(19) Deutsches Patent- und Markenamt

(10) **DE 11 2007 000 812 B4** 2023.06.07

(12) # Patentschrift

(21) Deutsches Aktenzeichen: **11 2007 000 812.1**
(86) PCT-Aktenzeichen: **PCT/US2007/064450**
(87) PCT-Veröffentlichungs-Nr.: **WO 2007/115003**
(86) PCT-Anmeldetag: **20.03.2007**
(87) PCT-Veröffentlichungstag: **11.10.2007**
(43) Veröffentlichungstag der PCT Anmeldung in deutscher Übersetzung: **22.01.2009**
(45) Veröffentlichungstag der Patenterteilung: **07.06.2023**

(51) Int Cl.: ***G06F 9/44*** (2018.01)
G06F 9/46 (2006.01)
G06F 9/52 (2006.01)

Innerhalb von neun Monaten nach Veröffentlichung der Patenterteilung kann nach § 59 Patentgesetz gegen das Patent Einspruch erhoben werden. Der Einspruch ist schriftlich zu erklären und zu begründen. Innerhalb der Einspruchsfrist ist eine Einspruchsgebühr in Höhe von 200 Euro zu entrichten (§ 6 Patentkostengesetz in Verbindung mit der Anlage zu § 2 Abs. 1 Patentkostengesetz).

(30) Unionspriorität:
11/394,687 **30.03.2006** **US**

(62) Teilung in:
11 2007 003 801.2

(73) Patentinhaber:
Intel Corporation, Santa Clara, Calif., US

(74) Vertreter:
BOEHMERT & BOEHMERT Anwaltspartnerschaft mbB - Patentanwälte Rechtsanwälte, 80336 München, DE

(72) Erfinder:
Rajwar, Ravi, Portland, Oreg., US; Akkary, Haitham, Portland, Oreg., US; Lai, Konrad, Vancover, Wash., US

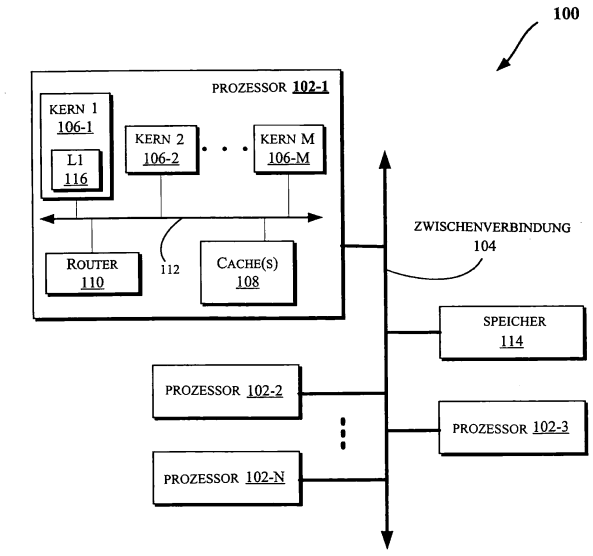
(56) Ermittelter Stand der Technik:

US	**2003 / 0 079 094**	**A1**
US	**2004 / 0 162 967**	**A1**

ANANIAN C. SCOTT [et al.]: Unbounded Transactional Memory. In: IEEE Xplore, Proceedings of the 11th Int'l Symposium on High-Performance Computer Architecture, 2005, HPCA-11 2005

(54) Bezeichnung: **VORRICHTUNG MIT EINER SPEICHEREINHEIT UND DREI LOGIKEN, VERFAHREN ZUM DURCHFÜHREN DER VERFAHRENSSCHRITTE DER VORRICHTUNG UNDSYSTEM MIT EINEM SPEICHER UND EINEM PROZESSOR ZUR BEREITSTELLUNG EINES EFFIZIENTEN MECHANISMUS' FÜR TRANSAKTIONALSPEICHERAUSFÜHRUNGEN IN OUT-OF-ORDER-PROZESSOREN**

(57) Zusammenfassung: Verfahren und Vorrichtung zum Bereitstellen von Transaktionalspeicherausführung in Out-of-Order-Prozessoren werden beschrieben. In einer Ausführungsform entspricht ein gespeicherter Wert der Anzahl von Transaktionalspeicherzugriffsanfragen, die unerledigt sind. Die gespeicherten Werte können in Übereinstimmung mit einer beschriebenen Ausführungsform verwendet werden, um eine verschachtelte Wiederherstellung im Falle eines Fehlers, Fehlfunktion, usw. bereitzustellen.



Beschreibung

HINTERGRUND

[0001] Die vorliegende Offenbarungsschrift bezieht sich generell auf den Bereich der Elektronik. Insbesondere bezieht sich eine Ausführungsform der Erfindung auf Transaktionalspeicher (TM)-Ausführung in Out-of-Order-Prozessoren.

[0002] Um die Leistungsfähigkeit zu verbessern, können manche Computersysteme mehrere Threads simultan ausführen. Generell kann ein Thread, bevor er auf eine Gemeinschaftsressource zugreift, einen Lock der Gemeinschaftsressource erwerben. In Situationen, in denen die Gemeinschaftsressource eine im Speicher gespeicherte Datenstruktur ist, können alle Threads, die versuchen, auf dieselbe Ressource zuzugreifen, die Ausführung ihrer Operationen angesichts der gegenseitigen Ausschließlichkeit, die von dem Locking-Mechanismus bereitgestellt wird, serialisieren. Dies kann sich nachteilig auf die Systemleistungsfähigkeit auswirken und kann Programmfunktionsfehler, z.B. aufgrund von Deadlock-Fehlern, verursachen.

[0003] Um den aus der Verwendung von Locking-Mechanismen resultierenden Leistungsverlust zu reduzieren, können manche Computersysteme Transaktionalspeicher verwenden. Transaktionalspeicher bezieht sich im Allgemeinen auf ein Synchronisationsmodell, das mehreren Threads erlaubt, simultan auf eine Gemeinschaftsressource zuzugreifen, ohne einen Locking-Mechanismus zu verwenden. Transaktionalspeicherausführung in Out-of-Order-Prozessoren kann jedoch ein Design komplizierter gestalten, z.B. aufgrund von aus spekulativer Verarbeitung resultierenden Fehlvorhersagen, die in Out-of-Order-Prozessoren auftreten.

[0004] In dem Artikel von Ananian C. Schott, et al.: „Unbounded TransactionalMemory," in IEEE Xplore, Proceedings of the 11th Int'1 Symposium on High-Performance Computer Architecture, 2005, HPCA-11 2005 ist eine sogenannte UTM-Architektur sowie eine LTM-Architektur beschrieben. Die UTM-Architektur erlaubt dabei Transaktionen, welche annähernd so groß sind wie der virtuelle Speicher. Es wird auch eine Semantik für eingebettete Transaktionen unterstützt, wobei innere Transaktionen in die von der äußeren Transaktion dargestellte atomare Region subsumiert sind. Die UTM-Archtektur erfordert dabei signifikante Änderungen sowohl an dem Prozessor als auch an dem Speicher-Subsystem einer aktuellen Computerarchitektur. Die LTM-Architektur unterstützt andererseits nicht Transaktionen von der Größe des virtuellen Speichers. Vielmehr begrenzt die LTM-Architektur den Fußabdruck einer Transaktion auf annähernd die Größe des physischen Speichers. Anders als ein UTM-Prozessor kann ein LTM-Prozessor mit einem konventionellen Prozessor pinkompatibel sein.

[0005] US 2003/0079094 A1 offenbart kritische Abschnitte von Multithread-Programmen, die normalerweise durch Sperren geschützt sind, die nur einem Thread Zugriff gewähren. Diese kritischen Abschnitte werden spekulativ von mehreren Threads gleichzeitig ausgeführt, wobei die Erfassung und Freigabe der Sperre entfällt. Nach Abschluss der spekulativen Ausführung ohne tatsächlichen Konflikt, wie unter Verwendung von Standard-Cache-Protokollen identifiziert werden kann, wird die spekulative Ausführung festgeschrieben, andernfalls wird die spekulative Ausführung gequetscht. Die spekulative Ausführung mit Elision der Sperrenerfassung ermöglicht ein höheres Maß an paralleler Ausführung in Multithread-Programmen mit aggressiver Sperrenverwendung.

[0006] US 2004/0162967 A1 offenbart eine Ausführungsform, welche die Ausführung eines Befehls zum Starten der transaktionalen Ausführung (STE) unterstützt, der den Beginn eines transaktional auszuführenden Befehlsblocks markiert. Beim Antreffen des STE-Befehls während der Ausführung eines Programms beginnt das System mit der transaktionalen Ausführung des Befehlsblocks, der dem STE-Befehl folgt. Änderungen, die während dieser Transaktionsausführung vorgenommen werden, werden nicht in den Architekturzustand des Prozessors übernommen, bis die Transaktionsausführung erfolgreich abgeschlossen ist.

KURZER ABRISS DER ERFINDUNG

[0007] Die Aufgabe der Erfindung besteht darin, einen effizienten Mechanismus für Transaktionalspeicherausführungen in Out-of-Order-Prozessoren bereitzustellen. Diese Aufgabe wird durch die Vorrichtung gemäß Patentanspruch 1, das Verfahren gemäß Patentanspruch 16 sowie das System gemäß Patentanspruch 21 gelöst. Bevorzugte Ausführungsformen sind Gegenstände der Unteransprüche

[0008] Der Patentanspruch 1 offenbart eine Vorrichtung, aufweisend: eine Speichereinheit zum Speichern eines Werts, der einer Anzahl von Transaktionalspeicherzugriffsanfragen entspricht, die unerledigt (uncommitted) sind; eine erste Logik zum Aktualisieren des in der Speichereinheit gespeicherten Werts für ein Auftreten eines Starts einer Transaktionalspeicherzugriffsanfrage und einer Erledigung der Transaktionalspeicherzugriffsanfrage, eine zweite Logik zum spekulativen Ausgeben einer Ladeoperation, nachdem eine vorherige der Transaktionalspeicherzugriffsanfragen auf dieselbe Stelle eines Speichers, die der Ladeoperation entspricht, zugegriffen hat, und eine dritte Logik zum erneuten Ausgeben der Ladeoperation nach einem Ausscheiden oder Erledigen, falls auf dieselbe Stelle vorher nicht zugegriffen wurde.

[0009] Der Patentanspruch 16 offenbart ein Verfahren, aufweisend: Aktualisieren eines gespeicherten Werts, der einer Anzahl von Transaktionalspeicherzugriffsanfragen entspricht, die unerledigt sind, als Antwort auf mindestens eine erste Instruktion, die einer Transaktionalspeicheranfrage entspricht; Durchführen einer oder mehrerer Operationen als Antwort auf eine zweite Instruktion, die der Transaktionalspeicheranfrage entspricht, spekulatives Ausgeben einer Ladeinstruktion, die der Transaktionalspeicheranfrage entspricht, falls ein Bit, das einem Abschnitt eines Cache entspricht, einen vorherigen transaktionalen Zugriff auf denselben Abschnitt des Cache anzeigt, und erneutes Ausgeben der Ladeoperation nach einem Ausscheiden oder Erledigen, falls auf denselben Abschnitt vorher nicht zugegriffen wurde.

[0010] Der Patentanspruch 21 offenbart ein System, aufweisend: einen Speicher zum Speichern von Daten; einen Prozessor, aufweisend: eine erste Logik zum Abrufen einer ersten Instruktion aus dem Speicher, die einem Start eines Transaktionalspeicherzugriffs entspricht, und einer zweiten Instruktion aus dem Speicher, die einem Ende des Transaktionalspeicherzugriffs entspricht; eine zweite Logik zum Aktualisieren eines in einer Speichereinheit gespeicherten Werts als Antwort auf eine oder mehrere der ersten Instruktion und der zweiten Instruktion, eine dritte Logik zum spekulativen Ausgeben einer Ladeoperation, nachdem eine vorherige der Transaktionalspeicherzugriffsanfragen auf dieselbe Stelle eines Speichers, die der Ladeoperation entspricht, zugegriffen hat, und eine vierte Logik zum erneuten Ausgeben der Ladeoperation nach einem Ausscheiden oder Erledigen, falls auf dieselbe Stelle vorher nicht zugegriffen wurde.

Figurenliste

[0011] Die detaillierte Beschreibung wird mit Bezug auf die begleitenden Figuren bereitgestellt. Die linkeste (n) Ziffer(n) eines Bezugszeichens in den Figuren identifiziert/identifizieren die Figur, in der das Bezugszeichen zuerst erscheint. Die Verwendung der gleichen Bezugszeichen in unterschiedlichen Figuren zeigt ähnliche oder identische Gegenstände an.

Fig. 1, **Fig. 6** und **Fig. 7** stellen Blockdiagramme von Ausführungsformen von Rechensystemen dar, die verwendet werden können, um verschiedene hier besprochene Ausführungsformen zu implementieren.

Fig. 2 stellt ein Blockdiagramm von Teilen eines Prozessorkerns gemäß einer Ausführungsform der Erfindung dar.

Fig. 3-4 stellen Blockdiagramme von Verfahren gemäß verschiedenen Ausführungsformen der Erfindung dar.

Fig. 5 stellt eine Ausführungsform eines Status- und Kontrollregisters einer Transaktion dar.

DETAILLIERTE BESCHREIBUNG

[0012] In der folgenden Beschreibung werden zahlreiche spezifische Details dargelegt, um ein tiefgehendes Verständnis von verschiedenen Ausführungsformen bereitzustellen. Einige Ausführungsformen können jedoch ohne diese spezifischen Details praktiziert werden. In anderen Fällen werden wohlbekannte Verfahren, Prozeduren, Komponenten und Schaltungen nicht detailliert beschrieben, um jeweilige Ausführungsformen nicht zu verschleiern.

