(19) **Bundesrepublik Deutschland**
Deutsches Patent- und Markenamt

(10) **DE 600 05 860 T2** 2004.08.05

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 244 962 B1**
(21) Deutsches Aktenzeichen: **600 05 860.3**
(86) PCT-Aktenzeichen: **PCT/US00/22458**
(96) Europäisches Aktenzeichen: **00 964 913.8**
(87) PCT-Veröffentlichungs-Nr.: **WO 01/050253**
(86) PCT-Anmeldetag: **16.08.2000**
(87) Veröffentlichungstag der PCT-Anmeldung: **12.07.2001**
(97) Erstveröffentlichung durch das EPA: **02.10.2002**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **08.10.2003**
(47) Veröffentlichungstag im Patentblatt: **05.08.2004**

(51) Int Cl.[7]: **G06F 9/38**

(30) Unionspriorität:

476322	**03.01.2000**	**US**
476570	**03.01.2000**	**US**
476578	**03.01.2000**	**US**
476204	**03.01.2000**	**US**

(73) Patentinhaber:
Advanced Micro Devices, Inc., Sunnyvale, Calif., US

(74) Vertreter:
Patentanwälte von Kreisler, Selting, Werner et col., 50667 Köln

(84) Benannte Vertragsstaaten:
DE, GB

(72) Erfinder:
KELLER, B., James, Palo Alto, US; HADDAD, W., Ramsey, Cupertino, US; MEIER, G., Stephan, Sunnyvale, US

(54) Bezeichnung: **ABLAUFSTEUERUNG ZUM AUSGEBEN UND WIEDERAUSGEBEN VON KETTEN ABHÄNGIGER BEFEHLE**

Beschreibung

Hintergrund der Erfindung

1. Technisches Gebiet

[0001] Diese Erfindung betrifft das Gebiet von Prozessoren und insbesondere Mechanismen zur Ablaufplanung von Befehlen innerhalb Prozessoren.

2. Stand der Technik

[0002] Superskalare Prozessoren versuchen eine hohe Leistungsfähigkeit zu erreichen, indem sie mehrere Befehle pro Taktzyklus ausgeben und ausführen und indem sie die höchst mögliche Taktfrequenz verwenden, die mit dem Design vereinbar ist. Ein Verfahren zum Steigern der Anzahl der pro Taktzyklus ausgeführten Befehle ist die unregelmäßige Ausführung. Bei der unregelmäßigen Ausführung können Befehle in einer anderen Reihenfolge ausgeführt werden als in der Programmsequenz (oder "Reihenfolge des Programms") angegeben ist. Gewisse Befehle, die in einer Programmsequenz nahe bei einander sind, können Abhängigkeiten haben, welche ihre gleichzeitige Ausführung verhindern, während nachfolgende Befehle in der Programmsequenz keine Abhängigkeiten von den vorherigen Befehlen haben müssen. Entsprechend kann eine außer der Reihe Ausführung die Leistungsfähigkeit des superskalaren Prozessors durch eine Steigerung der Anzahl der gleichzeitig ausgeführten Befehle (im Mittel) erhöhen. Ein weiteres Verfahren, das sich auf die ungeordnete Ausführung bezieht, ist die spekulative Ausführung, bei der Befehle ausgeführt werden, die auf andere Befehle folgen, welche die Ausführung des Programms veranlassen können, einem anderen Pfad zu folgen als dem Pfad, der die spekulativen Befehle enthält. Zum Beispiel können Befehle spekulativ sein, falls die Befehle auf einen bestimmten Befehl folgen, der eine Ausnahme verursachen könnte. Befehle sind auch spekulativ, wenn die Befehle einem vorher gesagten, bedingten Verzweigungsbefehl folgen, der noch nicht ausgeführt worden ist. Auf ähnliche Weise können Befehle ungeordnet oder spekulativ im Ablauf geplant, ausgegeben usw. werden.
[0003] Bedauerlicherweise führt die Ablaufplanung von Befehlen für die ungeordnete oder spekulative Ausführung zu zusätzlicher Komplexität bei der Hardware in dem Prozessor. Der Ausdruck „Ablaufplanung" bezieht sich im Allgemeinen auf die Auswahl eines Befehls für die Ausführung. Typischerweise versucht der Prozessor Befehle so schnell wie möglich für den Ablauf zu planen, um die durchschnittliche Rate der Befehlsausführung zu maximieren (zum Beispiel durch Ausführen von Befehlen außer der Reihe, um Abhängigkeiten und die Verfügbarkeit von Hardware für verschiedene Typen von Befehlen zu behandeln). Diese Komplexitäten können die Taktfrequenz begrenzen, bei denen der Prozessor arbeiten kann. Insbesondere müssen die Abhängigkeiten zwischen den Befehlen von der Hardware der Ablaufplanung berücksichtigt werden. Im Allgemeinen bezieht sich, wie hier verwendet, der Ausdruck „Abhängigkeit" auf eine Beziehung eines ersten Befehls und eines nachfolgenden zweiten Befehls in der Reihenfolge des Programms, welche die Ausführung des ersten Befehls vor der Ausführung des zweiten Befehls erfordert. Eine Vielzahl von Abhängigkeiten kann definiert werden. Zum Beispiel tritt eine Abhängigkeit vom Quelloperanden auf, wenn ein Quelloperand des zweiten Befehls ein Zieloperand des ersten Befehls ist.
[0004] Im Allgemeinen haben Befehle einen oder mehrere Quelloperanden und einen oder mehrere Zieloperanden. Die Quelloperanden sind Eingangswerte, die in Übereinstimmung mit der Definition des Befehls zu manipulieren sind, um ein oder mehrere Ergebnisse zu erzeugen (welche die Zieloperanden sind). Quellund Zieloperanden können Speicheroperanden sein, die in einer Speicherstelle außerhalb des Prozessors gespeichert werden, oder können Registeroperanden sein, die in in dem Prozessor enthaltenen Registerspeicherstellen gespeichert werden. Die von dem Prozessor verwendete Befehlssatzarchitektur definiert eine Anzahl von architekturisierten Registern. Diese Register sind von der Befehlssatzarchitektur zu existieren definiert und Befehle können kodiert werden unter Verwendung der architekturisierten Register als Quell- und Zieloperanden. Ein Befehl gibt ein bestimmtes Register als einen Quell- oder Zieloperanden über eine Registernummer (oder Registeradresse) in einem Operandenfeld eines Befehls an. Die Registernummer identifiziert das ausgewählte Register einzigartig unter den architekturisierten Registern. Ein Quelloperand wird von einer Quellregisternummer identifiziert und ein Zieloperand wird von einer Zielregisternummer identifiziert.
[0005] Zusätzlich zu den Abhängigkeiten der Operanden, können von dem Prozessor ein oder mehrere Typen von Ordnungsabhängigkeiten durchgesetzt werden. Ordnungsabhängigkeiten können zum Beispiel verwendet werden, um die verwendete Hardware zu vereinfachen oder um eine korrekte Ausführung des Programms zu erzeugen. Durch das Zwingen von gewissen Befehlen, in einer Reihenfolge im Hinblick auf gewisse andere Befehle ausgeführt zu werden, kann die Hardware zum Behandeln von Konsequenzen der außer der Reihe Ausführung der Befehle unterdrückt werden. Zum Beispiel können Befehle, die speziellen Register, welche einen allgemeinen Betriebszustand des Prozessors enthalten, aktualisieren, die Ausführung einer Vielzahl von nachfolgenden Befehlen beeinflussen, die nicht explizit auf die speziellen Register zugreifen. Im Allgemei-

nen können Ordnungsabhängigkeiten von Mikroarchitektur zu Mikroarchitektur variieren.

[0006] Während der Mechanismus für die Ablaufplanung Abhängigkeiten respektiert, ist es wünschenswert, bei der ungeordneten und/oder spekulativen Ablaufplanung so aggressiv wie möglich zu sein, in einem Versuch, den realisierten Gewinn der Leistungsfähigkeit zu maximieren. Jedoch wird, je aggressiver der Mechanismus für die Ablaufplanung ist (das heißt je weniger Bedingungen, welche einen bestimmten Befehl daran hindern, für den Ablauf geplant zu werden), desto wahrscheinlicher wird das Auftreten eines nicht korrekt ausgeführten Befehls. Der Mechanismus zur Wiederherstellung von nicht korrekt ausgeführten Befehlen war im Allgemeinen, den nicht korrekt ausgeführten Befehl und alle nachfolgenden Befehle aus der Pipeline des Prozessors zu entfernen und den nicht korrekt ausgeführten Befehl (und nachfolgende Befehle) erneut abzurufen. Oft wird die Entfernung und das erneute Abrufen von der Entdeckung einer nicht korrekten Ausführung aus Gründen der Einfachheit der Hardware verzögert (zum Beispiel bis der nicht korrekt ausgeführte Befehl der älteste Befehl in der Bearbeitung ist). Die durchschnittliche Anzahl von tatsächlich pro Taktzyklus ausgeführten Befehlen sinkt wegen der nicht korrekten Ausführung und den nachfolgenden Ereignissen zur Entfernung. Für aggressive Mechanismen für die Ablaufplanung, welche häufiger auf nicht korrekte Ausführung treffen, kann die Verschlechterung der Leistungsfähigkeit, welche diesen Mechanismen zur Wiederherstellung zuzurechnen ist, erheblich sein. Entsprechend ist ein Mechanismus zur Wiederherstellung von einer nicht korrekten, spekulativen Ausführung gewünscht, der die Gewinne der Leistungsfähigkeit, die durch eine aggressive spekulative oder ungeordnete Ablaufplanung möglich gemacht werden, sichert.

[0007] Das US Patent mit der Nummer 5,987,594 beschreibt einen Prozessor, der kodierte Befehle unter Verwendung einer Ablaufplanungseinheit ausführt, welche kodierte Befehle empfängt und sie für die Ausführung ausgibt. Der Prozessor gibt Speicheroperationen, welche in einem Cachespeicher nicht treffen, wenn Daten an den Cachespeicher zurück gegeben werden, und Befehle, die von den genannten Speicheroperationen abhängen, erneut aus. Des weiteren kann der Prozessor eine ungeordnete Ausführung von Speicheroperationen durchführen durch Detektierung von „Lesen nach Schreiben" Gefahren und Weiterleitung der Schreibdaten an den Lese Befehl.

Offenbarung der Erfindung

[0008] Die oben ausgeführten Probleme werden zu einem großen Teil von einem Ablaufplaner, wie hier beschrieben, gelöst. Der Ablaufplaner gibt Befehlsoperationen für die Ausführung aus, hält aber auch die Befehlsoperationen zurück. Falls eine bestimmte Befehlsoperation nachfolgend als nicht korrekt ausgeführt befunden wird, kann die bestimmte Befehlsoperation von dem Ablaufplaner erneut ausgegeben werden. Vorteilhafterweise kann die Strafe für das nicht korrekte Planen des Ablaufs von Befehlsoperationen im Vergleich zum Entfernen der bestimmten Befehlsoperation und jüngeren Befehlsoperationen aus der Pipeline und zum erneuten Abrufen der bestimmten Befehlsoperation reduziert werden. Die Leistungsfähigkeit kann wegen der reduzierten Strafe für eine nicht korrekte Ausführung reduziert werden. Des weiteren kann der Ablaufplaner einen aggressiveren Mechanismus für die Ablaufplanung verwenden, da die Strafe für eine nicht korrekte Ausführung verringert ist.

[0009] Des weiteren kann der Ablaufplaner die Angaben für Abhängigkeiten für jede Befehlsoperation, die ausgegeben worden ist, beibehalten. Falls die bestimmte Befehlsoperation erneut ausgegeben wird, können die Befehlsoperationen, die von der bestimmten Befehlsoperation abhängig sind (direkt oder indirekt) über die Angaben zur Abhängigkeit identifiziert werden. Der Ablaufplaner gibt auch die abhängigen Befehlsoperationen erneut aus. Befehlsoperationen, die in der Reihenfolge des Programms einer bestimmten Befehlsoperation nachfolgen, aber die nicht von der bestimmten Befehlsoperation abhängig sind, werden nicht erneut ausgegeben. Entsprechend kann die Strafe für die nicht korrekte Ablaufplanung von Befehlsoperationen im Hinblick auf das Entfernen der bestimmten Befehlsoperation und aller jüngeren Befehlsoperationen und das erneute Abrufen der bestimmten Befehlsoperation weiter reduziert werden. Die Leistungsfähigkeit kann somit weiter gesteigert werden.

[0010] Allgemein betrachtet, wird ein Ablaufplaner betrachtet. Der Ablaufplaner weist einen Befehlspuffer zum Speichern einer ersten Befehlsoperation, eine mit dem Befehlspuffer verbundene Ausgabeauswahlschaltung und eine Steuerschaltung auf. Die Ausgabeauswahlschaltung ist zum Auswählen einer ersten Befehlsoperation zur Ausgabe aus dem Befehlspuffer konfiguriert. An die Ausgabeauswahlschaltung angeschlossen ist die Steuerschaltung konfiguriert, um einen ersten Ausführungszustand der ersten Befehlsoperation zu halten. Die Steuerschaltung ist konfiguriert, um den ersten Ausführungszustand in einen Ausführungszustand zu ändern in Reaktion auf die Ausgabeauswahlschaltung, welche die erste Befehlsoperation für die Ausgabe auswählt. Des weiteren ist die Steuerschaltung konfiguriert, um den ersten Ausführungszustand in einen nicht ausgeführten Zustand in Reaktion auf ein erstes Signal zu ändern, das anzeigt, dass die erste Befehlsoperation nicht korrekt ausgeführt ist.

[0011] Des weiteren wird ein Prozessor betrachtet, der einen Ablaufplaner und eine Ausführungseinheit aufweist. Der Ablaufplaner ist konfiguriert zum Speichern einer ersten Befehlsoperation und zum Ausgeben der

Befehlsoperation für die Ausführung. Der Ablaufplaner ist konfiguriert zum Halten eines ersten Ausführungszustandes entsprechend der ersten Befehlsoperation und ist konfiguriert zum Ändern des ersten Ausführungszustandes in einen Ausführungszustand in Reaktion auf das Ausgeben der ersten Befehlsoperation. An den Ablaufplaner angeschlossen zum Empfangen der ersten Befehlsoperation in Reaktion auf eine Ausgabe davon durch den Ablaufplaner, ist die Ausführungseinheit konfiguriert, die erste Befehlsoperation auszuführen. Die Steuerschaltung ist konfiguriert, um den ersten Ausführungszustand in einen nicht ausgeführten Zustand zu ändern in Reaktion auf ein erstes Signal, das anzeigt, dass die erste Befehlsoperation nicht korrekt ausgeführt ist. Darüber hinaus wird ein Computersystem betrachtet, das einen Prozessor und ein Eingangs/Ausgangs (I/O) Gerät, das konfiguriert ist, zwischen dem Computersystem und weiteren Computersystemen, an die das I/O Gerät angeschlossen werden kann, zu kommunizieren, umfasst.

[0012] Des weiteren wird ein Verfahren betrachtet. Eine erste Befehlsoperation wird von einem Ablaufplaner an eine Ausführungseinheit ausgegeben. Die erste Befehlsoperation wird nach der Ausgabe in dem Ablaufplaner zurück gehalten. Ein erstes Signal wird empfangen, dass die erste Befehlsoperation nicht korrekt ausführt. Die erste Befehlsoperation wird in Reaktion auf den Empfang des ersten Signals erneut ausgegeben.

[0013] Darüber hinaus wird ein Prozessor betrachtet. Der Prozessor weist einen Ablaufplaner und eine Ausführungseinheit auf. Der Ablaufplaner ist konfiguriert, um eine erste Befehlsoperation zu speichern und um die erste Befehlsoperation für die Ausführung auszugeben. Der Ablaufplaner ist konfiguriert, um die erste Befehlsoperation nach der Ausgabe zurück zu halten, und ist angeschlossen, um ein erstes Signal zu empfangen, das anzeigt, dass die erste Befehlsoperation nicht korrekt ausgeführt wurde. In Reaktion auf das erste Signal ist der Ablaufplaner konfiguriert, die Befehlsoperation in Reaktion auf das erste Signal erneut auszugeben. An den Ablaufplaner angeschlossen, um die erste Befehlsoperation in Reaktion auf die Ausgabe davon durch den Ablaufplaner zu empfangen, wobei die Ausführungseinheit konfiguriert ist, die erste Befehlsoperation auszuführen.

Kurze Beschreibung der Zeichnungen

[0014] Weitere Aufgaben und Vorteile der Erfindung werden bei dem Studium der folgenden detaillierten Beschreibung und der Bezugnahme auf die begleitenden Zeichnungen offenbar, in denen:

[0015] **Fig.** 1 ein Blockdiagramm eines Ausführungsbeispiels eines Prozessors ist.

[0016] **Fig.** 2 ein beispielhaftes Pipelinediagramm ist, welches von einem in **Fig.** 1 gezeigten Ausführungsbeispiel des Prozessors verwendet werden kann.

[0017] **Fig.** 3 ein Blockdiagramm ist, das ein Ausführungsbeispiel der in **Fig.** 1 gezeigten Abbildungseinheit, des Ablaufplaners, des Ganzzahl Ausführungskerns und der Lade/Speicher-Einheit detaillierter darstellt.

[0018] **Fig.** 4 ein Blockdiagramm eines Ausführungsbeispiels des in den **Fig.** 1 und 3 gezeigten Ablaufplaners ist.

[0019] **Fig.** 5 ein Blockdiagramm eines Ausführungsbeispiels eines Abhängigkeitsvektors ist.

[0020] **Fig.** 6 ein Blockdiagramm eines Ausführungsbeispiels eines Abhängigkeitspuffers ist.

[0021] **Fig.** 7 ein Blockdiagramm eines Ausführungsbeispiels eines Bereichs des in **Fig.** 6 detaillierter gezeigten Abhängigkeitspuffers ist.

[0022] **Fig.** 8 ein Zustandsmaschinendiagramm im Hinblick auf eine Befehlsoperation innerhalb eines Ausführungsbeispiels des Ablaufplaners ist.

[0023] **Fig.** 9 ein Blockdiagramm ist, das Zustandinformation darstellt, die für jede Befehlsoperation in einem Ausführungsbeispiel des Ablaufplaners gespeichert ist.

[0024] **Fig.** 10 ein Zeitablaufdiagramm ist, welches das Auflösen einer Abhängigkeitskette darstellt.

[0025] **Fig.** 11 ein Zeitablaufdiagramm ist, welches die Ausgabe und die erneute Ausgabe von Befehlsoperationen von einem Ausführungsbeispiel des Ablaufplaners darstellt.

[0026] **Fig.** 12 ein Zeitablaufdiagramm ist, welches die Ausgabe und die nicht spekulative erneute Ausgabe von Befehlsoperationen von einem Ausführungsbeispiel des Ablaufplaners darstellt.

[0027] **Fig.** 13 ein Diagramm eines beispielhaften Eintrags in einem Ausführungsbeispiel des in **Fig.** 4 gezeigten physikalischen Adresspuffers zusammen mit einer beispielhaften Logik zum Arbeiten auf diesem Eintrag ist.

[0028] **Fig.** 14 ein Diagramm eines beispielhaften Eintrags in einem Ausführungsbeispiel des in **Fig.** 4 gezeigten Speicheridentifiziererpuffers zusammen mit einer beispielhaften Logik zum Arbeiten auf diesem Eintrag ist.

[0029] **Fig.** 15 ein Zeitablaufdiagramm eines Ausführungsbeispiels des erneuten Versuchens eines Ladevorgangs in Reaktion auf eine Speicheradresse, welche die Ladeadresse trifft, und des nachfolgenden Auflösens der abhängigen Operationen ist.

[0030] **Fig.** 16 ein Blockdiagramm eines ersten Ausführungsbeispiels eines Computersystems einschließlich des in **Fig.** 1 gezeigten Prozessors ist.

[0031] **Fig.** 17 ein Blockdiagramm eines zweiten Ausführungsbeispiels eines Computersystems einschließ-

lich des in **Fig.** 1 gezeigten Prozessors ist.

[0032] Während die Erfindung zahlreichen Modifikationen und alternativen Ausbildungen unterworten werden kann, sind bestimmte Ausführungsbeispiele davon als Beispiel in den Zeichnungen gezeigt und werden hier detailliert beschrieben werden. Es sollte jedoch verstanden werden, dass die Zeichnungen und die dazu gehörige detaillierte Beschreibung nicht beabsichtigt sind, die Erfindung auf die bestimmte offenbarte Form zu begrenzen, sondern dass die Erfindung vielmehr alle Modifikationen, Äquivalente und Alternativen umfassen soll, die in den Geist und den Umfang der vorliegenden Erfindung fallen, wie sie in den angefügten Ansprüchen definiert ist.

Weg(e) zum Ausführen der Erfindunq

Übersicht über den Prozessor

[0033] Es wird nun auf **Fig.** 1 Bezug genommen, in der ein Ausführungsbeispiel eines Prozessors **10** gezeigt ist. Weitere Ausführungsbeispiele sind möglich und werden betrachtet. In dem Ausführungsbeispiel aus **Fig.** 1 enthält der Prozessor **10** eine Zeilenvorhersage **12**, einen Befehls-Cachespeicher (B-Cachespeicher) **14**, eine Ausrichtungseinheit **16**, eine Verzweigungsvorhersage/Abruf PC Erzeugungseinheit **18**, eine Vielzahl von Dekodiereinheiten **24A**– **24D**, eine Vorhersagefehltreffer-Dekodiereinheit **26**, eine Mikrocode-Einheit **28**, eine Abbildungseinheit **30**, eine Rückzugswarteschlange **32**, eine architekturisierte Umbenennungsdatei **34**, eine zukünftige Datei **20**, einen Ablaufplaner **36**, eine Ganzzahl-Registerdatei **38A**, eine Gleitkomma-Registerdatei **38B**, einen Ganzzahl-Ausführungskern **40A**, einen Gleitkomma-Ausführungskern **40B**, eine Lade/Speichereinheit **42**, einen Daten-Cachespeicher (D-Cachespeicher) **44**, eine externe Interfaceeinheit **46** und einen PC Silo **48**. Die Zeilenvorhersage **12** ist verbunden mit der Vorhersagefehltreffer-Dekodiereinheit **26**, der Verzweigungsvorhersage/Abruf PC Erzeugungseinheit **18**, dem PC Silo **48** und der Ausrichtungseinheit **16**. Der B-Cachespeicher **14** ist verbunden mit der Ausrichtungseinheit **16** und der Verzweigungsvorhersage/Abruf PC Erzeugungseinheit **18**, welche ferner mit dem PC Silo **48** verbunden ist. Die Ausrichtungseinheit **16** ist des weiteren mit der Vorhersagefehltreffer-Dekodiereinheit **26** und den Dekodiereinheiten **24A**–**24D** verbunden. Die Dekodiereinheiten **24A**–**24D** sind ferner mit der Abbildungseinheit **30** verbunden, und die Dekodiereinheit **24D** ist mit der Mikrocode-Einheit **28** verbunden. Die Abbildungseinheit **30** ist verbunden mit der Rückzugswarteschlange **32** (die mit der architekturisierten Umbenennungsdatei **34** verbunden ist), der zukünftigen Datei **20**, dem Ablaufplaner **36** und dem PC Silo **48**. Die architekturisierte Umbenennungsdatei **34** ist mit der zukünftigen Datei **20** verbunden. Der Ablaufplaner **36** ist mit den Registerdateien **38A**–**38B** verbunden, welche des weiteren miteinander und mit den entsprechenden Ausführungskernen **40A**–**40B** verbunden sind. Die Ausführungskerne **40A**–**40B** sind ferner mit der Lade/Speichereinheit **42** und dem Ablaufplaner **36** verbunden. Der Ausführungskern **40A** ist darüber hinaus mit dem D-Cachespeicher **44** verbunden. Die Lade/Speichereinheit **42** ist verbunden mit dem Ablaufplaner **36**, dem D-Cachespeicher **44** und der externen Interfaceeinheit **46**. Der D-Cachespeicher **44** ist mit den Registerdateien **38** verbunden. Die externe Interfaceeinheit **46** ist mit einer externen Schnittstelle **52** und dem B-Cachespeicher **14** verbunden. Elemente, die hier durch eine Bezugsnummer gefolgt von einem Buchstaben bezeichnet werden, werden gemeinsam durch die Bezugszahl allein bezeichnet. Zum Beispiel werden die Dekodiereinheiten **24A**–**24D** gemeinsam als Dekodiereinheiten **24** bezeichnet.

[0034] In dem Ausführungsbeispiel aus **Fig.** 1 verwendet der Prozessor **10** eine komplexe Befehlssatzberechnungs- (CISC) Befehlssatzarchitektur mit variabler Bitlänge. Zum Beispiel kann der Prozessor **10** die x86 Befehlssatzarchitektur (auch als IA-32 bezeichnet) verwenden. Weitere Ausführungsbeispiele können andere Befehlssatzarchitekturen verwenden, einschließlich Befehlssatzarchitekturen mit fester Länge und reduzierten Befehlssatzberechnungs- (RISC) Befehlssatzarchitekturen. Gewisse in **Fig.** 1 gezeigte Merkmale können in solchen Architekturen unterdrückt werden. Des weiteren kann, falls gewünscht, jedes der obigen Ausführungsbeispiele eine 64 Bit Architektur verwenden.

[0035] Die Verzweigungsvorhersage/Abruf PC Erzeugungseinheit **18** ist konfiguriert, um dem B-Cachespeicher **14**, der Zeilenvorhersage **12** und dem PC Silo **48** eine Abrufadresse (Abruf PC) zur Verfügung zu stellen. Die Verzweigungsvorhersage/Abruf PC Erzeugungseinheit **18** kann einen geeigneten Mechanismus zur Vorhersage von Verzweigungen enthalten, der als Hilfe bei der Erzeugung von Abrufadressen verwendet wird. In Reaktion auf die Abrufadresse stellt die Zeilenvorhersage **12** Ausrichtungsinformation, die einer Vielzahl von Befehlen entspricht, der Ausrichtungseinheit **16** zur Verfügung und kann eine nächste Abrufadresse zum Abholen von Befehlen zur Verfügung stellen, die dem von der bereit gestellten Befehlsinformation identifizierten Befehl nach folgen. Die nächste Abrufadresse kann, wie gewünscht, der Verzweigungsvorhersage/Abruf PC Erzeugungseinheit **18** zur Verfügung gestellt werden oder kann dem B-Cachespeicher **14** direkt zur Verfügung gestellt werden. Die Verzweigungsvorhersage/Abruf PC Erzeugungseinheit **18** kann eine Fangadresse von dem PC Silo **48** empfangen (falls ein Fang detektiert ist) und die Fangadresse kann den Abruf PC aufweisen, der von der Verzweigungsvorhersage/Abruf PC Erzeugungseinheit **18** erzeugt wurde. Anderenfalls kann der Abruf PC unter Verwendung der Information zu der Verzweigungsvorhersage und der Information von der Zei-

lenvorhersage **12** erzeugt werden. Im Allgemeinen speichert die Zeilenvorhersage **12** Information entsprechend zu Befehlen, die zuvor spekulativ von dem Prozessor **10** abgerufen wurden. In einem Ausführungsbeispiel enthält die Zeilenvorhersage **12** 2K Einträge, wobei jeder Eintrag eine Gruppe von einem oder mehreren Befehlen angibt, die hier als eine „Zeile" von Befehlen bezeichnet werden. Die Zeile der Befehle kann nebeneinander von der Befehle verarbeitenden Pipeline des Prozessors **10** verarbeitet werden durch die Platzierung in dem Ablaufplaner **36**.

[0036] Der B-Cachespeicher **14** ist ein Cachespeicher hoher Geschwindigkeit zum Speichern von Befehlsbytes. In Übereinstimmung mit einem Ausführungsbeispiel kann der B-Cachespeicher **14** zum Beispiel eine 128 KByte, vier Wege Satz assoziative Organisation aufweisen, die Cachezeilen mit 64 Byte verwendet. Jedoch kann jede B-Cachespeicher Struktur geeignet sein (einschließlich direkt abgebildeter Strukturen).

[0037] Die Ausrichtungseinheit **16** empfängt die Information zur Ausrichtung von Befehlen von der Zeilenvorhersage **12** und Befehlsbytes, die der Abrufadresse entsprechen, von dem B-Cachespeicher **14**. Die Ausrichtungseinheit **16** wählt Befehlsbytes in jeder der Dekodiereinheiten **24A**–**24D** in Übereinstimmung mit der zur Verfügung gestellten Information zur Ausrichtung von Befehlen aus. Genauer gesagt stellt die Zeilenvorhersage **12** einen Befehlszeiger zur Verfügung, der jeder Dekodiereinheit **24A**–**24D** entspricht. Der Befehlszeiger ortet einen Befehl innerhalb der abgerufenen Befehlsbytes zur Beförderung an die entsprechende Dekodiereinheit **24A**–**24D**. In einem Ausführungsbeispiel können bestimmte Befehle zu mehr als einer Dekodiereinheit **24A**–**24D** befördert werden. Entsprechen kann in dem gezeigten Ausführungsbeispiel eine Zeile von Befehlen von der Zeilenvorhersage **12** bis zu **4** Befehle enthalten, obwohl andere Ausführungsbeispiele mehr oder weniger Dekodiereinheiten **24** enthalten können, um für mehr oder weniger Befehle in einer Zeile zu sorgen.

