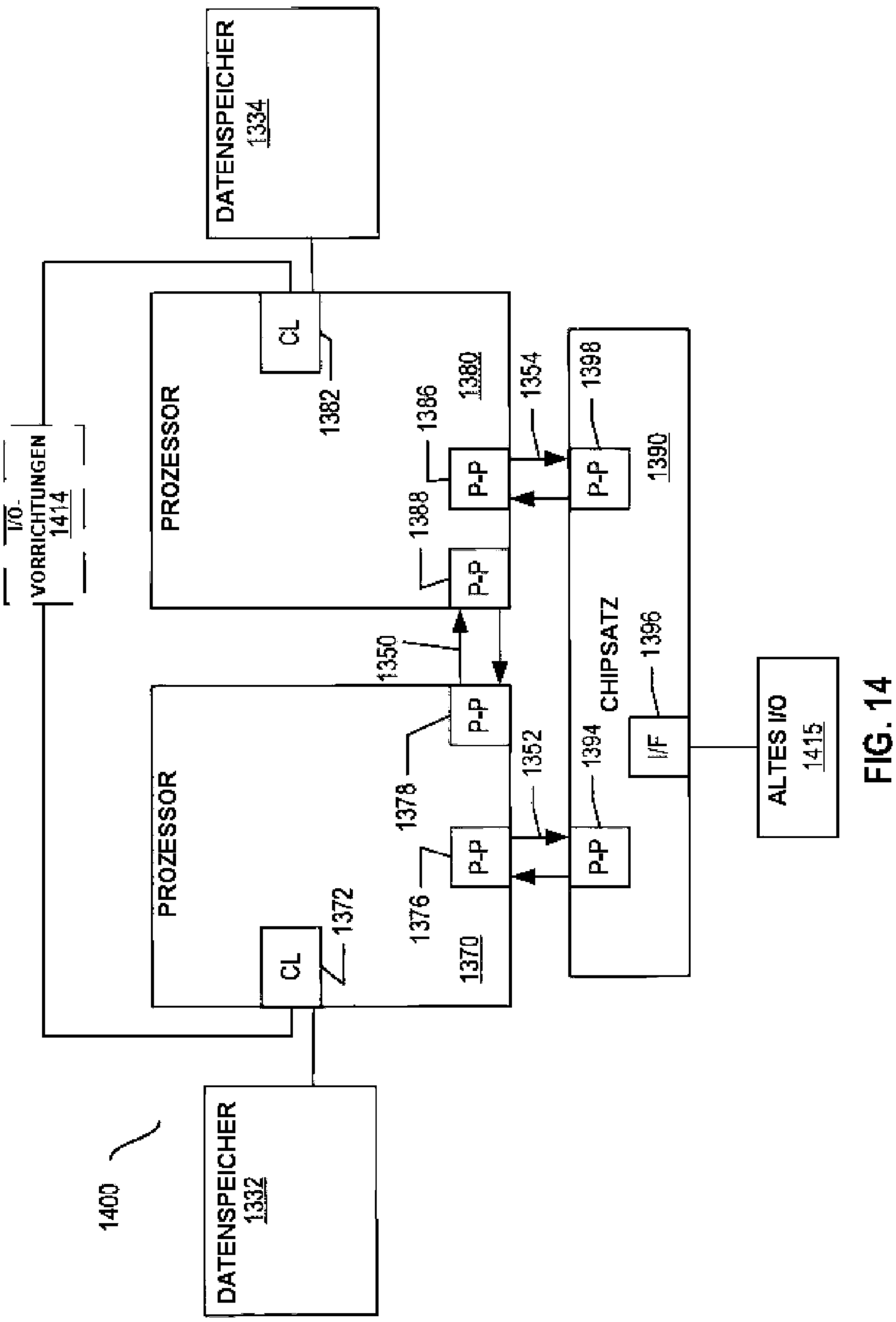

1400
DATENSPEICHER
1332
DATENSPEICHER
1334
I/O-
VORRICHTUNGEN
1414
PROZESSOR
CL
1372
1370
1376
1378
1380
1382
1386
1388
1350
1352
1354
P-P
CHIPSATZ
1390
1394
1396
1398
I/F
ALTES I/O
1415