[0013] Einige der hier besprochenen Ausführungsformen können effiziente Mechanismen für Transaktionalspeicherausführung in Out-of-Order-Prozessoren bereitstellen, wie z.B. den mit Bezug auf **Fig. 1-7** besprochenen Prozessoren. Insbesondere stellt **Fig. 1** ein Blockdiagramm eines Rechensystems 100 gemäß einer Ausführungsform der Erfindung dar. Das System 100 kann einen oder mehrere Prozessoren 102-1 bis 102-N umfassen (die hier generell als „Prozessoren 102“ oder „Prozessor 102“ bezeichnet werden). Die Prozessoren 102 können über eine Zwischenverbindung oder einen Bus 104 kommunizieren. Jeder Prozessor kann verschiedene Komponenten umfassen, von denen einige nur mit Bezug auf Prozessor 102-1 um der Klarheit

willen besprochen werden. Entsprechend kann jeder der verbleibenden Prozessoren 102-2 bis 102-N dieselben oder ähnliche mit Bezug auf den Prozessor 102-1 besprochene Komponenten umfassen.

[0014] In einer Ausführungsform kann der Prozessor 102-1 einen oder mehrere Prozessorkerne 106-1 bis 106-N (die hier als „Kerne 106“ oder allgemeiner als „Kern 106“ bezeichnet werden), einen Cache 108 (der einen oder mehrere private oder Gemeinschafts-Caches umfassen kann) und/oder einen Router 110 umfassen. Die Prozessorkerne 106 können auf einem einzigen integrierten Schaltungs (IC)-Chip implementiert sein. Darüber hinaus kann der Chip einen oder mehrere Gemeinschafts- und/oder private Caches (wie z.B. Cache 108), Busse oder Zwischenverbindungen (wie z.B. einen Bus oder Zwischenverbindung 112), Speichercontroller oder andere Komponenten umfassen.

[0015] In einer Ausführungsform kann der Router 110 verwendet werden, um zwischen verschiedenen Komponenten des Prozessors 102-1 und/oder Systems 100 zu kommunizieren. Darüber hinaus kann der Prozessor 102-1 mehr als einen Router 110 umfassen. Des Weiteren kann die Vielzahl der Router (110) kommunizierend sein, um Routing von Daten zwischen verschiedenen Komponenten innerhalb oder außerhalb des Prozessors 102-1 zu ermöglichen.

[0016] Der Cache 108 kann Daten (z.B. Instruktionen umfassend) speichern, die von einer oder mehreren Komponenten des Prozessors 102-1, wie z.B. den Kernen 106, benutzt werden. Der Cache 108 kann z.B. lokal in einem Speicher 114 gespeicherte Daten für einen schnelleren Zugriff durch die Komponenten des Prozessors 102 zwischenspeichern. Wie in **Fig. 1** gezeigt, kann der Speicher 114 mit den Prozessoren 102 über die Zwischenverbindung 104 kommunizieren. In einer Ausführungsform kann der Cache 108 ein Last Level Cache (LLC) sein. Jeder der Kerne 106 kann ferner einen Level 1 (L1)-Cache (116) umfassen (der hier allgemein als „L1-Cache 116“ bezeichnet wird). Des Weiteren kann der Prozessor 102-1 ferner einen Mid-Level-Cache umfassen, der von mehreren Kernen (106) gemeinsam benutzt wird. Verschiedene Komponenten des Prozessors 102-1 können mit dem Cache 108 über einen Bus (z.B. den Bus 112) und/oder einen Speichercontroller oder Hub direkt kommunizieren.

[0017] **Fig. 2** stellt ein Blockdiagramm von Teilen eines Prozessorkerns 106 gemäß einer Ausführungsform der Erfindung dar. In einer Ausführungsform stellen die in **Fig. 2** gezeigten Pfeile den Fluss der Instruktionen durch den Kern 106 dar. Einer oder mehrere Prozessorkerne (wie z.B. der Prozessorkern 106) können auf einem einzigen integrierten Schaltungschip (oder Die), wie z.B. mit Bezug auf **Fig. 1** besprochen, implementiert sein. Darüber hinaus kann der Chip einen oder mehrere Gemeinschafts- und/oder private Caches (z.B. Cache 108 aus **Fig. 1**), Zwischenverbindungen (z.B. Zwischenverbindungen 104 und/oder 112 aus **Fig. 1**), Speichercontroller oder andere Komponenten umfassen. In einer Ausführungsform kann der in **Fig. 2** gezeigte Prozessorkern 106 zum Ausführen von Transaktionalspeicherzugriffsanfragen in Hardware verwendet werden (was hier allgemein als beschränkter Transaktionalspeicher (RTM) bezeichnet werden kann).

[0018] Wie in **Fig. 2** dargestellt, kann der Prozessorkern 106 eine Fetch-Einheit 202 zum Abrufen von Instruktionen zur Ausführung durch den Kern 106 umfassen. Die Instruktionen können aus jeder Speichervorrichtung, wie z.B. dem Speicher 114 und/oder den mit Bezug auf **Fig. 6** und **Fig. 7** besprochenen Speichervorrichtungen, abgerufen werden. Der Kern 106 kann ebenfalls eine Decodiereinheit 204 zum Decodieren der abgerufenen Instruktion umfassen. So kann z.B. die Decodiereinheit 204 die abgerufene Instruktion in eine Vielzahl von uOps (Mikrooperationen) decodieren.

[0019] Zusätzlich kann der Kern 106 eine Schedule-Einheit 206 umfassen. Die Schedule-Einheit 206 kann verschiedene dem Speichern von decodierten Instruktionen zugeordneten Operationen durchführen (die z.B. von der Decodiereinheit 204 empfangen wurden), bis die Instruktionen zum Abfertigen bereit sind, z.B. bis alle Quellwerte einer decodierten Instruktion verfügbar sind. Die Schedule-Einheit 206 kann in einer Ausführungsform decodierte Instruktionen für eine Ausführungseinheit 208 zur Ausführung einplanen und/oder ausgeben (oder abfertigen). Die Ausführungseinheit 208 kann die abgefertigten Instruktionen ausführen, nachdem sie decodiert (z.B. durch die Decodiereinheit 204) und abgefertigt (z.B. durch die Schedule-Einheit 206) worden sind. Die Ausführungseinheit 208 kann in einer Ausführungsform mehr als eine Ausführungseinheit, wie z.B. eine Speicherausführungseinheit, eine Integer-Ausführungseinheit, eine Fließkomma-Ausführungseinheit oder andere Ausführungseinheiten, umfassen. Des Weiteren kann die Ausführungseinheit 208 Instruktionen out-of-order ausführen. Somit kann in einer Ausführungsform der Prozessorkern 106 ein Out-of Order-Prozessorkern sein. Der Kern 106 kann ferner eine Retirement-Einheit 210 umfassen. Die Retirement-Einheit 210 kann ausgeführte Instruktionen, nachdem sie erledigt (committed) worden sind, ausscheiden. Das Ausscheiden der ausgeführten Instruktionen kann in einer Ausführungsform im Erledigen eines Prozess-

orzustands ausgehend von der Ausführung der Instruktionen, im Deallozieren von von den Instruktionen verwendeten physikalischen Registern, usw. resultieren.

[0020] Wie in **Fig. 2** gezeigt, kann der Kern 106 einen Reorder-Puffer (ROB) 212 zum Speichern von Informationen über In-Flight-Instruktionen (oder uOps) für einen Zugriff durch verschiedene Komponenten des Prozessorkerns 106 umfassen. Der Kern 106 kann des Weiteren eine RAT (Register Alias Table) 214 zum Führen einer Abbildung von logischen (oder Architektur-) Registern (wie z.B. denen durch Operanden von Softwareinstruktionen identifizierten) auf entsprechende physikalische Register umfassen. In einer Ausführungsform kann jeder Eintrag in der RAT 214 eine ROB-Kennung umfassen, die jedem physikalischen Register zugeordnet wurde. Zusätzlich kann ein Ladepuffer 216 und ein Speicherpuffer 218 (die hier kollektiv als Speicherordnungspuffer (MOB) bezeichnet werden können) ausstehende Speicheroperationen speichern, die nicht aus einem Hauptspeicher geladen oder entsprechend in einen Hauptspeicher zurückgeschrieben haben (z.B. einen Speicher, der extern zum Prozessorkern 106 angeordnet ist, wie z.B. Speicher 114). Eine MOB-Logik 219 kann verschiedene auf die Puffer 216 und 218 bezogene Operationen durchführen, wie z.B. hier mit Bezug auf **Fig. 3** und **Fig. 4** besprochen wird.

[0021] Des Weiteren kann der Prozessorkern 106 eine Buseinheit 220 umfassen, um Kommunikation zwischen Komponenten des Prozessorkern 106 und anderen Komponenten (wie z.B. den mit Bezug auf **Fig. 1** besprochenen Komponenten) über einen oder mehrere Busse (z.B. Busse 104 und/oder 112) zu ermöglichen. Ein oder mehrere Füllpuffer 222 können temporär Daten, die (z.B. über die Busse 104 und/oder 112) von dem Speicher 114 empfangen wurden, vor dem Speichern der empfangenen Daten im Cache 116 speichern.

[0022] Wie in **Fig. 2** gezeigt, kann der Cache 116 eine oder mehrere Cache-Zeilen 224 (z.B. Cache-Zeilen 0 bis W) umfassen. In einer Ausführungsform kann jede Zeile des Cache 116 ein Transaction-Read-Bit 226 und/oder ein Transaction-Write-Bit 228 für jeden in dem Kern 106 ausgeführten Thread umfassen. Die Bits 226 und 228 können, wie mit Bezug auf **Fig. 3** besprochen, gesetzt oder gelöscht sein, um z.B. einen (Lade- und/oder Speicher-) Zugriff auf die entsprechende Cache-Zeile durch eine Transaktionalspeicherzugriffsanfrage anzuzeigen. Obwohl in **Fig. 2** jede Cache-Zeile 224 mit einem entsprechenden Bit 226 und 228 gezeigt ist, sind ebenfalls andere Konfigurationen möglich. Ein Transaction-Read-Bit 226 (oder Transaction-Write-Bit 228) kann z.B. einem ausgewählten Abschnitt des Cache 116 entsprechen, wie z.B. ein Cache-Block oder ein anderer Abschnitt des Cache 116. Die Bits 226 und/oder 228 können ebenfalls an anderen Stellen als im Cache 116 gespeichert werden, wie z.B. im Cache 108 aus **Fig. 1**, dem Speicher 114 oder z.B. einem Opfer-Cache.

[0023] Wie des Weiteren mit Bezug auf **Fig. 3** besprochen wird, kann der Kern 106 einen Transaktionstiefenzähler 230 zum Speichern eines Werts umfassen, der der Anzahl der Transaktionalspeicherzugriffsanfragen entspricht, die unerledigt (uncommitted) bleiben. Der in dem Zähler 230 gespeicherte Wert kann z.B. die Verschachtelungstiefe von mehreren Transaktionalspeicherzugriffsanfragen, die sich auf denselben Thread beziehen, anzeigen. In einem Fall können mehrere Transaktionalspeicherzugriffsanfragen resultieren, wenn eine Transaktion innerhalb einer ausstehenden Transaktion initiiert wird (wie z.B. über einen Bibliotheksaufruf oder andere verschachtelte Prozedur). Der Zähler 230 kann als ein beliebiger Typ einer Speichervorrichtung, wie z.B. ein Hardwareregister oder eine in einem Speicher (z.B. dem Speicher 114 oder Cache 116) gespeicherte Variable, implementiert werden. Der Kern 106 kann ebenfalls eine Transaktionstiefenzählerlogik 232 zum Aktualisieren des in dem Zähler 230 gespeicherten Werts umfassen. Wie mit Bezug auf **Fig. 3** des Weiteren besprochen wird, kann der Kern 106 zusätzlich eine Transaktions-Checkpoint-Logik 234 zum Checkpointen (oder Speichern) des Zustands von verschiedenen Komponenten des Kerns 106 und eine Transaktionswiederherstellungslogik 236 zum Wiederherstellen des Zustands von verschiedenen Komponenten des Kerns 106 umfassen. Der Kern 106 kann ebenfalls eine Cache-Logik 239 umfassen, die verschiedene Operationen durch Zugreifen auf den Cache 116 durchführen kann, wie hier des Weiteren z.B. mit Bezug auf **Fig. 3** und **Fig. 4** beschrieben wird. Zusätzlich kann der Kern 106 einen oder mehrere zusätzliche Register 240 umfassen, die sich auf verschiedene Transaktionalspeicherzugriffsanfragen beziehen, wie z.B. Status und Kontrolle einer Transaktion (TXSR) (die hier des Weiteren mit Bezug auf **Fig. 4** besprochen wird), Transaktionsinstruktionszeiger (TXIP) (der z.B. ein Instruktionszeiger auf eine Instruktion am Anfang (oder unmittelbar davor) der entsprechenden Transaktion sein kann) und/oder Transaktions-Stack-Zeiger (TXSP) (der z.B. ein Stack-Zeiger auf das oberste Element eines Stacks sein kann, der verschiedene Zustände einer oder mehrerer Komponenten des Kerns 106 speichert), wie des Weiteren mit Bezug auf **Fig. 3** besprochen wird.

[0024] Insbesondere stellt **Fig. 3** ein Blockdiagramm einer Ausführungsform eines Verfahrens 300 zum Ausführen einer oder mehrerer Operationen, die sich auf eine Transaktionalspeicherzugriffsanfrage beziehen, dar. In einer Ausführungsform können verschiedene mit Bezug auf **Fig. 1-2** und **Fig. 6-7** besprochene Komponenten zum Durchführen einer oder mehrerer der mit Bezug auf **Fig. 3** besprochenen Operationen benutzt werden.