[0038] Die Dekodiereinheiten **24A**–**24D** dekodieren die ihnen zur Vertügung gestellten Befehle und jede Dekodiereinheit **24A**–**24D** erzeugt Informationen, die eine oder mehrere den Befehlen entsprechende Befehlsoperationen (oder ROPs) identifizieren. In einem Ausführungsbeispiel kann jede Dekodiereinheit **24A**–**24D** bis zu zwei Befehlsoperationen pro Befehl erzeugen. Wie hier verwendet ist eine Befehlsoperation (oder ROP) eine Operation, für die eine Ausführungseinheit innerhalb der Ausführungskerne **40A**–**40B** konfiguriert ist, sie als ein einzelnes Gebilde auszuführen. Einfache Befehle können einer einzelnen Befehlsoperation entsprechen, während komplexere Befehle mehrfachen Befehlsoperationen entsprechen können. Gewisse der komplexeren Befehle können innerhalb der Mikrocode-Einheit **28** als Mikrocode Routinen implementiert sein (abgerufen von einem Nur-Lese-Speicher darin über die Dekodiereinheit **24D** in diesem Ausführungsbeispiel). Des weiteren können andere Ausführungsbeispiele eine einzelne Befehlsoperation für jeden Befehl verwenden (das heißt Befehl und Befehlsoperation können in derartigen Ausführungsbeispielen gleichbedeutend sein).

[0039] Der PC Silo **48** speichert die abgerufenen Adressen und Befehlsinformationen für jede Befehlsabrufung und ist verantwortlich für die Umleitung des Abrufens von Befehlen bei Ausnahmen (wie Befehlsfängen, wie sie von der von dem Prozessor **10** verwendeten Befehlssatzarchitektur definiert sind, falschen Vorhersagen für Verzweigungen und weiteren mikroarchitektonisch definierten Fängen). Der PC Silo **48** kann einen Ringpuffer zum Speichern der Abrufadresse und Befehlsinformation, die den mehrfachen Zeilen von Befehlen entspricht, welche innerhalb des Prozessors **10** unerledigt sind, umfassen. In Reaktion auf die Zurückziehung einer Zeile von Befehlen kann der PC Silo **48** den entsprechenden Eintrag verwerfen. In Reaktion auf eine Ausnahme kann der PC Silo **48** der Verzweigungsvorhersage/Abruf PC Erzeugungseinheit **18** eine Fangadresse zur Vertügung stellen. Die Rückzugs- und Ausnahmeinformation kann von dem Ablaufplaner **36** zur Verfügung gestellt werden. In einem Ausführungsbeispiel weist die Abbildungseinheit **30** jedem Befehl eine Sequenznummer (R#) zu, um die Reihenfolge der innerhalb des Prozessors **10** unerledigten Befehle zu identifizieren. Der Ablaufplaner **36** kann die R#s zu dem PC Silo **48** zurück geben, um die Befehlsoperationen zu identifizieren, die Ausnahmen erfahren oder Befehlsoperationen zurück ziehen.

[0040] Auf die Detektierung eines Fehltreffers in der Zeilenvorhersage **12** dirigiert die Ausrichtungseinheit **16** die entsprechenden Befehlsbytes von dem B-Cachespeicher **14** an die Vorhersagefehltreffer-Dekodiereinheit **26**. Die Vorhersagefehltreffer-Dekodiereinheit **26** dekodiert den Befehl, wobei einer Zeile von Befehlen jegliche Beschränkungen aufgezwungen werden, für die der Prozessor **10** entworfen ist (zum Beispiel maximale Anzahl von Befehlsoperationen, maximale Anzahl von Befehlen, Beenden bei Verzweigungsbefehlen usw.) sobald eine Zeile beendet ist, stellt die Vorhersagefehltreffer-Dekodiereinheit **26** die Information der Zeilenvorhersage 12 zum Speichern zur Verfügung. Es ist zu bemerken, dass die Vorhersagefehltreffer-Dekodiereinheit **26** konfiguriert sein kann, um Befehle abzusenden, wenn sie dekodiert werden. Alternativ kann die Vorhersagefehltreffer-Dekodiereinheit **26** die Zeile von Befehlsinformationen dekodieren und sie der Zeilenvorhersage **12** zum Speichern zur Verfügung stellen. Nachfolgend kann die fehltreffende Abrufadresse erneut in der Zeilenvorhersage **12** versucht werden und ein Treffer könnte detektiert werden.

[0041] Zusätzlich zu dem Dekodieren von Befehlen nach einem Fehltreffer in der Zeilenvorhersage **12** kann die Vorhersagefehltreffer-Dekodiereinheit **26** konfiguriert sein, um Befehle zu dekodieren, wenn die von der Zeilenvorhersage **12** zur Verfügung gestellte Befehlsinformation ungültig ist. In einem Ausführungsbeispiel versucht der Prozessor **10** nicht, Information in der Zeilenvorhersage **12** mit den Befehlen in dem B-Cachespei-

cher **14** kohärent zu halten (zum Beispiel wenn Befehle in dem B-Cachespeicher **14** ersetzt oder ungültig gemacht werden, kann die Befehlsinformation nicht aktiv ungültig gemacht werden). Die Dekodiereinheiten **24A**–**24D** können die zur Verfügung gestellte Befehlsinformation verifizieren und können der Vorhersagefehltreffer-Dekodiereinheit **26** signalisieren, wenn ungültige Befehlsinformation detektiert ist. In Übereinstimmung mit einem bestimmten Ausführungsbeispiel werden die folgenden Befehlsoperationen von dem Prozessor **10** unterstützt: Ganzzahl (einschließlich arithmetischen, logischen, Schiebe/Rotations- und Verzweigungsoperationen), Gleitkomma (einschließlich Multimediaoperationen und Lade/Speicher.

[0042] Die dekodierten Befehlsoperationen und Quell- und Zielregisternummern werden der Abbildungseinheit **30** zur Verfügung gestellt. Die Abbildungseinheit **30** ist konfiguriert, um eine Umbenennung der Register durchzuführen durch eine Zuweisung physikalischer Registernummern (PR#s) für jeden Zielregisteroperanden und jeden Quellregisteroperanden für jede Befehlsoperation. Die physikalische Registernummer identifiziert Register innerhalb der Registerdateien 38A–38B. Die Abbildungseinheit **30** stellt des weiteren einen Hinweis auf die Abhängigkeiten für jede Befehlsoperation zur Verfügung, durch zur Verfügung stellen von R#s der Befehlsoperationen, die jede physikalische Registernummer aktualisieren, die einem Quelloperanden der Befehlsoperation zugewiesen ist. Die Abbildungseinheit **30** aktualisiert die zukünftige Datei **20** mit den physikalischen Registernummern, die jedem Zielregister zugewiesen sind (und dem R# der entsprechenden Befehlsoperation) basierend auf der entsprechenden logischen Registernummer. Des weiteren speichert die Abbildungseinheit **30** die logischen Registernummern der Zielregisters, die zugewiesenen physikalischen Registernummern und die zuvor zugewiesenen physikalischen Registernummern in der Rückzugswarteschlange **32**. Wenn Befehle zurück gezogen werden (der Abbildungseinheit **30** von dem Ablaufplaner **36** angezeigt), aktualisiert die Rückzugswarteschlange **32** die architekturisierte Umbenennungsdatei **34** und macht jegliche Register frei, die nicht länger in Benutzung sind. Entsprechend identifizieren die physikalischen Registernummern in der architekturisierten Umbenennungsdatei **34** die physikalischen Register, welche den übergebenen architekturalen Zustand des Prozessors **10** speichern, während die zukünftige Datei **20** den spekulativen Zustand des Prozessors **10** repräsentiert. In anderen Worten speichert die architekturisierte Umbenennungsdatei **34** eine physikalische Registernummer, die jedem logischen Register entspricht, darstellend den übergebenen Registerzustand für jedes logische Register. Die zukünftige Datei **20** speichert eine physikalische Registernummer, die jedem logischen Register entspricht, darstellend den spekulativen Registerzustand für jedes logische Register.

[0043] Die Zeile von Befehlsoperationen, die physikalischen Registernummern für die Quelle und die physikalischen Registernummern für das Ziel werden in dem Ablaufplaner **36** in Übereinstimmung mit den von dem PC Silo **48** zugewiesenen R#s gespeichert. Des weiteren können Abhängigkeiten für eine bestimmte Befehlsoperation als Abhängigkeiten von anderen Befehlsoperationen, die in dem Ablaufplaner gespeichert sind, notiert werden. In einem Ausführungsbeispiel verbleiben die Befehlsoperationen in der Vorhersagefehltreffer-Dekodiereinheit **26** bis sie zurück gezogen werden.

[0044] Der Ablaufplaner **36** speichert jede Befehlsoperation bis die für diese Befehlsoperation notierten Abhängigkeiten zufrieden gestellt sind. In Reaktion auf die zeitliche Planung einer bestimmten Befehlsoperation zur Ausführung kann der Ablaufplaner **36** bestimmen, zu welchem Taktzyklus diese bestimmte Befehlsoperation die Registerdateien **38A**–**38B** aktualisieren wird. Verschiedene Ausführungseinheiten in den Ausführungskernen **40A**–**40B** können verschiedene Anzahlen von Stufen der Pipeline verwenden (und damit unterschiedliche Latenzzeiten). Des weiteren können gewisse Befehle innerhalb einer Pipeline mehr Latenz als andere erfahren. Entsprechend wird ein Countdown erzeugt, der die Latenzzeit für diese bestimmte Befehlsoperation misst (in Nummern von Taktzyklen). Der Ablaufplaner **36** erwartet die angegebene Nummer von Taktzyklen (bis die Aktualisierung geschieht vor oder zusammenfallend mit den abhängigen die Registerdatei lesenden Befehlsoperationen) und zeigt dann an, dass die Befehlsoperationen zeitlich geplant werden können, die von dieser bestimmten Befehlsoperation abhängig sind. Es ist zu bemerken, dass der Ablaufplaner **36** einen Befehl zeitlich planen kann, sobald seine Abhängigkeiten zufrieden gestellt worden sind (das heißt außer der Reihe, was seine Reihenfolge innerhalb der Warteschlange des Ablaufplaners betrifft).

[0045] Ganzzahl und Lade/Speicher Befehlsoperationen lesen Quelloperanden in Übereinstimmung mit den physikalischen Registernummern für die Quelle aus der Registerdatei **38A** und werden für die Ausführung zu dem Ausführungskern **40A** befördert. Der Ausführungskern **40A** führt die Befehlsoperation aus und aktualisiert die dem Ziel innerhalb der Registerdatei **38A** zugewiesenen physikalischen Register. Des weiteren meldet der Ausführungskern **40A** die R# der Befehlsoperation und Ausnahmeinformation hinsichtlich der Befehlsoperation (falls vorhanden) dem Ablaufplaner **36**. Die Registerdatei **38B** und der Ausführungskern **40B** können auf ähnliche Weise im Hinblick auf Gleitkomma Befehlsoperationen arbeiten (und können der Lade/Speichereinheit **42** Speicherdaten für Gleitkomma Speichervorgänge zur Verfügung stellen). Es ist zu bemerken, dass Operanden für abhängige Operationen direkt zu den abhängigen Operationen umgeleitet werden können, falls die Operationen, von denen sie abhängig sind, gleichzeitig abschließen.

[0046] In einem Ausführungsbeispiel kann der Ausführungskern **40A** zum Beispiel zwei Ganzzahleinheiten, eine Verzweigungseinheit und zwei Adresserzeugungseinheiten (mit entsprechenden Übersetzungsseitenblickpuffern oder TLBs) enthalten. Der Ausführungskern **40B** kann einen Gleitkomma/Multimedia-Multiplizie-

rer, einen Gleitkomma/Multimedia-Addieren und eine Speicherdateneinheit zum Liefern von Speicherdaten an die Lade/Speichereinheit **42** enthalten. Weitere Konfigurationen von Ausführungseinheiten sind möglich, einschließlich einem kombinierten Gleitkomma/Ganzzahl Ausführungskern.

[0047] Die Lade/Speichereinheit **42** stellt eine Schnittstelle zu dem D-Cachespeicher **44** zur Verfügung, um Speicheroperationen durchzuführen und um Fülloperationen für Speicheroperationen, die den D-Cachespeicher **44** nicht treffen, zeitlich zu planen. Lade-Speicheroperationen können von dem Ausführungskern **40A** beendet werden durch Ausführen einer Adresserzeugung und Weiterleiten von Daten an die Registerdateien **38A**–**38B** (von dem D-Cachespeicher **44** oder der Speicherwarteschlange innerhalb der Lade/Speichereinheit **42**). Speicheradressen können dem D-Cachespeicher **44** präsentiert werden, sobald sie von dem Ausführungskern **40A** erzeugt sind (direkt über Verbindungen zwischen dem Ausführungskern **40A** und dem D-Cachespeicher **44**). Die Speicheradressen werden einem Eintrag in der Speicherwarteschlange zugeordnet. Die Speicherdaten können neben einander zur Verfügung gestellt werden, oder nach einander zur Verfügung gestellt werden, in Übereinstimmung mit der Wahl des Entwicklers. Auf das Zurückziehen der Befehlsoperation werden die Daten in dem D-Cachespeicher **44** gespeichert (obwohl einige Verzögerung zwischen dem Zurückziehen und der Aktualisierung des D-Cachespeichers **44** sein kann). Des weiteren kann die Lade/Speichereinheit **42** einen Lade/Speicher Puffer enthalten zum Speichern von Lade/Speicher Adressen, die den D-Cachespeicher **44** nicht treffen, für nachfolgende Füllvorgänge des Cachespeichers (über die externe Interfaceeinheit **46**) und zum erneuten Versuchen der nicht treffenden Lade/Speicher Operationen. Die Lade/Speichereinheit **42** ist ferner konfiguriert, um Abhängigkeiten von Lade/Speicher Operationen zu behandeln.

[0048] Der D-Cachespeicher **44** ist ein Cachespeicher mit hoher Geschwindigkeit zum Speichern von Daten, auf die von dem Prozessor **10** zugegriffen werden. Während der D-Cachespeicher **44** jede geeignete Struktur aufweisen kann (einschließlich direkt abbildender oder Satz assoziativer Strukturen), kann ein Ausführungsbeispiel des D-Cachespeichers **44** einen 128 KByte, vier Wege Satz assoziativen Cachespeicher aufweisen, der Cachezeilen mit 64 Byte hat.

[0049] Die externe Interfaceeinheit **46** ist zur Kommunikation mit anderen Geräten über die externe Schnittstelle **52** konfiguriert. Jede geeignete externe Schnittstelle **52** kann verwendet werden, einschließlich Schnittstellen zu L2 Cachespeichern und einem externen Bus oder Bussen zum Anschließen des Prozessors **10** an andere Geräte. Die externe Interfaceeinheit **46** ruft Füllvorgänge für den B-Cachespeicher **14** und den D-Cachespeicher **44** ab, ebenso wie sie verworfene aktualisierte Cachezeilen von dem D-Cachespeicher **44** an die externe Schnittstelle schreibt. Des weiteren kann die externe Interfaceeinheit **46** auch von dem Prozessor **10** erzeugte Lesevorgänge und Schreibvorgänge ausführen, die nicht zwischen gespeichert werden können.

[0050] Es wird nun auf **Fig.** 2 Bezug genommen, in der ein beispielhaftes Pipelinediagramm gezeigt ist, das einen beispielhaften Satz von Stufen einer Pipeline zeigt, die von einem Ausführungsbeispiel des Prozessors **10** verwendet werden können. Weitere Ausführungsbeispiele können verschiedene Pipelines verwenden, Pipelines, die mehr oder weniger Stufen der Pipeline verwenden als die in **Fig.** 2 gezeigte Pipeline. Die in **Fig.** 2 gezeigten Stufen sind von senkrechten, gestrichelten Linien unterteilt. Jede Stufe ist ein Taktzyklus eines Taktsignals, das zur Taktung von Speicherelementen (zum Beispiel Registern, Auffangregistern, Flops usw.) innerhalb des Prozessors **10** verwendet wird.

[0051] Wie in **Fig.** 2 dargestellt enthält die beispielhafte Pipeline eine Stufe CAM0, eine Stufe CAM1, eine Stufe Zeilenvorhersage (LP), eine Stufe Befehls-Cachespeicher (IC), eine Stufe Ausrichtung (AL), eine Stufe Dekodierung (DEC), eine Stufe Abbildung1 (M1), eine Stufe Abbildung2 (M2), eine Stufe Ablaufplanung Schreiben (WR SC), eine Stufe Ablaufplanung Lesen (RD SC), eine Stufe Registerdatei Lesen (RF RD), eine Stufe Ausführung (EX), eine Stufe Registerdatei Schreiben (RF WR) und eine Stufe Zurückziehen (RET). Einige Befehle benutzen mehrfache Taktzyklen in der Stufe Ausführung. Zum Beispiel sind Speicheroperationen, Gleitkomma Operationen und Ganzzahl Multiplikationsoperationen in auseinander gezogener Darstellung in **Fig.** 2 gezeigt. Speicheroperationen können eine Stufe Adresserzeugung (AGU), eine Stufe Übersetzung (TLB), eine Stufe Daten-Cachespeicher **1** (DC1) und eine Stufe Daten-Cachespeicher **2** (DC2) enthalten. Auf ähnliche Weise können Gleitkomma Operationen bis zu vier Stufen Ausführung Gleitkomma (FEX1–FEX4) enthalten und Ganzzahl Multiplikationen bis zu vier (IM1–IM4) Stufen enthalten.

[0052] Während der Stufen CAM0 und CAM1 vergleicht die Zeilenvorhersage **12** die von der Verzweigungsvorhersage/Abruf PC Erzeugungseinheit **18** zur Verfügung gestellte Abrufadresse mit den Adressen der darin gespeicherten Zeilen. Des weiteren wird die Abrufadresse von einer virtuellen Adresse (zum Beispiel einer linearen Adresse in der x86 Architektur) zu einer physikalischen Adresse während der Stufen CAM0 und CAM1 übersetzt. In Reaktion auf die Detektierung eines Trefers während der Stufen CAM0 und CAM1 wird die entsprechende Information der Zeile von der Zeilenvorhersage während der Stufe Zeilenvorhersage gelesen. Der Lesevorgang endet während der Stufe Befehls-Cachespeicher.

[0053] Es ist zu bemerken, dass, während die in **Fig.** 2 dargestellte Pipeline zwei Taktzyklen zur Detektierung eines Treffers in der Zeilenvorhersage **12** für eine Abrufadresse verwendet, andere Ausführungsbeispiele einen einzelnen Taktzyklus (und Stufe) zur Durchführung dieser Operation verwenden können. Darüber hinaus stellt in einem Ausführungsbeispiel die Zeilenvorhersage **12** eine nächste Abrufadresse für den B-Cachespei-

cher **14** und einen nächsten Eintrag in der Zeilenvorhersage **12** für einen Treffer zur Vertügung und daher können die Stufen CAM0 und CAM1 übersprungen werden für Abrufvorgänge, die von einem vorherigen Treffer in der Zeilenvorhersage **12** stammen.

[0054] Die von dem B-Cachespeicher **14** zur Verfügung gestellten Befehlsbytes werden von der Ausrichtungseinheit **16** für die Dekodiereinheiten **24A**–**24D** ausgerichtet während der Stufe Ausrichtung in Reaktion auf die entsprechende Zeileninformation von der Zeilenvorhersage **12**. Es ist zu bemerken, dass einige Befehle auf mehr als eine Dekodiereinheit **24A**–**24D** ausgerichtet sein können. Die Dekodiereinheiten **24A**–**24D** dekodieren während der Stufe Dekodierung die zur Verfügung gestellten Befehle, die sowohl den Befehlen entsprechende ROPs als auch Operandeninformation identifizieren. Die Abbildungseinheit **30** erzeugt während der Stufe Abbildung1 ROPs von den zur Verfügung gestellten Informationen und führt eine Umbenennung der Register durch (Aktualisierung der zukünftigen Datei **20**). Während der Stufe Abbildung2 werden die ROPs und zugewiesenen Umbenennungen in der Rückzugswarteschlange **32** aufgezeichnet. Des weiteren werden die ROPs bestimmt, von denen jede ROP abhängig ist. Jede ROP kann registerabhängig von früheren ROPs sein, wie in der zukünftigen Datei ausgezeichnet ist, und kann auch andere Typen aufweisen (zum Beispiel Abhängigkeiten von einem vorherigen Serialisierungsbefehl usw.).

[0055] Die erzeugten ROPs werden während der Stufe Ablaufplanung in den Ablaufplaner **36** geschrieben. Bis zu dieser Stufe fließen die ROPs, die von einer bestimmten Zeile an Information geortet sind, als eine Einheit durch die Pipeline. Es ist zu bemerken, dass ROPs, die eine Mikrocode Routine aufweisen, eine Ausnahme zu der zuvor erwähnten. Aussage sein können, weil sie von dem Mikrocode ROM über mehrere Taktzyklen gelesen werden könnten. Jedoch können die ROPs, nachdem sie in den Ablaufplaner **36** geschrieben sind, zu verschiedenen Zeiten unabhängig durch die verbleibenden Stufen fließen. Im Allgemeinen verbleibt eine bestimmte ROP in dieser Stufe, bis sie von dem Ablaufplaner **36** zur Ausführung ausgewählt wird (zum Beispiel nachdem die ROPs, von denen die bestimmte ROP abhängig ist, wie oben beschrieben zur Ausführung ausgewählt worden sind). Entsprechend kann eine bestimmte ROP zwischen der Stufe Ablaufplanung Schreiben und der Stufe Ablaufplanung Lesen einen oder mehrere Taktzyklen an Verzögerung erfahren. Während der Stufe Ablaufplanung Lesen nimmt die bestimmte ROP an der Auswahllogik innerhalb des Ablaufplaners **36** teil, wird zur Ausführung ausgewählt und von dem Ablaufplaner **36** gelesen. Die bestimmte ROP schreitet dann in der Stufe Registerdatei Lesen fort zu den Lese Register Operationen von einer der Registerdateien **38A**–**38B** (abhängig von dem Typ der ROP).

[0056] Die bestimmte ROP und die Operanden werden dem entsprechenden Ausführungskern **40A** oder **40B** zur Verfügung gestellt und die Befehlsoperation wird während der Stufe Ausführung auf den Operanden ausgeführt. Zum Beispiel werden Speicher Befehlsoperationen (zum Beispiel Ladevorgänge und Speichervorgänge) ausgeführt durch eine Stufe Adresserzeugung (in der die Datenadressen der Speicherstelle, auf die von der Speicher Befehlsoperation zugegriffen wird, erzeugt wird), eine Stufe Übersetzung (in der die von der Stufe Adresserzeugung zur Verfügung gestellte virtuelle Datenadresse übersetzt wird) und ein Paar von Stufen Daten-Cachespeicher, in denen auf den D-Cachespeicher **44** zugegriffen wird. Gleitkomma Operationen können bis zu vier Taktzyklen an Ausführung verwenden und Ganzzahl Multiplikationen verwenden ähnlich bis zu vier Taktzyklen an Ausführung.

[0057] Nach Abschluss der Stufe oder der Stufen Ausführung aktualisiert die bestimmte ROP während der Stufe Registerdatei Schreiben ihre zugewiesenen physikalischen Register. Schließlich wird die bestimmte ROP zurück gezogen, nachdem jede vorherige ROP zurück gezogen ist (in der Stufe Zurückziehung). Wieder können einer oder mehrere Taktzyklen für eine bestimmte ROP zwischen der Stufe Registerdatei Schreiben und der Stufe Zurückziehung ablaufen. Des weiteren kann eine bestimmte ROP an jeder Stufe der Pipeline angehalten werden aufgrund von Bedingungen zum Anhalten der Pipeline, wie im Stand der Technik gut bekannt sind.

Ablaufplaner

[0058] Es wird nun auf **Fig.** 3 Bezug genommen, wo ein Blockdiagramm gezeigt ist, das ein Ausführungsbeispiel einer Abbildungseinheit **30**, einer zukünftigen Datei **20**, eines Ablaufplaners **36**, eines Ganzzahl Ausführungskerns **40A** und einer Lade/Speicher-Einheit **42** darstellt. Eine gewisse beispielhafte Verbindung ist in der **Fig.** 3 dargestellt, als auch gewisse interne Details eines Ausführungsbeispiels der Einheiten außer dem Ablaufplaner **36**. Weitere Ausführungsbeispiele sind möglich und werden betrachtet. In dem Ausführungsbeispiel von **Fig.** 3 ist die Abbildungseinheit **30** mit den Dekodiereinheiten **24A**– **24D**, der zukünftigen Datei **20** und dem Ablaufplaner **36** verbunden. Der Ablaufplaner **36** ist des weiteren mit der externen Interfaceeinheit **46**, dem Ganzzahl Ausführungskern **40A** und der Lade/Speicher-Einheit **42** verbunden. In dem Ausführungsbeispiel von **Fig.** 3 enthält die Abbildungseinheit **30** eine Zielumbenennungsschaltung **60**, eine Inline Abhängigkeitsüberprüfungsschaltung **62**, eine Abhängigkeitsordnungsschaltung **64**, einen Satz von Abhängigkeitsordnungsregistern **66A**–**66N** und einen Mux **68**. Die Zielumbenennungsschaltung **60**, die Inline Abhängigkeitsüberprüfungsschaltung **62** und die Abhängigkeitsordnungsschaltung **64** sind angeschlossen, um von den Dekodierein-

heiten **24A**–**24N** Befehlsoperationen zu empfangen. Die Zielumbenennungsschaltung **60** ist mit dem Mux **68** und dem Ablaufplaner **36** verbunden. Die Inline Abhängigkeitsüberprüfungsschaltung **62** ist mit dem Mux **68** verbunden, der darüber hinaus mit der zukünftigen Datei **20** verbunden ist. Die zukünftige Datei **20** ist angeschlossen, um Identifizieren für Quelloperanden zu empfangen, die den von der Abbildungseinheit **30** empfangenen Befehlsoperationen entsprechen. Die Abhängigkeitsordnungsschaltung **64** ist mit den Abhängigkeitsordnungsregistern **66A**–**66N** und mit dem Ablaufplaner **36** verbunden. Die Lade/Speicher-Einheit **42** umfasst eine Speicherwarteschlange **70**, die angeschlossen ist, um eine physikalische Adresse von dem Ganzzahl Ausführungskern **40A** zu empfangen. Der Ganzzahl Ausführungskern **40A** enthält eine Adresserzeugungseinheit **40AA**, die mit einem Übersetzungsseitenblickpuffer (TLB) **40AB** verbunden ist.

[0059] Im Allgemeinen empfängt die Abbildungseinheit **30** Befehlsoperationen von den Dekodiereinheiten **24A**–**24D**. Die Abbildungseinheit **30** führt für jede Befehlsoperation eine Umbenennung der Register durch und bestimmt die Abhängigkeiten für jede Befehlsoperation von älteren Operationen, die in dem Ablaufplaner **36** in Verarbeitung sind (oder gleichzeitig an den Ablaufplaner **36** abgeschickt werden). Die Abbildungseinheit **30** stellt die Befehlsoperationen und die Umbenennungen der Register dem Ablaufplaner **36** zum Speichern zur Verfügung (und spätere Ausgabe zur Ausführung). Des weiteren stellt die Abbildungseinheit **30** eine Angabe der Abhängigkeiten jeder Befehlsoperation zur Verfügung (dargestellt als die Quelloperanden-Abhängigkeiten und die Ordnungsabhängigkeiten in **Fig.** 3). Genauer gesagt identifiziert die Abbildungseinheit **30** die ältesten Befehlsoperationen durch R# (die Nummer, welche die Befehlsoperation in dem Ablaufplaner **36** identifiziert). Die PR#s der den Operanden zugeordneten physikalischen Register werden dem Ablaufplaner **36** zur Verfügung gestellt zur Ausgabe mit der Befehlsoperation, aber werden nicht bei der Bestimmung der Abhängigkeiten verwendet. Der Ablaufplaner **36** speichert die Befehlsoperationen und die entsprechenden Abhängigkeiten und plant den Ablauf der Befehlsoperationen in Reaktion auf die entsprechenden Abhängigkeiten, die befriedigt werden. Die für den Ablauf geplanten Befehlsoperationen werden an die Ausführungskerne **40A**–**40B** ausgegeben, die Ressourcen zur Ausführung haben, die konfiguriert sind zur Ausführung dieser Befehlsoperation.