FIG. 14

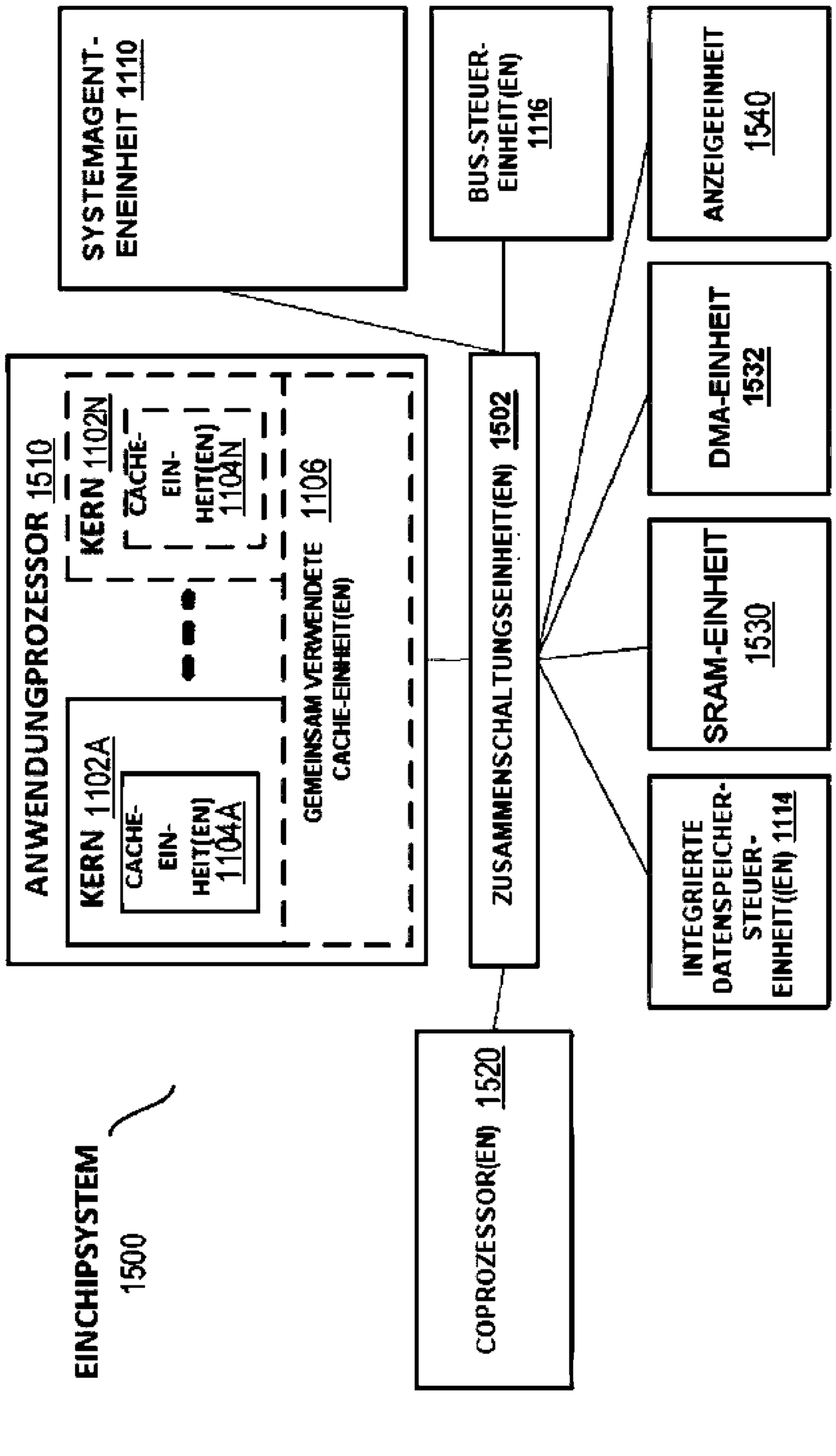

EINCHIPSYSTEM
1500
SYSTEMAGENT-EINHEIT 1110
BUS-STEUER-EINHEIT(EN) 1116
ANZEIGEEINHEIT 1540
ANWENDUNGPROZESSOR 1510
KERN 1102A
CACHE-EIN-HEIT(EN) 1104A
KERN 1102N
CACHE-EIN-HEIT(EN) 1104N
GEMEINSAM VERWENDETE CACHE-EINHEIT(EN) 1106
ZUSAMMENSCHALTUNGSEINHEIT(EN) 1502
DMA-EINHEIT 1532
SRAM-EINHEIT 1530
INTEGRIERTE DATENSPEICHER-STEUER-EINHEIT((EN) 1114
COPROZESSOR(EN) 1520