[0025] Bezug nehmend auf **Fig. 1-3** wird in einer Operation 302 eine Transaktionalspeicherzugriffsanfrage (wie z.B. eine oder mehrere Lade- oder Speicheroperationen eines Speichers) empfangen. Die Fetch-Einheit 202 kann z.B. eine Instruktion abrufen, die den Start einer Transaktionalspeicherausführung anzeigt. In Übereinstimmung mit mindestens einer Instruktionssatzarchitektur kann die Instruktion, die den Start einer Transaktionalspeicherausführung anzeigt, ein TXMBEG <user handler IP> sein, wobei user handler IP eine Handler-Prozedur eines Benutzers identifiziert, auf die die Ausführung im Falle eines Abbruchs, Fehlers oder anderer Fehlkonditionen umgeleitet werden kann. Eine transaktionale Endinstruktion (wie z.B. TXMEND in Übereinstimmung mit mindestens einer Instruktionssatzarchitektur) kann ebenfalls das Ende einer Transaktion anzeigen. In einer Ausführungsform können alle Operationen zwischen der TXMBEG-Instruktion und TXMEND als transaktional markiert werden. In einer Ausführungsform können die transaktionalen Operationen standardmäßig transaktional (und z.B. durch einen Instruktionspräfix oder Parameter explizit nicht-transaktional) sein, um z.B. die transaktionale Verwendung von nicht-transaktionalen Altbibliotheken ohne Codeänderungen zu ermöglichen. Alternativ können die transaktionalen Operationen standardmäßig nicht-transaktional (und z.B. durch einen Instruktionspräfix oder Parameter explizit transaktional) sein. In einer Ausführungsform kann ein spezieller Satz von explizit nicht-transaktionalen Instruktionen benutzt werden, die nicht-transaktional behandelt werden, obwohl sie innerhalb einer Transaktion, z.B. zwischen einer TXMBEG-Instruktion und einer TXMEND-Instruktion, vorkommen können. In einer Ausführungsform können ebenfalls nicht-transaktionale Speicheroperationen, die innerhalb einer Transaktion (z.B. zwischen einer TXMBEG-Instruktion und einer TXMEND-Instruktion) auftreten, durch die Ausführungseinheit 208 als Write-Through-Operationen zum Speicher 114 ausgeführt werden. In einer Ausführungsform kann die Einstellung der standardmäßigen Behandlung der Operationen mit der TXMBEG-Instruktion bereitgestellt werden, z.B. bereitgestellt durch einen entsprechenden Parameter oder Präfix. In einer Ausführungsform kann die Einstellung des Standards durch ein Modus-Bit in einem Kontrollregister bereitgestellt werden.

[0026] In einer Operation 304 kann die Logik 232 den Zähler 230 aktualisieren (z.B. den Zähler 230 abhängig von der Implementierung inkrementieren oder dekrementieren). In einer Ausführungsform kann die Ausführung der TXMBEG-Instruktion (z.B. durch die Ausführungseinheit 208) im Aktualisieren des Zählers 230 resultieren. Alternativ kann der Zähler 230 zum Ausgabezeitpunkt aktualisiert werden, z.B. wenn die Schedule-Einheit 206 die TXMBEG-Instruktion ausgibt. Die Transaktions-Checkpoint-Logik 234 kann ebenfalls in der Operation 304 den Zustand von verschiedenen Komponenten (z.B. Zähler 230 und/oder Register 240) checkpointen. Die Logik 234 kann z.B. den Zustand einer oder mehrerer Komponenten des Kerns 106 in einer Speichervorrichtung (z.B. in dem Cache 116, Cache 108 und/oder Speicher 114) speichern. Wie hier des Weiteren beschrieben wird, kann die Logik 234, da der Kern 106 mehr als eine Transaktionalspeicherzugriffsanfrage zum selben Zeitpunkt (und/oder spekulativ) durchführen kann, mehr als einen Zustand von verschiedenen Komponenten des Kerns 106 speichern, und in einer Ausführungsform können die verschiedenen gecheckpointeten Zustände in einer Datenstruktur, die als ein Stack implementiert ist, gespeichert werden. In einer Ausführungsform kann die Logik 234 einen der äußersten Transaktionalspeicheranfrage entsprechenden gecheckpointeten Zustand speichern.

[0027] In einer Operation 306 können eine oder mehrere Instruktionen (oder Operationen), die der Transaktionalspeicherzugriffsanfrage der Operation 302 entsprechen, ausgeführt werden, z.B. durch die Ausführungseinheit 208. Während einer Ausführung von Instruktionen in Operation 306 kann jedes Mal, wenn auf einen Abschnitt des Cache 116 zugegriffen wird, das entsprechende Bit (z.B. Bits 226 und/oder 228) aktualisiert werden, z.B. abhängig von der Implementiert gesetzt oder gelöscht werden. Die Cache-Logik 239 kann z.B. transaktionale Instruktionen, die auf den Cache 116 zugreifen, identifizieren und die entsprechenden Bits 226 und/oder 228 aktualisieren. In einer Ausführungsform können die Instruktionen der Operation 306 explizit oder implizit als transaktionale oder nicht-transaktionale Operationen identifiziert werden, z.B. mittels eines Präfix (oder Kontroll-Bits), der mit der Instruktion bereitgestellt und/oder an einer entsprechenden Stelle gespeichert wird, wie z.B. innerhalb eines entsprechenden Eintrags im ROB 212.

[0028] In einer Operation 308 fährt das Verfahren 300, falls eine andere Transaktionalspeicherzugriffsanfrage empfangen wird, mit der Operation 304 fort. Andernfalls wird, falls keine zusätzliche Transaktionalspeicherzugriffsanfrage in Operation 308 empfangen wird, in einer Operation 310 festgelegt, ob ein Konflikt oder eine Abbruchbedingung, die der Ausführung der Instruktionen der Operation 306 entspricht, existiert. Falls

ein Konflikt mit einer anderen Instruktion (die eine einem anderen Thread entsprechende Instruktion sein kann, der z.B. auf demselben oder einem Differenzprozessorkern (difference processor core) ausgeführt wird) besteht, wird die Transaktion in einer Operation 312 abgebrochen. Eine Konfliktinstruktion kann z.B. eine Snoop-Invalidierung für einen Eintrag im Cache 116 senden, auf den die Transaktion in Operation 302 zugreift (oder den Zugriff darauf markiert hat, der durch einen in den Bits 226 und/oder 228 gespeicherten Wert angezeigt wird). Darüber hinaus kann, falls das Transaction-Write-Bit 228 des Abschnitts des Cache 116 einen vorherigen Schreibzugriff auf denselben Abschnitt des Cache anzeigt, eine Snoop-Leseanfrage von einem anderen Thread auf denselben Abschnitt des Cache 116 in Operation 312 abgebrochen werden. Der Abbruch in Operation 310 kann ebenfalls aufgrund eines implementierungsspezifischen Ereignisses, das einen Abbruch erzwingt, (z.B. ein nicht-zwischenspeicherbares (UC) Ereignis, ein I/O-Ereignis, ein Pufferüberlauf, usw.) erfolgen.

[0029] Falls in einer Ausführungsform ein Snoop in einer Invalidierung von im Speicher gespeicherten Daten (wie z.B. im Cache 116 gespeichert) resultiert, kann die Ladeinstruktion am obersten Element des Ladepuffers 216 in die Snoop-Überprüfung einbezogen werden und eine vollständige Adressenüberprüfung kann (statt einer partiellen Adressenüberprüfung), z.B. durch die MOB-Logik 219, durchgeführt werden. Falls die Ladeinstruktion am obersten Element des Ladepuffers 216 mit dem Snoop in Operation 310 kollidiert, kann die entsprechende Transaktionalspeicheranfrage in Operation 312 abgebrochen werden. Alternativ kann eine explizite Abbruchanfrage die Durchführung von Operation 310 verursachen. So kann z.B. in Übereinstimmung mit mindestens einer Instruktionssatzarchitektur die Instruktion, die den Abbruch einer Transaktionalspeicherausführung anzeigt, TXMABT sein. Die Abbruchinstruktion kann in einem expliziten Abbruch resultieren, der im Falle von mehreren verschachtelten Transaktionalspeicherzugriffsanfragen (wie z.B. durch den im Zähler 230 gespeicherten Wert angezeigt) in einem Zurückkehren zu einem vorherigen Zustand und einem Aufrufen des entsprechenden Handlers (z.B. dem innersten Handler im Fall von mehreren ausstehenden Transaktionalspeicherzugriffsanfragen, die mit der entsprechenden TXNMEG-Instruktion, wie z.B. mit Bezug auf Operation 302 besprochen, identifiziert werden können) resultieren kann. In einer Ausführungsform kann der Abbruch in einem Zurückkehren zu dem gecheckpointeten Zustand, der der äußersten Transaktion entspricht, und einem Aufrufen des entsprechenden äußersten TXMBEG-Handlers resultieren. In einer Ausführungsform kann die Transaktionswiederherstellungslogik 236 in Operation 312 einen vorherigen Zustand von verschiedenen Komponenten des Kerns 106, wie mit Bezug auf **Fig. 2** besprochen, wiederherstellen.

[0030] In einer Ausführungsform kann das Checkpointen in Operation 304 mittels eines Copy-on-Write-Verfahrens durchgeführt werden, um einen Registerwiederherstellungszustand in einem Bit, das in den Registern 240 und/oder einem entsprechenden Eintrag der RAT 214 vorhanden ist, aufzuzeichnen. So kann z.B. ein 1-Bit-Array jeder RAT-Instanzierung zugeordnet werden (alternativ kann nur ein solches Array verwendet werden). Während der im Zähler 230 gespeicherte Wert eine ausstehende Transaktion anzeigt (z.B. in einer Ausführungsform TXND > 0) und während danach die Decodiereinheit 204 eine Registerumbenennungsoperation durchführt, wird das Array geprüft. Falls das entsprechende Bit keinen vorherigen Zugriff anzeigt, wird eine uOp in den Instruktionsfluss und den ROB 212 eingefügt, um die entsprechenden Daten in einen der Register 240 zu schreiben, z.B. vor dem Umbenennen. Falls das entsprechende Bit einen vorherigen Zugriff anzeigt, ist kein Kopieren notwendig, da die entsprechenden Daten bereits vor einem Schreiben kopiert worden sind. Falls eine Branch-Fehlvorhersage auftritt, können die verschiedenen 1-Bit-Arrays miteinander mittels OR logisch verknüpft werden, z.B. um das weitergehende Start-Array zu bilden. Falls dies gemacht wird, müssen die wiederholten Fehlvorhersagen nicht in wiederholten Copy-on-Write-Operationen resultieren, da die Existenz einer Kopie von gegebenen Daten in einem Register des Kerns 106 garantiert wird. Dies kann den Leistungs-Overhead von Copy-on-Write beschränken.

[0031] In einer Ausführungsform werden, wenn der Cache 116 geräumt wird, die Einträge, die ein aktives (z.B. gesetztes) Bit 226 oder Bit 228 haben, zuletzt geräumt. Falls kein weiterer verfügbarer Platz im Cache 116 verbleibt, kann die Programmausführung an einen Benutzer-Handler umgeleitet werden (wie z.B. der Handler der entsprechenden TXNMEG-Instruktion). Zusätzlich können die Füllpuffer 222 verwendet werden, um nicht-transaktionale Operation(en) (z.B. Speichern von Daten) zu bedienen, während eine Transaktion aktiv aber temporär suspendiert ist.

[0032] Falls keine Konflikte existieren, wird in Operation 314 bestimmt, ob die einer Transaktionalspeicherzugriffsanfrage entsprechende(n) Transaktionalspeicherzugriffsoperation(en) (wie z.B. durch eine TXMBEG- und entsprechende TXMEND-Instruktion identifiziert) erfolgreich erledigt werden (z.B. durch die Retirement-Einheit 210). Falls die Transaktionalspeicherzugriffsoperation(en) unerledigt bleibt/bleiben, fährt das Verfahren 300 mit der Operation 306 fort, um die entsprechende(n) Instruktion(en) auszuführen. Sobald die einer

Transaktionalspeicherzugriffsanfrage entsprechende(n) Transaktionalspeicherzugriffsoperation(en) erfolgreich erledigt wird/werden, aktualisiert die Logik 232 den Zähler 230 in einer Operation 316. In einer Ausführungsform kann die Ausführung (z.B. durch die Ausführungseinheit 208) oder Ausgabe (z.B. durch die Schedule-Einheit 206) der TXMEND-Instruktion im Erledigen einer Transaktionalspeicherzugriffsanfrage und somit der Durchführung von Operation 316 resultieren.

[0033] In einer Operation 318 wird bestimmt, ob irgendwelche anderen Transaktionalspeicherzugriffsanfragen zugeordnete(n) Transaktionalspeicherzugriffsoperation(en) unerledigt bleiben. In einer Ausführungsform kann die Operation durch Logik 232 durchgeführt werden, z.B. durch Bestimmen, ob der im Zähler 230 gespeicherte Wert die Existenz von verbleibenden Transaktionalspeicherzugriffsanfragen anzeigt (z.B. wenn der Zähler 230 in einer Ausführungsform nicht gelöscht ist). Falls andere in Operation 318 auszuführende(n) Operation(en) verbleibt/verbleiben, fährt das Verfahren 300 mit der Operation 306 fort. Sobald keine weiteren Transaktionalspeicherzugriffsanfragen entsprechenden Operationen bleiben (z.B. wenn der Zähler 230 gelöscht ist), werden in einer Operation 320 die entsprechenden Daten (z.B. innerhalb des Cache 116, Cache 108 und/oder Speicher 114) atomar in Übereinstimmung mit Operationen der erledigten Transaktionalspeicherzugriffsanfrage(n) aktualisiert. Die TXMEND-Instruktion kann ebenfalls einen Erledigungspunkt für die entsprechende Transaktionalspeicherzugriffsanfrage anzeigen, falls der im Zähler 230 gespeicherte Wert anzeigt, dass die TXMEND dem äußersten Transaktionalspeicherzugriff entspricht (z.B. wenn der Zähler 230 gelöscht ist). Die entsprechenden Bits 226 und/oder 228 können in einer Ausführungsform ebenfalls aktualisiert werden (z.B. in einer Ausführungsform gelöscht werden), um anzuzeigen, dass keine weiteren Transaktionalspeicherzugriffsanfragen im Kern 106 ausstehen. Entsprechend kann die TXMEND-Instruktion in einer atomaren Erledigung resultieren, z.B. durch Sicherstellen, dass alle Spekulationen und/oder Ausnahmen aufgelöst wurden, alle Cache-Blöcke, die dem Transaktionsschreibsatz gehören, sich in einem exklusiven Zustand befinden und/oder alle Blöcke, die dem Lesesatz gehören, gegenwärtig vorhanden und valide sind. Hier besprochene Lese- oder Schreibsätze können sich allgemein auf Sätze von Daten innerhalb des Speichers (z.B. Cache 116) beziehen, die jeweils den Bits 226 und 228 entsprechen. In einer Ausführungsform werden die eingehenden Snoops blockiert, während der entsprechende Schreibsatz erledigt wird.