[0060] Gewisse Befehlsoperationen könnten die Ausführung nicht abschließen, wenn sie ausgegeben werden. Zum Beispiel könnte in dem gezeigten Ausführungsbeispiel Speicheroperationen die Ausführung nicht abschließen. Falls ein Befehl die Ausführung nicht abschließt, wird er von einer Einheit, die in die Ausführung der Befehlsoperation einbezogen ist, „erneut versucht". Das erneute Versuchen einer Befehlsoperation umfasst das Signalisieren an den Ablaufplaner **36**, dass die Befehlsoperation zurück gezogen wird. Der Ablaufplaner **36** hält die ausgegebenen Befehlsoperationen zurück und, falls die ausgegebenen Befehlsoperationen zurück gezogen werden, gibt der Ablaufplaner **36** dann die Befehlsoperationen erneut aus. Insbesondere in einem Ausführungsbeispiel hält der Ablaufplaner **36** einen Ausführungszustand für jede Befehlsoperation aufrecht. In Reaktion auf ein erneutes Versuchen einer zuvor ausgegebenen Befehlsoperation setzt der Ablaufplaner **36** den Ausführungszustand der Befehlsoperation auf einen „nicht ausgeführt" Zustand zurück. Nachfolgend kann die Befehlsoperation erneut ausgegeben werden. Des weiteren behält der Ablaufplaner **36** die Abhängigkeiten von jeder ausgegebenen Befehlsoperation. Alle Befehlsoperationen, die direkt oder indirekt von der zurück gezogenen Befehlsoperation abhängig sind, werden ebenfalls in den nicht ausgeführten Zustand zurück gebracht. Es ist zu bemerken, dass eine Gruppe von Befehlsoperationen, in der die erste aus der Gruppe der Befehlsoperationen von einer bestimmten Befehlsoperation abhängig ist und in der jede andere Befehlsoperation innerhalb der Gruppe von den anderen Befehlsoperationen abhängig ist und durch die andere Befehlsoperation indirekt von der bestimmten Befehlsoperation abhängig ist, hier als eine „Abhängigkeitskette" bezeichnet wird.

[0061] Das Zurücksetzen des Ausführungszustands auf nicht ausgeführt in Reaktion auf einen erneuten Versuch der Befehlsoperation oder einer anderen Befehlsoperation, von der die Befehlsoperation direkt oder indirekt abhängig ist, wird hier als „rückgängig machen" dieser Befehlsoperation bezeichnet.

[0062] Indem Befehlsoperationen erlaubt wird, zurück gezogen zu werden (und in Reaktion auf ein erneutes Versuchen erneut ausgegeben zu werden), kann der Ablaufplaner **36** Befehlsoperationen aggressiv für die Ausführung im Ablauf planen und kann sich von einer nicht korrekten Ablaufplanung erholen durch die erneute Ausgabe der nicht korrekt im Ablauf geplanten Befehlsoperationen zu einem späteren Zeitpunkt. Die Strafe für eine nicht korrekte Ablaufplanung kann erheblich geringer sein als das Entfernen der nicht korrekt im Ablauf geplanten Befehlsoperation und aller jüngeren Befehlsoperationen und das erneute Abrufen beginnend an der nicht korrekt im Ablauf geplanten Befehlsoperation.

[0063] Die Abbildungseinheit **30** verwendet die Zielumbenennungsschaltung **60**, die Inline Abhängigkeitsüberprüfungsschaltung **62**, die zukünftige Datei **20** und die Abhängigkeitsordnungsschaltung **64**, um die Abhängigkeiten für jede Befehlsoperation zu bestimmen. Die Zielumbenennungsschaltung **60** empfängt eine Angabe für jede Befehlsoperation, ob diese Befehlsoperation einen Register-Zieloperanden hat oder nicht, und die Nummer des Zielregisters, falls die Befehlsoperation einen Register-Zieloperanden hat. Falls die Befehlsoperation einen Register-Zieloperanden hat, weist die Zielumbenennungsschaltung **60** der Befehlsoperation eine freie physikalische Registernummer zu. Die zugewiesenen PR#s werden dem Ablaufplaner **36** mit den Befehlsoperationen zur Verfügung gestellt. Des weiteren stellt die Zielumbenennungsschaltung **60** die R#s und die

PR#s von jedem Befehlsoperand dem Mux **68** zur Verfügung.

[0064] Die zukünftige Datei **20** stellt für jede Registernummer des Quelloperanden die PR# und die R# der Befehlsoperation zur Verfügung, die zuletzt das entsprechende architekturisierte Register als einen Zieloperanden hatte. Insbesondere kann die zukünftige Datei **20** eine Tabelle aufweisen mit Einträgen für jedes architekturisierte Register (und in Mikrocode verwendenden Ausführungsbeispielen, jedes temporäre Mikrocoderegister). Die Registernummern der Quelloperanden werden verwendet, um die Einträge der Register auszuwählen, die als Quelloperanden für die Befehlsoperationen angegeben sind. Jeder Eintrag speichert die R# der ältesten Befehlsoperation (vor der aktuellen Zeile von Befehlsoperationen), um dieses Register zu aktualisieren, und die PR# des physikalischen Registers, das dem Ziel dieser ältesten Befehlsoperation zugewiesen ist. Des weiteren enthält die zukünftige Datei **20** ein Gültig Bit (V) in jedem Eintrag. Das Gültig Bit zeigt an, ob die für dieses Register aufgezeichnete R# gültig ist oder nicht (das heißt ob die entsprechende Befehlsoperation in dem Ablaufplaner **36** immer noch gültig ist oder nicht). Das Gültig Bit wird gesetzt auf die Versendung der der R# entsprechenden Befehlsoperation in den Ablaufplaner **36** und wird zurück gesetzt, wenn die Befehlsoperation zurück gezogen wird. Das Gültig Bit wird dem Ablaufplaner **36** zur Verfügung gestellt, wenn der Eintrag als die Abhängigkeit des Quelloperanden ausgewählt wird. Der Ablaufplaner **36** zeichnet für diesen Quelloperanden keine Abhängigkeit auf, falls das Gültig Bit nicht gesetzt ist, und zeichnet eine Abhängigkeit auf, falls das Gültig Bit gesetzt ist.

[0065] Die Inline Abhängigkeitsüberprüfungsschaltung **62** empfängt die Nummern der Quell- und Zielregister von jeder Befehlsoperation und führt eine Überprüfung der Abhängigkeit innerhalb der Zeile der von der Abbildungseinheit **30** empfangenen Befehlsoperationen durch. Die Inline Abhängigkeitsüberprüfungsschaltung **62** vergleicht die Nummern der Zielregister mit von jeder älteren Befehlsoperation in der Zeile mit den Nummern der Zielregister einer bestimmten Befehlsoperation innerhalb der Zeile. Falls eine Übereinstimmung für einen der Quelloperanden gefunden wurde, übersteuert die Inline Abhängigkeitsüberprüfungsschaltung **62** die R#s und die PR#s von der zukünftigen Datei **20**, die dem Quelloperanden entsprechen, mit den entsprechenden von der Zielumbenennungsschaltung **60** zur Verfügung gestellten R# und PR#. Falls keine Übereinstimmung gefunden wurde, stellen die R# und die PR# von der zukünftigen Datei **20** die korrekte Registerumbenennungs- und Abhängigkeits- R# für diesen Quelloperanden zur Verfügung. Die Inline Abhängigkeitsüberprüfungsschaltung **62** erzeugt Multiplexerauswahlleitungen für den Mux 68, um den geeigneten R# und PR# für jeden Quelloperanden von jeder Befehlsoperation auszuwählen. Es ist zu bemerken, dass der Mux **68** jede geeignete Auswahlschaltung zum Auswählen der Abhängigkeiten des Quelloperanden darstellen kann. Zum Beispiel kann der Mux **68** separate Multiplexer für jeden möglichen Quelloperanden von jeder möglichen Befehlsoperation innerhalb der Zeile darstellen.

[0066] Die Inline Abhängigkeitsüberprüfungsschaltung kann des weiteren die Nummern der Zielregister für jede Befehlsoperation innerhalb der Zeile vergleichen, um die älteste Befehlsoperation innerhalb der Zeile zu bestimmen, um jedes architekturisierte Register zu aktualisieren, das ein Zieloperand für eine oder mehrere Befehlsoperationen innerhalb der Zeile ist. Die zukünftige Datei **20** kann dann in den Einträgen, die den Zieloperanden der Zeile entsprechen, mit den R#s und den PR#s aktualisiert werden, die von der Zielumbenennungsschaltung **60** zugewiesen wurden. Der Pfad der Aktualisierung ist wegen der Übersichtlichkeit der Zeichnung in der **Fig.** 3 nicht gezeigt.

[0067] Die Abhängigkeitsordnungsschaltung **64** verfolgt Abhängigkeiten von der Reihenfolge, die im Hinblick auf gewisse Befehlsoperationen aufgezeichnet werden können. Zum Beispiel sind in einem Ausführungsbeispiel, das die x86 Befehlssatzarchitektur verwendet, Abhängigkeiten von der Reihenfolge definiert für: (i) Segmentladevorgänge, welche eine Abhängigkeit von der Reihenfolge für jede nachfolgende Speicheroperation verursachen, (ii) Aktualisierungen von Steuerworten für Gleitkomma, welche eine Abhängigkeit von der Reihenfolge für jede nachfolgende Gleitkomma Befehlsoperation verursachen. Im Allgemeinen führt jede Befehlsoperation, die eine Serialisierungssperre für nachfolgende Befehlsoperationen erzeugt, zu einer Abhängigkeit von der Reihenfolge von den serialisierenden Befehlsoperationen zu den nachfolgenden betroffenen Befehlsoperationen. Eine "Serialisierungssperre" ist eine Sperre in der Programmsequenz, um die eine ungeordnete oder spekulative Ausführung verboten ist. Einige Befehlssatzarchitekturen haben Befehle, deren alleinige Funktion es ist, die Serialisierungssperre zur Verfügung zu stellen.

[0068] Die oben erwähnten Abhängigkeiten von der Reihenfolge können nach verfolgt werden unter Verwendung der Abhängigkeitsordnungsregister **66A**–**66N**. Die Abhängigkeitsordnungsschaltung **64** speichert in Reaktion auf eine Befehlsoperation, welche eine Abhängigkeit von der Reihenfolge erzeugt, die R# der Befehlsoperation in einem der Abhängigkeitsordnungsregister **66A**–**66N**. Ein Abhängigkeitsordnungsregister **66A**–**66N** kann für jede von dem Prozessor **10** detektierte Abhängigkeit von der Reihenfolge zur Verfügung gestellt werden. Zusätzlich kann ein Gültig Bit enthalten sein und kann in Reaktion auf das Aufzeichnen einer R# gesetzt werden und zurück gesetzt werden auf das zurück ziehen der entsprechenden Befehlsoperation (ähnlich wie bei dem Gültig Bit in der zukünftigen Datei **20**). In Reaktion auf eine Befehlsoperation, die über eine bestimmte Abhängigkeit von der Reihenfolge als von der Reihenfolge abhängig definiert ist, stellt die Abhängigkeitsordnungsschaltung **64** die entsprechende R# als eine der Abhängigkeiten von der Reihenfolge für

diese Befehlsoperation zur Verfügung.

[0069] Zusätzlich zu den obigen besonderen Situationen kann die Abhängigkeitsordnungsschaltung **64** eine Tabelle verwenden, um vorherige Ereignisse von Lade-Speicheroperationen zu verfolgen, welche vor den älteren Speicher-Speicheroperationen im Ablauf geplant waren und nachfolgend als abhängig von dieser älteren Speicher-Speicheroperation befunden wurde (für den Speicheroperanden, auf den von dem Ladevorgang zugegriffen wurde). Die Tabelle kann eine erste Tabelle aufweisen, die mit der Abrufadresse der Lade-Speicheroperation indiziert wird und mit der Abrufadresse der älteren Speicher-Speicheroperation trainiert wird, wenn die Abhängigkeit während der Ausführung detektiert wird. Die zweite Tabelle wird mit der Abrufadresse der Speicher-Speicheroperation indiziert und wird auf das Versenden der Speicher-Speicheroperationen mit der R# der Speicher-Speicheroperation aktualisiert. Falls die Lade-Speicheroperation ein Treffer in der Tabelle ist, wird die entsprechende R# als eine Abhängigkeit von der Reihenfolge für die Lade-Speicheroperation zur Verfügung gestellt.

[0070] Wie oben erwähnt plant der Ablaufplaner **36** den Ablauf und gibt eine Befehlsoperation in Reaktion auf die Detektierung, dass jede Abhängigkeit von dieser Befehlsoperation befriedigt ist, an einen geeigneten Ausführungskern aus. Genauer gesagt werden Speicheroperationen an eine Adresserzeugungseinheit **40AA** innerhalb des Ausführungskerns **40A** ausgegeben. Die Adresserzeugungseinheit **40AA** empfängt die Registeroperanden von der Ganzzahl Registerdatei **38A** und erzeugt die Adresse des der Speicheroperation entsprechenden Speicheroperanden. Die Adresse ist die virtuelle Adresse, welche mittels eines Schemas zur Adressübersetzung, das von der von dem Prozessor **10** verwendeten Befehlssatzarchitektur angegeben ist, in eine physikalische Adresse zum Zugreifen auf den Speicher (und den D-Cachespeicher **44**) übersetzt wird. Der TLB **40AB** ist ein Cachespeicher für die Ergebnisse von vorherigen Übersetzungen, was eine schnelle Übersetzung der virtuellen Adressen, welche darin treffen, in entsprechende physikalische Adressen und was eine schnelle Bestimmung der verschiedenen Attribute erlaubt, die den entsprechenden Speicherstellen über den Mechanismus zur Übersetzung zugewiesen wurden. Die Kombination von AGU **40AA** und TLB **40AB** stellt der Lade/Speicher-Einheit **42** (und parallel dazu dem D-Cachespeicher **44** und dem Ablaufplaner **36**) eine physikalische Adresse zur Verfügung.

[0071] Die Lade/Speicher-Einheit **42** bestimmt, ob die Speicheroperation die Ausführung erfolgreich abschließt oder ob sie zurück zu ziehen ist. Falls eine Situation zum Rückzug detektiert wird, legt die Lade/Speicher-Einheit **42** das erneutes Versuchen Signal an dem Ablaufplaner **36** an und stellt den Grund für den erneuten Versuch über die erneuter Versuchstyp Signale zur Verfügung. In einem Ausführungsbeispiel können Speicheroperationen aus den folgenden Gründen zurückgezogen werden:

(i) die Speicheroperation ist eine Lade-Speicheroperation, welche den D-Cachespeicher **44** nicht trifft;

(ii) die Speicheroperation erfordert einen Puffer in der Lade/Speicher-Einheit **42**, der voll ist (zum Beispiel ein Fehltreffer-Puffer zum Speichern von Fehltrefferadressen, die von der externen Interfaceeinheit **46** aus dem Hauptspeicher abgerufen werden);

(iii) die Speicheroperation erfährt einen Bankkonflikt innerhalb des D-Cachespeichers **44** mit einer anderen Speicheroperation, die gleichzeitig auf den D-Cachespeicher **44** zugreift;

(iv) die Speicheroperation ist eine Speicher-Speicheroperation und erfordert eine Überprüfung auf selbst modifizierenden Code (SMC);

(v) die Speicheroperation ist eine Lade-Speicheroperation, die eine oder mehrere Speicheroperationen in der Speicherwarteschlange **70** trifft (das heißt die eine oder die mehreren Speicheroperationen unterstützen zumindest ein Byte des Speicheroperanden des Speicheroperanden, auf dem von der Lade-Speicheroperation zugegriffen wird) und die Speicherwarteschlange **70** ist nicht fähig, die entsprechenden Daten weiterzuleiten;

(vi) die Speicheroperation ist nicht spekulativ auszuführen.

[0072] Der Grund (i) ist als getrennter erneuter Versuchstyp kodiert, für den der Ablaufplaner **36** eine übereinstimmende Fülladresse erwartet, die von der externen Interfaceeinheit **46** vor der Ablaufplanung und dem erneuten Ausgeben der Lade-Speicheroperation zur Verfügung gestellt wird. Die externe Interfaceeinheit **46** stellt die Fülladresse zur Verfügung, um anzuzeigen, dass die Daten von der Fülladresse an den D-Cachespeicher **44** zum Speichern zur Verfügung gestellt werden (und damit, dass entsprechende Lade-Speicheroperationen Treffer in dem D-Cachespeicher **44** sein könnten). Der Ablaufplaner **36** zeichnet die physikalische Adresse der Lade-Speicheroperation auf (zur Verfügung gestellt von dem Ausführungskern **40A**) zum Vergleich mit der Fülladresse. Die Gründe (ii), (iii) und (v) können als ein einzelner erneuter Versuchstyp kodiert sein, auf den der Ablaufplaner **36** antworten kann durch erneutes Planen des Ablaufs der entsprechenden Speicheroperation ohne jegliche besondere Warteerfordernisse. Der Grund (iv) ist als ein erneuter Versuchstyp kodiert und der Ablaufplaner **36** kann die entsprechende Speicher-Speicheroperation nachdem die SMC Überprüfung abgeschlossen worden ist für die erneute Ausgabe im Ablauf planen. Der Grund (vi) ist als ein erneuter Versuchstyp kodiert und der Ablaufplaner **36** plant den Ablauf der Speicheroperation zur erneuten Ausgabe nachdem die entsprechende Speicheroperation nicht spekulativ wird. In Übereinstimmung mit einem

bestimmten Ausführungsbeispiel ist eine Speicheroperation als nicht spekulativ auszuführen, wenn die Speicheroperation auf einen Speicheroperanden zugreift, der eine Seitengrenze überquert (das heißt zumindest ein Byte des Speicheroperanden ist in einer zweiten Seite gespeichert, die von einer zweiten Adressübersetzung übersetzt wurde, die anders ist als die erste Adressübersetzung), die Übersetzung zeigt an, dass der Typ des Speichers des Speicheroperanden nicht spekulativ ist oder die Speicheroperation trifft fehl in dem TLB. Der erste und der letzte Grund für das nicht spekulative Ausführen sind beim Entwurf gewählt, um die Hardware zu vereinfachen, und der mittlere Grund wird von der von dem Prozessor **10** verwendeten Befehlssatzarchitektur gefordert.

[0073] Es ist zu bemerken, dass, während die obige Beschreibung sich auf die erneute, nicht spekulative Ausgabe gewisser Speicheroperationen bezieht, andere Befehlsoperationen ebenfalls nicht spekulativ erneut ausgegeben werden können. Zum Beispiel kann jede Befehlsoperation, die eine Ausnahme erfährt (zum Beispiel eine Falle oder einen von der Architektur angegebenen Fehler oder eine für die von dem Prozessor **10** implementierte Mikroarchitektur definierte mikroarchitekturale Ausnahme), nicht spekulativ erneut ausgegeben werden. Auf diese Weise kann Information, die auf die Ausnahme bezogen ist, während der nicht spekulativen Ausführung aufgezeichnet werden. Daher kann der Aufwand an Hardware, der zum Speichern und Verfolgen von Ausnahmeinformation verwendet wird, minimiert werden.

[0074] Die Speicherwarteschlange **70** stellt über die Treffer und Speicher R# Signale zusätzliche Information hinsichtlich der Lade-Speicheroperationen zur Verfügung, welche Speicher-Speicheroperationen innerhalb der Speicherwarteschlange treffen. Die Treffer und Speicher R# werden unabhängig davon zur Verfügung gestellt, ob ein erneuter Versuch der Lade-Speicheroperation auftritt oder nicht. Das Treffer Signal zeigt an, dass ein Treffer in der Speicherwarteschlange detektiert wurde, und die Speicher R# ist die R# des Speichervorgangs, der von dem Ladevorgang getroffen wurde. Diese Information kann verwendet werden, um einen erneuten Versuch der Lade-Speicheroperation zu veranlassen, falls der Speichervorgang, der von dem Ladevorgang getroffen wurde nachfolgend erneut ausgeführt wird (und eine andere Adresse empfängt). Die Verwendung der Speicher R# wird detaillierter unten beschrieben. Es ist zu bemerken, dass, während die Speicher R# in diesem Beispiel verwendet wird, jeglicher Identifizieren, der den Speichervorgang identifiziert, verwendet werden kann. Zum Beispiel kann die Nummer der Speicherwarteschlange zur Verfügung gestellt werden, welche den Eintrag der Speicherwarteschlange innerhalb der Speicherwarteschlange **70** identifiziert, der von dem Ladevorgang getroffen wird. Derartige Ausführungsbeispiele werden betrachtet.

[0075] Wie oben bemerkt, könnte die Speicherwarteschlange **70** nicht fähig sein, in allen Fällen, wenn eine Lade-Speicheroperation eine Speicher-Speicheroperation in der Speicherwarteschlange **70** trifft, die Daten weiter zu leiten. Zum Beispiel können zahlreiche Bytes des Lade-Speicheroperanden von verschiedenen Speichervorgängen in der Speicherwarteschlange **70** zur Verfügung gestellt werden. Jedoch kann die Speicherwarteschlange **70** die Anzahl der separaten Speichervorgänge begrenzen, von denen Bytes eines bestimmten Lade-Speicheroperanden weiter geleitet werden könnten. Falls zum Beispiel die Speicherwarteschlange **70** fähig ist, die Daten von bis zu zwei Speicher-Speicheroperationen weiter zu leiten, verhindert ein Treffen auf zwei oder mehr Speicher-Speicheroperationen für verschiedene Bytes von diesem bestimmten Lade-Speicheroperanden die Weiterleitung von allen Bytes des bestimmten Lade-Speicheroperanden. Des weiteren können einige Ausführungsbeispiele der Speicherwarteschlange **70** die Adresse des Lade-Speicheroperanden vor dem Empfangen der Speicherdaten empfangen. Falls die Speicherdaten nicht zur Verfügung stehen, ist die Speicherwarteschlange **70** nicht fähig, die Speicherdaten weiter zu leiten, sogar wenn ein Treffer detektiert würde.

[0076] Es ist zu bemerken, dass Befehlsoperationen hier als "älter" oder "jünger" als andere Befehlsoperationen seiend bezeichnet werden. Eine erste Befehlsoperation ist "älter" als eine zweite Befehlsoperation, wenn die erste Befehlsoperation in der Reihenfolge des Programms vor der zweiten Befehlsoperationen ist. Andererseits ist eine erste Befehlsoperation "jünger" als eine zweite Befehlsoperation, wenn die erste Befehlsoperation in der Reihenfolge des Programms der zweiten Befehlsoperation nachfolgend ist. Wie hier verwendet bezieht sich der Ausdruck "erneute Ausgabe" auf die Ausgabe einer Befehlsoperation, die zuvor ausgegeben wurde (und als nicht korrekt ausgeführt befunden wurde, entweder direkt über einen erneuten Versuch oder indirekt über die von dem Ablaufplaner **36** für diese Befehlsoperation aufgezeichneten Abhängigkeiten). Des weiteren bezieht sich der Ausdruck "Speicheroperation" wie er hier verwendet wird auf eine Befehlsoperation, die eine Speicheroperation hat. Lade-Speicheroperationen haben einen Speicher-Quelloperanden als einen Quelloperanden (und einen Register-Zieloperanden) und geben den Transfer von Daten von dem Speicher-Quelloperanden zu dem Register-Zieloperanden an. Speicher-Speicheroperationen haben einen Register-Quelloperanden und einen Speicher-Zieloperanden und geben den Transfer von Daten von dem Register-Quelloperanden zu dem Speicher-Zieloperanden an. Es ist zu bemerken, dass, obwohl **Fig.** 3 eine Adresserzeugungseinheit **40AA** und einen entsprechenden TLB **40AB** darstellt, verschiedene Ausführungsbeispiele eine beliebige Anzahl von Adresserzeugungseinheiten und TLBs enthalten können. Die Lade/Speicher-Einheit **42** kann separate erneutes Versuchen Signale, erneuter Versuchstyp Signale, Treffer Signale und Speicher R#s für Speicheroperationen, die jeder AGU entsprechen, zur Verfügung stellen.

[0077] Nun Bezug nehmend auf **Fig.** 4 ist ein Blockdiagramm eines Ablaufplaners **36** gezeigt. Weitere Aus-

führungsbeispiele sind möglich und werden betrachtet. Wie in **Fig.** 4 gezeigt umfasst der Ablaufplaner **36** einen Befehlsoperation (ROP) Puffer **80**, eine Ausgabeauswahlschaltung **82**, eine Rückzugsgrenzschaltung **84**, eine ROP Steuerschaltung **86**, einen Abhängigkeitspuffer **88**, einen Puffer für physikalische Adressen **90**, einen Speichern R# Puffer **92**, eine Rückzugsschaltung **94** und eine Abhängigkeitsdekodiererschaltung **96**. Der ROP Puffer **80** ist angeschlossen zum Empfangen von Befehlsoperationen (einschließlich solcher Information wie unmittelbare oder Versetzungsdaten, usw.) und von zugewiesenen PR#s von der Abbildungseinheit **30** und ist verbunden, um ausgegebene Befehlsoperationen und PR#s den Registerdateien **38A**–**38B** und den Ausführungskernen **40A**–**40B** zur Verfügung zu stellen. Der ROP Puffer **80** ist des weiteren an die Ausgabeauswahlschaltung **82** angeschlossen, welche an die ROP Steuerschaltung **86** angeschlossen ist. Die Rückzugsgrenzschaltung **84** ist mit der Rückzugsschaltung **94** und der ROP Steuerschaltung **86** verbunden, welche angeschlossen ist an die Rückzugsschaltung **94**, den Abhängigkeitspuffer **88**, den Puffer für physikalische Adressen **90** und den Speichern R# Puffer **92**. Die ROP Steuerschaltung **86** ist des weiteren angeschlossen, um die erneutes Versuchen und erneuter Versuchstyp Signale von der Lade/Speicher-Einheit **42** zu empfangen. Die Abhängigkeitsdekodiererschaltung **96** ist angeschlossen zum Empfangen von der Quellabhängigkeits R#s und der Ordnungsabhängigkeits R#s von der Abbildungseinheit **30** und ist mit dem Abhängigkeitspuffer **88** verbunden. Der Puffer für physikalische Adressen **90** ist angeschlossen zum Empfang einer Fülladresse von der externen Interfaceeinheit **46** und einer oder mehrerer physikalischer Adressen von dem Ausführungskern **40A**. Der Speichern R# Puffer **92** ist angeschlossen zum Empfang eines oder mehrerer Treffersignale und einer oder mehrerer Speicher R#s von der Lade/Speicher-Einheit **42**.

[0078] Die Abhängigkeitsdekodiererschaltung **96** empfängt die R#s, welche Befehlsoperationen identifizieren, von denen jede in den Ablaufplaner **36** geschriebene Befehlsoperation abhängig ist, und dekodiert die R#s in Abhängigkeitsangaben für die entsprechende Befehlsoperation. Wie oben bemerkt wird, falls ein R# als ungültig angezeigt wird (zum Beispiel von der zukünftigen Datei **20**), dann eine auf dieser R# basierende Abhängigkeit nicht angezeigt. Im Gegensatz zu der Belieferung der Abhängigkeitsdekodiererschaltung **96** kann die Abbildungseinheit **30** die Angaben über die Abhängigkeit für jede Befehlsoperation direkt erzeugen (zum Beispiel durch Bereitstellung eines Abhängigkeitsvektors für jede Befehlsoperation wie den in **Fig.** 5 gezeigten). Im Allgemeinen wird eine Angabe zur Abhängigkeit einer ersten Befehlsoperation und einer zweiten Befehlsoperation zugewiesen und identifiziert eine Abhängigkeit (oder ein Fehlen davon) der erstem Befehlsoperation von der zweiten Befehlsoperation. Zum Beispiel kann jede Befehlsoperation ein Bit aufweisen, das, wenn gesetzt, eine Abhängigkeit der ersten Befehlsoperation von der zweiten Befehlsoperation anzeigt, und wenn nicht gesetzt ein Fehlen einer Abhängigkeit der ersten Befehlsoperation von der zweiten Befehlsoperation anzeigt. Die gesetzten und nicht gesetzten Bedeutungen des Bits können in einem Ausführungsbeispiel umgekehrt werden und weitere Kodierungen der Angaben zur Abhängigkeit sind möglich.

[0079] Die Abhängigkeitsdekodiererschaltung **96** stellt die Angaben über die Abhängigkeit dem Abhängigkeitspuffer **88** zum Speichern zur Verfügung. Der Abhängigkeitspuffer **88** weist mehrere Abhängigkeitseinträge auf, von denen jedem zwei Einträge in dem ROP Puffer **80** zugewiesen sind. Der Abhängigkeitseintrag speichert die Angabe über die Abhängigkeit, welche die Abhängigkeit oder das Fehlen derselben einer ersten Befehlsoperation, die in einem der zwei Einträge in dem ROP Puffer **80** gespeichert ist, von einer zweiten Befehlsoperation, die in dem anderen der zwei Einträge gespeichert ist, identifiziert. Falls die Angabe über die Abhängigkeit eine Abhängigkeit anzeigt, dann ist die erste Befehlsoperation nicht für die Ablaufplanung auswählbar bis die zweite Befehlsoperation die Abhängigkeit befriedigt.