FIG. 15

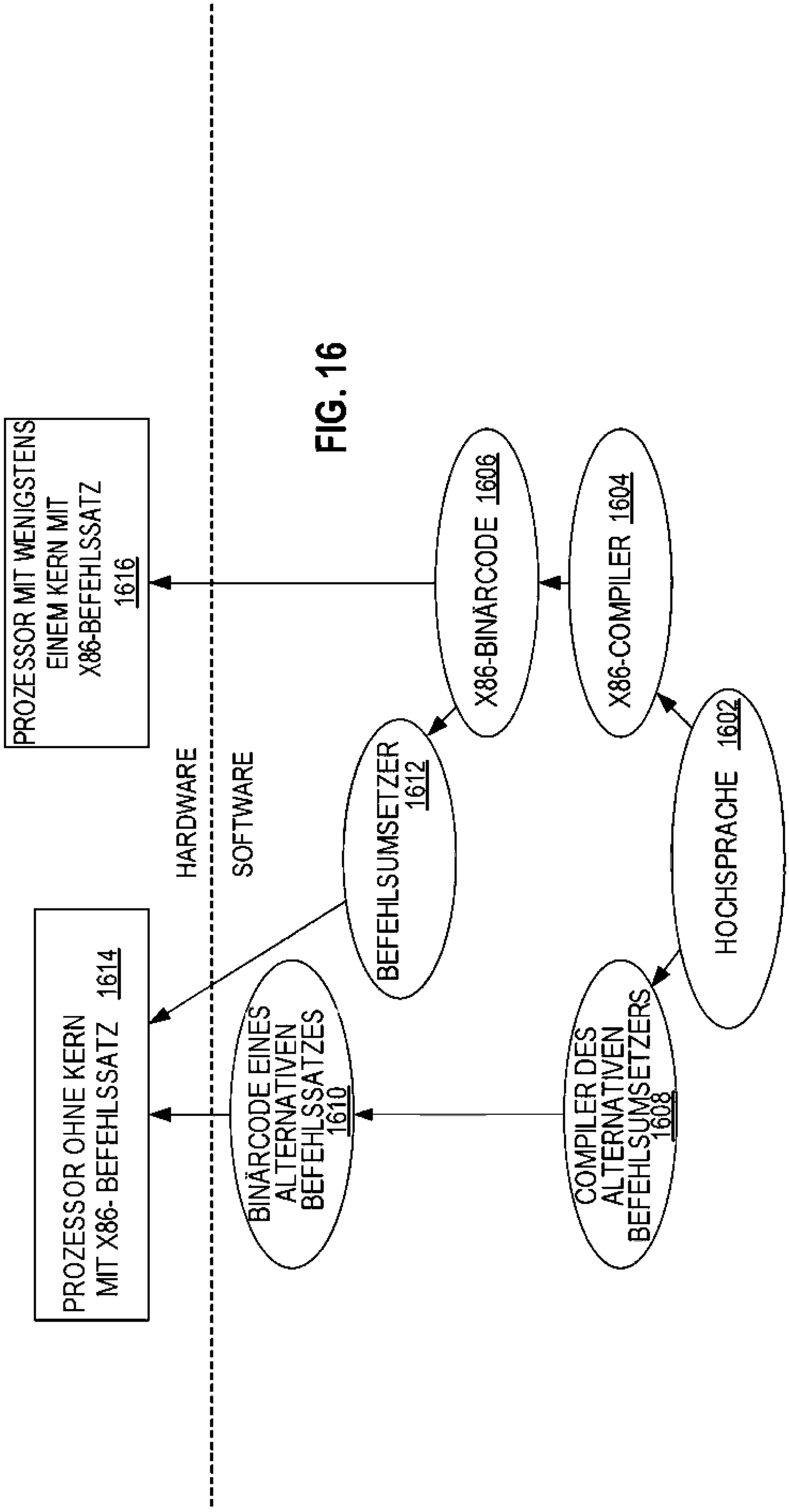

PROZESSOR MIT WENIGSTENS EINEM KERN MIT X86-BEFEHLSSATZ 1616
HARDWARE
SOFTWARE
X86-BINÄRCODE 1606
X86-COMPILER 1604
BEFEHLSUMSETZER 1612
HOCHSPRACHE 1602
PROZESSOR OHNE KERN MIT X86- BEFEHLSSATZ 1614
BINÄRCODE EINES ALTERNATIVEN BEFEHLSSATZES 1610
COMPILER DES ALTERNATIVEN BEFEHLSUMSETZERS 1608

FIG. 16