[0034] In einer Ausführungsform kann der Kern 106 verschiedene Instruktionen (oder Operationen), die mit Bezug auf das Verfahren 300 beschrieben wurden, spekulativ ausführen, wie des Weiteren mit Bezug auf **Fig. 4** beschrieben wird. Verschiedene Operationen des Verfahrens 300 können ebenfalls out-of-order durchgeführt werden. So können z.B. Operationen 306, 308 und 310 in einer beliebigen Reihenfolge durchgeführt werden. Darüber hinaus können in einer Ausführungsform Snoop-Anfragen vor einer Durchführung von Operationen 310, 312, 314 und/oder 320 blockiert werden und nach der Durchführung der Operationen 310, 312, 314 und/oder 320 entblockiert werden.

[0035] **Fig. 4** stellt ein Blockdiagramm einer Ausführungsform eines Verfahrens 400 zum Ausführen einer oder mehrerer einer Transaktionalspeicherzugriffsanfrage entsprechenden Ladeoperationen dar. In einer Ausführungsform können verschiedene mit Bezug auf **Fig. 1-2** und **Fig. 6-7** besprochene Komponenten benutzt werden, um eine oder mehrere der mit Bezug auf **Fig. 4** besprochenen Operationen durchzuführen.

[0036] Bezug nehmend auf **Fig. 1-4** kann in einer Operation 402 die Decodiereinheit 204 eine abgerufene Instruktion decodieren und feststellen, ob die abgerufene Instruktion einer Transaktionalspeicherzugriffsladeoperation entspricht. Falls die Instruktion einer Transaktionalspeicherzugriffsladeoperation nicht entspricht, kann der Kern 106 die Instruktion entsprechend verarbeiten (404). So kann z.B. für eine Transaktionalspeicherzugriffsspeicheroperation der Kern 106 die Speicheroperation (oder Instruktion) an den Cache 116 nach dem Ausscheiden der entsprechenden Transaktionalspeicherzugriffsanfrage ausgeben. In einer Ausführungsform kann/können die der Speicheroperation entsprechende(n) Adresse oder Daten spekulativ berechnet und im Speicherpuffer 218 gespeichert werden. Sobald die entsprechende Transaktionalspeicherzugriffsanfrage ausgeschieden ist, kann das Speichern an den Cache in Übereinstimmung mit den vorberechneten Daten ausgegeben werden. In einer Ausführungsform kann das ausgeschiedene Speichern des Weiteren ein Setzen des zugegriffenen Cache-Blocks (z.B. des entsprechenden TXw-Bits 228) verursachen, falls das Bit nicht gesetzt ist, wenn das Speichern an den Cache 116 ausgegeben wird.

[0037] In einer Operation 406 kann die Logik 239 bestimmen, ob eine vorherige Transaktionalspeicherzugriffsanfrage auf dieselbe Stelle im Cache 116 zugegriffen hat, die von der Ladeinstruktion der Operation 402 adressiert wurde. In einer Ausführungsform kann die Logik 239 auf das entsprechende Bit 226 des Cache 116 zugreifen, um die Operation 406 durchzuführen. Falls auf dieselbe Stelle vorher zugegriffen wurde, kann die Schedule-Einheit 206 die Ladeinstruktion spekulativ ausgeben. Andernfalls kann die MOB-Logik 219 die Ladeinstruktion nach dem Ausscheiden (oder Erledigen, wie durch einen entsprechenden Ein-

trag im ROB 212 angezeigt wird) in einer Operation 410 erneut ausgeben und dann das entsprechende TXr-Bit 226 im Cache 116 in Operation 412 setzen. In einer Ausführungsform kann das entsprechende Bit 226, falls darauf durch ein vorheriges transaktionales Laden zugegriffen wurde, ebenfalls im Ladepuffer 216 gespeichert werden. Sobald eine Ladeoperation erledigt ist, wie durch einen entsprechenden Eintrag im ROB 212 angezeigt wird, kann der ROB 212 ein Signal an den Ladepuffer 216 (oder die MOB-Logik 219) senden, um anzuzeigen, dass die entsprechende Ladeoperation erledigt ist. Somit kann ein Bereitstellen des Bits 226 im Ladepuffer 216 ein Wiederzugreifen auf den Cache 116 für die Operationen 410 und 412 beschränken, um z.B. die Cache-Bandbreite zu optimieren.

[0038] In einer Ausführungsform können die Lade- oder Speicheroperationen einer Transaktionalspeicherzugriffsanfrage implizit oder explizit als transaktionale oder nicht-transaktionale Operationen identifiziert werden, z.B. indem beim Laden erlaubt wird, Lese- oder Schreibsätze ohne Serialisierungen und ohne Aktualisierung auszugeben, auf den Cache 116 aber wieder zugegriffen wird, wenn sich die Operation am obersten Element des entsprechenden Lade- oder Speicherpuffers (z.B. Puffer 216 bzw. 218) befindet, um explizit die Lade- oder Speicheradresse als zu dem Lese- oder Schreibsatz der Transaktion zugehörig zu identifizieren. Dies ermöglicht ein spekulatives Ausgeben von Lade- und Speichervorgängen ohne fälschliches Aktualisieren der Lese/Schreibsätze und ermöglicht somit eine präzise Identifikation von Lade- und Speichervorgängen und kann des Weiteren automatisch Branch-Fehlvorhersagen behandeln. Die Speicheroperationen können die Lese/Schreibsätze aktualisieren, wenn die Speicherinstruktionen den ältesten Abschnitt des Speicherpuffers 218 belassen (der in einer Ausführungsform als eine Seniorspeicherschlange bezeichnet werden kann). Darüber hinaus können implizite Einfügungen die Unterstützung von Altbibliotheken in den hier besprochenen Ausführungsformen ermöglichen.

[0039] **Fig. 5** stellt eine Ausführungsform eines Zustands- und Kontrollregisters einer Transaktion (TXSR) 500 dar. In einer Ausführungsform kann das TXSR-Register 500 verwendet werden, um verschiedene z.B. mit Bezug auf **Fig. 2-3** hier besprochene Ausführungsformen zu implementieren. Die Register 240 aus **Fig. 2** können z.B. das TXSR-Register 500 in einer Ausführungsform umfassen. Darüber hinaus können die mit Bezug auf **Fig. 2-4** besprochenen Operationen durch Zugreifen auf das Register 500 durchgeführt werden.

[0040] Wie in **Fig. 5** gezeigt, kann das TXSR-Register 500 einen oder mehrere Einträge 502 (oder in einer Ausführungsform Bits) umfassen. Jeder Eintrag 502 kann den dargestellten Namen (504) in Übereinstimmung mit mindestens einer Instruktionssatzarchitektur umfassen. Ebenfalls kann jeder Eintrag 502 eine entsprechende Klasse 506 umfassen, z.B. um Bits, die ähnlich sind, in einer Kategorie zu gruppieren. **Fig. 5** zeigt zusätzlich eine jedem Eintrag 502 entsprechende Beschreibung 508. In einer Ausführungsform sind die Klasseninformationen 506 und Beschreibung 508 für Informationszwecke angegeben und können nicht im aktuellen Register 500 gespeichert werden. Falls einer der Einträge 502 den durch die Beschreibung 508 identifizierten Zustand anzeigt, können in einer Ausführungsform eine oder mehrere Aktionen 510 ergriffen werden. In **Fig. 5** können TXDR und TXFR Register (z.B. enthalten in den Registern 240 aus **Fig. 2**) für Debug-Zwecke sein. TXDR kann z.B. die Adresse einer Speicherinstruktion oder die Snoop-Adresse eines Datenkonflikts speichern, bevor eine Transaktion die Operation abbricht. Darüber hinaus kann TXFR den Instruktionszeiger (IP) einer Fehlinstruktion speichern, die einen Abbruch in einer Transaktion verursacht hat. In einer Ausführungsform können die in TXDR und TXFR gespeicherten Werte durch eine Transaktions-Handler-Routine (z.B. ein Handler, der durch die TXMBEG identifiziert wird) nach dem Abbruch gelesen werden, um einer Software zu ermöglichen, die Adressen, die den Abbruch verursachten, zu bestimmen.

[0041] Bezug nehmend auf **Fig. 1-5** kann in einer Ausführungsform die mit Bezug auf **Fig. 3** beschriebene TXMBEG-Instruktion eine oder mehrere der folgenden zwei Aktionen abhängig vom Zustand des TXSR-Registers 500 durchführen:

1. Die TXMBEG-Instruktion kann die Ausführung in einen atomaren Ausführungsmodus überleiten, in dem der Instruktion folgende Operationen in einem wiederherstellbaren Modus ausgeführt werden. Alle Aktualisierungen, die von Instruktionen, während sich der Kern 106 in diesem Modus befindet, durchgeführten werden, können zu jedem Zeitpunkt verworfen werden. Um die Wiederherstellbarkeit des Registerzustands zu ermöglichen, können einige der Register 240 durch die Logik 234 gecheckpointet werden. Operationen, die, falls sie ausgeführt werden, nicht annulliert werden können, können in einem Fehler, verworfenen Transaktionsaktualisierungen und/oder einer Übertragung der Kontrolle auf den durch die TXMBEG-Instruktion identifizierten Handler resultieren.

2. Die TXMBEG-Instruktion kann ebenfalls abhängig vom Zustand des TXSR-Registers 500 eine bedingte Kontrollübertragung durchführen. Falls die entsprechenden TXSR-Bits gesetzt sind, kann die TXMBEG-Instruktion die Programmkontrolle zu einem unterschiedlichen Punkt in dem der Transaktio-

nalspeicherzugriffsanfrage entsprechenden Instruktionsstrom übertragen, ohne Rücksprunginformationen aufzuzeichnen. Der Zielpunkt (Ziel)-Operand kann die Adresse, zu der die Instruktion springt, angeben. Falls eine TXMBEG eine bedingte Kontrollübertragung durchführt, kann der Kern 106 nicht in einen wiederherstellbaren Ausführungsmodus übergehen und die Ausführung kann zu der einer JMP (oder Sprung)-Instruktion in Übereinstimmung mit mindestens einer Instruktionssatzarchitektur ähnlich sein.

[0042] In einer Ausführungsform kann der folgende Pseudocode die mit Bezug auf einige Einträge 502 des in **Fig. 5** gezeigten Registers 500 oben besprochenen Operationen darstellen:

```
IF TXSR.exec _state = X_ABORTED
       THEN
              JMP zur Zieloperandenadresse
       ELSE
              IF TXSR.exec _state = X_ACTIVE
                     THEN
                            TXND <- TXND + 1
                            ckptSP = SP
                     ELSE
                            TXND <- 1
                            TXSR.exec _state = X_ACTIVE
                            ckptSP = SP
                            ckptIP = IP (des äußersten Beginne atomar)
              FI
FI
```

[0043] In dem obigen Code kann zusätzlich zum SP-Register eine Vielzahl von Registern (z.B. Register 240) gespeichert werden. In einer Ausführungsform kann der zu speichernde Satz von Registern von der Software zu der Hardware kommuniziert werden. Bezug nehmend auf **Fig. 1-5** kann in einer Ausführungsform die mit Bezug auf **Fig. 3** besprochene TXMEND-Instruktion den folgenden Pseudocode abhängig vom Zustand des TXSR-Registers 500 durchführen (in einer Ausführungsform kann das untere tmpmemstatus ein kontinuierlich durchgeschalteter (continuously gated) Zustand sein, um anzuzeigen, ob sich alle Cache-Blöcke im Cache 116 in einem entsprechenden Zustand befinden):

```
IF TXSR.exec _state != X_ACTIVE
       THEN
              Illegal Opcode ERROR
FI
IF –TXND != 0
       THEN
              Done; (*verschachtelter Block ausgeschieden*)
FI
IF (RTM)
 THEN
       IF (tmpmemstatus)
              THEN
                     TXSR.exec _state = X _COMMITTING
                     commit_cache_rtm_mode()
                     TXSR <- 0
                     Done
              ELSE (*konnte Speicherzustand nicht erledigen*)
                     TXSR.exec_state = X_ABORTING
                     abort_cache_rtm_mode()
                     TXSR.exec _state = X_ABORTED
                     TXSR.attempt_state++ (*um die Anzahl von fehlge-
                                   schlagenen Versuchen zu verfolgen*)
                     JMP to TXIP
                     (*stelle SP usw. wieder her*)
       FI
```

[0044] Bezug nehmend auf **Fig. 1-5** kann in einer Ausführungsform die mit Bezug auf **Fig. 3** besprochene TXMABT-Instruktion einem Benutzer ermöglichen, explizit die atomare Blockausführung abzubrechen. Der Effekt kann einem Abbrechen der Ausführung ähnlich sein und alle Aktualisierungen werden verworfen (bis auf die Bits des Registers 500). In einer Ausführungsform kann die TXMABT-Instruktion den folgenden Pseudocode abhängig vom Zustand des TXSR-Registers 500 durchführen:

```
IF TXSR.exec _state != X_ACTIVE
       THEN
              Illegal Opcode ERROR
FI
discard_speculative_updates_cache()
restore_architectural_register_state() (* SP/IP *)
TXSR.exec _state = X_ABORTED
TXSR.attempt _state++
JMP to TXIP
```

[0045] In einer Ausführungsform zeigt die untere Tabelle 1 beispielhafte Cache-Zustände (z.B. des Cache 116 aus **Fig. 2**), Bit-Werte für die Bits 226 und 228 aus **Fig. 2** und eine entsprechende Beschreibung. In Tabelle 1 zeigt „M“ einen modifizierten Cache-Zustand an, „E“ zeigt einen exklusiven Cache-Zustand and, „S“ zeigt einen Gemeinschafts-Cache-Zustand an, „I“ zeigt einen invaliden Cache-Zustand an, „X“ zeigt einen Do-Not-Care-Zustand an und „N/A“ bedeutet nicht anwendbar.