[0080] Die ROP Steuerschaltung **86** überwacht die Angaben über die Abhängigkeit innerhalb des Abhängigkeitspuffers **88** und die Befriedigung von diesen Abhängigkeiten und identifiziert diejenigen Befehlsoperationen, welche für die Ablaufplanung auswählbar sind. Die ROP Steuerschaltung **86** identifiziert die auswählbaren Befehlsoperationen für die Ausgabeauswahlschaltung **82**, welche die auswählbaren Befehlsoperationen abtastet, um Befehlsoperationen für die Ausgabe an die Ausführungskerne **40A**–**40B** auswählt. Ausgewählte Befehlsoperationen werden in Reaktion auf die Ausgabeauswahlschaltung **82** von dem ROP Puffer **80** gelesen und den Registerdateien **38A**–**38B** und den Ausführungskernen **40A**–**40B** für die Ausführung zur Verfügung gestellt. Im Allgemeinen ist die Ausgabeauswahlschaltung **82** konfiguriert, um eine Befehlsoperation für jede Ausführungseinheit in jedem der Ausführungskerne **40A**– **40B** auszuwählen (falls eine Befehlsoperation von jenem Typ ist, der für die Ablaufplanung auswählbar ist). Die ausgewählte Befehlsoperation ist die älteste Befehlsoperation von diesem Typ, die für die Ablaufplanung auswählbar ist. In einem Ausführungsbeispiel tastet die Ausgabeauswahlschaltung **82** die auswählbaren Befehlsoperationen zweimal pro Taktzyklus ab, um die Auswahl von zwei Befehlsoperationen von einem gegebenen Typen zu ermöglichen. Die zweite Abtastung wählt eine zweite Befehlsoperation zur Ausgabe an eine zweite Ausführungseinheit eines bestimmten Typs aus (zum Beispiel zwei Adresserzeugungseinheiten und zwei ALUs werden in einem Ausführungsbeispiel des Ausführungskerns **40A** zur Verfügung gestellt). In der zweiten Abtastung wird die während der ersten Abtastung ausgewählte Befehlsoperation maskiert (das heißt erscheint nicht auswählbar), so dass die zweitälteste Befehlsoperation des entsprechenden Typs ausgewählt werden kann.

[0081] In einer bestimmten Ausführung kann die Ausgabeauswahlschaltung **82** unabhängige Auswahlschal-

tungen für jeden Typ von Befehlen haben. Jede Auswahlschaltung kann parallel zu dem Betrieb der anderen Auswahlschaltungen nach Befehlsoperationen von dem entsprechenden abtasten. Jeder Befehlstyp kann verschiedene Ressourcen zur Ausführung (zum Beispiel Ausführungseinheiten) von den anderen Befehlstypen verwenden, was eine unabhängige Operation der Auswahlschaltungen erlaubt.

[0082] Die Ausgabeauswahlschaltung **82** meldet (der ROP Steuerschaltung **86**) welche Befehlsoperationen für die Ausgabe ausgewählt worden sind. Die ausgewählten Befehlsoperationen werden als im Ablauf geplant bezeichnet und die Befehlsoperationen werden ausgegeben (oder erneut ausgegeben) sobald sie von dem ROP Puffer **80** gelesen worden sind. Die ROP Steuerschaltung **86** unterhält einen Ausführungszustand für jede Befehlsoperation. Der Ausführungszustand kann im weitesten Sinn definiert sein, um einen "nicht ausgeführten" Zustand, einen "ausführend" Zustand und einen "erledigt" Zustand zu umfassen. Jeder dieser Zustände kann mehrere Zustände aufweisen, wie in der beispielhaften Zustandsmaschine dargestellt, die in **Fig.** 8 gezeigt ist, je nach Wahl des Entwicklers. Eine Befehlsoperation wird als nicht ausgeführt angesehen bei der Speicherung in den Ablaufplaner **36**, bis die Befehlsoperation ausgegeben ist. Der Ausführungszustand der Befehlsoperation wird in ausführend geändert in Reaktion auf das Ausgegeben werden und wechselt nachfolgend in den erledigt Zustand auf die Erledigung der Ausführung. Der Ausführungszustand der Befehlsoperation kann zu jedem Zeitpunkt in den nicht ausgeführten Zustand geändert werden (oder kann "rückgängig gemacht" sein), wenn die Befehlsoperation erneut versucht wird (zum Beispiel über die erneutes Versuchen Signale von der Lade/Speicher-Einheit **42**) oder wenn eine andere Befehlsoperation von der diese Befehlsoperation abhängt (direkt oder indirekt) rückgängig gemacht wird. Die ROP Steuerschaltung **86** kann im Allgemeinen eine bestimmte Befehlsoperation als für die Ablaufplanung auswählbar identifizieren, falls die bestimmte Befehlsoperation einen Ausführungszustand von nicht ausgeführt hat und falls jede Abhängigkeit der bestimmten Befehlsoperation befriedigt worden ist.

[0083] Da der Ausführungszustand einer Befehlsoperation in nicht ausgeführt geändert wird in Reaktion auf ein erneutes Versuchen für diese Befehlsoperation, kann die Befehlsoperation auswählbar werden für eine erneute Ablaufplanung und für eine erneute Ausgabe in Reaktion auf das erneute Versuchen. Jedoch können gewisse Typen von erneuten Versuchen angeben, dass die Befehlsoperation nicht erneut im Ablauf zu planen ist bis zu dem Auftreten eines nachfolgenden Ereignisses (zum Beispiel wird eine Fülladresse in dem Fall einer Lade-Speicheroperation zur Verfügung gestellt, die nicht trifft, oder die Befehlsoperation wird nicht spekulativ). In derartigen Fällen kann die ROP Steuerschaltung **86** den Ausführungszustand der zurück gezogenen ROP in nicht ausgeführt ändern, aber kann nicht signalisieren, dass die Befehlsoperation für die Ablaufplanung auswählbar ist bis das nachfolgende Ereignis auftritt.

[0084] Da die Angaben über die Abhängigkeit nicht aus dem Abhängigkeitspuffer **88** in Reaktion auf die Ausgabe der entsprechenden Befehlsoperationen gelöscht werden, können Befehlsoperationen innerhalb einer Kette von Abhängigkeiten spekulativ ausgegeben werden, wenn die Abhängigkeiten befriedigt werden. Die Abhängigkeiten anderer Befehlsoperationen von einer bestimmten Befehlsoperation werden als nicht befriedigt erneut kategorisiert, wenn die bestimmte Befehlsoperation rückgängig gemacht wird, und daher werden diese anderen Befehlsoperationen ebenfalls rückgängig gemacht. Auf diese Weise wird eine spekulativ ausgegebene Kette von Abhängigkeiten rückgängig gemacht und in Reaktion auf ein erneutes Versuchen der ersten Befehlsoperation in der Kette erneut ausgegeben.

[0085] Zusätzlich zu den während der Ausführung einer Lade-Speicheroperation gemeldeten erneuten Versuchen können Lade-Speicheroperationen auch aufgrund älterer Speicher-Speicheroperationen erneut versucht werden, die nachfolgend zu der Lade-Speicheroperation ausgeben. Der Puffer für physikalische Adressen **90** wird zur Verfügung gestellt für die Detektierung dieser Szenarien für das erneute Versuchen. Im Allgemeinen werden Lade-Speicheroperationen nicht angezeigt (über die Angaben zur Abhängigkeit innerhalb des Abhängigkeitspuffers **88**) als abhängig seiend von älteren Speicher-Speicheroperationen. Stattdessen werden Lade-Speicheroperationen im Ablauf geplant ohne Berücksichtigung von älteren Speicher-Speicheroperationen (mit der Ausnahme, in einem Ausführungsbeispiel, des oben beschriebenen Mechanismus der Ordnungsabhängigkeit). Es ist jedoch möglich, dass eine Lade-Speicheroperation von einer älteren Lade-Speicheroperation abhängig sein kann, falls die ältere Lade-Speicheroperation wenigstens ein Byte der Speicheroperanden aktualisiert, auf das von der Lade-Speicheroperation zugegriffen wird. Um diese Szenarien zu detektieren, speichert der Puffer für physikalische Adressen **90** die physikalischen Adressen, auf die von dem Ladevorgang zugegriffen wird (empfangen von dem Ausführungskern **40A**). der Puffer für physikalische Adressen **90** enthält die gleiche Anzahl von Einträgen wie der ROP Puffer **80**, wobei jeder Eintrag fähig ist, Informationen über physikalischen Adressen für eine Lade-Speicheroperation und zugewiesen zu einem entsprechenden Eintrag in dem ROP Puffer **80** zu speichern. Der einer ausführenden Lade-Speicheroperation entsprechende Eintrag wird mit der physikalischen Adresse der Lade-Speicheroperation aktualisiert.

[0086] Während der Ausführung von Speicher-Speicheroperationen wird die physikalische Adresse, die von der Speicher-Speicheroperation aktualisiert wurde, von dem Ausführungskern **40A** zur Verfügung gestellt. Der Puffer für physikalische Adressen **90** vergleicht die Speicheradresse mit den physikalischen Adressen in dem Puffer für physikalische Adressen **90**, welche jüngeren Lade-Speicheroperationen entsprechen. Anders aus-

gedrückt ist der Vergleich der Adressen maskiert auf diejenigen Einträge in dem Puffer für physikalische Adressen **90**, welche den Befehlsoperationen entsprechen, die jünger sind als die ausführende Speicher-Speicheroperation. Falls ein Treffer der Speicheradresse auf einer Ladeadresse detektiert wird, wird die entsprechende Lade-Speicheroperation rückgängig gemacht (der Puffer für physikalische Adressen **90** signalisiert der ROP Steuerschaltung **86**, dass die entsprechende Lade-Speicheroperation getroffen worden ist, und die ROP Steuerschaltung **86** ändert den Ausführungszustand der entsprechenden Lade-Speicheroperation auf nicht ausgeführt). Die entsprechende Lade-Speicheroperation wird später erneut ausgegeben. Während der Ausführung nach dem erneuten Ausgeben wird die Lade-Speicheroperation entweder die ältere Speicher-Speicheroperation in der Speicher-Warteschlange **70** treffen (und die Speicherdaten werden weiter geleitet oder die Lade-Speicheroperation wird zurück gezogen) oder die ältere Lade-Speicheroperation wird den Cachespeicher oder den Hauptspeicher aktualisieren lassen. In jeden Fall empfängt die Lade-Speicheroperation den korrekten Speicheroperanden nach dem erneuten Ausgeben und erfolgreich abgeschlossener Ausführung. Es ist zu bemerken, dass in einem Ausführungsbeispiel, falls eine Lade-Speicheroperation wegen eines älteren Speichervorgangs, der die entsprechende physikalische Adresse in dem Puffer für physikalische Adressen **90** trifft, rückgängig gemacht wird, die Lade-Speicheroperation in die Tabelle in der Schaltung für Ordnungsabhängigkeiten **64** trainiert werden kann.

[0087] Während der Puffer für physikalische Adressen **90** einen Mechanismus zum Erholen von nicht korrekter Ablaufplanung einer Lade-Speicheroperation vor einer älteren Speicher-Speicheroperation, von der die Lade-Speicheroperation abhängig ist, zur Verfügung stellt, kann ein weiteres Problem existieren, welches die Lade-Speicheroperation veranlasst, rückgängig gemacht zu werden. Sogar wenn die Lade-Speicheroperation nach der Speicher-Speicheroperation, von der sie abhängt, im Ablauf geplant wird und die Speicherdaten von der Speicherwarteschlange in der Lade/Speicher-Einheit **42** weiter geleitet werden, kann die Speicher-Speicheroperation selbst rückgängig gemacht werden. Die Adressoperanden der Speicher-Speicheroperation (verwendet, um die Adresse des Speicheroperanden der Speicher-Speicheroperation zu bilden) können während der erneuten Ausgabe anders sein (das heißt das Empfangen eines nicht korrekten Adressoperanden kann der Grund für die erneute Ausgabe sein) und daher könnte die Speicheradresse nicht den Puffer für physikalische Adressen **90** während der Ausführung der erneuten Ausgabe treffen und die Lade-Speicheroperation veranlassen, ungültig gemacht zu werden. Der Ablaufplaner **36** ist mit dem Speichern R# Puffer **92** ausgestattet, um diese Möglichkeit zu behandeln.

[0088] In Antwort auf die Detektierung eines Treffers einer Lade-Speicheroperation auf einen Speichervorgang in der Speicher-Warteschlange **70** stellt die Lade/Speicher-Einheit **42** dem Ablaufplaner **36** ein Treffer Signal und die Speicher R# der Speicher-Speicheroperation, die von der Lade-Speicheroperation getroffen wird, zur Verfügung. Ähnlich wie der Puffer für physikalische Adressen **90** enthält der Speichern R# Puffer **92** die gleiche Anzahl von Einträgen wie der ROP Puffer **80**. Jeder der Einträge ist einem entsprechenden Eintrag in dem ROP Puffer **80** zugeordnet. Falls das Treffer Signal für eine ausführende Lade-Speicheroperation ausgegeben ist, speichert der Speichern R# Puffer **92** die Speicher R#, die von der Lade/Speicher-Einheit **42** zur Verfügung gestellt wird.

[0089] Die Lade/Speicher-Einheit **42** stellt ebenfalls die R# eines ausführenden Speichervorgangs an den Speichern R# Puffer **92** zur Verfügung. Die Speicher R# wird mit den in dem Speichern R# Puffer **92** gespeicherten R#s verglichen. Falls eine Übereinstimmung detektiert wird, signalisiert der Speichern R# Puffer **92** der ROP Steuerschaltung **86**, dass die entsprechende Lade-Speicheroperation rückgängig zu machen ist. Die ROP Steuerschaltung **86** ändert den Ausführungszustand der entsprechenden Lade-Speicheroperation auf nicht ausgeführt in Antwort auf das Signal. Nachfolgend wird die Lade-Speicheroperation erneut im Ablauf geplant und erneut ausgegeben. Es ist zu bemerken, dass die Speicher R# während der Ausführung der Speicher-Speicheroperation durch den Ausführungskern **40A** zur Verfügung gestellt werden kann, falls dies gewünscht ist.

[0090] Zusätzlich zu der Detektierung der Abhängigkeiten von Speichervorgängen zu Ladevorgängen, wie oben beschrieben, kann der Puffer für physikalische Adressen **90** für weitere Zwecke verwendet werden. Zum Beispiel kann der Puffer für physikalische Adressen **90** verwendet werden, um festzustellen, wann eine Lade-Speicheroperation erneut auszugeben ist, die im D-Cachespeicher **44** fehl getroffen hat. Die Lade-Speicheroperation wird nachfolgend zu den entsprechenden Daten erneut ausgegeben, die von der externen Interfaceeinheit **46** zur Verfügung gestellt werden. Entsprechend stellt die externe Interfaceeinheit **46** eine Fülladresse zur Verfügung, die Fülldaten identifiziert, welche dem D-Cachespeicher **44** zur Verfügung gestellt werden. Der Puffer für physikalische Adressen **90** vergleicht die Fülladresse mit den darin gespeicherten Adressen und signalisiert der ROP Steuerschaltung **86** jegliche Übereinstimmungen. In Antwort zeichnet die ROP Steuerschaltung **86** auf, dass die Daten für die Lade-Speicheroperation zur Verfügung gestellt worden sind und dass die Lade-Speicheroperation erneut im Ablauf geplant werden kann (angenommen dass weitere Abhängigkeiten. der Lade-Speicheroperation befriedigt sind).

[0091] Die externe Interfaceeinheit **46** kann des weiteren Sondierungsadressen zur Vertügung stellen, die den auf dem externen Interface empfangenen Sondierungen entsprechen. Im Allgemeinen werden Sondierun-

gen verwendet, um die Kohärenz des Speichers in Computersystemen aufrecht zu erhalten und geben einen Block im Cachespeicher, der von einem anderen Gerät angenommen wird, und den richtigen Zustand des Cachespeichers für den Block des Cachespeichers an, falls der Prozessor **10** eine Kopie des Block des Cachespeichers hat. Falls die Sondierungsadresse eine Lade physikalische Adresse in dem Puffer für physikalische Adressen **90** trifft könnte der entsprechende Ladevorgang erfordern, erneut im Ablauf geplant zu werden, um die Kohärenz und die Regeln der Speicherordnung, die von der von dem Prozessor **10** verwendeten Befehlssatzarchitektur beizubehalten. Zum Beispiel gibt die x86 Befehlssatzarchitektur eine strenge Speicherordnung an. Daher könnte ein spekulativer Ladevorgang, der von einer Sondierung getroffen wird, erneut im Ablauf geplant werden müssen, falls vorherige Speicheroperationen in dem Ablaufplaner **36** existieren und nicht ausgeführt worden sind.

[0092] Wie zuvor erwähnt speichert der ROP Puffer **80** die Befehlsoperationen und gibt die Befehlsoperationen an die Registerdateien **38A**–**38B** und die Ausführungskerne **40A**–**40B** in Antwort auf die Ausgabeauswahlschaltung **82** aus.

[0093] Der ROP Puffer **80** weist eine Vielzahl von Einträgen auf, von denen jeder zum Speichern einer Befehlsoperation fähig ist. Der einer bestimmten Befehlsoperation zugewiesene Eintrag wird durch die R# der Befehlsoperation identifiziert. Entsprechend hat jeder Eintrag in dem ROP Puffer **80**: (i) einen entsprechenden ersten zugewiesenen Satz von Abhängigkeitseinträgen in dem Abhängigkeitspuffer **88**, welche Angaben über die Abhängigkeit der Befehlsoperation in diesem Eintrag von anderen Befehlsoperationen in dem Ablaufplaner **36** speichert; (ii) einen entsprechenden zweiten zugewiesenen Satz von Abhängigkeitseinträgen, welche die Angaben über die Abhängigkeit von anderen Befehlsoperationen in dem Ablaufplaner **36** von der Befehlsoperation in diesem Eintrag speichert; (iii) einen entsprechenden Eintrag in dem Puffer für physikalische Adressen und (iv) einen entsprechenden Eintrag in dem Speichern R# Puffer. Gemeinsam werden die Einträge in den verschiedenen Puffern des Ablaufplaners **36**, die einer gegebenen R# entsprechen, hier als ein "Eintrag im Ablaufplaner" bezeichnet.

[0094] Die Rückzugsgrenzauswahlschaltung **84** und die Rückzugsschaltung **94** arbeiten zusammen, um Befehlsoperationen aus dem Ablaufplaner **36** zurück zu ziehen. Die ROP Steuerschaltung **86** zeigt der Rückzugsgrenzauswahlschaltung an, welche Befehlsoperationen einen Ausführungszustand von abgeschlossen haben. Die Rückzugsgrenzauswahlschaltung **84** tastet die Angaben von dem Anfang des Ablaufplaners **36** ab (das heißt der ältesten Befehlsoperation in dem Ablaufplaner **36**) auf entweder die erste Befehlsoperation mit einem Ausführungszustand, der nicht abgeschlossen ist, oder eine vorbestimmte maximale Anzahl von Befehlsoperationen ist abgetastet worden und alte sind in einem abgeschlossenen Zustand. Die Rückzugsgrenzauswahlschaltung **84** bestimmt daher die jüngste Befehlsoperation, welche zurück gezogen sein könnte und die Rückzugsschaltung **94** bestimmt, wie viele Befehlsoperationen tatsächlich zurück gezogen sind. Die Rückzugsschaltung **94** sendet die R# der letzten Befehlsoperation, die zurück gezogen wird, und kommuniziert zu der ROP Steuerschaltung **86**, welche Befehlsoperationen zurück gezogen werden. Für jede zurück gezogene Befehlsoperation macht die ROP Steuer-schaltung **86** den entsprechenden Eintrag in dem ROP Puffer **80**, dem Puffer für physikalische Adressen **90** und dem Speichern R# Puffer **92** ungültig. Des weiteren setzt die ROP Steuerschaltung **86** für jede zurück gezogene Befehlsoperation jeden Abhängigkeitseintrag in dem Abhängigkeitspuffer **88**, der eine Abhängigkeit einer Befehlsoperation von einer zurück gezogenen Befehlsoperation anzeigt.

[0095] Wie hier verwendet bezieht sich der Ausdruck "Puffer" auf einen Speicher, der konfiguriert ist, um Gegenstände an Information zu speichern. Der Puffer kann einen oder mehrere Einträge enthalten, von denen jeder eine Speicherstelle in dem Speicher ist, die ausreichend Speicherplatz hat, um einen, der Gegenstände an Information zu speichern, für den der Puffer entworfen ist.

[0096] Es ist zu bemerken, dass, während der Puffer für physikalische Adressen **90** und der Speichern R# Puffer **92** beschrieben sind als die gleiche Anzahl von Einträgen wie der ROP Puffer **80** zu haben sein, andere Ausführungsbeispiele Puffer mit weniger Einträgen einsetzen können. Jeder Eintrag in dem Puffer **90** oder **92** kann zum Beispiel eine Markierung enthalten, welche den Eintrag in dem ROP Puffer **80** identifiziert, der die entsprechende Lade-Speicheroperation speichert. Es ist ferner zu bemerken, dass wie zuvor erwähnt die Nummer der Speicher-Warteschlange statt der Speicher R# verwendet werden kann, um die erneute Ausgabe der Speicher-Speicheroperationen zu detektieren, welche als von einer Lade-Speicheroperation getroffen detektiert wurden.

[0097] Es wird nun auf **Fig.** 5 Bezug genommen, in der ein Blockdiagramm eines Ausführungsbeispiels eines Abhängigkeitsvektors **100** gezeigt ist. Der Abhängigkeitsvektor **100** enthält eine Vielzahl von Abhängigkeitsangaben **102A**–**102N**. Jede Abhängigkeitsangabe **102A**–**102N** zeigt die Abhängigkeit (oder das Fehlen davon) einer Befehlsoperation an, die dem Abhängigkeitsvektor **100** auf eine andere Befehlsoperation in dem Ablaufplaner **36** entspricht. Die Befehlsoperation kann daher von einer zufälligen Anzahl von anderen Befehlsoperationen abhängig sein. Des weiteren können, da die Abhängigkeiten in Überein-stimmung mit der Befehlsoperation und nicht mit dem Typ der Abhängigkeit aufgezeichnet werden, die Abhängigkeiten aus willkürlichen Gründen erzeugt werden (zum Beispiel um den Entwurf des Prozessors **10** zu vereinfachen). Wie zuvor er-

wähnt kann der Abhängigkeitsvektor **100** durch die Dekodierung von Abhängigkeits R#s, die von der Abbildungseinheit **30** zur Verfügung gestellt werden, und das Setzen der verbleibenden Abhängigkeitsangaben, um keine Abhängigkeit anzuzeigen, erzeugt werden. Alternativ kann die Abbildungseinheit **30** dem Ablaufplaner **36** Abhängigkeitsvektoren in der in **Fig.** 5 gezeigten Form zum Speichern zur Verfügung stellen.

[0098] Es wird nun auf **Fig.** 6 Bezug genommen, in der ein Blockdiagramm eines Ausführungsbeispiels des Abhängigkeitspuffers **88** gezeigt ist. Weitere Ausführungsbeispiele sind möglich und werden betrachtet. In dem Ausführungsbeispiel von **Fig.** 6 umfasst der Abhängigkeitspuffer **88** eine Vielzahl von Abhängigkeitseinträgen einschließlich der Abhängigkeitseinträge **104A**–**104L**. Die Abhängigkeitseinträge, welche Abhängigkeiten einer bestimmten Befehlsoperation identifizieren, die in einem bestimmten Eintrag des Ablaufplaners **36** gespeichert ist (das heißt ein Eintrag in dem ROP Puffer **80** und entsprechende Einträge in dem Puffer für physikalische Adressen **90** und in dem Speichern R# Puffer **92**), werden als Zeilen und Spalten von Abhängigkeitseinträgen angeordnet. Jede Zeile von Abhängigkeitseinträgen speichert die Abhängigkeitsangaben, welche die Abhängigkeiten einer bestimmten Befehlsoperation in einem bestimmten Eintrag des Ablaufplaners angeben. Zum Beispiel werden die Abhängigkeitseinträge, welche die Abhängigkeiten der Befehlsoperation in dem Eintrag 0 des Ablaufplaners identifizieren, in den Abhängigkeitseinträgen **104A**–**104G** gespeichert (und den dazwischen liegenden Einträgen dieser Zeile, nicht gezeigt). Die in den Abhängigkeitseinträgen **104A**–**104G** gezeigten beispielhaften Abhängigkeitsangaben stellen die Abhängigkeit der Befehlsoperation in dem Eintrag 0 des Ablaufplaners von der Befehlsoperation in dem Eintrag N–2 (Abhängigkeitseintrag **104F**) des Ablaufplaners dar. Des weiteren gibt jede Spalte von Abhängigkeitseinträgen die Abhängigkeiten jeder anderen Befehlsoperation von einer bestimmten Befehlsoperation an. Zum Beispiel werden die Abhängigkeiten jeder anderen Befehlsoperation von der Befehlsoperation in dem Eintrag 0 des Ablaufplaners in den Abhängigkeitseinträge **104H**–**104L** aufgezeichnet. Die gezeigten beispielhaften Abhängigkeitsangaben stellen eine Abhängigkeit der Befehlsoperation in dem Eintrag **2** des Ablaufplaners von der Befehlsoperation in dem Eintrag 0 (Abhängigkeitseinträge **104I**) des Ablaufplaners dar.

[0099] Der Abhängigkeitspuffer **88** ist angeschlossen zum Empfangen eines Satzes von Eingangssignalen (Block(0) bis Block(N-1)). Jedes Blocksignal entspricht einem der Einträge des Ablaufplaners. Das Blocksignal zeigt, wenn angelegt, an, dass die in dem entsprechenden Eintrag des Ablaufplaners gespeicherte Befehlsoperation nicht befriedigte Abhängigkeiten auf diese Befehlsoperation hat. Wenn nicht angelegt, zeigt das Blocksignal an, dass die Abhängigkeiten von dieser Befehlsoperation befriedigt worden sind. Im Allgemeinen wird das Blocksignal angelegt auf das Schreiben der entsprechenden Befehlsoperation in den Ablaufplaner **36** und wird zurückgenommen während der Ausführung der entsprechenden Befehlsoperation. Falls die Befehlsoperation zurück gezogen oder anders rückgängig gemacht wird, wird das Blocksignal zurückgenommen bis die entsprechende Befehlsoperation erneut ausgeführt wird. Die Blocksignale werden von der ROP Steuerschaltung **86** angelegt und zurückgenommen in Übereinstimmung mit dem Ausführungszustand der entsprechenden Befehlsoperation. Jedes Blocksignal wird zu den Abhängigkeitseinträgen weiter geleitet, welche Abhängigkeiten anderer Befehlsoperationen von der entsprechenden Befehlsoperation aufzeichnen. Zum Beispiel wird Block(0) zu den Abhängigkeitseinträgen **104N**–**104L** weiter geleitet. Wenn das Blocksignal zurück genommen wird, werden die entsprechenden Abhängigkeiten als befriedigt angesehen. Wenn zum Beispiel Block (0) zurück genommen wird, ist die Abhängigkeit der Befehlsoperation in dem Eintrag **2** des Ablaufplaners von der Befehlsoperation in dem Eintrag 0 des Ablaufplaners befriedigt.

[0100] Der Abhängigkeitspuffer **88** stellt des weiteren eine Vielzahl von Ausgangssignalen (Not_Blocked(0) bis Not_Blocked(N-1)) zur Verfügung. Jedes Not_Blocked Signal entspricht einem der Einträge des Ablaufplaners. Das Not_Blocked Signal zeigt, wenn angelegt, an, dass die Abhängigkeiten der in dem entsprechenden Eintrag des Ablaufplaners gespeicherten Befehlsoperation befriedigt worden sind. Wenn zurück genommen zeigt das Not_Blocked Signal an, dass die Abhängigkeiten der in dem entsprechenden Eintrag des Ablaufplaners gespeicherten Befehlsoperation nicht befriedigt worden sind. Im Allgemeinen ist das Not_Blocked Signal nicht angelegt bis das letzte Blocksignal, das einer Abhängigkeit der entsprechenden Befehlsoperation entspricht, zurück genommen ist, und dann wird das Not_Blocked Signal angelegt. Befehlsoperationen, für welche das Not_Blocked Signal angelegt ist, sind für die Ablaufplanung auswählbar, zumindest im Hinblick auf die Abhängigkeiten von dieser Befehlsoperation (das heißt andere Bedingungen, wie der erneute Versuchstyp, der das Warten auf ein nachfolgendes Ereignis angibt, können die Ablaufplanung verhindern). Jedes Not_Blocked Signal wird zu den Abhängigkeitseinträgen weiter geleitet, welche Abhängigkeiten der entsprechenden Befehlsoperation aufzeichnen. Zum Beispiel wird Not_Blocked(0) zu den Abhängigkeitseinträgen **104A**–**104G** weiter geleitet. Die Not_Blocked Signale können jeweils eine verdrahtete ODER Leitung sein, welche auf angelegt vorgeladen wird und dann von einem oder mehreren Abhängigkeitseinträgen zurück genommen wird, für welche das entsprechende Blocksignal angelegt ist und die Abhängigkeitsangabe eine Abhängigkeit anzeigt.