Tabelle 1 - Beispielhafte Cache-Zustand und Bit-Beschreibung

TXR	***TXw***	***CACHE-ZUSTAND***	***BESCHREIBUNG***
0	1	MESI	N/A (falls TXw gesetzt ist, dann ist TXr ebenfalls gesetzt)
1	X	I	N/A (TX-Bits haben keine Bedeutung, wenn der Cache-Zustand invalide ist)
1	1	ES	N/A (ES-Cache-Zeile kann nicht dirty sein)
0	0	X	keine zusätzliche Bedeutung
1	0	MES	Lesen einer Zeile innerhalb der Transaktion (Falls M, dann ist Zeile im Cache dirty, wurde in Transaktion gelesen, aber nicht in Transaktion beschrieben.)
1	1	M	Zeile wurde innerhalb der Transaktion beschrieben. Früherer Wert wurde auf die nächste Stufe zurückgeschrieben.

[0046] In einer Ausführungsform zeigt die untere Tabelle 2 Anfragen des Prozessorkerns 106, Cache-Zustände (z.B. des Cache 116 aus **Fig. 2**), Bit-Werte für die Bits 226 und 228 aus **Fig. 2** und eine entsprechende Aktion. In Tabelle 2 zeigt „M“ einen modifizierten Cache-Zustand an, „E“ zeigt einen exklusiven Cache-Zustand an, „S“ zeigt einen Gemeinschafts-Cache-Zustand an, „I“ zeigt einen invaliden Cache-Zustand an, „X“ zeigt einen Do-Not-Care-Zustand an, „LD“ bedeutet eine Ladeanfrage, „ST“ bedeutet eine Speicheranfrage, „LDntx“ bedeutet eine explizite nicht-transaktionale Ladeanfrage, „STntx“ bedeutet eine explizite nicht-transaktionale Speicheranfrage, „Done“ bedeutet keine zusätzlichen Operationen gegenüber dem, was bereits in dem Basissystem passieren würde (die existierende Cache-Hit-Kondition mit keiner anderen Aktion), „[TXr=? TXw=?]“ zeigt den Endzustand des Cache-Blocks für dieses Ereignis, „NA“ bedeutet, dass dieser Eintrag nicht auftreten sollte, „FAULT“ bedeutet, dass eine Fehlerkondition markiert sein kann, und „Issue miss“, dass eine Miss-Anfrage ausgegeben wird.

Tabelle 2 - Beispielhafte Anfrage des Kerns, Cache-Zustand, Bit-Wert und Aktion

Anfrage des Kerns	***TXr***	***TXw***	***Cache-Zustand***	***Aktion***
X	0	1	MESI	NA
X	1	X	I	NA
X	1	1	ES	NA

Anfrage des Kerns	*TXr*	*TXw*	*Cache-Zustand*	*Aktion*
LD	0	0	MES	[TXr=1 TXw=0]
LD	1	0	MES	Done
LD	1	1	MES	Done
LD	0	0	I	Issue miss [TXr=1 TXw=0]
ST	0	0	M	WB_first_retain_own() [TXr= 1 TXw=1]
ST	0	0	E	[TXr=1 TXw=1]
ST	0	0	S	Issue miss (ohne Datenübertragung) [TXr=1 TX w= 1]
ST	0	0	I	Issue miss [TXr=1 TXw=1]
ST	1	0	M	WB_first_retain_own() [TXr= 1 TXw=1]
ST	1	0	E	[TXr=1 TXw=1]
ST	1	0	S	Issue miss (ohne Datenübertragung) [TXr=1 TX w= 1]
ST	1	1	M	Done
LDntx	0	0	I	Issue miss [TXr=0 TXw=0]
LDntx	1	X	MES	Done
LDntx	0	0	MES	Done
STntx	0	0	S	Issue miss (ohne Datenübertragung) [TXr=0 TXw=0]
STntx	0	0	I	Issue miss [TXr=0 TXw=0]
STntx	0	0	ME	Done
STntx	1	0	MES	FAULT
STntx	1	1	M	FAULT

[0047] **Fig. 6** stellt ein Blockdiagramm einer Ausführungsform eines Rechensystems 600 dar. Das Rechensystem 600 kann eine oder mehrere Zentraleinheit(en) (CPUs) oder Prozessoren 602 umfassen, die mit einer Zwischenverbindung (oder Bus) 604 kommunizieren. In einer Ausführungsform können die Prozessoren 602 dieselben wie oder ähnlich zu den Prozessoren 102 aus **Fig. 1** sein. Ebenfalls kann die Zwischenverbindung 604 dieselbe wie die oder ähnlich zu den mit Bezug auf **Fig. 1-2** diskutierten Zwischenverbindungen 104 und/oder 112 sein. Die Prozessoren 602 können jeden Typ eines Prozessors umfassen, wie z.B. ein Allzweckprozessor, ein Netzwerkprozessor (z.B. einen Prozessor, der über ein Computernetzwerk kommunizierte Daten verarbeitet) oder anderer Prozessor, umfassend einen Reduced Instruction Set Computer (RISC)-Prozessor oder einen Complex Instruction Set Computer (CISC)-Prozessor. Darüber hinaus können die Prozessoren 602 ein Single- oder Multiple-Core-Design aufweisen, z.B. einen oder mehrere Prozessorkerne (106) umfassen, wie mit Bezug auf **Fig. 1-2** beschrieben. Die Prozessoren 602 mit einem Multiple-Core-Design können unterschiedliche Typen von Prozessorkernen auf demselben integrierten Schaltungs (IC)-Die integrieren. Die Prozessoren 602 mit einem Multiple-Core-Design können ebenfalls als symmetrische oder asymmetrische Multi-Prozessoren implementiert sein.

[0048] Wie in **Fig. 6** gezeigt, kann ein Chipsatz 606 mit der Zwischenverbindung 604 kommunizieren. Der Chipsatz 606 kann einen Speicherkontroll-Hub (MCH) 608 umfassen. Der MCH 608 kann einen Speichercontroller 610 umfassen, der mit dem Speicher 114 kommuniziert. Der Speicher 114 kann Daten speichern, z.B. umfassend Sequenzen von Instruktionen, die durch die Prozessoren 602 oder jede andere mit dem Rechensystem 600 kommunizierende Vorrichtung ausgeführt werden. In einer Ausführungsform der Erfindung kann der Speicher 114 einen oder mehrere flüchtige Storage (oder Speicher)-Vorrichtungen umfassen, wie z.B. Direktzugriffsspeicher (RAM), dynamisches RAM (DRAM), synchrones DRAM (SDRAM), statisches RAM (SRAM) oder andere flüchtige Speichervorrichtungen. Nichtflüchtiger Speicher kann ebenfalls verwendet werden, wie z.B. eine Festplatte. Zusätzliche Vorrichtungen können über die Zwischenverbindung 604 kommunizieren, wie z.B. mehrere Prozessoren und/oder mehrere Systemspeicher.

[0049] Der MCH 608 kann zusätzlich eine mit einem Graphikbeschleuniger 616 kommunizierende Graphikschnittstelle 614 umfassen. Die Graphikschnittstelle 614 kann in einer Ausführungsform mit dem Graphikbeschleuniger 616 über einen Accelerated Graphics Port (AGP) kommunizieren. In einer Ausführungsform der Erfindung kann ein Display (wie z.B. ein Flachbildschirm) mit der Graphikschnittstelle 614 über z.B. einen Signalkonverter kommunizieren, der eine digitale Repräsentation eines in einer Speichervorrichtung, wie z.B. Videospeicher oder Systemspeicher, gespeicherten Bildes in Display-Signale übersetzt, die von dem Display interpretiert und dargestellt werden. In zahlreichen Ausführungsformen kann das von der Display-Vorrichtung erzeugte Display-Signal unterschiedliche Kontrollvorrichtungen durchlaufen, bevor es von dem Display interpretiert und nachfolgend auf dem Display dargestellt wird.

[0050] Des Weiteren kann eine Hub-Schnittstelle 618 die Kommunikation zwischen dem MCH 608 und einem Eingabe/Ausgabe (I/O)-Kontroll-Hub (ICH) 620 ermöglichen. Der ICH 620 kann eine Schnittstelle zu mit dem Rechensystem 600 kommunizierenden I/O-Vorrichtungen bereitstellen. Der ICH 620 kann mit einem Bus 622 durch eine Peripherie-Bridge (oder Controller) 624 kommunizieren, wie z.B. eine Peripheral Component Interconnect (PCI)-Bridge oder ein Universal Serial Bus (USB)-Controller. Die Bridge 624 kann einen Datenweg zwischen dem Prozessor 602 und Peripherievorrichtungen bereitstellen. Andere Topologietypen können benutzt werden. Mehrere Busse können ebenfalls mit dem ICH 620, z.B. durch mehrere Bridges oder Controller, kommunizieren. Darüber hinaus können andere mit dem ICH 620 kommunizierende Peripheriegeräte in verschiedenen Ausführungsformen der Erfindung Integrated Drive Electronics (IDE) oder Small Computer System Interface (SCSI)-Festplatte(n), USB-Port(s), eine Tastatur, eine Maus, parallele(n) Port(s), serielle(n) Port(s), Diskettenlaufwerk(e) oder digitale Daten unterstützende Schnittstellen (z.B. Digital Video Interface (DVI)) umfassen.

[0051] Der Bus 622 kann mit einer Audiovorrichtung 626, einem oder mehreren Plattenlaufwerk(en) 628 und einem Netzwerkadapter 630 kommunizieren. Der Netzwerkadapter 630 kann mit einem Computernetzwerk 631 kommunizieren, um z.B. verschiedenen Komponenten des Systems 600 zu ermöglichen, Daten über das Netzwerk 631 zu senden und/oder zu empfangen. Andere Vorrichtungen können über den Bus 622 kommunizieren. Verschiedene Komponenten (wie z.B. der Netzwerkadapter 630) können ebenfalls mit dem MCH 608 in einigen Ausführungsformen der Erfindung kommunizieren. Zusätzlich können der Prozessor 602 und der MCH 608 kombiniert werden, um einen einzigen Chip zu bilden. Des Weiteren kann der Graphikbeschleuniger 616 innerhalb des MCH 608 in weiteren Ausführungsformen der Erfindung enthalten sein.

[0052] In einer Ausführungsform kann das Rechensystem 600 flüchtigen und/oder nicht-flüchtigen Speicher (oder Storage) umfassen. Nicht-flüchtiger Speicher kann z.B. einen oder mehrere der Folgenden umfassen: Festwertspeicher (ROM), programmierbarer ROM (PROM), löschbarer PROM (EPROM), elektrischer EPROM (EEPROM), ein Plattenlaufwerk (z.B. 628), eine Floppy-Diskette, eine Compact Disk ROM (CD-ROM), eine Digital Versatile Disk (DVD), Flash-Speicher, eine magnetooptische Platte oder andere Typen von nicht-flüchtigen maschinenlesbaren Datenträgern zum Speichern elektronischer Daten (z.B. Instruktionen umfassend).

[0053] **Fig. 7** stellt ein Rechensystem 700 dar, das gemäß einer Ausführungsform der Erfindung in einer Point-to-Point (PtP)-Konfiguration angeordnet ist. **Fig. 7** zeigt insbesondere ein System, in dem Prozessoren, Speicher und Eingabe/Ausgabe-Vorrichtungen miteinander durch eine Vielzahl von Point-to-Point-Schnittstellen verbunden sind. Die mit Bezug auf **Fig. 1-6** besprochenen Operationen können durch eine oder mehrere Komponenten des Systems 700 durchgeführt werden.

[0054] Wie in **Fig. 7** dargestellt, kann das System 700 mehrere Prozessoren umfassen, von denen nur zwei, Prozessoren 702 und 704, um der Klarheit willen gezeigt werden. Die Prozessoren 702 und 704 können jeder einen lokalen Speichercontroller-Hub (MCH) 706 und 708 umfassen, um eine Kommunikation mit Speichern 710 und 712 zu ermöglichen. Die Speicher 710 und/oder 712 können verschiedene Daten, wie z.B. die mit Bezug auf den Speicher 114 aus **Fig. 1**, **Fig. 2** und **Fig. 6** diskutierten, speichern.

[0055] In einer Ausführungsform können die Prozessoren 702 und 704 einer der mit Bezug auf **Fig. 6** besprochenen Prozessoren 602 sein. Die Prozessoren 702 und 704 können Daten über eine Point-to-Point (PtP)-Schnittstelle 714 mittels PtP-Schnittstellenschaltungen 716 bzw. 718 austauschen. Jeder der Prozessoren 702 und 704 kann ebenfalls Daten mit einem Chipsatz 720 über individuelle PtP-Schnittstellen 722 und 724 mittels Point-to-Point-Schnittstellenschaltungen 726, 728, 730 und 732 austauschen. Der Chipsatz 720 kann des Weiteren Daten mit einer Hochleistungsgraphikschaltung 734 über eine Hochleistungsgraphikschnittstelle 736 z.B. mittels einer PtP-Schnittstellenschaltung 737 austauschen.

[0056] Mindestens eine Ausführungsform der Erfindung kann innerhalb der Prozessoren 702 und 704 bereitgestellt werden. Einer oder mehrere der Kerne 106 aus **Fig. 1** oder **Fig. 2** kann sich z.B. innerhalb der Prozessoren 702 und 704 befinden. Andere Ausführungsformen der Erfindung können jedoch in anderen Schaltungen, logischen Einheiten oder Vorrichtungen innerhalb des Systems 700 aus **Fig. 7** existieren. Des Weiteren können andere Ausführungsformen der Erfindung über mehrere Schaltungen, logische Einheiten oder Vorrichtungen, die in **Fig. 7** dargestellt sind, verteilt werden.

[0057] Der Chipsatz 720 kann mit einem Bus 740 mittels einer PtP-Schnittstellenschaltung 741 kommunizieren. Der Bus 740 kann eine oder mehrere Vorrichtungen umfassen, die mit ihm kommunizieren, wie z.B. eine Bus-Bridge 742 und I/O-Vorrichtungen 743. Die Bus-Bridge 743 kann über einen Bus 744 mit anderen Vorrichtungen, wie z.B. eine Tastatus/Maus 745, Kommunikationsvorrichtungen 746 (wie z.B. Modems, Netzwerkschnittstellenvorrichtungen (z.B. der Netzwerkadapter 630 aus **Fig. 6**) oder andere Kommunikationsvorrichtungen, die mit dem Computernetzwerk 631 kommunizieren können), Audio-I/O-Vorrichtungen und/oder eine Datenspeichervorrichtung 748, kommunizieren. Die Datenspeichervorrichtung 748 kann Code 749 speichern, der durch die Prozessoren 702 und/oder 704 ausgeführt werden kann.

[0058] In zahlreichen Ausführungsformen der Erfindung können die hier z.B. mit Bezug auf **Fig. 1-7** besprochenen Operationen als Hardware (z.B. Schaltung), Software, Firmware oder Kombinationen davon implementiert sein, die als ein Computerprogrammprodukt bereitgestellt werden können, z.B. umfassend einen maschinenlesbaren oder computerlesbaren Datenträger mit darauf gespeicherten Instruktionen (oder Softwareprozeduren), die verwendet werden, um einen Computer zum Durchführen eines hier besprochenen Prozesses zu programmieren. Der Ausdruck „Logik“ kann ebenfalls beispielsweise Software, Hardware oder Kombinationen von Software und Hardware umfassen. Der maschinenlesbare Datenträger kann eine Speichervorrichtung, wie z.B. die mit Bezug auf **Fig. 1-7** beschriebene, umfassen. Zusätzlich können solche computerlesbaren Datenträger als ein Computerprogrammprodukt heruntergeladen werden, wobei das Programm von einem entfernten Computer (z.B. einem Server) auf einen anfragenden Computer (z.B. einen Client) mittels Datensignalen, die in einer Trägerwelle verkörpert sind, oder anderen Verbreitungsmedien über eine Kommunikationsverbindung (z.B. einen Bus, ein Modem oder eine Netzwerkverbindung) übertragen werden kann. Entsprechend sollte hier eine Trägerwelle als einen maschinenlesbaren Datenträger umfassend betrachtet werden.

Patentansprüche

1. Vorrichtung, aufweisend:
eine Speichereinheit zum Speichern eines Werts, der einer Anzahl von Transaktionalspeicherzugriffsanfragen entspricht, die unerledigt (uncommitted) sind;
eine erste Logik zum Aktualisieren des in der Speichereinheit gespeicherten Werts für ein Auftreten eines Starts einer Transaktionalspeicherzugriffsanfrage und einer Erledigung der Transaktionalspeicherzugriffsanfrage,
eine zweite Logik zum spekulativen Ausgeben einer Ladeoperation, nachdem eine vorherige der Transaktionalspeicherzugriffsanfragen auf dieselbe Stelle eines Speichers,
die der Ladeoperation entspricht, zugegriffen hat, und
eine dritte Logik zum erneuten Ausgeben der Ladeoperation nach einem Ausscheiden oder Erledigen, falls auf dieselbe Stelle vorher nicht zugegriffen wurde.

2. Vorrichtung nach Anspruch 1, des Weiteren aufweisend eine vierte Logik zum Speichern von Daten, die einem Zustand einer oder mehrerer Komponenten eines Prozessors entsprechen, zumindest teilweise, als Antwort auf eine der Transaktionalspeicherzugriffsanfragen.

3. Vorrichtung nach Anspruch 2, des Weiteren aufweisend eine fünfte Logik zum Wiederherstellen des Zustands der einen oder mehreren Komponenten des Prozessors nachdem die Erledigung mindestens einer von einer oder mehreren Operationen, die einer der Transaktionalspeicherzugriffsanfragen entsprechen, fehlschlägt.

4. Vorrichtung nach Anspruch 2, des Weiteren aufweisend eine fünfte Logik zum Verursachen einer Ausführung von einer oder mehrerer Operationen, die mindestens einer der Transaktionalspeicherzugriffsanfragen entsprechen, nach einer Indikation, dass die eine oder mehreren Operationen atomar erledigt werden kann.

5. Vorrichtung nach Anspruch 4, wobei die fünfte Logik die Ausführung der einen oder mehreren Operationen verursacht, nachdem der in der Speichereinheit gespeicherte Wert anzeigt, dass eine letzte der Transaktionalspeicherzugriffsanfragen erledigt wurde.

6. Vorrichtung nach Anspruch 2, wobei die eine oder mehrere Komponenten des Prozessors ein oder mehrere Register umfasst.

7. Vorrichtung nach Anspruch 1, wobei sich die Transaktionalspeicherzugriffsanfragen auf denselben Thread beziehen.

8. Vorrichtung nach Anspruch 1, des Weiteren aufweisend einen Cache zum Speichern von Daten, wobei der Cache ein oder mehrere Bits zum Anzeigen eines Zugriffs, der einer der Transaktionalspeicherzugriffsanfragen entspricht, auf einen Abschnitt des Cache aufweist.

9. Vorrichtung nach Anspruch 8, wobei der Abschnitt des Cache eines oder mehrere aus einer Gruppe, umfassend eine Cache-Zeile und einen Cache-Block, ist.

10. Vorrichtung nach Anspruch 8, wobei ein oder mehrere Einträge im Cache, die mindestens einer der Transaktionalspeicherzugriffsanfragen entsprechen, nach anderen Einträgen im Cache geräumt werden.

11. Vorrichtung nach Anspruch 1, des Weiteren aufweisend eine vierte Logik zum spekulativen Ausführen mindestens einiger von einer oder mehreren Operationen, die den Transaktionalspeicherzugriffsanfragen entsprechen.

12. Vorrichtung nach Anspruch 1, wobei mindestens einige von einer oder mehreren Operationen, die den Transaktionalspeicherzugriffsanfragen entsprechen, als transaktional oder nicht-transaktional identifiziert werden.

13. Vorrichtung nach Anspruch 1, des Weiteren aufweisend eine vierte Logik zum Abbrechen mindestens einer der Transaktionalspeicherzugriffsanfragen als Antwort auf eines oder mehrere aus einer Gruppe, umfassend:
einen Konflikt mit einer anderen Operation;
ein implementierungsspezifisches Ereignis, das einen Abbruch erzwingt; und
eine Anfrage für einen expliziten Abbruch.

14. Vorrichtung nach Anspruch 1, des Weiteren aufweisend eine Vielzahl von Prozessorkernen, wobei mindestens einer aus der Vielzahl von Prozessorkernen eines oder mehrere aus einer Gruppe, umfassend die erste Logik und die Speichereinheit, umfasst.

15. Vorrichtung nach Anspruch 1, des Weiteren aufweisend eine vierte Logik zum Verursachen einer Ausführung von einem nicht-transaktionalen Speichern, das dem Transaktionalspeicherzugriff entspricht, als Write-Through-Operationen auf einen Speicher.

16. Verfahren, aufweisend:
Aktualisieren eines gespeicherten Werts, der einer Anzahl von Transaktionalspeicherzugriffsanfragen entspricht, die unerledigt sind, als Antwort auf mindestens eine erste Instruktion, die einer Transaktionalspeicheranfrage entspricht; Durchführen einer oder mehrerer Operationen als Antwort auf eine zweite Instruktion, die der Transaktionalspeicheranfrage entspricht,
spekulatives Ausgeben einer Ladeinstruktion, die der Transaktionalspeicheranfrage entspricht, falls ein Bit, das einem Abschnitt eines Cache entspricht, einen vorherigen transaktionalen Zugriff auf denselben Abschnitt des Cache anzeigt, und
erneutes Ausgeben der Ladeoperation nach einem Ausscheiden oder Erledigen, falls auf denselben Abschnitt vorher nicht zugegriffen wurde.

17. Verfahren nach Anspruch 16, des Weiteren aufweisend Aktualisieren des gespeicherten Werts als Antwort auf die zweite Instruktion.

18. Verfahren nach Anspruch 16, des Weiteren aufweisend ein Setzen von Kontrollpunkten in einer oder mehren Komponenten eines Prozessors als Antwort auf die erste Instruktion.

19. Verfahren nach Anspruch 16, des Weiteren aufweisend Identifizieren einer oder mehrerer Instruktionen, die der Transaktionalspeicheranfrage entsprechen, als transaktional oder nicht-transaktional.

20. Verfahren nach Anspruch 19, wobei das Identifzieren der einen oder mehreren Instruktionen implizit oder explizit durchgeführt wird.

21. System, aufweisend:
einen Speicher zum Speichern von Daten;
einen Prozessor, aufweisend:
eine erste Logik zum Abrufen einer ersten Instruktion aus dem Speicher, die einem Start eines Transaktionalspeicherzugriffs entspricht, und einer zweiten Instruktion aus dem Speicher, die einem Ende des Transaktionalspeicherzugriffs entspricht; eine zweite Logik zum Aktualisieren eines in einer Speichereinheit gespeicherten Werts als Antwort auf eine oder mehrere der ersten Instruktion und der zweiten Instruktion,
eine dritte Logik zum spekulativen Ausgeben einer Ladeoperation, nachdem eine vorherige der Transaktionalspeicherzugriffsanfragen auf dieselbe Stelle eines Speichers,
die der Ladeoperation entspricht, zugegriffen hat, und
eine vierte Logik zum erneuten Ausgeben der Ladeoperation nach einem Ausscheiden oder Erledigen, falls auf dieselbe Stelle vorher nicht zugegriffen wurde.

22. System nach Anspruch 21, des Weiteren aufweisend eine fünfte Logik zum Verursachen einer Ausführung von einer oder mehrerer Operationen auf dem Speicher nach einer Indikation, dass die eine oder mehreren Speicheroperationen atomar erledigt werden kann.

23. System nach Anspruch 21, des Weiteren aufweisend eine fünfte Logik zum Wiederherstellen eines Zustands einer oder mehrerer Komponenten des Prozessors nachdem die Erledigung mindestens einer von einer oder mehreren Operationen, die dem Transaktionalspeicherzugriff entsprechen, fehlschlägt.

24. System nach Anspruch 21, wobei der in der Speichereinheit gespeicherte Wert einer Anzahl von Transaktionalspeicherzugriffsanfragen entspricht, die unerledigt sind.

25. System nach Anspruch 21, wobei die zweite Logik den in der Speichereinheit gespeicherten Wert als Antwort auf eine Ausführung oder ein Ausgeben der ersten oder zweiten Instruktion aktualisiert.

26. System nach Anspruch 21, des Weiteren aufweisend einen Cache zum Speichern mindestens einiger der in dem Speicher gespeicherten Daten, wobei der Cache ein oder mehrere Bits zum Anzeigen eines Zugriffs, der dem Transaktionalspeicherzugriff entspricht, auf einen Abschnitt des Cache aufweist.

27. System nach Anspruch 26, des Weiteren aufweisend eine fünfte Logik zum Aktualisieren von in dem Cache gespeicherten Daten als Antwort auf eine oder mehrere Operationen, die dem Transaktionalspeicherzugriff entsprechen.