[0101] Durch das Aufzeichnen von Abhängigkeiten basierend auf der Position der Befehlsoperationen innerhalb des Ablaufplaners (zum Beispiel durch R#) im Gegensatz zu einer Basierung auf einer Ressource oder eines Abhängigkeitsgrunds, kann der Abhängigkeitspuffer **88** leichter zu implementieren sein und mit höheren Frequenzen betreibbar sein. Die Verdrahtung innerhalb des Abhängigkeitspuffers **88** kann höchst üblich sein

(das heißt kein Bereich des Abhängigkeitspuffers ist überladen im Hinblick auf die Verdrahtung und es wenig Überlappung der Leiter). Die Normalität erleichtert die Implementierung und kann zu einem Betrieb mit höheren Frequenzen beitragen (zum Beispiel durch erlauben einer dichteren Implementierung des Abhängigkeitspuffers **88**).

[0102] Es ist zu bemerken, dass die Abhängigkeitseinträge auf der Diagonalen von der oberen linken zu der unteren Rechten, wie in der **Fig.** 6 gezeigt, eine Abhängigkeit der Befehlsoperation von sich selbst anzeigen würden. Diese Abhängigkeitseinträge könnten nicht implementiert sein (wie durch die punktierten Kästen, welche diese Einträge repräsentieren, dargestellt).

[0103] Wie hier verwendet, bezieht sich der Ausdruck "angelegt" auf die Bereitstellung eines logisch wahren Werts für ein Signal oder ein Bit. Ein Signal oder Bit kann angelegt werden, wenn es einen Wert transportiert, der ein Fehlen einer bestimmten Bedingung anzeigt. Ein Signal oder Bit kann als angelegt definiert sein, wenn es einen logischen Wert von Null befördert, oder umgekehrt, wenn es einen logischen Wert von Eins befördert, und das Signal oder Bit kann als nicht angelegt definiert sein, wenn der gegenteilige logische Wert befördert wird.

[0104] Es wird nun auf **Fig.** 7 Bezug genommen, in der ein Blockdiagramm detaillierter gezeigt ist, das einen Bereich eines Ausführungsbeispiels des Abhängigkeitspuffers **88** und der ROP Steuerschaltung **86** darstellt. Weitere Ausführungsbeispiele sind möglich und werden betrachtet. In Übereinstimmung mit der **Fig.** 7 weist die ROP Steuerschaltung **86** eine Vielzahl von unabhängigen Schaltungen auf, von denen jede einem Eintrag in dem Ablaufplaner **36** entspricht. Zum Beispiel wird der Eintrag (i) in dem Ablaufplaner in der **Fig.** 7 dargestellt. Eine ROP Steuerschaltung(i) **86A** ist dargestellt zum Nachverfolgen der Zustands der Ausführung der in dem Eintrag (i) gespeicherten Befehlsoperation. Des weiteren sind einige Abhängigkeitseinträge **104M**–**104N** gezeigt, welche Abhängigkeiten der in dem Eintrag (i) gespeicherten Befehlsoperation speichern. Insbesondere sind die Abhängigkeitseinträge gezeigt, welche eine Abhängigkeit der in dem Eintrag (i) gespeicherten Befehlsoperation von der in dem Eintrag (j) (Abhängigkeitseintrag **104M**) gespeicherten Befehlsoperation und von der in dem Eintrag (j + 1) (Abhängigkeitseintrag **104N**) gespeicherten Befehlsoperation anzeigen. Die Block(i) und die Not_Blocked(i) Signale sind gezeigt, ebenso wie die Block(j) und die Block(j + 1) Signale. Die ROP Steuerschaltung(i) **86A** ist angeschlossen zur Bereitstellung des Block(i) Signals und ist angeschlossen zum Empfangen des Not_Blocked(i) Signals. Des weiteren ist die ROP Steuerschaltung(i) **86A** angeschlossen zum Empfangen eines retry_PA(i) Signals und eines fill_hit(i) Signals von dem Speichern R# Puffer **92**, einem erneuter Versuch Signal und erneuter Versuchtyp Signalen von der Lade/Speicher-Einheit **42**, einem almost_done Signal von den Ausführungskernen **40A**–**40B** und einem pick(i) Signal von der Ausgabeauswahlschaltung **82**. Des weiteren ist die ROP Steuerschaltung(i) **86A** angeschlossen, um ein request(i) Signal der Ausgabeauswahlschaltung **82** zur Verfügung zu stellen.

[0105] Die ROP Steuerschaltung(i) **86A** fängt an, die Abhängigkeiten der in dem Eintrag (i) gespeicherten Befehlsoperation zu überwachen, sobald die Befehlsoperation in den in den Eintrag (i) geschrieben ist. Bis die Befehlsoperation die Abhängigkeiten anderer Befehlsoperationen von dieser Befehlsoperation befriedigt hat, gibt die ROP Steuerschaltung(i) **86A** das Block(i) Signal aus (welches an die Abhängigkeitseinträge weiter geleitet wird, welche anderer Befehlsoperationen von der Befehlsoperation aufzeichnen, wie in **Fig.** 6 dargestellt). Die Befehlsoperation hat die nicht befriedigte Abhängigkeiten, während der Zustand der Ausführung der Befehlsoperation in dem nicht ausgeführt zustand ist und während der Zustand der Ausführung in dem ausführenden Zustand ist, aber nicht nahe genug zum Abschluss der Ausführung ist, um befriedigte Abhängigkeiten zu haben. Des weiteren überwacht die ROP Steuerschaltung(i) **86A** das Not_Blocked(i) Signal, um festzustellen, wann die Abhängigkeiten der Befehlsoperation befriedigt worden sind.

[0106] Jeder Abhängigkeitseintrag **104**, der eine Abhängigkeitsangabe einer Befehlsoperation von einer anderen Befehlsoperation speichert, ist angeschlossen, um das Not_Blocked(i) Signal zurück zu nehmen, um anzuzeigen, dass die Befehlsoperation blockiert ist. Zum Beispiel ist der Abhängigkeitseintrag **104M** mit einem UND Gatter **106A** und einem Transistor **108A** verbunden und der Abhängigkeitseintrag **104N** ist mit einem UND Gatter **106B** und einem Transistor **108B** verbunden. Falls die gespeicherte Abhängigkeitsangabe und der Abhängigkeitseintrag eine Abhängigkeit anzeigen und das entsprechende Blocksignal angelegt ist, aktiviert das UNG Gatter den entsprechenden Transistor, der das Not_Blocked(i) Signal zurück nimmt. Andererseits deaktiviert das UND Gatter den entsprechenden Transistor und dieser Transistor legt das Not_Blocked(i) Signal nicht an, wenn die Abhängigkeitsangabe keine Abhängigkeit anzeigt oder das Blocksignal nicht angelegt ist. Entsprechend blockieren Befehlsoperationen, von denen die Befehlsoperation in dem Eintrag (i) nicht abhängig ist, nicht die Ausgabe von dieser Befehlsoperation. Befehlsoperationen, von denen die Befehlsoperation in dem Eintrag (i) abhängig ist, blockieren die Ausgabe dieser Befehlsoperation bis die Abhängigkeit befriedigt ist (angezeigt durch die Rücknahme des entsprechenden Blocksignals).

[0107] In Antwort auf das Anlegen des Not_Blocked Signals legt die ROP Steuerschaltung(i) 86A das request(i) Signal an die Ausgabeauswahlschaltung **82** an. Die Ausgabeauswahlschaltung **82** tastet die request(i) Signale zusammen mit ähnlichen Signalen von anderen Steuerschaltungen, die anderen Einträgen entsprechen, ab. Sobald die Ausgabeauswahlschaltung **82** den Ablauf der Befehlsoperation in dem Eintrag (i) für die

Ausgabe plant, legt die Ausgabeauswahlschaltung **82** das pick(i) Signal an. In Antwort auf das pick(i) Signal ändert die ROP Steuerschaltung(i) den Zustand der Ausführung auf ausführend. Wie oben bemerkt, zeichnet der Ablaufplaner **36** in dem vorliegenden Ausführungsbeispiel die Latenz der Befehlsoperation auf und zählt die Taktzyklen von der Ausgabe der Befehlsoperation, um den Punkt festzustellen, an dem Abhängigkeiten befriedigt sind. Weitere Ausführungsbeispiele können zum Beispiel Abschlusssignale von den Ausführungseinheiten empfangen oder jeglichen anderen alternativen Mechanismus zur Feststellung, wann Abhängigkeiten befriedigt sind, verwenden. Des weiteren haben in dem vorliegendem Ausführungsbeispiel gewisse Befehlsoperationen eine variable Latenz oder haben eine längere Latenz als erwünscht zu zählen ist. Für derartige Befehlsoperationen können die Ausführungskerne **40A**–**40B** ein almost done Signal zur Verfügung stellen. Das almost done Signal wird angelegt, wenn die Ausführungskerne feststellen, dass eine Befehlsoperation mit variabler Latenz eine voreingestellte Anzahl von Taktzyklen vor dem Abschluss erreicht hat. Das almost done Signal kann von der ROP Steuerschaltung(i) 86A verwendet werden, um anzufangen, Zyklen bis zu der voreingestellten Anzahl zu zählen, an welchem Punkt die Befehlsoperation die Ausführung abgeschlossen hat.

[0108] Falls die Befehlsoperation eine Speicheroperation ist, tastet die ROP Steuerschaltung(i) **86A** jedes erneuter Versuch Signal von der Lade/Speicher-Einheit **42** während des Taktzyklus ab, in dem der Zustand des erneuten Versuchs für die in dem Eintrag (i) gespeicherte Befehlsoperation zur Verfügung gestellt wird. In Reaktion auf ein angelegtes erneuter Versuch Signal ändert die ROP Steuerschaltung(i) **86** den Zustand der Ausführung auf nicht ausgeführt und legt das Block(i) Signal erneut an. Auf diese Weise wird die Befehlsoperation in einen Zustand vor der Ausgabe zurück gebracht und nachfolgende Befehlsoperationen in einer Abhängigkeitskette mit der Befehlsoperation werden ebenfalls in einen Zustand vor der Ausgabe zurück gebracht (durch Zurücknehmen der entsprechenden Not_Blocked Signale). Zusätzlich tastet die ROP Steuerschaltung(i) **86A** die erneuter Versuchstyp Signale ab, falls das erneuter Versuch Signal angelegt ist. Falls der erneute Versuchstyp das Auftreten eines nachfolgenden Ereignisses erfordert bevor die Befehlsoperation erneut ausgegeben ist, zeichnet die ROP Steuerschaltung(i) **86A** das zu suchende Ereignis auf und verhindert eine erneute Anfrage nach einer erneuten Ausgabe (durch erneutes Anlegen des request(i) Signals), bis das nachfolgende Ereignis auftritt.

[0109] Zusätzlich zu dem erneuten versucht werden während der Ausführung, können Lade-Speicheroperationen erneut versucht werden, weil eine physikalische Adresse einer ausführenden Speicher-Speicheroperation die physikalische Adresse der Lade-Speicheroperation (gespeichert in dem Puffer für physikalische Adressen **90**) trifft oder die R# der ausführenden Speicher-Speicheroperation die Speichern R# trifft, die für die Lade-Speicheroperation aufgezeichnet wurde. Der Puffer für physikalische Adressen **90** legt ein retry_PA(i) Signal an, um den vorigen Fall an die ROP Steuerschaltung(i) 86A zu kommunizieren (und kann ähnliche Signale für jeden anderen Eintrag enthalten). Der Speichern R# Puffer **92** legt ein retry_stq(i) Signal an, um den späteren Fall zu kommunizieren (und kann ähnliche Signale für jeden anderen Eintrag enthalten). In Antwort auf das Anlegen eines der beiden Signale ändert die ROP Steuerschaltung(i) **86A** den Zustand der Ausführung auf nicht ausgeführt und legt das Block(i) Signal erneut an. Unter der Annahme, dass das Not_Blocked(i) Signal angelegt ist, kann die ROP Steuerschaltung(i) **86A** das request(i) Signal anlegen, um eine erneute Ablaufplanung und eine erneute Ausgabe der Befehlsoperation anzufordern.

[0110] Zusätzlich zu den erneuter versuch, retry_PA(i) und retry_stq(i) Signalen kann der Zustand der Ausführung der Befehlsoperation auf nicht ausgeführt zurück gebracht werden, falls das Not_Blocked(i) Signal zurück genommen wird. Dieser Mechanismus wird verwendet, um den abgeschlossen Zustand einer Abhängigkeitskette rückgängig zu machen, wenn eine Befehlsoperation an dem Anfang der Kette rückgängig gemacht wird, um die erneute Ausgabe der Befehlsoperationen in der Abhängigkeitskette zu veranlassen. Entsprechend ändert, falls das Not_Blocked(i) Signal nicht angelegt ist, die ROP Steuerschaltung(i) **86A** den Zustand der Ausführung auf nicht ausgeführt und legt das Block(i) Signal wieder an (was nachfolgend weitere Not_Blocked Signale veranlassen kann, zurück zu nehmen, was die Abhängigkeitskette weiter rückgängig macht).

[0111] Der Puffer für physikalische Adressen **90** stellt der ROP Steuerschaltung(i) 86A ein zusätzliches Signal zur Verfügung, um anzuzeigen, falls eine von der externen Interfaceeinheit **46** zur Verfügung gestellte Adresse die physikalische Adresse des Ladevorgangs in dem Puffer für physikalische Adressen **90** trifft, als fill_hit(i) in **Fig.** 7 gezeigt. Der Puffer für physikalische Adressen **90** legt das fill_hit(i) Signal an, um anzuzeigen, dass die von der externen Interfaceeinheit **46** zur Verfügung gestellte Adresse die dem Eintrag (i) zugeordnete physikalische Adresse in dem Puffer für physikalische Adressen **90** trifft. Die externe Interfaceeinheit **46** kann auch Füll/Sondierung Signale zur Verfügung stellen, um den Typ der zur Verfügung gestellten Adresse anzuzeigen. Falls die Füll/Sondierung Signale Füllen anzeigen, dann ist das Anlegen des fill_hit(i) eine Angabe, dass die Fülldaten für die Zeile des Cachespeichers einschließlich der physikalischen Adresse für die Lade-Speicheroperation zur Verfügung gestellt werden. Falls die Lade-Speicheroperation an der Ablaufplanung gehindert wird, wegen einer Detektierung eines Fehltreffers im Cachespeicher während einer vorherigen Ausgabe, kann die Lade-Speicheroperation für ein erneutes Planen des Ablaufs auswählbar sein und die ROP Steuerschaltung(i) 86A kann das request(i) Signal in Antwort auf den Treffer der Fülladresse anlegen. Das oben erwähnte Aus-

führungsbeispiel stellt auch Adressen von der externen Interfaceeinheit **46** zur Verfügung, um Sondierungen auszuführen. Falls das fill_hit(i) Signal angelegt ist und die Füll/Sondierung Signale von der externen Interfaceeinheit **46** eine Sondierung anzeigen, dann wird ein Sondierungstreffer detektiert, der eine korrigierende Aktion erfordern könnte. In einem Ausführungsbeispiel kann das Anlegen des fill_hit(i) Signals für eine Sondierung die ROP Steuerschaltung(i) **86A** veranlassen, den Zustand der Ausführung auf nicht ausgeführt zu ändern. Weitere Ausführungsbeispiele können kompliziertere Mechanismen versuchen, um eine Speicherordnung sicher zu stellen, ohne Befehlsoperation unnötig erneut auszugeben. Zum Beispiel kann die ROP Steuerschaltung(i) **86A** den Treffer durch die Sondierungsadresse aufzeichnen. Falls eine ältere Lade-Speicheroperation nachfolgend aus dem Ablaufplaner zurückgezogen wird, kann dann die ROP Steuerschaltung(i) **86A** den Zustand der Ausführung auf nicht ausgeführt ändern. Weitere Alternativen sind ebenfalls möglich.

[0112] Es wird nun auf **Fig.** 8 Bezug genommen, in der eine beispielhafte Zustandsmaschine gezeigt ist, welche von einem Ausführungsbeispiel der ROP Steuerschaltung(i) **86A** verwendet werden kann. Weitere Steuerschaltungen können ähnliche Zustandsmaschinen verwenden. Weitere Ausführungsbeispiele sind möglich und werden betrachtet. In dem Ausführungsbeispiel von **Fig.** 8 enthält die Zustandsmaschine einen Ungültig Zustand **110**, einen Blockiert Zustand **112**, einen Anforderung Zustand **114**, einen Ausführung Variable (ExecV) Zustand **118**, einen Exec6 Zustand **120**, einen Exec5 Zustand **122**, einen Exec4 Zustand **124**, einen Exec3 Zustand **126**, einen Exec2 Zustand **128**, einen Exec1 Zustand **130** und einen Abgeschlossen Zustand **132**.

[0113] Die Zustandsmaschine fängt in dem Ungültig Zustand **110** an, wenn der entsprechende Eintrag keine Befehlsoperation speichert. In Reaktion auf das Schreiben einer Befehlsoperation in den entsprechenden Eintrag geht die Zustandsmaschine entweder in den Blockiert Zustand **112** oder in den Anforderung Zustand **114** über. Der Blockiert Zustand **112** wird ausgewählt, wenn das Not_Blocked(i) Signal nicht angelegt ist, und der Anforderung Zustand **114** wird ausgewählt, wenn das Not_Blocked(i) Signal angelegt ist. In anderen Ausführungsbeispielen können Befehlsoperationen mit voreingestellten Warteereignissen in den Ablaufplaner geschrieben werden, welche die Befehlsoperation blockieren, im Ablauf geplant zu werden, sogar wenn alle Abhängigkeiten befriedigt sind (auf ähnliche Weise zu den Ereignissen, die ein erneutes Planen des Ablaufs verhindern nachdem eine Befehlsoperation in den nicht ausgeführt Zustand zurück gebracht worden ist). Derartige Befehlsoperationen können einen Übergang in den Blockiert Zustand **112** veranlassen sogar wenn das Not_Blocked(i) Signal angelegt ist.

[0114] Die Zustandsmaschine verbleibt in dem Blockiert Zustand **112** bis die Befehlsoperation nicht mehr blockiert wird. Während der Übergang von dem Ungültig Zustand **110** in den Blockiert Zustand **112** oder in den Anforderung Zustand **114** in dem vorliegenden Ausführungsbeispiel auf dem Not_Blocked(i) Signal basieren kann, berücksichtigt der Übergang von dem Blockiert Zustand **112** in den Anforderung Zustand **114** die Auswirkungen von Situationen des erneuten Versuchens, die angeben, dass ein nachfolgendes Ereignis auftritt bevor die Befehlsoperation auswählbar ist für ein erneutes Planen des Ablaufs. Das Kasten **134** in der **Fig.** 8 enthält eine Gleichung für den Ausdruck für den blockierten Übergang, der an den Pfeilen in **Fig.** 8 für das oben beschriebene Ausführungsbeispiel verwendet wird. Genauer gesagt ist eine Befehlsoperation blockiert, wenn das Not_Blocked(i) Signal nicht angelegt ist oder eine vorherige Ausgabe in der Feststellung endete, dass die Befehlsoperation nicht spekulativ (blocked_non_spec) auszuführen ist und immer noch spekulativ ist, oder eine vorherige Ausgabe in einem Fehltreffer im Cachespeicher (blocked_until_fill) endete und die Fülldaten noch nicht zur Verfügung gestellt worden sind. Weitere Ausführungsbeispiele können zusätzliche Ereignisse enthalten, die falls gewünscht ein erneutes Planen des Ablaufs blockieren. Sobald die Befehlsoperation nicht mehr blockiert ist geht die Zustandsmaschine von dem Blockiert Zustand **112** in den Anforderung Zustand **114** über.

[0115] Während die Zustandsmaschine in dem Anforderung Zustand **114** ist, legt die ROP Steuerschaltung(i) **86A** das request(i) Signal an. Falls die Befehlsoperation während des Anforderung Zustands **114** wieder blockiert wird, geht die Zustandsmaschine in den Blockiert Zustand **112** über. Die Zustandsmaschine geht von dem Anforderung Zustand **114** in einen der Zustände **118**–**128** (basierend auf der Latenz der Befehlsoperation) in Reaktion auf ein Anlegen des pick(i) Signals über. Der Zustand, in den in Reaktion auf das pick(i) Signal übergegangen wurde, kann in einem Ausführungsbeispiel der Stufe Lesen Ablaufplanung der Pipeline von **Fig.** 2 entsprechen.

[0116] Das vorliegende Ausführungsbeispiel unterstützt Latenzen von zwei bis sechs Taktzyklen und eine variable Latenz größer als sechs Taktzyklen. Die Zustandsmaschine verbleibt in dem ExecV Zustand **118** bis das almost done Signal von den Ausführungskernen **40A**–**40B** angelegt wird und dann in den Exec6 Zustand **120**. Jeder der Exec6 Zustand **120** bis Exec2 Zustand **128** geht in den nächst niedrigeren Zustand in der Latenzkette über, wenn die Befehlsoperation nicht rückgängig gemacht ist, wie in **Fig.** 8 gezeigt. Von dem Exec1 Zustand **130** geht die Zustandsmaschine in den Abgeschlossen Zustand **132** über, wenn die Befehlsoperation nicht rückgängig gemacht ist. Schließlich geht die Zustandsmaschine von dem Abgeschlossen Zustand **132** in den Ungültig Zustand **110** über, wenn die Befehlsoperation nicht vor dem Zurückziehen rückgängig gemacht ist.

[0117] Zur Übersichtlichkeit der Zeichnung ist das pick(i) Signal gezeigt, wie es zu einem Auswahlknoten **116**

geht, von dem einer der Zustände **118**–**128** betreten wird. Der Auswahlknoten **116** wird nur verwendet, um die Unordnung in der Zeichnung zu reduzieren und ist nicht beabsichtigt, einen separaten Zustand darzustellen.

[0118] In dem vorliegenden Ausführungsbeispiel. ist die Latenz der Befehlsoperation für Zwecke der Zustandsmaschine von **Fig.** 8 die Anzahl der Taktzyklen bevor die Befehlsoperation die Abhängigkeiten von dieser Befehlsoperation befriedigt hat. Diese Latenz kann auslaufen, bevor die Befehlsoperation Informationen zu dem Zustand der Ausführung zurück gibt (zum Beispiel ob die Befehlsoperation eine Ausnahme erfährt oder nicht). Jedoch nimmt die Zustandsmaschine Vorteil von der Verzögerung der Pipeline zwischen einer Befehlsoperation, die im Ablauf geplant wird, und dieser Befehlsoperation, welche Operanden von den Registerdateien **38A**–**38B** liest, um anzuzeigen, dass Abhängigkeiten befriedigt sind bevor die Abhängigkeiten tatsächlich physikalisch mittels einer Aktualisierung der Registerdateien befriedigt sind. Entsprechend wird das Block(i) Signal zurück genommen, wenn die Befehlsoperation in dem vorliegenden Ausführungsbeispiel den Exec2 Zustand **128** erreicht und verbleibt nicht angelegt, wenn die Zustandsmaschine in dem Exec1 Zustand **130**, dem Abgeschlossen Zustand **132** oder dem Ungültig Zustand **134** ist (siehe Kasten **134**). Das Block(i) Signal wird für andere Zustände angelegt.

[0119] An jedem Punkt nach der Ablaufplanung (pick(i) ist angelegt) kann die Befehlsoperation rückgängig gemacht werden und kehrt in den nicht ausgeführt Zustand zurück. Diese Operation ist in **Fig.** 8 dargestellt durch jeden der Zustände **118**–**132**, der einen Übergang basierend auf einer "undo" Gleichung (Kasten **134**) zu einem zentralen Punkt **136** hat, von dem ein Übergang entweder in den Blockiert Zustand **112** oder den Anforderung Zustand **114** ausgeführt wird basierend auf der in dem Kasten **134** dargestellten Blockierungsgleichung. Der zentrale Punkt **136** wird nur verwendet, um die Unordnung in der Zeichnung zu reduzieren und ist nicht beabsichtigt, einen separaten Zustand darzustellen. Für jeden der Zustände, der einen Übergang zu dem zentralen punkt **136** zeigt, wird ein Übergang in den Blockiert Zustand **112** ausgeführt, wenn die Rückgängig-Gleichung wahr ist und die Blockierungsgleichung wahr ist, und ein Übergang in den Anforderung Zustand **114** wird ausgeführt, wenn die Rückgängig-Gleichung wahr ist und die Blockierungsgleichung falsch ist.

[0120] In dem vorliegenden Ausführungsbeispiel wird eine Befehlsoperation "rückgängig gemacht" (das heißt kehrt in einem Zustand der Ausführung von nicht ausgeführt) zurück, wenn die Befehlsoperation direkt erneut versucht wird oder wenn das Not_Blocked(i) Signal zurück genommen wird. Die Rückgängig-Gleichung in dem Kasten **134** stellt die Bedingung für das erneute Versuchen als einen retry_this_op Wert dar, um anzuzeigen, dass die Befehlsoperation in dem Eintrag (i) zurückgezogen wurde. Ein Kasten **138** ist ferner gezeigt, der den retry_this op Wert als eine Gleichung darstellt, die wahr sein kann, wenn das retry_PA(i) Signal oder das retry_stq(i) Signal angelegt ist, oder wenn die Befehlsoperation während der Ausführung erneut versucht wird (zum Beispiel das erneuter Versuch Signal von der Lade/Speicher-Einheit **42**). Die retry this op Gleichung stellt ferner die Abtastung des erneuter Versuch Signals dar, wenn die Befehlsoperation in dem Exec1 Zustand **130** ist. In dem vorliegenden Ausführungsbeispiel werden Situationen des erneuten Versuchs von der Lade/Speicher-Einheit **42** gemeldet, wenn die entsprechende Befehlsoperation in dem Exec1 Zustand **130** ist. Weitere Ausführungsbeispiele können den Zustand an verschiedenen Punkten während der Ausführung der Befehlsoperation melden, in Übereinstimmung mit den Wünschen des Entwicklers.

[0121] Wie zuvor erwähnt kann der Zustand der Ausführung einer Befehlsoperation weithin nicht ausgeführt, ausführend und abgeschlossen Zustände umfassen.

[0122] Für das Ausführungsbeispiel von **Fig.** 8 kann der nicht ausgeführt Zustand den Blockiert Zustand **112** oder den Anforderung Zustand **114** aufweisen. Der ausführende Zustand kann die Ausführung Zustände **118**–**130** umfassen. Der abgeschlossen Zustand kann den Abgeschlossen Zustand **132** umfassen. Es ist zu bemerken, dass die Anzahl der Ausführung Zustände **118**–**130** von der Implementierung abhängig ist und je nach Wahl des Entwicklers variiert werden kann. Des weiteren kann der Punkt in der Ausführung der Befehlsoperation, an dem Abhängigkeiten befriedigt werden je nach Wahl des Entwicklers variiert werden. Die Variation kann teilweise auf der Anzahl der Stufen der Pipeline zwischen der Stufe, in der die abhängige Befehlsoperation für den Ablauf geplant ist, und einer bestimmten Stufe, in der die Befriedigung der Abhängigkeiten, wie Operanden- oder Ordnungsabhängigkeiten, wie benötigt befriedigt werden, basieren. In dem vorliegenden Ausführungsbeispiel ist die bestimmte Stufe die Stufe Lesen Registerdatei.

[0123] Nun Bezug nehmend auf **Fig.** 9 ist ein Register **140** gezeigt, das von der ROP Steuerschaltung(i) **86A** verwendet werden kann, um Zustände der Zustandsmaschine von **Fig.** 8 und weitere Zustände, wie sie gewünscht sein könnten, zu speichern. Weitere Ausführungsbeispiel sind möglich und werden betrachtet. In dem Ausführungsbeispiel von **Fig.** 9 kann das Register **140** einen Zustand **142**, eine blocked_non spec Angabe **144**, eine blocked_until_fill Angabe **146** und weitere Informationen **148** speichern.

[0124] Der Zustand **142** speichert den derzeitigen Zustand der in **Fig.** 8 dargestellten Zustandsmaschine. Die Zustände können auf jegliche geeignete Weise in dem Zustand **142** kodiert sein. Das Register **142** wird jeden Taktzyklus in Übereinstimmung mit den Übergängen der Zustände, wie in der **Fig.** 8 dargestellt, aktualisiert.

[0125] Die blocked_non spec Angabe **144** kann gesetzt werden, um ein Blockieren in Reaktion auf den Empfang des erneuter Versuch Signals von der Lade/Speicher-Einheit **42** während der Ausführung der Befehlsoperation anzuzeigen, wenn der erneute Versuchtyp anzeigt, dass die Befehlsoperation nicht spekulativ auszu-

führen ist. Die blocked_non spec Angabe **144** kann in der in dem Kasten **134** in **Fig.** 8 gezeigten Blockierungsgleichung verwendet werden. Insbesondere ist, während die blocked_non spec Angabe **144** die Blockierung anzeigt, die Befehlsoperation an der Anforderung einer Ablaufplanung gehindert bis die Befehlsoperation nicht spekulativ wird. In Reaktion darauf, dass die Befehlsoperation nicht spekulativ wird, kann die blocked_non spec Angabe gesetzt werden, um nicht blockiert anzuzeigen und die Befehlsoperation kann für den Ablauf geplant werden. In einem bestimmten Ausführungsbeispiel wird die Befehlsoperation nicht spekulativ, wenn jede ältere Befehlsoper ration in dem Ablaufplaner **36** einen Zustand der Ausführung von abgeschlossen hat.

[0126] Die blocked_until fill Angabe **146** kann gesetzt werden, um ein Blockieren in Reaktion auf den Empfang des erneuter Versuch Signals von der Lade/Speicher-Einheit **42** während der Ausführung der Befehlsoperation anzuzeigen, wenn der erneute Versuchtyp anzeigt, dass die Befehlsoperation in dem D-Cachespeicher **44** fehl trifft. Die blocked_until fill Angabe **146** kann in der in dem Kasten **134** in **Fig.** 8 gezeigten Blockierungsgleichung verwendet werden. Insbesondere ist, während die blocked_until fill Angabe **146** die Blockierung anzeigt, die Befehlsoperation an der Anforderung einer Ablaufplanung gehindert bis die entsprechenden Fülldaten zur Verfügung gestellt werden. In Reaktion darauf, dass die als zur Verfügung gestellt angezeigt werden, kann die blocked_until_fill Angabe gesetzt werden, um nicht blockiert anzuzeigen und die Befehlsoperation kann für den Ablauf geplant werden.

[0127] Weitere Informationen können wie gewünscht in dem Weitere Informationen Feld **148** aufgezeichnet werden. Zum Beispiel können gewisse Ausführungsbeispiele die Zurückziehung einer Speicheroperation verhindern bis eine SMC Überprüfung durchgeführt wird. Das Weitere Informationen Feld **148** kann das Erfordernis auf die SMC Überprüfung zu warten aufzeichnen und kann die Beendigung der SMC Überprüfung aufzeichnen. Jegliche andere Information kann aufgezeichnet werden. Des weiteren werden Ausführungsbeispiele, bei denen keine weitere Information aufgezeichnet wird, betrachtet werden.

[0128] Nun wird auf **Fig.** 10 Bezug genommen, in der ein Zeitablaufdiagramm gezeigt ist, das ein Beispiel des Rückgängigmachens einer Abhängigkeitskette in Übereinstimmung mit einem Ausführungsbeispiel des Ablaufplaners **36** darstellt. Taktzyklen werden von vertikalen punktierten Linien unterteilt, mit einem Identifizieren für jeden Taktzyklus an der Spitze zwischen den vertikalen punktierten Linien, die diesen Taktzyklus abgrenzen. Zustände für jede der Befehlsoperationen (wie von der ROP Steuerschaltung **86** aufgezeichnet) sind ebenfalls in **Fig.** 10 gezeigt (angrenzend zu dem Wort "State" und die R#s der entsprechenden Befehlsoperation in Klammern), wobei "done" den Abgeschlossen Zustand **132** anzeigt, und "blkd" den Blockiert Zustand **112** anzeigt. Die **Fig.** 10 umfasst einen Kasten **150**, der zwei Abhängigkeitsketten darstellt. Die erste Abhängigkeitskette beginnt mit einer Befehlsoperation I0, einer R# von **10** zugewiesen, und enthält ferner die Befehlsoperationen I1, I2 und I3. die Befehlsoperation I1 ist abhängig von I0 und hat einen R# von **15**. Die Befehlsoperation I2 ist abhängig von I1 und hat einen R# von **23**. Die Befehlsoperation I3 ist abhängig von I2 und hat einen R# von **34**. Die Befehlsoperation I4 ist in einer zweiten Abhängigkeitskette eingeleitet von I0 und ist somit abhängig von I0. Die Befehlsoperation I4 hat einen R# von 45. I1 und I4 sind direkt abhängig von I0, während I2 und I3 indirekt abhängig von I0 sind. Die Block und Not_Blocked Signale für jede Befehlsoperation sind in **Fig.** 10 dargestellt (wobei die R# der Befehlsoperation in Klammern gezeigt ist). Gewisse Ereignisse, welche andere Ereignisse veranlassen, sind durch Pfeile von dem Ereignis zu dem sich ergebenden Ereignis dargestellt. Zum Beispiel verursacht die Rücknahme von Not_Blocked(**10**) eine Änderung von State(**10**) zu blockiert, dargestellt durch einen Pfeil von der Rücknahme von Not_Blocked(**10**) zu dem blockierten Zustand von State(**10**).

[0129] Während des Taktzyklus clk0 ist jede der Befehlsoperationen in dem abgeschlossen Zustand. Entsprechend sind die entsprechenden Block Signale nicht angelegt und die Not_Blocked Signale sind angelegt. Während des Taktzyklus clk1 wird das Not_Blocked(**10**) Signal zurück genommen (wegen einer oder mehrerer Befehlsoperationen, von denen die Rückgängigmachung von I0 abhängt). In Reaktion auf die Rücknahme von Not_Blocked(**10**) kehrt die Zustandsmaschine für I0 (State(**10**)) in den blockierten Zustand zurück und somit wird das Block(**10**) Signal in dem Taktzyklus clk2 erneut angelegt. In Reaktion auf das Anlegen von Block(**10**) und die aufgezeichnete Abhängigkeit von I1 und I4 von I0, werden die Not_Blocked(**15**) und Not_Blocked(**45**) Signale zurück genommen (Taktzyklus clk2). Die Rücknahme der Not_Blocked(**15**) und Not_Blocked(**45**) Signale führen wiederum zu dem rückgängig machen von I1 und I4 (State(**15**) und State(**45**) wechseln in den blockierten Zustand in dem Taktzyklus clk3). Nachfolgend werden I2 und I3 rückgängig gemacht wegen ihrer direkten Abhängigkeiten von I1 beziehungsweise I2 und damit wegen ihrer indirekten Abhängigkeit von I0. an dem Ende des Taktzyklus clk5 sind die Abhängigkeitsketten in dem beispielhaften Beispiel rückgängig gemacht und die Zustände der Ausführung entsprechend jeder Befehlsoperation (I0 bis I4) sind in dem nicht ausgeführt Zustand. Nachfolgend können die Befehlsoperationen Befriedigung ihrer Abhängigkeiten erfahren und können wiederum erneut ausgegeben werden, wenn die Befehlsoperationen in der Abhängigkeitskette erneut ausgeben und die Abhängigkeiten von anderen Befehlsoperationen in den Abhängigkeitsketten erfüllen.

[0130] Es ist zu bemerken, dass während die Block und Not_Blocked Signale in **Fig.** 10 gezeigt sind (und in den **Fig.** 11, 12 und 15 unten) als angelegt oder nicht angelegt während eines bestimmten Taktzyklus, die Block Signale während eines ersten Teils des Taktzyklus inaktiv sein können, um es den Not_Blocked Signalen

zu erlauben, vorgeladen zu werden, und dann können die Block Signale während des zweiten Teils des Taktzyklus pulsieren (und Not_Blocked Signale können entladen werden oder vorgeladen bleiben, in Übereinstimmung mit den aufgezeichneten Abhängigkeiten). Des weiteren stellen die Zeitablaufdiagramme der **Fig.** 10, 11, 12 und 15 den Übergang der Not_Blocked Signale basierend auf dem Übergang der dargestellten Block Signale dar. Entsprechend stellen die Beispie lein Beispiel dar, in dem die Abhängigkeiten der dargestellten Abhängigkeitsketten die letzten zu befriedigenden Abhängigkeiten für jede Befehlsoperation in der Abhängigkeitskette sind. Falls weitere Abhängigkeiten unbefriedigt bleieben würden die Not_Blocked Signale nicht angelegt bleiben bis zur Erfüllung von diesen anderen Abhängigkeiten. Auf ähnliche Weise stellen die Zeitablaufdiagramme für die Übersichtlichkeit der Zeichnungen Befehlsoperationen dar, die in Reaktion auf eine Anforderung unmittelbar für den Ablauf geplant werden. Jedoch kann die Ablaufplanung für einen oder mehrere Taktzyklen verzögert werden, falls jüngere Befehlsoperationen des gleichen Typs die Ablaufplanung anfordern.

[0131] Es wird nun auf **Fig.** 11 Bezug genommen, in der ein Zeitablaufdiagramm die Ausgabe und die erneute Ausgabe von beispielhaften Befehlsoperationen in einer Abhängigkeitskette darstellt, wobei die erneute Ausgabe wegen eines erneuten Versuchs der ersten Befehlsoperation in der Abhängigkeitskette auftritt. Taktzyklen werden von vertikalen punktierten Linien unterteilt, mit einem Identifizieren für jeden Taktzyklus an der Spitze zwischen den vertikalen punktierten Linien, die diesen Taktzyklus abgrenzen. Ein Kasten **152** stellt die beispielhafte Abhängigkeitskette dar, welche die Befehle I0 bis I2 von dem Beispiel aus **Fig.** 10 sind. Die Block und Not_Blocked Signale für jede Befehlsoperation sind dargestellt, ebenso wie die Zustände von jeder Befehlsoperation (wie von der ROP Steuerschaltung **86** aufgezeichnet), ähnlich wie bei dem Beispiel von **Fig.** 10. Die in der **Fig.** 11 dargestellten Zustände umfassen blockierte und abgeschlossene Zustände, dargestellt durch "blkd" und "done" in der **Fig.** 11 ähnlich wie in **Fig.** 10. Ebenso sind der Anforderung Zustand **114**, der Exec2 Zustand **128** und der Exec1 Zustand **130** als "rgst", "ex2" beziehungsweise "ex1" dargestellt. Wieder ähnlich zu der **Fig.** 10 sind gewisse Ereignisse, die andere Ereignisse veranlassen als Pfeile von den verursachenden Ereignissen zu den sich ergebenden Ereignissen dargestellt. In diesem Beispiel sind beide Befehlsoperationen I0 und I1 von einer Latenz von 2.

[0132] Der Taktzyklus clk0 stellt jede der Befehlsoperationen I0–I2 in einem blockierten Zustand dar, wartend auf die Erfüllung von Abhängigkeiten bevor sie für die Ausgabe auswählbar werden. Jedes der Not_Blocked Signale ist nicht angelegt und jedes der Block Signale ist angelegt. Während des Taktzyklus clk1 wird das Not_Blocked(**10**) Signal angelegt. In Reaktion auf das Anlegen von Not_Blocked(**10**) wechselt State(**10**) in den Anforderung Zustand während des Taktzyklus clk2. I0 wird für die Ausgabe ausgewählt und damit geht Block(**10**) in den Exec2 Zustand in dem Taktzyklus clk3 über. In dem Taktzyklus clk4 geht State(**10**) in die Exec1 Zustände über.

[0133] In Reaktion auf den Exec2 Zustand von State(**10**), wird während des Taktzyklus clk4 Block(**10**) zurück genommen (was wiederum dazu führt, dass Not_Blocked(**15**) angelegt wird). State(**15**) geht in dem Taktzyklus clk4 in Reaktion auf das Anlegen von Not_Blocked(**15**) in den Anforderung Zustand über und in dem Taktzyklus clk5 in Reaktion auf ausgewählt zu werden in den Exec2 Zustand.

[0134] Während des Exec1 Zustands von State(**10**) (Taktzyklus clk4) detektiert die ROP Steuerschaltung **86** einen erneuten Versuch von I0 (dargestellt in **Fig.** 10 mittels des retry(R#10) Signals). Der erneute Versuch veranlasst ein rückgängig machen von I0 und damit geht State(**10**) in dem Taktzyklus clk5 in einen nicht ausgeführt Zustand über. Genauer gesagt geht State(**10**) in den Anforderung Zustand über, weil das Not_Blocked(**10**) Signal während des Taktzyklus clk4 angelegt wird. In Reaktion auf State(**10**), das in einen nicht ausgeführt Zustand zurück kehrt, wird das Block(**10**) Signal erneut angelegt (und damit wird Not_Blocked(**15**) zurück genommen. Die Rücknahme von Not_Blocked(**15**)führt zu einer Rückkehr von State(**15**) in einen nicht ausgeführten Zustand (Taktzyklus clk6).

[0135] Der erneute Versuch von I0 in diesem Beispiel ist von einem Typ des erneuten Versuchs, der eine unmittelbare erneute Ausgabe von I0 erlaubt. Entsprechend ist State(**10**) im Taktzyklus clk5 in dem Anforderung Zustand. I0 wird für die Ausführung ausgewählt und daher geht State(**10**) in den Taktzyklen clk6, clk7 beziehungsweise clk8 in die Exec2, Exec1 und abgeschlossen Zustände über. Während der erneuten Ausführung von I0 tritt ein erneuter Versuch nicht auf. Es ist jedoch zu bemerken, dass erneute Versuche mehrere Male auftreten können bevor eine Befehlsoperation erfolgreich abschließt.

[0136] Sobald State(**10**) während der erneuten Ausführung von I0 den Exec2 Zustand erreicht (Taktzyklus clk6) wird das Block(**10**) Signal zurück genommen und das Not_Blocked(**15**) Signal wird angelegt. In Reaktion auf das Anlegen des Not_Blocked(**15**) Signals geht State(**15**) in den Anforderung Zustand über (Taktzyklus clk7) und nachfolgend in den Exec2 Zustand in Reaktion darauf, zur Ausgabe ausgewählt zu werden (Taktzyklus clk8). State(**15**) geht in den Taktzyklen clk9 beziehungsweise clk10 in die Exec1 und abgeschlossen Zustände über.

[0137] In Reaktion auf das Erreichen von dem Exec2 Zustand durch State(**15**) (Taktzyklus clk8) wird das Block(**15**) Signal zurück genommen. Das Not_Blocked(**23**) Signal wird während Taktzyklus clk8 in Reaktion auf die Rücknahme von Block(**15**) angelegt und damit geht State(**23**) in den Anforderung Zustand während Taktzyklus clk9 zurück. Die Ausgabe von I2 kann während eines späteren Taktzyklus geschehen (nicht gezeigt).

[0138] Es wird nun auf **Fig.** 12 Bezug genommen, in der ein Zeitablaufdiagramm eine erneute Ausgabe von einer Befehlsoperationen darstellt, mit einem Grund für eine erneute Ausgabe, der das Auftreten eines nachfolgendes Ereignisses erfordert, bevor die erneute Ablaufplanung der Befehlsoperation durchgeführt wird. Genauer gesagt stellt **Fig.** 12 einen erneuten Versuch einer Befehlsoperation dar, die nicht spekulativ auszuführen ist. Taktzyklen werden von vertikalen punktierten Linien unterteilt, mit einem Identifizierer für jeden Taktzyklus an der Spitze zwischen den vertikalen punktierten Linien, die diesen Taktzyklus abgrenzen. Ein Kasten **152** stellt die beispielhafte Abhängigkeitskette dar, welche die gleichen Befehlsoperationen I0 bis I2 und Abhängigkeiten von dem Beispiel aus **Fig.** 11 sind. Die Block und Not_Blocked Signale für jede Befehlsoperation sind dargestellt, ebenso wie die Zustände von jeder Befehlsoperation (wie von der ROP Steuerschaltung **86** aufgezeichnet), ähnlich wie bei dem Beispiel von **Fig.** 11. Die in der **Fig.** 12 dargestellten Zustände sind ähnlich wie in **Fig.** 11 dargestellt. Wieder ähnlich zu der **Fig.** 11 sind gewisse Ereignisse, die andere Ereignisse veranlassen als Pfeile von dem Ereignis zu dem sich ergebenden Ereignis dargestellt. In diesem Beispiel ist die Befehlsoperation I0 von einer Latenz von 2.

[0139] Die Taktzyklen clk0 bis clk6 sind ähnlich. zu den entsprechenden Taktzyklen clk0 bis clk6 aus **Fig.** 11 mit der Ausnahme, dass das erneute Versuchen von I0 in dem Taktzyklus clk4 als ein erneutes Versuchen angezeigt wird, weil I0 nicht spekulativ auszuführen ist. Daher wurde I0 spekulativ ausgegeben und seine nicht spekulative Natur wurde nach der Ausgabe detektiert. Der Ablaufplaner **36** löst diese Situation durch rückgängig machen von I0, um nicht spekulativ zu werden bevor eine erneute Ausgabe erlaubt wird. Genauer gesagt kann die ROP Steuerschaltung **86** in Reaktion auf den erneuten Versuchstyp als "wartend auf nicht spekulativ" seiend die Blocked_non spec Angabe entsprechend zu I0 setzen. Damit wird die ROP Steuerschaltung **86** daran gehindert, eine Ablaufplanung von I0 anzufordern bevor I0 nicht spekulativ wird, sogar obwohl das Not_Blocked(**10**) Signal angelegt ist. Einige Anzahlen von Taktzyklen können ablaufen und dann kann die ROP Steuerschaltung **86** feststellen, dass I0 nicht spekulativ ist (zum Beispiel Taktzyklus clkn in **Fig.** 12, dargestellt durch das Anlegen von dem non-spec(R#10) Signal in **Fig.** 12). Wie zuvor erwähnt kann eine Befehlsoperation nicht spekulativ sein, wenn in Übereinstimmung mit einem Ausführungsbeispiel jede vorherige Befehlsoperation (in der Reihenfolge des Programms) innerhalb des Ablaufplaners **36** in dem abgeschlossen Zustand ist.

[0140] In Reaktion darauf, dass I0 nicht spekulativ wird geht State(**10**) in den Anforderung Zustand über (Taktzyklus clkn + 1). Nachfolgend wird I0 für die Ausgabe ausgewählt (Exec2 Zustand von State(**10**) in dem Taktzyklus clkn + 2) und führt aus. Die abhängigen Befehlsoperationen I1 und I2 können nachfolgend ausgeführt werden, sobald ihre Abhängigkeiten von I0 erfüllt sind.

[0141] Es wird nun auf **Fig.** 13 Bezug genommen, in der ein beispielhafter Eintrag **160** des Puffers für physikalische Adressen gezeigt ist, der von einem Ausführungsbeispiel des Puffers für physikalische Adressen **90** verwendet werden kann. Des weiteren ist eine beispielhafte kombinatorische Logikschaltung **172** gezeigt. Die Schaltung **172** kann verwendet werden, um die fill_hit(i) und retry_PA(i) Signale zu erzeugen. Weitere Ausführungsbeispiele sind möglich und werden betrachtet. Genauer gesagt kann jede geeignete kombinatorische Logikschaltung verwendet werden, einschließlich jeglicher Boolean Äquivalente der in **Fig.** 13 gezeigten Logik. Des weiteren kann die in dem Eintrag **160** gespeicherte Information in Form und Inhalt geändert werden, in Übereinstimmung mit den Wünschen des Entwicklers. In dem Ausführungsbeispiel von **Fig.** 13 umfasst der Eintrag **160** ein Gültig Bit **162**, ein erstes Laden PA Feld **164**, ein erstes Bytemaskierungsfeld **166**, ein zweites Laden PA Feld **168** und ein zweites Bytemaskierungsfeld **170**.

[0142] Im Allgemeinen wird, falls die Befehlsoperation in dem Eintrag des Befehlspuffers, dem der Eintrag **160** zugeordnet ist, eine Lade-Speicheroperation ist, der Eintrag **160** mit der physikalischen Adressinformation des Speicheroperanden aktualisiert, auf den von der Lade-Speicheroperation zugegriffen wird (der "Lade Speicheroperand") und das Gültig Bit **162** wird gesetzt. In dem vorliegenden Ausführungsbeispiel wird die Information von der physikalischen Adresse des Quadwortes einschließlich des ersten Bytes des Lade Speicheroperanden (erstes Laden PA Feld **164**) und einer Bytemaskierung repräsentiert, die anzeigt, welche Bytes in dem Quadwort Teil des Lade Speicheroperanden sind (erstes Bytemaskierungsfeld **166**). Die Bytemaskierung weist ein Bit für jedes Byte in dem Quadwort auf. Falls das Bit gesetzt ist, ist das entsprechende Byte Teil des Lade Speicheroperanden. Falls das Bit nicht gesetzt ist, ist das entsprechende Byte kein Teil des Lade Speicheroperanden.

[0143] Lade Speicheroperanden können beliebig in dem Speicher ausgerichtet sein. Entsprechend können ein oder mehrere Bytes des Lade Speicheroperanden in einem Quadwort sein und ein oder mehrere Bytes des Lade Speicheroperanden können in dem nächsten darauf folgenden Quadwort sein. Daher stellt der Eintrag **160** das zweite Laden PA Feld **168** und das zweite Bytemaskierungsfeld **170** zur Vertügung. Das zweite Laden PA Feld **168** speichert die physikalische Adresse des nächsten darauf folgenden Quadworts in das erste Laden PA Feld **168**. In dem vorliegenden Ausführungsbeispiel wird der in der Seite befindliche Teil der physikalischen Adresse in dem zweite Laden PA Feld **168** gespeichert. Weil Lade-Speicheroperationen, die eine Seitengrenze überschreiten in dem vorliegenden Ausführungsbeispiel nicht spekulativ sind, ist es ausreichend, lediglich den in der Seite befindlichen Teil des nächsten darauf folgenden Quadworts zu speichern (weil die Lade-Speicheroperation, wenn eine Seite überschritten wird, nicht spekulativ erneut ausgegeben wird und daher keine älte-

ren Speichervorgänge nachfolgend zu der erneuten Ausgabe der Lade-Speicheroperation ausgegeben werden). Weitere Ausführungsbeispiele können die Gesamtheit des nächsten darauf folgenden Quadworts speichern oder falls gewünscht jeden geeigneten Teil. Des weiteren können andere Ausführungsbeispiele, während das vorliegende Ausführungsbeispiel Adressen mit einer Auflösung von einem Quadwort speichert, jegliche andere geeignete Auflösung verwenden (zum Beispiel Oktawort, Doppelwort, usw.). Das zweite Bytemaskierungsfeld **170**, ähnlich zu dem ersten Bytemaskierungsfeld **166**, zeigt an, welche Bytes in dem nächsten darauf folgenden Quadwort Teil des Lade Speicheroperanden sind.

[0144] Der Ausführungskern **40A** stellt die Speicher physikalische Adresse und die entsprechende Bytemaskierung während der Ausführung von Speicher-Speicheroperationen zur Verfügung. Die Schaltung **172** vergleicht die entsprechenden Teile der Speicher physikalische Adresse mit den in dem ersten Laden PA Feld **164** und in dem zweiten Laden PA Feld **168** gespeicherten Werten. Des weiteren werden entsprechende Speicher Bytemaskierungen zur Verfügung gestellt. Die UND Gatter, welche die Speicher und Lade Bytemaskierungen in der Schaltung **172** empfangen, stellen eine Logik dar, welche feststellt, ob zumindest ein Byte der Lade Bytemaskierung und zumindest ein entsprechendes Bit in der Speicher Bytemaskierung gesetzt sind, was anzeigt, dass zumindest ein Byte des Lade Speicheroperanden von der Lade-Speicheroperation aktualisiert wird. Zum Beispiel kann ein UND Gatter für jedes Bit verwendet werden, deren Ausgänge mit einem ODER verknüpft sind. Falls der Eintrag **160** gültig ist, stimmen die physikalischen Adressteile überein und zumindest ein Byte in dem entsprechenden Quadwort ist Teil des Lade Speicheroperanden und wird von der Speicher-Speicheroperation aktualisiert, dann kann das retry_PA(i) Signal erzeugt werden. Es ist zu bemerken, dass das retry_PA(i) Signal ebenfalls maskiert werden kann, wenn die Speicher-Speicheroperation in der Reihenfolge des Programms nicht vor der Lade-Speicheroperation ist (nicht gezeigt in **Fig.** 3).

[0145] Es ist zu bemerken, dass Speicher Speicheroperanden ebenfalls willkürlich im Speicher ausgerichtet sein können. Entsprechend können ein oder mehrere Bytes des Speicher Speicheroperanden in einem Quadwort sein und ein oder mehrere Bytes des Speicher Speicheroperanden können in dem nächsten darauf folgenden Quadwort sein. Daher kann die Speicher PA+1 (ähnlich zu der Lade PA+1) mit den gespeicherten Lade PAs verglichen werden, um eine Speicher PA zu detektieren, die eine Lade PA trifft. Die folgende Formel kann das retry_PA(i) Signal darstellen (in dem die MATCH(A(n : 0),B(n : 0)) Funktion eine binäre 1 zurück gibt, wenn zumindest ein Bit in A(n : 0) gesetzt ist ein entsprechendes Bit in B(n : 0) gesetzt ist):

```
Retry_PA(i) = V & Load_PA(39:12) == Store_PA(39:12) &
                    ((Load_PA(11:3) == Store_PA(11:3) & MATCH(Store_Byte_Mask(7:0),
                    Load_Byte_Mask(7:0))) ||
                    ((Load_PA(11:3)+1 == Store PA(11:3) & MATCH(Store_Byte_Mask(6:0),
                    Load_Byte_Mask(14:8))) ||
((Load_PA(11:3)      ==      Store_PA(11:3)+1      &
MATCH(Store_Byte_Mask(14:8),
Load_Byte_Mask(6:0))) ||
((Load_PA(11:3)+1    ==      Store_PA(11:3)+1      &
MATCH(Store_Byte_Mask(14:8),
Load_Byte_Mask(14:8))))
```

[0146] Es ist ferner zu bemerken, dass der letzte der vier Ausdrücke (Load_PA(11 : 3) + 1 und Store_PA(11 : 3) + 1 vergleichend) redundant ist und in dem vorliegenden Ausführungsbeispiel eliminiert werden kann, weil, für einen Speicheroperanden mit einem gültigen Byte in dem nächsten darauf folgenden Quadwort, der Speicheroperand zumindest ein gültiges Byte (Byte **7**, repräsentiert von dem Maskierungsbit **7**) in dem ersten Quadwort hat. Daher wird eine Übereinstimmung in dem vierten Ausdruck nur angetroffen, wenn auch eine Übereinstimmung in dem ersten Ausdruck (Load_PA(11 : 3) und Store_PA(11 : 3) vergleichend) angetroffen wird.

[0147] Des weiteren wird der Eintrag **160** mit den von der externen Interfaceeinheit **46** zur Verfügung gestellten Füll/Sondierung Adressen verglichen. In dem dargestellten Ausführungsbeispiel wird die für einen Füllvorgang zur Verfügung gestellte Zeile des Cachespeichers dem Puffer für physikalische Adressen 90 zum Vergleich zur Verfügung gestellt. Ein entsprechender Teil des ersten Laden PA Feld **164** und des zweiten Laden PA Feld **168** kann mit der Fülladresse verglichen werden. Falls eine Übereinstimmung detektiert wird, kann das fill_hit(i) Signal angelegt werden. In anderen Ausführungsbeispielen kann die Zeile des Cachespeichers dem D-Cachespeicher **44** als eine Vielzahl von Paketen zur Verfügung gestellt werden. Der Teil der Adresse, der

die Zeile des Cachespeichers identifiziert, und das zur Verfügung gestellte Paket können in diesen Ausführungsbeispielen verglichen werden.

[0148] Es ist ferner zu bemerken, dass das retry_PA(i) Signal maskiert werden kann, wenn die Speicher-Speicheroperation, die dem store_PA entspricht, jünger ist als die Lade-Speicheroperation, die dem Eintrag **160** entspricht.

[0149] Es wird nun auf **Fig.** 14 Bezug genommen, in der ein beispielhafter Eintrag **180** des Puffers für Speichern R# Adressen gezeigt ist, der von einem Ausführungsbeispiel des Speichern R# Puffer **92** verwendet werden kann. Des weiteren ist eine beispielhafte kombinatorische Logikschaltung **190** gezeigt. Die Schaltung **190** kann verwendet werden, um das retry_stq(i) Signal zu erzeugen. Weitere Ausführungsbeispiele sind möglich und werden betrachtet. Genauer gesagt kann jede geeignete kombinatorische Logikschaltung verwendet werden, einschließlich jeglicher Boolean Äquivalente der in **Fig.** 14 gezeigten Logik. Des weiteren kann die in dem Eintrag **180** gespeicherte Information in Form und Inhalt geändert werden, in Übereinstimmung mit den Wünschen des Entwicklers. In dem Ausführungsbeispiel von **Fig.** 14 umfasst der Eintrag **180** ein Gültig Bits **182** und **186** und Speichern R# Felder **184** und **188**.