28. System nach Anspruch 21, des Weiteren umfassend eine Audiovorrichtung.

Es folgen 7 Seiten Zeichnungen

Anhängende Zeichnungen

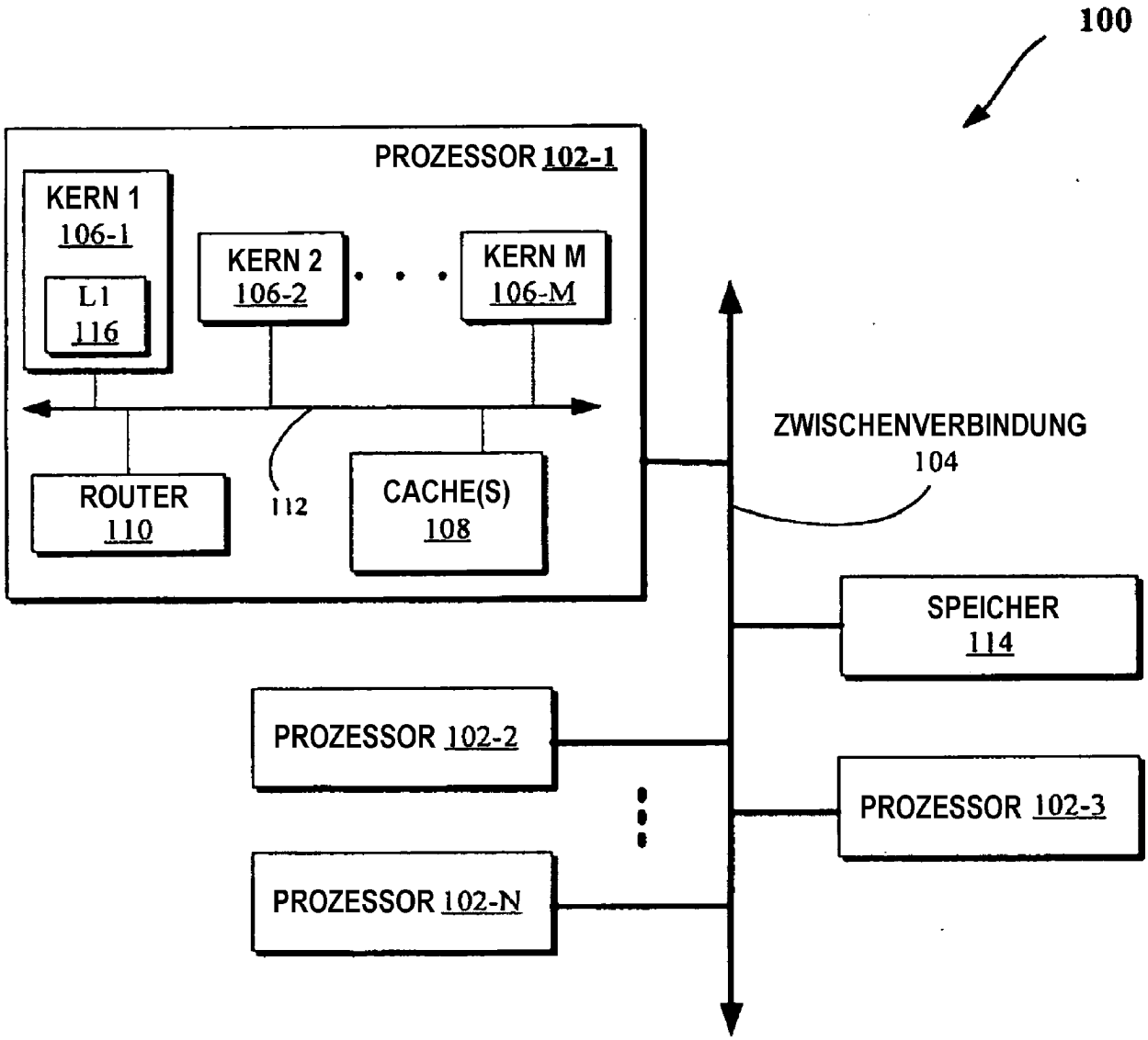


FIG. 1

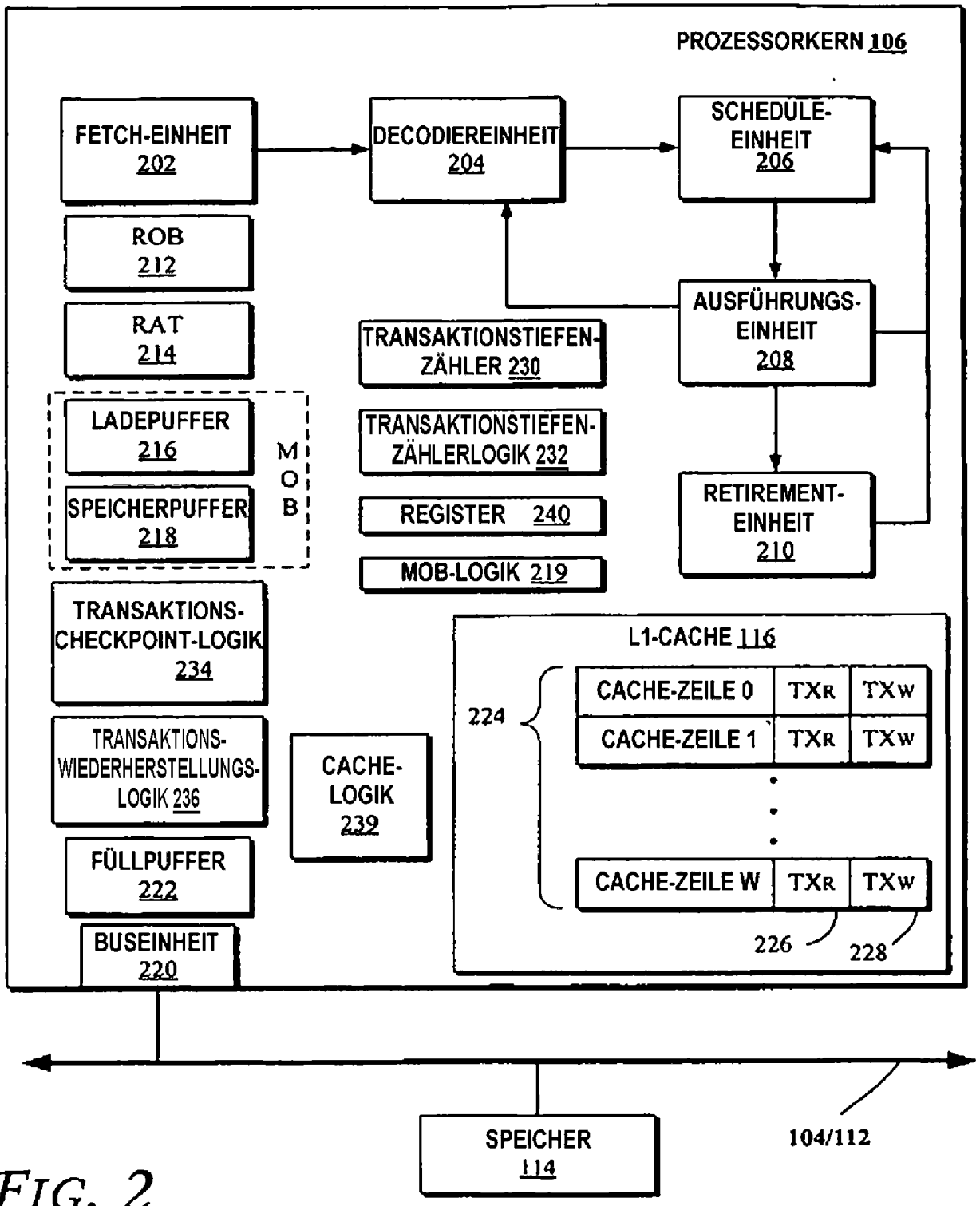

PROZESSORKERN 106
FETCH-EINHEIT 202
DECODIEREINHEIT 204
SCHEDULE-EINHEIT 206
ROB 212
RAT 214
AUSFÜHRUNGS-EINHEIT 208
TRANSAKTIONSTIEFEN-ZÄHLER 230
LADEPUFFER 216
SPEICHERPUFFER 218
MOB
TRANSAKTIONSTIEFEN-ZÄHLERLOGIK 232
REGISTER 240
MOB-LOGIK 219
RETIREMENT-EINHEIT 210
TRANSAKTIONS-CHECKPOINT-LOGIK 234
L1-CACHE 116
224
CACHE-ZEILE 0
TXR
TXW
CACHE-ZEILE 1
CACHE-ZEILE W
TRANSAKTIONS-WIEDERHERSTELLUNGS-LOGIK 236
CACHE-LOGIK 239
FÜLLPUFFER 222
BUSEINHEIT 220
226
228
SPEICHER 114
104/112

FIG. 2

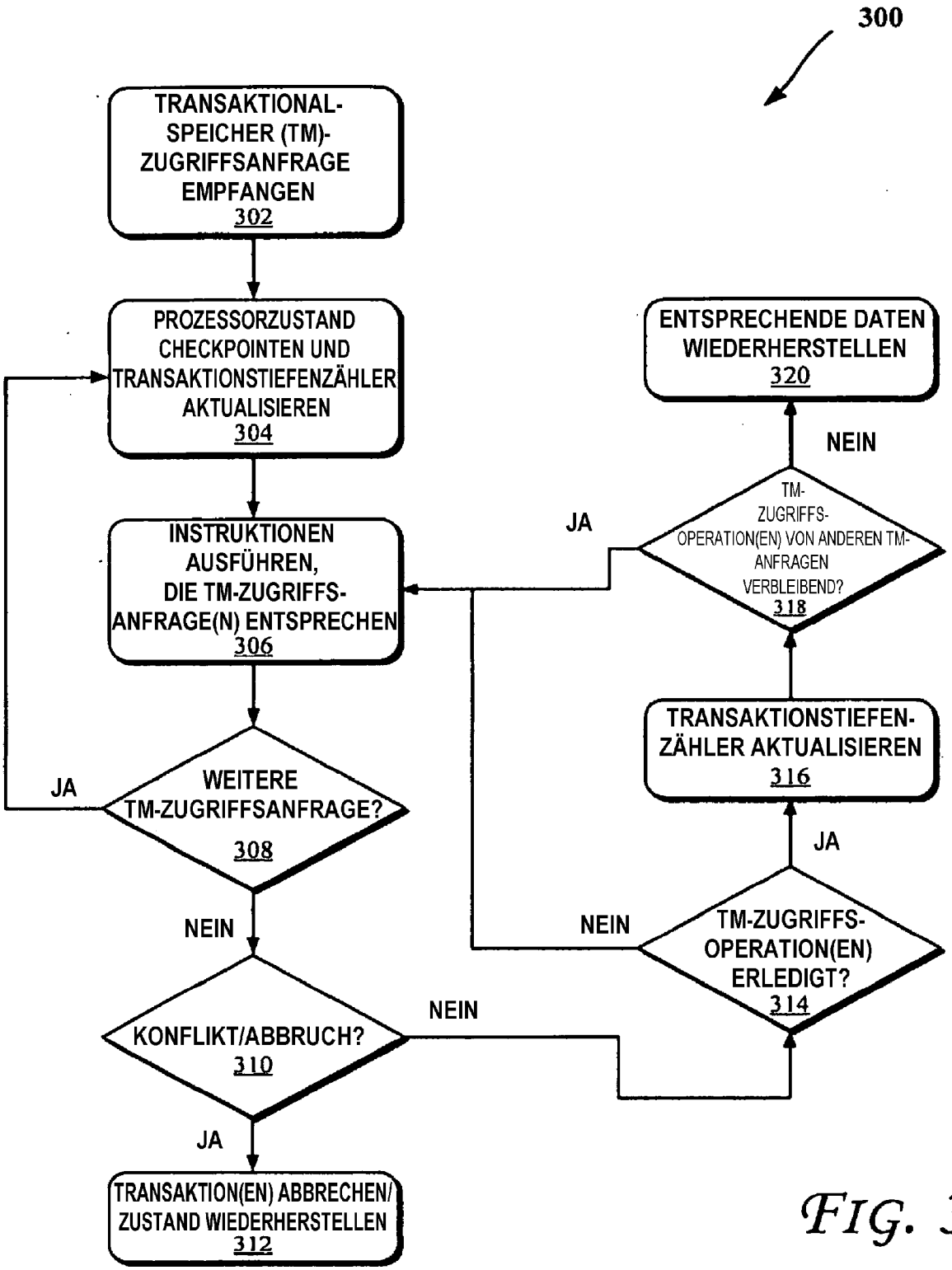

300
TRANSAKTIONAL-SPEICHER (TM)-ZUGRIFFSANFRAGE EMPFANGEN 302
PROZESSORZUSTAND CHECKPOINTEN UND TRANSAKTIONSTIEFENZÄHLER AKTUALISIEREN 304
INSTRUKTIONEN AUSFÜHREN, DIE TM-ZUGRIFFS-ANFRAGE(N) ENTSPRECHEN 306
WEITERE TM-ZUGRIFFSANFRAGE? 308
JA
NEIN
KONFLIKT/ABBRUCH? 310
JA
TRANSAKTION(EN) ABBRECHEN/ ZUSTAND WIEDERHERSTELLEN 312
NEIN
TM-ZUGRIFFS-OPERATION(EN) ERLEDIGT? 314
NEIN
JA
TRANSAKTIONSTIEFEN-ZÄHLER AKTUALISIEREN 316
TM-ZUGRIFFS-OPERATION(EN) VON ANDEREN TM-ANFRAGEN VERBLEIBEND? 318
JA
NEIN
ENTSPRECHENDE DATEN WIEDERHERSTELLEN 320

FIG. 3

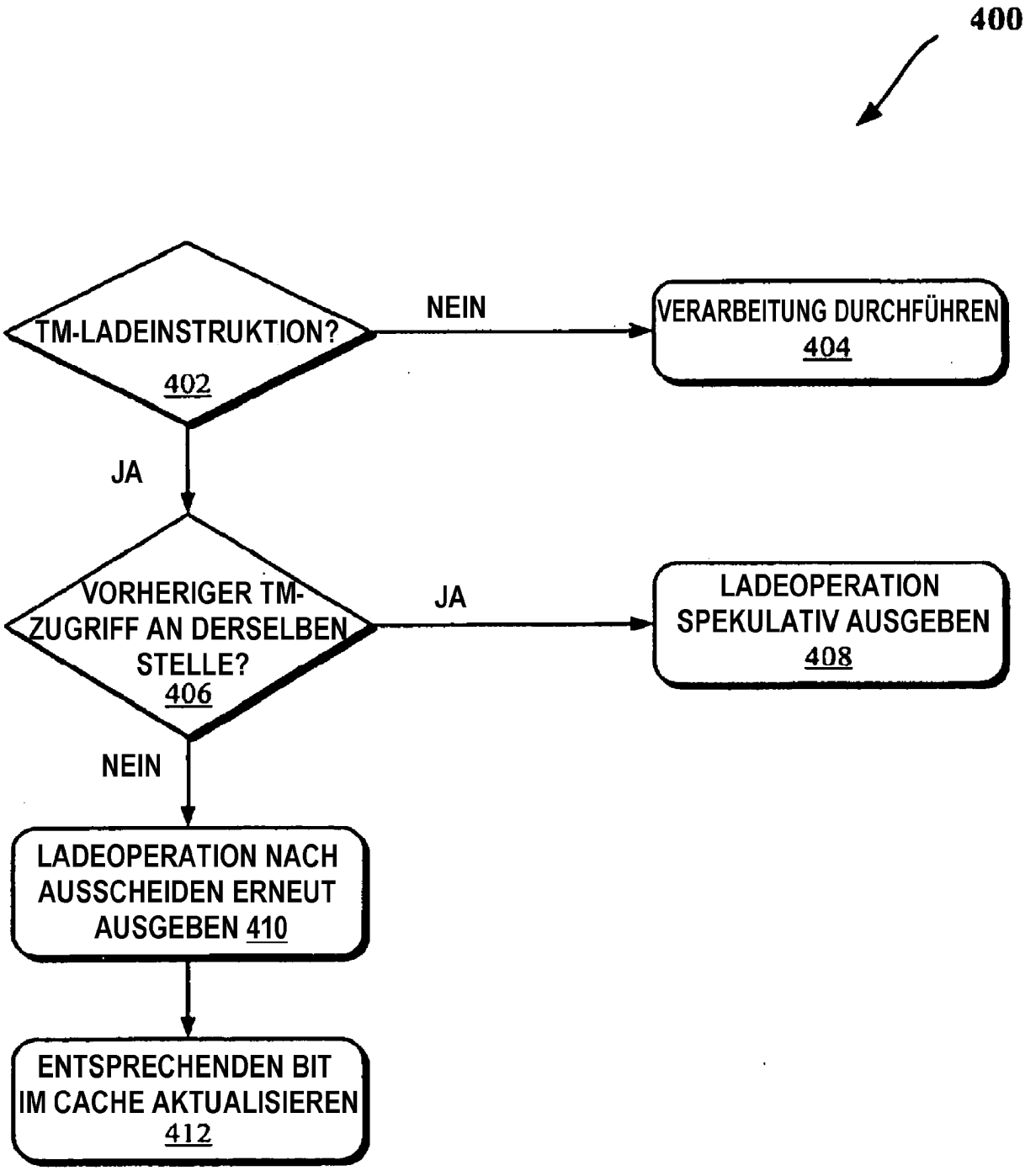
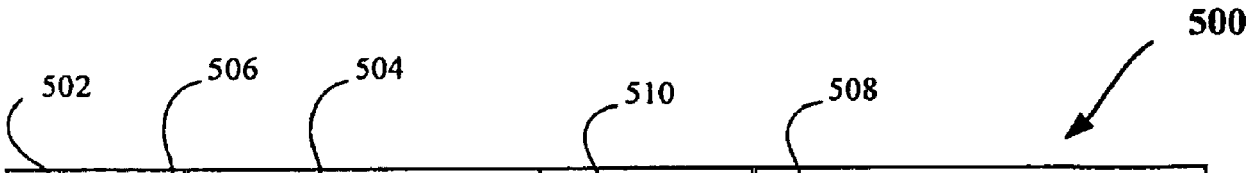