[0150] Im Allgemeinen wird, falls die Befehlsoperation in dem Eintrag des Befehlspuffers, dem der Eintrag **180** zugeordnet ist, eine Lade-Speicheroperation ist, der Eintrag **180** mit dem Speicher R# eines Speichervorgangs in der Speicher-Warteschlange **70** aktualisiert, der von der Lade-Speicheroperation getroffen wird. Das vorliegende Ausführungsbeispiel stellt die Weiterleitung von bis zu zwei Speicher-Speicheroperationen an eine Lade-Speicheroperation zur Verfügung und daher sind zwei Speichern R# Felder **184** und **188** vorgesehen, um die R# von jedem weiter geleiteten Speichervorgang aufzuzeichnen. Entsprechende Gültig Bits **182** und **186** sind gesetzt, wenn die entsprechenden weiterleitenden Speichervorgänge detektiert werden. Weitere Ausführungsbeispiele könnten lediglich von einem Speichervorgang weiterleiten und der Eintrag **180** könnte lediglich ein Speicher R# aufzeichnen. Noch weitere Ausführungsbeispiele können von mehr als zwei Speichervorgängen weiterleiten und der Eintrag **180** kann eine entsprechende Anzahl von Speicher R#s aufzeichnen.

[0151] Sobald Speicher-Speicheroperationen ausgeführt werden stellt die Lade/Speicher-Einheit **42** die R# der Speicher-Speicheroperation dem Speichern 2# Puffer **92** zur Verfügung. Die R# wird mit den in dem Eintrag **180** gespeicherten R# verglichen und falls eine Übereinstimmung detektiert wird (und das entsprechende Gültig Bit gesetzt ist), legt die Schaltung **190** das retn stq(i) Signal an. Wie oben erwähnt können in einer weiteren Alternative die Nummern der Speicherwarteschlange in dem Puffer **92** gespeichert werden und die Nummern der Speicherwarteschlange können zum Vergleich zur Verfügung gestellt werden.

[0152] Es wird nun auf **Fig.** 15 Bezug genommen, in der ein Zeitablaufdiagramm eine erneute Ausgabe von einer Lade-Speicheroperation über einen Treffer in dem Puffer für physikalische Adressen **90** darstellt. Ein erneuter Versuch einer Lade-Speicheroperation über einen Treffer in dem Speichern R# Puffer **92** kann ähnlich sein. Taktzyklen werden von vertikalen punktierten Linien unterteilt, mit einem Identifizieren für jeden Taktzyklus an der Spitze zwischen den vertikalen punktierten Linien, die diesen Taktzyklus abgrenzen. Ein Kasten **192** stellt die beispielhafte Abhängigkeitskette dar, welche die gleichen Befehlsoperationen I0 bis I2 und Abhängigkeiten von dem Beispiel aus **Fig.** 11 sind (mit der Ausnahme, dass I0 nun eine Lade-Speicheroperation ist). Die Block und Not_Blocked Signale für jede Befehlsoperation sind dargestellt, ebenso wie die Zustände von jeder Befehlsoperation (wie von der ROP Steuerschaltung **86** aufgezeichnet), ähnlich wie bei dem Beispiel von **Fig.** 11. Die in der **Fig.** 12 dargestellten Zustände sind ähnlich wie in **Fig.** 11 dargestellt. Des weiteren sind die Exec4 und Exec3 Zustände als "ex4" beziehungsweise "ex3" dargestellt. Wieder ähnlich zu der **Fig.** 11 sind gewisse Ereignisse, die andere Ereignisse veranlassen als Pfeile von dem Ereignis zu dem sich ergebenden Ereignis dargestellt. In diesem Beispiel ist die Lade-Speicheroperation I0 von einer Latenz von **4**.

[0153] In dem Taktzyklus clk0 ist jede der Befehlsoperationen I0–I2 ausgegeben und ausgeführt worden und sind daher in dem abgeschlossen Zustand. Entsprechende Block Signale sind nicht angelegt und Not_Blocked Signale sind angelegt. Jedoch wird für R#10 ein Treffer in dem Puffer für physikalische Adressen **90** detektiert (retry_PA(**10**) wird während des Taktzyklus clk0 angelegt). In Reaktion geht State(**10**) in dem Taktzyklus clk1 in den Anforderung Zustand über. Des weiteren wird das Block(**10**) Signal angelegt und I1 und I2 werden nach einander während der Taktzyklen clk2 bis clk3 rückgängig gemacht.

[0154] Die Lade-Speicheroperation I0 wird in den Taktzyklen clk2 bis clk6 für die Ausführung ausgewählt und läuft durch die ausführenden Stufen in den abgeschlossen Zustand. In Reaktion darauf, dass I0 in dem Taktzyklus clk4 den Exec2 Zustand erreicht, wird das Block(**10**) Signal zurückgenommen (und damit wird das Not_Blocked(**15**) Signal angelegt. Die Befehlsoperationen I1 und I2 werden damit erneut für den Ablauf geplant und wie in **Fig.** 15 gezeigt erneut ausgegeben.

[0155] **Fig.** 15 stellt dar, dass eine Lade-Speicheroperation vor Speicher-Speicheroperationen ausgegeben und ausgeführt werden kann, von denen die Lade Befehlsoperation abhängt. Nachfolgend können die Speicher-Speicheroperationen ausgegeben werden und die Abhängigkeit detektiert werden. Die Abhängigkeit wird berücksichtigt durch eine erneute Ausgabe der Lade-Speicheroperation (und ihrer Abhängigkeitsketten) durch den Ablaufplaner 36, sobald die Abhängigkeit erkannt wird. Ein ähnliches Zeitablaufdiagramm mit dem retry_stq(**10**) Signal angelegt stellt die Detektierung einer falschen Abhängigkeit einer Lade-Speicheroperation

von einer vorherigen Speicher-Speicheroperation dar, welche nicht korrekt ausgeführt wurde und nachfolgend erneut ausgegeben wurde. Wiederum behandelt der Ablaufplaner **36** die Situation durch eine erneute Ausgabe der Lade-Speicheroperation und ihrer Abhängigkeitsketten. Eine korrekte Operation kann bei minimaler Abnahme der Leistungsfähigkeit zur Verfügung gestellt werden und daher kann eine aggressive spekulative Ausführung durchgeführt werden und eine höhere Leistungsfähigkeit kann erreicht werden.

Computersysteme

[0156] Es wird nun auf **Fig.** 16 Bezug genommen, in der ein Blockdiagramm gezeigt ist, das ein Ausführungsbeispiel eines Computersystems **200** einschließlich des Prozessors **10** darstellt, der über eine Busbrücke **202** mit einer Vielzahl von Systemkomponenten verbunden ist. Weitere Ausführungsbeispiele sind möglich und werden betrachtet. In dem dargestellten System ist ein Hauptspeicher **204** über einen Speicherbus **206** mit der Busbrücke **202** verbunden und eine Grafiksteuerung **208** ist über einen AGP Bus **210** mit der Busbrücke **202** verbunden. Schließlich ist eine Vielzahl von PCI Geräten **212A**–**212B** über einen PCI Bus **214** mit der Busbrücke **202** verbunden. Eine zweite Busbrücke **216** kann ferner vorgesehen sein, um einem oder mehreren EISA oder ISA Geräten **218** über einen EISA/ISA Bus **220** eine elektrische Schnittstelle zu bieten. Der Prozessor **10** ist über einen CPU Bus **224** mit der Busbrücke **202** und mit einem optionalen L2 Cachespeicher **228** verbunden. Zusammen können der CPU Bus **224** und die Schnittstelle zu dem L2 Cachespeicher **228** eine externe Schnittstelle **52** aufweisen.

[0157] Die Busbrücke **202** stellt eine Schnittstelle zwischen dem Prozessor 10, dem Hauptspeicher **204**, der Grafiksteuerung **208** und an den PCI Bus **214** angebrachten Geräten zur Verfügung. Wenn eine Operation von einem der an die Busbrücke **202** angeschlossenen Geräte empfangen wird, identifiziert die Busbrücke **202** das Ziel der Operation (zum Beispiel ein bestimmtes Gerät oder in dem Fall des PCI Busses **214**, dass das Ziel auf dem PCI Bus **214** ist). Die Busbrücke **202** führt die Operation zu dem angezielten Gerät. Die Busbrücke **202** übersetzt im Allgemeinen eine Operation von dem Protokoll, das von dem Quellgerät oder -bus benutzt wird, in das Protokoll, das von dem Zielgerät oder -bus benutzt wird.

[0158] Zusätzlich zur Bereitstellung einer Schnittstelle zu einem ISA/EISA Bus für den PCI Bus **214** kann die zweite Busbrücke **216** wie gewünscht weitere Funktionalität enthalten. Eine Eingangs/Ausgangssteuerung (nicht gezeigt), entweder außerhalb von der oder in die zweite Busbrücke **216** integriert, kann auch in dem Computersystem **200** enthalten sein, um operationelle Unterstützung für eine Tastatur und Maus **222** und für verschiedene serielle und parallele Anschlüsse wie gewünscht zur Verfügung zu stellen. Eine externe Cachespeichereinheit (nicht gezeigt) kann ferner an den CPU Bus **224** zwischen dem Prozessor **10** und der Busbrücke **202** in anderen Ausführungsbeispielen angeschlossen sein. Alternativ kann der externe Cachespeicher an die Busbrücke **202** angeschlossen sein und die Cachespeichersteuerlogik für den externen Cachespeicher kann in der Busbrücke **202** integriert sein. Ein L2 Cachespeicher **228** ist des weiteren in einer hinter dem Prozessor **10** angeordneten Konfiguration gezeigt. Es ist zu bemerken, dass der L2 Cachespeicher **228** getrennt von dem Prozessor **10**, in einem Einsatz (zum Beispiel Slot **1** oder Slot A) für den Prozessor **10** integriert oder sogar integriert auf einem Halbleitersubstrat in dem Prozessor **10** sein kann.

[0159] Der Hauptspeicher **204** ist ein Speicher, in dem Anwendungsprogramme gespeichert werden und von dem der Prozessor **10** hauptsächlich ausführt. Ein geeigneter Hauptspeicher **204** weist DRAM (dynamischen wahlfreien Zugriffsspeicher) auf. Zum Beispiel kann eine Vielzahl von Bänken an SDRAM (Synchrones DRAM) oder an Rambus DRAM (RDRAM) geeignet sein.

[0160] Die PCI Geräte **212A**–**212B** sind beispielhaft für eine Vielzahl von peripheren Geräten, wie zum Beispiel Netzwerkschnittstellenkarten, Videobeschleunigern, Audiokarten, Festplatten oder Diskettenlaufwerken oder Laufwerkssteuerungen, SCSI (kleine Computer Systemschnittstelle) Adaptern oder Telefoniekarten. Auf ähnliche Weise ist das ISA Gerät **218** beispielhaft für verschiedene Typen von peripheren Geräten, wie einem Modem, einer Soundkarte und einer Vielzahl von Daten sammelnden Karten, wie GPIB oder Feldbus Schnittstellenkarten.

[0161] Die Grafiksteuerung **208** ist vorgesehen, um Text und Bilder auf einer Anzeige **256** sichtbar zu machen. Die Grafiksteuerung **208** kann einen typischen Grafikbeschleuniger verwenden, der allgemein im Stand der Technik bekannt ist, um dreidimensionale Datenstrukturen sichtbar zu machen, die effektiv aus und in den Hauptspeicher **204** geschoben werden können. Die Grafiksteuerung **208** kann daher ein Master von dem AGP Bus **210** sein, so dass es Zugriff auf eine Zielschnittstelle innerhalb der Busbrücke **202** anfordern und empfangen kann, um dadurch Zugriff auf den Hauptspeicher **204** zu bekommen. Ein fest zugeordneter Grafikbus erlaubt eine schnelle Erlangung von Daten aus dem Hauptspeicher **204**. Für gewisse Operationen kann die Grafiksteuerung **208** des weiteren konfiguriert sein, um auf dem AGP Bus Transaktionen nach dem PCI Protokoll zu erzeugen. Die AGP Schnittstelle der Busbrücke **202** kann daher Funktionalitäten enthalten, um sowohl Transaktionen nach dem AGP Protokoll als auch Ziel- und Urhebertransaktionen nach dem PCI Protokoll zu unterstützen. Die Anzeige **226** ist jegliche elektronische Anzeige, auf der ein Bild oder ein Text dargestellt werden kann. Eine geeignete Anzeige umfasst eine Kathodenstrahlröhre („CRT"), ein Flüssigkristalldisplay

(„LCD") usw.

[0162] Es ist zu bemerken, dass, während die AGP, PCI und ISA oder EISA Busse in der obigen Beschreibung als Beispiele benutzt worden sind, jede Busarchitektur wie gewünscht ersetzt werden kann. Es ist weiter zu bemerken, dass das Computersystem **200** ein Mehrfachprozessorsystem sein kann, das zusätzliche Prozessoren enthält (zum Beispiel Prozessor **10a**, der als optionale Komponente des Computersystems **200** gezeigt ist). Der Prozessor **10a** kann ähnlich zu dem Prozessor **10** sein. Genauer gesagt kann der Prozessor **10a** eine identische Kopie des Prozessors **10** sein. Der Prozessor **10a** kann über einen unabhängigen Bus (wie in **Fig.** 11 gezeigt) mit der Busbrücke **202** verbunden sein oder kann den CPU Bus **224** mit dem Prozessor **10** teilen. Des weiteren kann der Prozessor **10a** mit einem optionalen L2 Cachespeicher **228a** verbunden sein, der dem L2 Cachespeicher **228** ähnlich ist.

[0163] Es wird nun auf **Fig.** 17 Bezug genommen, in der ein weiteres Ausführungsbeispiel eines Computersystems **300** gezeigt ist. Weitere Ausführungsbeispiele sind möglich und werden betrachtet. In dem Ausführungsbeispiel von **Fig.** 17 enthält das Computersystem **300** mehrere Verarbeitungsknoten **321A**, **312B**, **312C** und **312D**. Jeder Verarbeitungsknoten ist mit einem entsprechenden Speicher **314A**–**314D** über eine Speichersteuerung **316A**–**316D** verbunden, die in jedem entsprechenden Verarbeitungsknoten **312A**–**312D** enthalten ist. Des weiteren enthalten die Verarbeitungsknoten **312A**–**312D** eine Interfacelogik, die zur Kommunikation zwischen den Verarbeitungsknoten **312A**–**312D** verwendet wird. Zum Beispiel enthält der Verarbeitungsknoten **312A** eine Intertacelogik **318A** zum Kommunizieren mit dem Verarbeitungsknoten **312B**, eine Intertacelogik **318B** zum Kommunizieren mit dem Verarbeitungsknoten **312C** und eine dritte Interfacelogik **318C** zum Kommunizieren mit noch einem weiteren Verarbeitungsknoten (nicht gezeigt). Auf ähnliche Weise enthält der Verarbeitungsknoten **312B** die Intertacelogiken **318D**, **318E** und **318F**; der Verarbeitungsknoten **312C** enthält die Intertacelogiken **318G**, **318H** und **318I** und der Verarbeitungsknoten **312D** enthält die Interacelogiken **318J**, **318K** und **318L**. Der Verarbeitungsknoten **312D** ist angeschlossen, um mit einer Vielzahl von Eingangs/Ausgangs Geräten (zum Beispiel Geräte **320A**–**320B** in verketteter Konfiguration) über die Interfacelogik **318L** zu kommunizieren. Weitere Verarbeitungsknoten können mit anderen I/O Geräten in ähnlicher Weise kommunizieren.

[0164] Die Verarbeitungsknoten **312A**–**312D** implementieren eine paketbasierte Verbindung für die Kommunikation zwischen den Verarbeitungsknoten. In dem vorliegenden Ausführungsbeispiel ist die Verbindung als Sätze von unidirektionalen Leitungen verwirklicht (zum Beispiel werden Leitungen **324A** verwendet, um Pakete von dem Verarbeitungsknoten **312A** zu dem Verarbeitungsknoten **312B** zu übermitteln und Leitungen **324B** werden verwendet, um Pakete von dem Verarbeitungsknoten **312B** zu dem Verarbeitungsknoten **312A** zu übermitteln). Weitere Sätze von Leitungen **324C**–**324h** werden wie in **Fig.** 17 dargestellt verwendet, um Pakete zwischen den anderen Verarbeitungsknoten zu übermitteln. Im Allgemeinen kann jeder Satz von Leitungen **324** eine oder mehrere Datenleitungen, eine oder mehrere den Datenleitungen entsprechende Taktleitungen und eine oder mehrere Steuerleitungen zur Anzeige des Typs des beförderten Pakets aufweisen. Die Verbindung kann für die Kommunikation zwischen den Verarbeitungsknoten auf eine mit einem Cachespeicher kohärente Weise betrieben werden oder für die Kommunikation zwischen einem Verarbeitungsknoten und einem I/O Gerät (oder einer Busbrücke zu einem I/O Bus von konventioneller Konstruktion, wie einem PCI Bus oder einem ISA Bus) auf nicht kohärente Weise betrieben werden. Des weiteren kann die Verbindung unter Benutzung einer verketteten Struktur zwischen den I/O Geräten, wie gezeigt, auf nicht kohärente Weise betrieben werden. Es ist zu bemerken, dass ein Paket, das von einem Verarbeitungsknoten zu einem Anderen übermittelt wird, einen oder mehrere andere in der Mitte liegende Knoten passieren kann. Zum Beispiel kann ein von dem Verarbeitungsknoten **312A** an den Verarbeitungsknoten **312D** übermitteltes Paket entweder durch den Verarbeitungsknoten **312B** oder den Verarbeitungsknoten **312C** wie in **Fig.** 17 gezeigt gelangen. Jeder geeignete Algorithmus zur Dirigierung kann verwendet werden. Weitere Ausführungsbeispiele des Computersystems **300** können mehr oder weniger Verarbeitungsknoten als in dem in **Fig.** 17 gezeigten Ausführungsbeispiel enthalten.

[0165] Im Allgemeinen können die Pakete mit einer oder mehr Bitzeiten auf den Leitungen **324** zwischen den Knoten übermittelt werden. Eine Bitzeit kann die steigende oder fallende Flanke des Taktsignals auf den entsprechenden Taktleitungen sein. Die Pakete können Anweisungspakete zum Einleiten von Transaktionen, Sondierungspakete zur Beibehaltung der Kohärenz des Cachespeichers und Antwortpakete sein, die auf Sondierungen und Anweisungen antworten.

[0166] Die Verarbeitungsknoten **312A**–**312D** können, zusätzlich zu einer Speichersteuerung und einer Interfacelogik, einen oder mehrere Prozessoren enthalten. Allgemein gesagt weist ein Verarbeitungsknoten mindestens einen Prozessor auf und kann optional eine Speichersteuerung zur Kommunikation mit einem Speicher und weitere Logik wie gewünscht enthalten. Insbesondere kann ein Verarbeitungsknoten **312A**–**312D** einen Prozessor **10** aufweisen. Die externe Interfaceeinheit **46** kann die Interfacelogik **318** innerhalb des Knotens enthalten, ebenso wie die Speichersteuerung **316**.

[0167] Die Speicher **314A**–**314D** können alle geeigneten Speichergeräte aufweisen. Zum Beispiel kann ein Speicher **314A**–**314D** einen oder mehrere RAMBUS DRAMs (RDRAMs), synchrone DRAMs (SDRAMs), sta-

tische RAMs usw. aufweisen. Der Adressraum des Computersystems **300** ist zwischen den Speichern **314A**–**314D** aufgeteilt. Jeder Verarbeitungsknoten **312A**–**312D** kann eine Speicherabbildung enthalten, die verwendet wird, um festzustellen, welche Adressen auf welche Speicher **314A**–**314D** abgebildet sind, und damit, an welchen Verarbeitungsknoten **312A**–**312D** eine Speicheranforderung für eine bestimmte Adresse dirigiert werden soll. In einem Ausführungsbeispiel ist der Kohärenzpunkt für eine Adresse in dem Computersystem **300** die Speichersteuerung **316A**–**316D**, die mit dem Speicher verbunden ist und den Adressen entsprechende Bytes speichert. Anders gesagt ist die Speichersteuerung **316A**–**316D** verantwortlich für die Sicherstellung, dass jeder Speicherzugriff auf den entsprechenden Speicher **314A**–**314D** auf eine mit dem Cachespeicher kohärente Weise geschieht. Die Speichersteuerungen **316A**–**316D** können eine Steuerschaltung aufweisen, um eine Schnittstelle zu dem Speicher **314A**–**314D** zu bilden. Des weiteren können die Speichersteuerungen **316A**– **316D** Anforderungswarteschlangen für die Aufreihung von Speicheranforderungen enthalten.

[0168] Im Allgemeinen kann die Interfacelogik **318A**–**318L** eine Vielzahl von Puffern zum Empfangen von Paketen von der Verbindung und zum Puffern von auf der Verbindung zu übermittelnden Paketen aufweisen. Das Computersystem **300** kann jeden geeigneten Mechanismus zur Flusssteuerung für die Übermittelung von Paketen verwenden. Zum Beispiel speichert in einem Ausführungs-Beispiel die Interfacelogik **318** einen Zählwert der Anzahl von jedem Typ des Puffers in dem Empfänger an der anderen Seite der Verbindung, mit dem diese Interfacelogik verbunden ist. Die Interfacelogik übermittelt kein Paket, es sei denn die empfangene Interfacelogik hat einen freien Puffer, um das Paket zu speichern. Wenn ein empfangender Puffer durch die Weiterleitung eines Pakets frei wird, übermittelt die empfangende Logik eine Nachricht zu der sendenden Interfacelogik, um anzuzeigen, dass der Puffer leer gemacht worden ist. Ein derartiger Mechanismus kann als ein „Coupon basiertes" System bezeichnet werden.

[0169] Die I/O Geräte **320A**–**320B** können alle geeigneten I/O Geräte sein. Zum Beispiel können die I/O Geräte **320A**–**320B** Netzwerk-Schnittstellenkarten, Videobeschleuniger, Audiokarten, Festplatten oder Diskettenlaufwerke oder Laufwerkcontroller, SCSI (Kleincomputer-Systemschnittstelle) Adapter und Telefoniekarten, Modems, Soundkarten und eine Vielzahl von Datenaquisitionskarten, wie GPIB oder Feldbusinterfacekarten sein.

[0170] Zahlreiche Variationen und Modifikationen werden den Fachleuten auf dem Gebiet offenbar werden, sobald die obige Offenbarung vollständig anerkannt ist. Es ist beabsichtigt, dass die folgenden Ansprüche interpretiert werden, um alle derartigen Variationen und Modifikationen zu umfassen.

Industrielle Anwendbarkeit

[0171] Diese Erfindung ist in dem Gebiet von Prozessoren und Computersystemen anwendbar.

Patentansprüche

1. Planungseinrichtung (**36**) mit:
einem Puffer (**80**) zum Speichern einer ersten Befehlsoperation; einer mit dem Puffer (**80**) gekoppelten Schaltung (**82**;**86**), die zum Auswählen der ersten Befehlsoperation zwecks Ausgebens aus dem Puffer (**80**), zum Halten der ersten Befehlsoperation nach dem Ausgeben in dem Puffer (**80**) und zum Wiederausgeben der ersten Befehlsoperation, wenn die erste Befehlsoperation inkorrekt ausgeführt worden ist, vorgesehen ist;
dadurch gekennzeichnet, dass
die Planungseinrichtung ferner einen mit der Schaltung (**82**;**86**) gekoppelten Adressenpuffer (**90**) aufweist, wobei die erste Befehlsoperation eine erste Speicheroperation ist und der Adressenpuffer (**90**) zum Speichern einer ersten Adresse vorgesehen ist, auf die die erste Speicheroperation zugreift, wobei der Adressenpuffer zum Empfangen der ersten Adresse, zum Empfangen der zweiten Adresse einer im Anschluss an die erste Speicheroperation erfolgenden Speichern-in-Speicheroperation und zum Vergleichen der zweiten Adresse mit im Adressenpuffer (**90**) gespeicherten Adressen vorgesehen ist, und wobei die erste Befehlsoperation inkorrekt ausgeführt worden ist, wenn die Speichern-in-Speicheroperation mindestens ein Byte aktualisiert, auf das die erste Befehlsoperation zugreift, und die Speichern-in-Speicheroperation der ersten Speicheroperation in der Programmreihenfolge vorangeht.

2. Planungseinrichtung nach Anspruch 1 zum Empfangen eines ersten Signals von einer Ausführungseinheit (**42**), das anzeigt, dass die erste Befehlsoperation inkorrekt ausgeführt worden ist, wobei die Schaltung (**82**;**86**) zum Wiederausgeben der ersten Befehlsoperation in Reaktion auf das erste Signal vorgesehen ist und das erste Signal ferner anzeigt, dass die erste Befehlsoperation nicht spekulativ ausgeführt werden soll, und wobei die Schaltung (**82**;**86**) zum Verzögern des Wiederausgebens der ersten Befehlsoperation vorgesehen ist, bis die erste Befehlsoperation nicht spekulativ ist.

3. Planungseinrichtung nach Anspruch 1 zum Empfangen eines ersten Signals von einer Ausführungsein-

heit (**42**), das anzeigt, dass die Befehlsoperation inkorrekt ausgeführt worden ist, wobei die Schaltung zum Wiederausgeben der ersten Befehlsoperation in Reaktion auf das erste Signal vorgesehen ist.

4. Planungseinrichtung nach Anspruch 1, bei der der Adressenpuffer (**90**) zum Empfangen einer Fülladresse vorgesehen ist, die anzeigt, dass Daten an einen Daten-Cachespeicher (**44**) gesendet werden, und wobei, wenn die erste Adresse Daten innerhalb der an den Daten-Cachespeicher gesendeten Daten anzeigt, die Schaltung zum Wiederausgeben der ersten Speicheroperation vorgesehen ist.

5. Planungseinrichtung nach Anspruch 1, ferner mit einem Kennzeichenpuffer (**92**), der zum Empfangen eines Speicherkennzeichens entsprechend einer Speichern-in-Speicheroperation vorgesehen ist, wobei die erste Befehlsoperation eine erste Speicheroperation ist, wobei die Speichern-in-Speicheroperation mindestens ein Byte aktualisiert, auf das die erste Speicheroperation zugreift, und die Speichern-in-Speicheroperation der ersten Speichern-in-Speicher-Speicheroperation in der Programmreihenfolge vorangeht, wobei der Kennzeichenpuffer (**92**) zum Speichern des Speicherkennzeichens in einen Eintrag entsprechend der ersten Speicheroperation in Reaktion auf das Ausführen der ersten Speicheroperation vorgesehen ist.

6. Planungseinrichtung nach Anspruch 5, bei dem der Kennzeichenpuffer (**92**) ferner zum Empfangen eines zweiten Speicherkennzeichens entsprechend einem Ausführungsspeicher vorgesehen ist, wobei der Kennzeichenpuffer (**92**) zum Vergleichen des zweiten Speicherkennzeichens mit dem Speicherkennzeichen vorgesehen ist und wobei die Schaltung zum Wiederausgeben der ersten Speicheroperation in Reaktion darauf, dass das Speicherkennzeichen dem zweiten Speicherkennzeichen gleich ist, vorgesehen ist.

7. Prozessor (**10**) mit:
einer Planungseinrichtung (**36**) nach einem der Ansprüche 1 bis 6; und einer mit der Planungseinrichtung gekoppelten Ausführungseinheit (**42**), die zum Ausführen der ersten Befehlsoperation vorgesehen ist.

8. Verfahren mit folgenden Schritten:
Ausgeben einer ersten Befehlsoperation von einer Planungseinrichtung (**36**) zu Ausführungszwecken;
Halten der ersten Befehlsoperation in der Planungseinrichtung (**36**) nach dem Ausgeben;
Wiederausgeben der ersten Befehlsoperation von der Planungseinrichtung (**36**) zu Ausführungszwecken in Reaktion darauf, dass die erste Befehlsoperation inkorrekt ausgeführt worden ist;
dadurch gekennzeichnet, dass
die erste Befehlsoperation eine erste Speicheroperation ist; wobei das Verfahren ferner folgende Schritte umfasst:
Speichern einer ersten Adresse der ersten Befehlsoperation in einem Adressenpuffer (**90**);
Ausgeben einer Speichern-in-Speicheroperation, die der ersten Befehloperation in der Programmreihenfolge vorangeht, nach dem Ausgeben der ersten Befehlsoperation;
Vergleichen der ersten Adresse in dem Puffer (**90**) mit einer Speicheradresse entsprechend der Speichern-in-Speicheroperation; und Detektieren, dass die erste Befehlsoperation inkorrekt ausgeführt worden ist, wenn der Vergleich anzeigt, dass die Speichern-in-Speicheroperation mindestens ein Byte aktualisiert, auf das die erste Speicheroperation zugreift.