400
TM-LADEINSTRUKTION?
402
NEIN
VERARBEITUNG DURCHFÜHREN
404
JA
VORHERIGER TM-ZUGRIFF AN DERSELBEN STELLE?
406
JA
LADEOPERATION SPEKULATIV AUSGEBEN
408
NEIN
LADEOPERATION NACH AUSSCHEIDEN ERNEUT AUSGEBEN 410
ENTSPRECHENDEN BIT IM CACHE AKTUALISIEREN
412

FIG. 4



#	KLASSE	NAME	AKTION	BESCHREIBUNG
1	AUSFÜHRUNGS-ZUSTAND (EXEC_ STATE)	X_IDLE		KEINE TRANSAKTIONALE AKTIVITÄT
2		X_ACTIVE		IN ATOMAREM BLOCK
3		X_ABORTED		ATOMARER BLOCK ABGEBROCHEN
4		X_COMMITTING		ATOMARER BLOCK WIRD ERLEDIGT
5		X_ABORTING		ATOMARER BLOCK WIRD ABGEBROCHEN
6	VERSUCHS-ZUSTAND (ATTEMPT_ STATE)	A_IDLE (00)		KEINE ABBRÜCHE
7		A_ABORT_FIRST (01)		DER ERSTE ABBRUCH
8		A_ABORT_MULTIPLE (1X)		NACHFOLGENDE ABBRÜCHE
9	QUELLE HANDLER-AUFRUF (SOURCE)	S_ABORT_INST_RETIRED		EXPLIZITER ABBRUCH DURCH PROGRAMMIERER
10		S_DISALLOWED_INSTTYPE	TXFR = IP	- SYSCALL, ANDERE ANWEISUNGEN, DIE AUSFÜHRUNGSBERECHTIGUNGSSTUFE ÄNDERN KÖNNEN - E/A-INSTRUKTIONEN
11		S_RESOURCE_SET_CONFLICT	TXFR = IP TXDR = ADRESSE	- BESCHRÄNKTER PLATZ IM CACHE - ERNEUTER VERSUCH KANN FEHLSCHLAGEN
12		S_DISALLOWED_MEMTYPE	TXFR = IP TXDR = ADRESSE	- NUR WB-SPEICHERTYP IST ZULÄSSIG - ERNEUTER VERSUCH SCHLÄGT WAHRSCHEINLICH FEHL
13		S_EXCEPTION	TXFR = FEHLER/ ABBRUCH/TRAP IP	KANN GESETZT WERDEN, UM EINEN TRANS-AKTIONSABBRUCH AUFGRUND VON AUSFÜHRUNGSAUSNAHME, FEHLER ODER TRAP ANZUZEIGEN
14		S_INTERRUPT		KANN GESETZT WERDEN, UM EINEN TRANS-AKTIONSABBRUCH AUFGRUND VON EXTERNER UNTERBRECHUNG ANZUZEIGEN
15		S_DATA_CONFLICT_CLEAN	TXDR = ADRESSE	- ERNEUTER VERSUCH ZEIGT MÖGLICHERWEISE KEINEN KONFLIKT - HARDWAREPRIORITÄTEN KÖNNEN DIESE EREIGNISSE BESEITIGEN
16		S_DATA_CONFLICT_DIRTY	TXDR = ADRESSE	- ERNEUTER VERSUCH ZEIGT MÖGLICHERWEISE KEINEN KONFLIKT - HARDWAREPRIORITÄTEN KÖNNEN DIESE EREIGNISSE BESEITIGEN
17		X_OVF__COMMITING		- ZEIGT AN, DASS EIN ATOMARER ÜBERLAUF-BLOCK ERLEDIGT WIRD - KANN DURCH MICROCODE-UNTERSTÜTZUNG BEI EINEM SNOOP-TREFFER IN M XSW GELADEN WERDEN
18		X_OVF_ABORTING		- ZEIGT AN, DASS EIN ATOMARER ÜBERLAUF-BLOCK ABGEBROCHEN WIRD - KANN DURCH MICROCODE-UNTERSTÜTZUNG BEI EINEM SNOOP-TREFFER IN M XSW GELADEN WERDEN
19	STEUERUNG	C_RTM_MODE		WIRD IM RTM-MODUS AUSGEFÜHRT
20		C_NON_GATED_OPS		WENN GESETZT, WERDEN LADE- UND SPEICHERUNGE-OPERATIONEN ALS NICHT DURCHGESCHALTETE LADE- UND SPEICHERUNGEOPERATIONEN AUSGEFÜHRT (Z B. ÄHNLICH DEM ANHALTEN EINER TRANS-AKTION, SOBALD IN DIESEN MODUS GEWECHSELT WIRD, EINE ANDERE TXMBEG VOR EINER WIEDER-HERSTELLUNG KANN EINEN FEHLER VERURSACHEN

FIG. 5

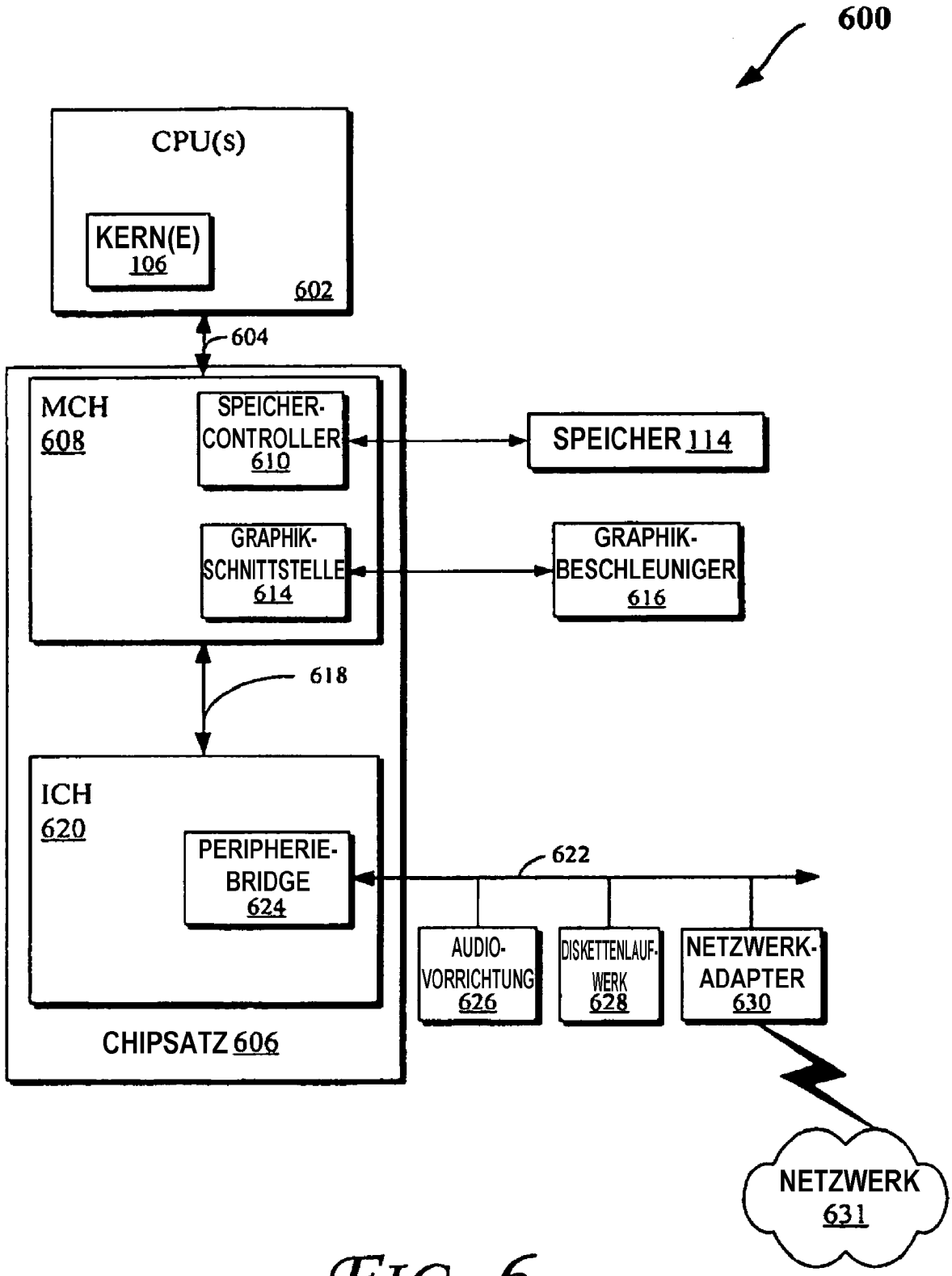

600
CPU(s)
KERN(E)
106
602
604
MCH
608
SPEICHER-
CONTROLLER
610
SPEICHER 114
GRAPHIK-
SCHNITTSTELLE
614
GRAPHIK-
BESCHLEUNIGER
616
618
ICH
620
PERIPHERIE-
BRIDGE
624
622
AUDIO-
VORRICHTUNG
626
DISKETTENLAUF-
WERK
628
NETZWERK-
ADAPTER
630
CHIPSATZ 606
NETZWERK
631

FIG. 6

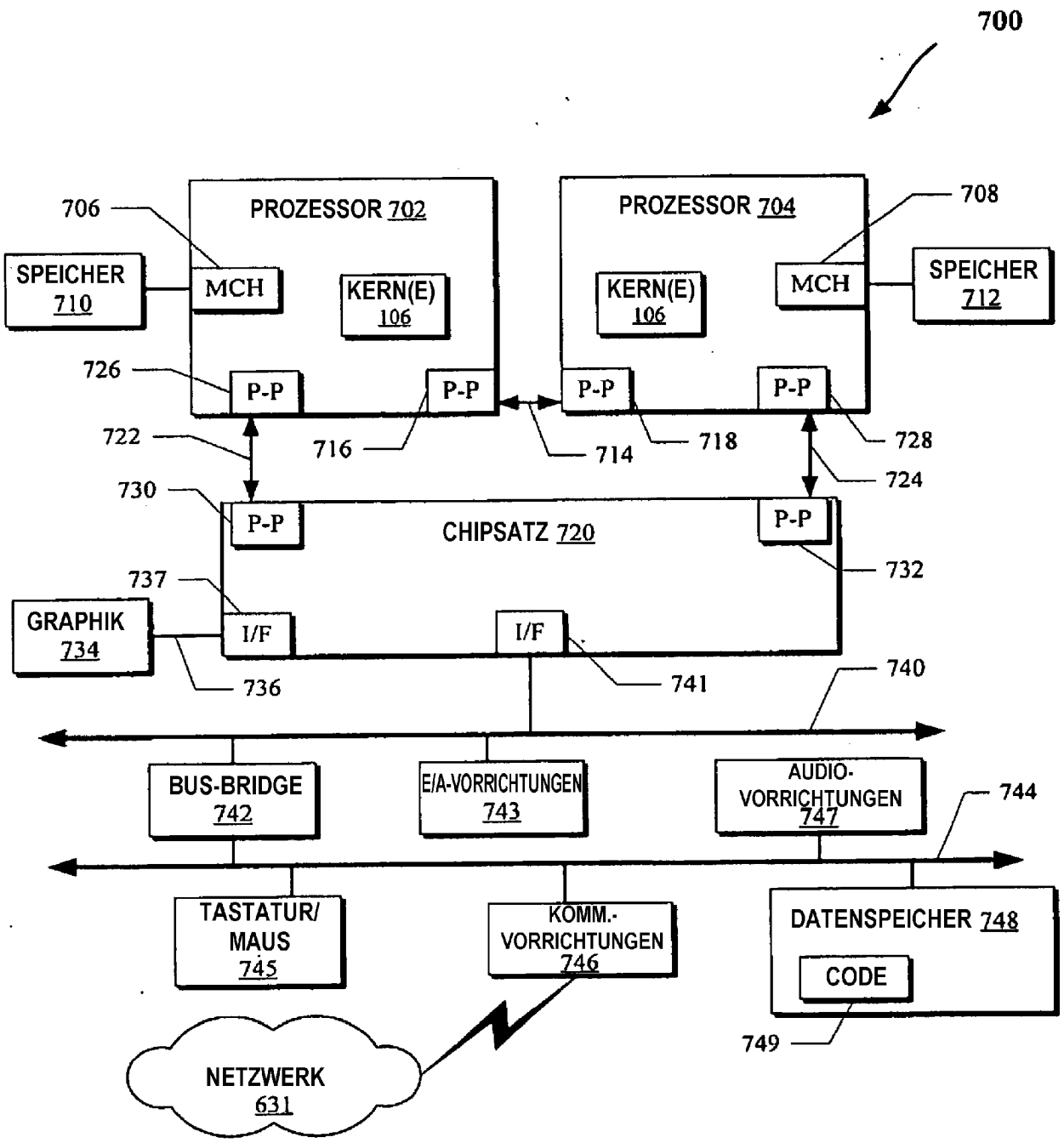


FIG. 7

 Das Dokument wurde durch die Firma Luminess hergestellt.