9. Verfahren nach Anspruch 8, bei dem die erste Befehlsoperation eine erste Speicheroperation ist, wobei das Verfahren ferner folgende Schritte umfasst:
Speichern einer ersten Adresse der ersten Befehlsoperation in einem Adressenpuffer (**90**);
Vergleichen einer Fülladresse, die anzeigt, dass Daten zu einem Daten-Cachespeicher (**44**) gesendet werden; und
Wiederausgeben der ersten Speicheroperation, wenn die erste Adresse Daten innerhalb der zu dem Daten-Cachespeicher gesendeten Daten anzeigt.

10. Verfahren nach Anspruch 8, bei dem die erste Befehlsoperation eine erste Speicheroperation ist, wobei das Verfahren ferner folgende Schritte umfasst:
Speichern eines Speicherkennzeichens entsprechend einer Speichern-in-Speicheroperation in einem Eintrag eines Kennzeichenpuffers (**92**) entsprechend der ersten Speicheroperation, wobei die Speichern-in-Speicheroperation mindestens ein Byte aktualisiert, auf das die erste Speicheroperation in der Programmreihenfolge zugreift;
Vergleichen eines zweiten Speicherkennzeichens entsprechend einer ausgeführten Speicherung mit dem Speicherkennzeichen in dem Kennzeichenpuffer (**92**); und
Wiederausgeben der ersten Speicheroperation in Reaktion darauf, dass der Vergleich eine Gleichheit anzeigt.

Es folgen 14 Blatt Zeichnungen

Anhängende Zeichnungen

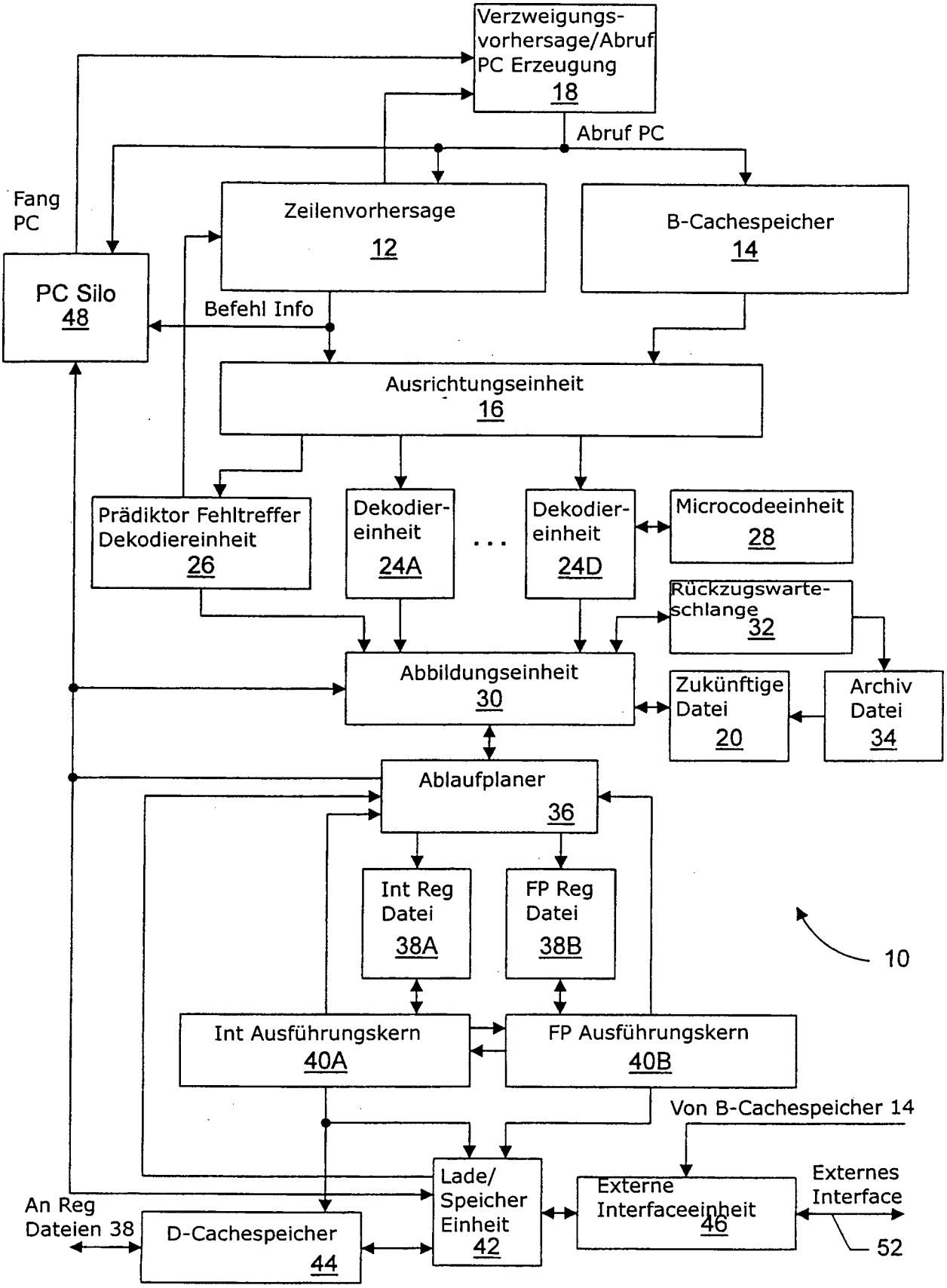


Fig. 1

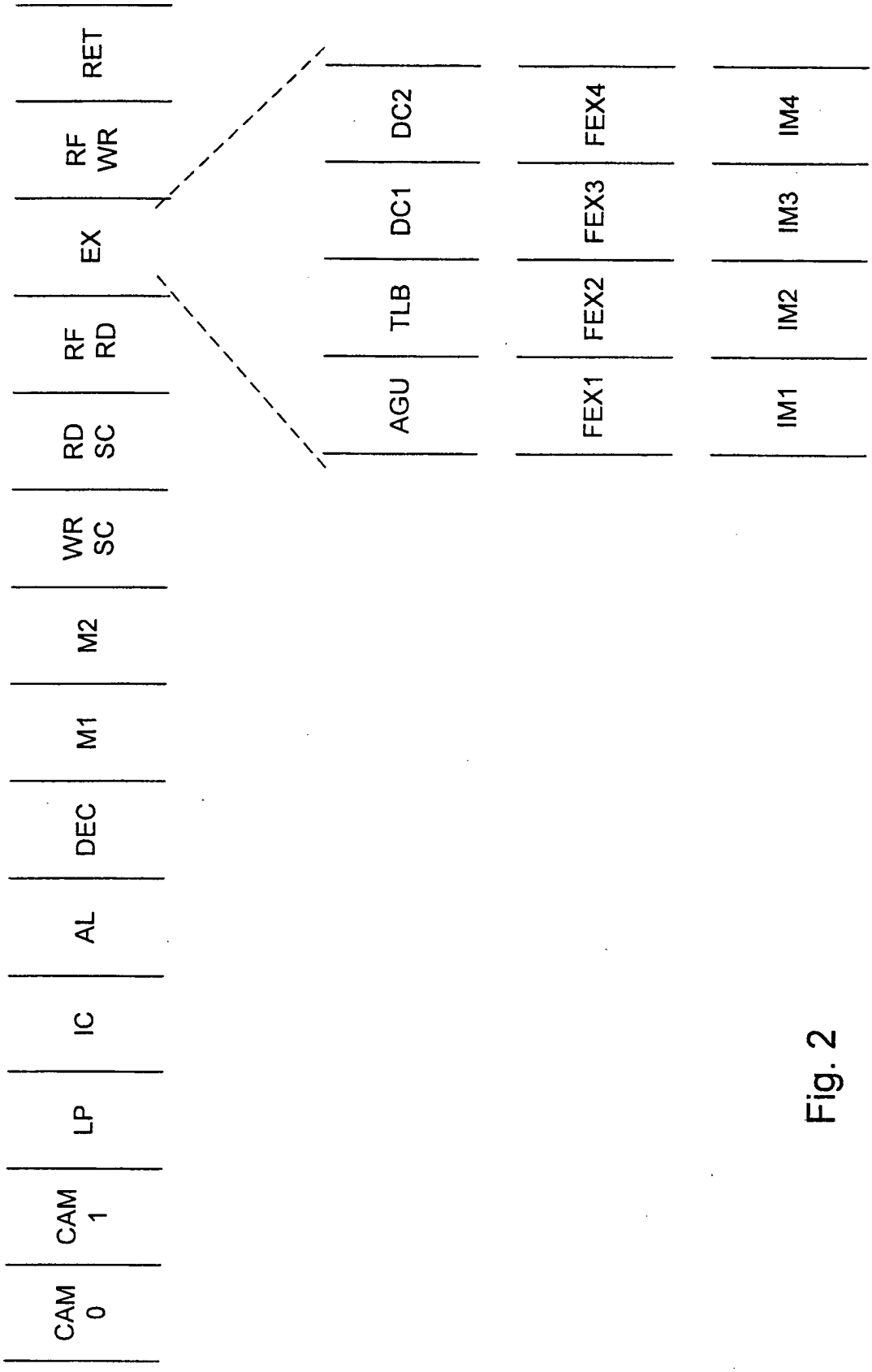

CAM 0
CAM 1
LP
IC
AL
DEC
M1
M2
WR SC
RD SC
RF RD
EX
RF WR
RET
AGU
TLB
DC1
DC2
FEX1
FEX2
FEX3
FEX4
IM1
IM2
IM3
IM4

Fig. 2

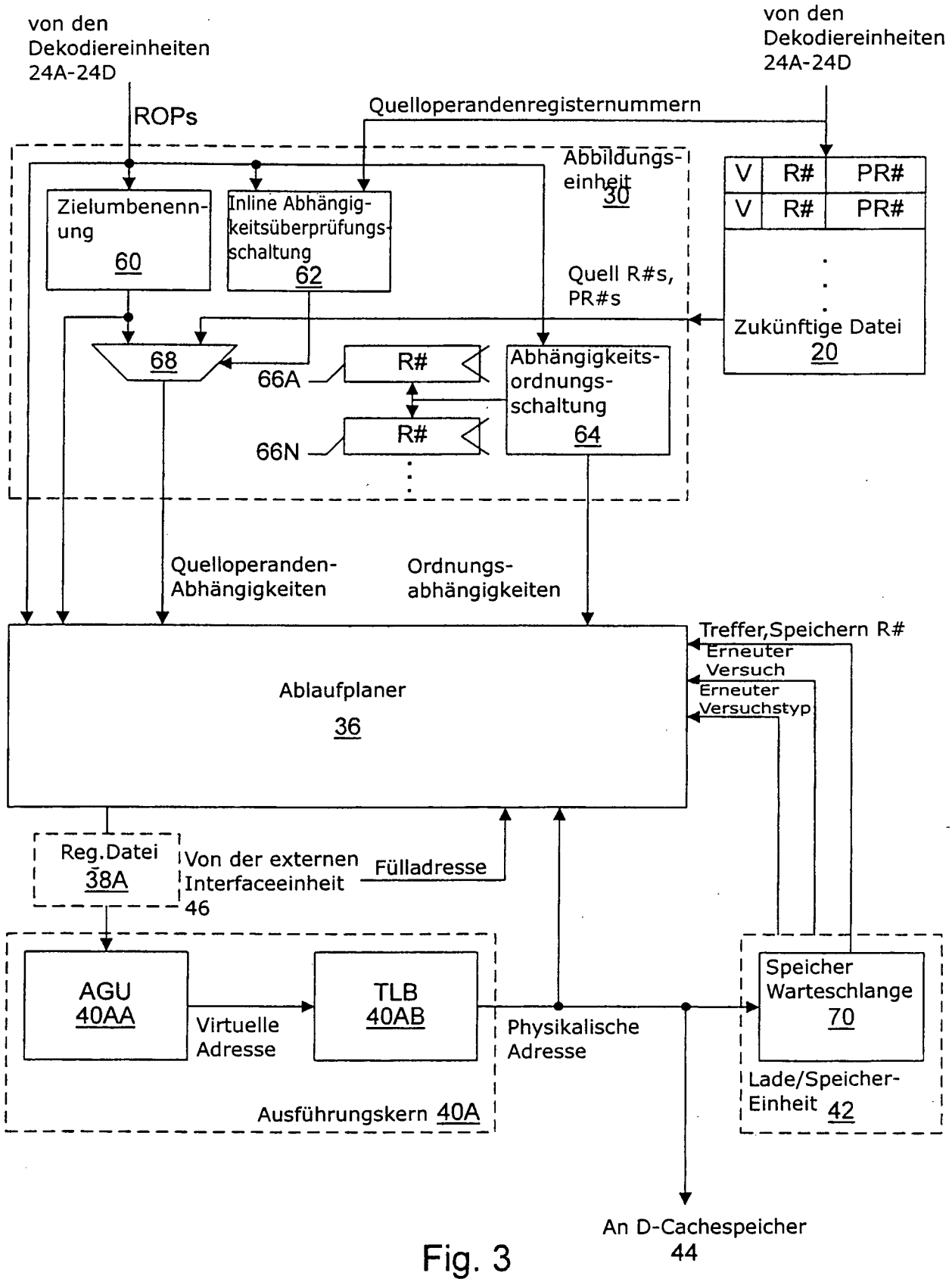

von den Dekodiereinheiten 24A-24D
ROPs
Quelloperandenregisternummern
von den Dekodiereinheiten 24A-24D
Abbildungs-einheit 30
Zielumbenennung 60
Inline Abhängigkeitsüberprüfungsschaltung 62
V R# PR#
V R# PR#
Quell R#s, PR#s
Zukünftige Datei 20
68
66A
R#
66N
R#
Abhängigkeits-ordnungs-schaltung 64
Quelloperanden-Abhängigkeiten
Ordnungs-abhängigkeiten
Treffer,Speichern R#
Erneuter Versuch
Erneuter Versuchstyp
Ablaufplaner 36
Reg.Datei 38A
Von der externen Interfaceeinheit 46
Fülladresse
AGU 40AA
Virtuelle Adresse
TLB 40AB
Physikalische Adresse
Speicher Warteschlange 70
Lade/Speicher-Einheit 42
Ausführungskern 40A
An D-Cachespeicher 44

Fig. 3

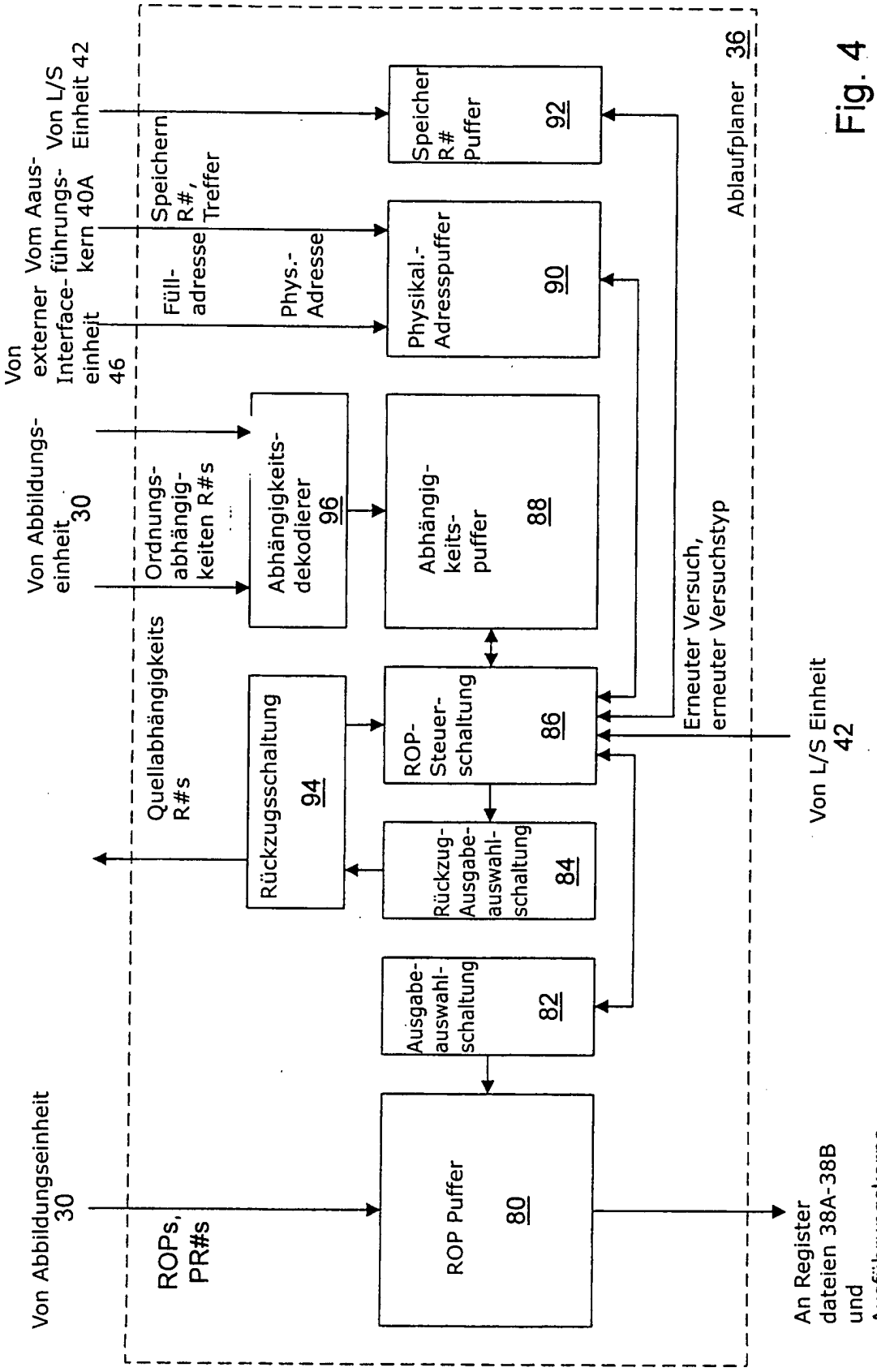

Von Abbildungseinheit 30
ROPs, PR#s
ROP Puffer 80
Ausgabe-auswahl-schaltung 82
Rückzug-Ausgabe-auswahl-schaltung 84
ROP-Steuer-schaltung 86
Rückzugsschaltung 94
Quellabhängigkeits R#s
Abhängigkeits-puffer 88
Abhängigkeits-dekodierer 96
Ordnungs-abhängig-keiten R#s
Von Abbildungs-einheit 30
Von externer Interface-einheit 46
Vom Aus-führungs-kern 40A
Füll-adresse
Phys.-Adresse
Physikal.-Adresspuffer 90
Speichern R#, Treffer
Speicher R# Puffer 92
Von L/S Einheit 42
Ablaufplaner 36
Erneuter Versuch, erneuter Versuchstyp
Von L/S Einheit 42
An Register dateien 38A-38B und Ausführungskerne 40A-40B

Fig. 4

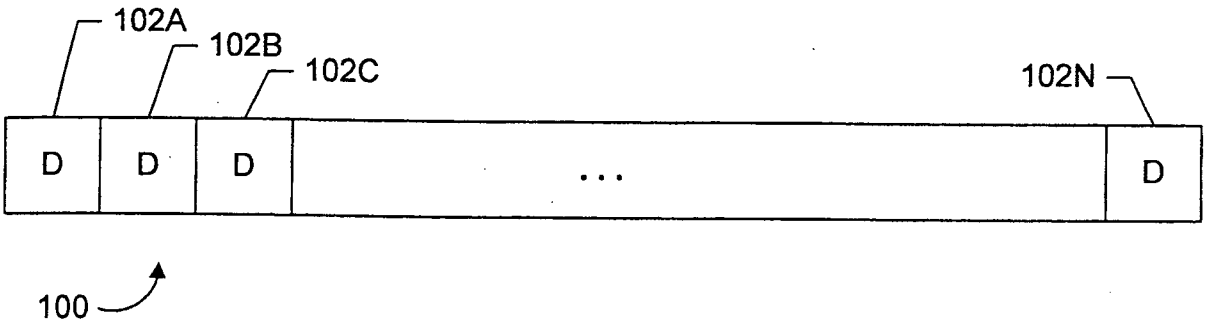

102A
102B
102C
102N
D
D
D
...
D
100

Fig. 5

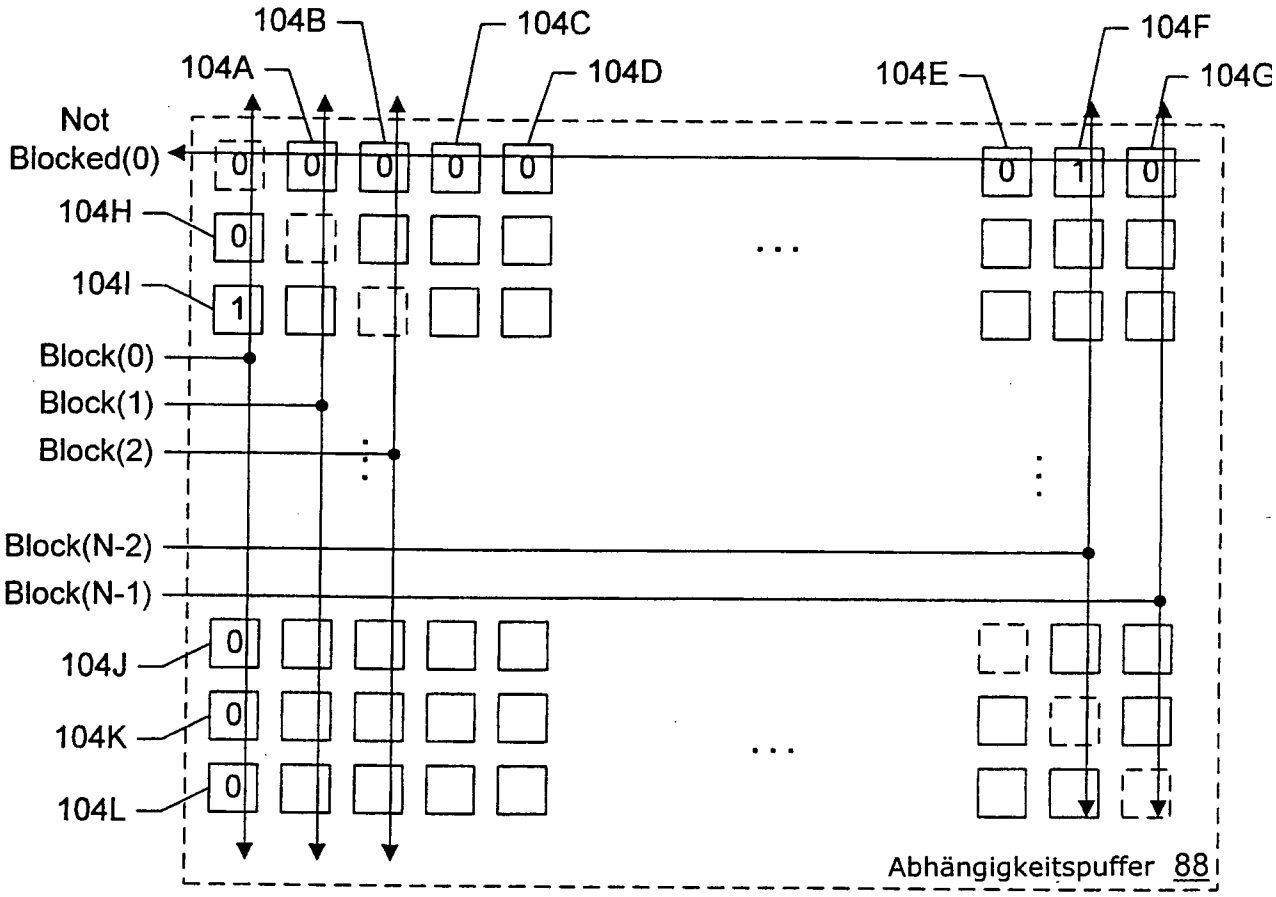

104A
104B
104C
104D
104E
104F
104G
Not Blocked(0)
104H
104I
Block(0)
Block(1)
Block(2)
Block(N-2)
Block(N-1)
104J
104K
104L
Abhängigkeitspuffer 88

Fig. 6

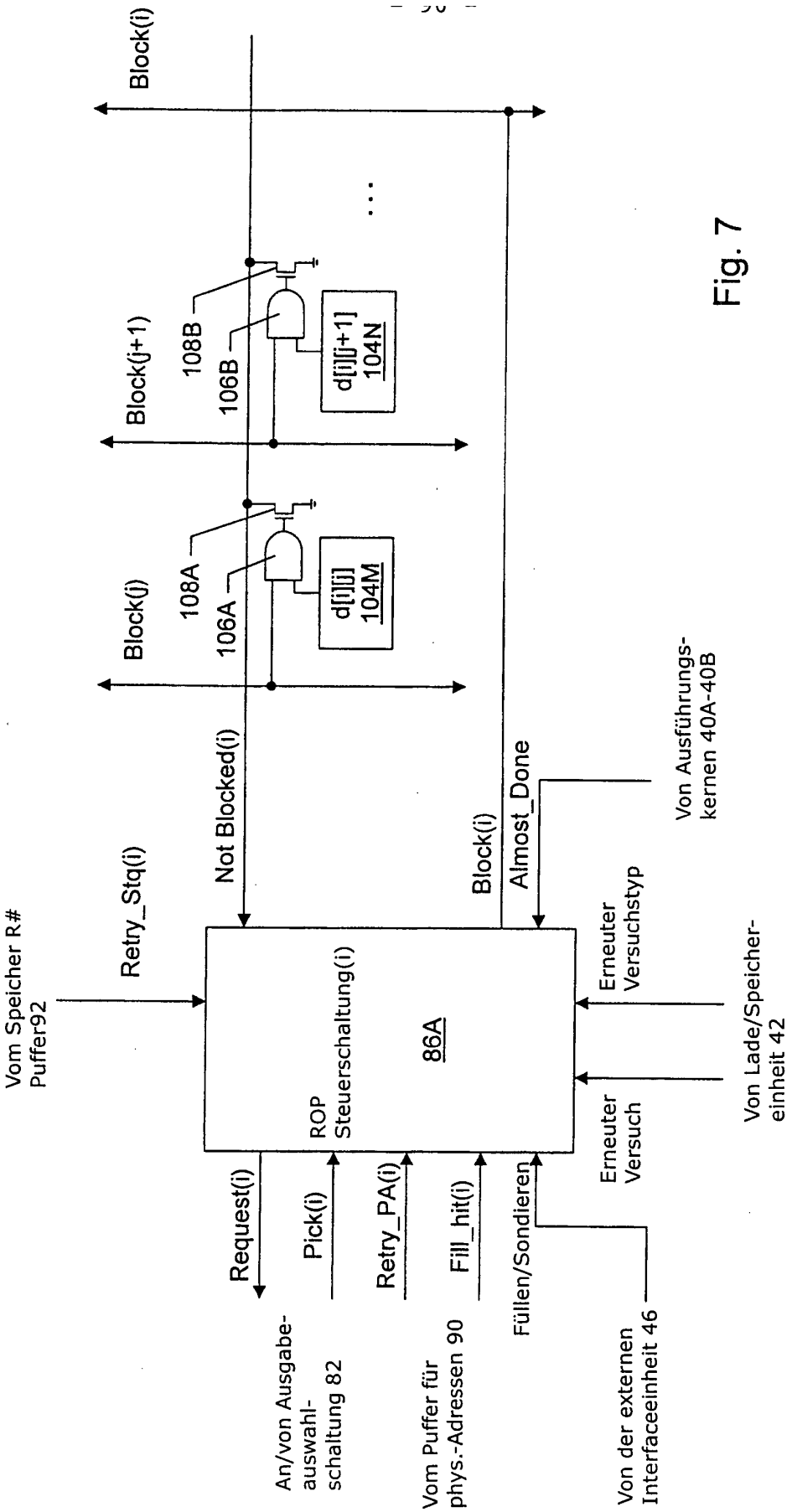

Block(i)
Block(j+1)
Block(j)
108B
106B
108A
106A
d[i][j+1]
104N
d[i][j]
104M
Not Blocked(i)
Block(i)
Almost_Done
Retry_Stq(i)
Vom Speicher R#
Puffer92
ROP
Steuerschaltung(i)
86A
Erneuter
Versuchstyp
Von Ausführungs-
kernen 40A-40B
Von Lade/Speicher-
einheit 42
Request(i)
Pick(i)
Retry_PA(i)
Fill_hit(i)
Füllen/Sondieren
Erneuter
Versuch
An/von Ausgabe-
auswahl-
schaltung 82
Vom Puffer für
phys.-Adressen 90
Von der externen
Interfaceeinheit 46

Fig. 7

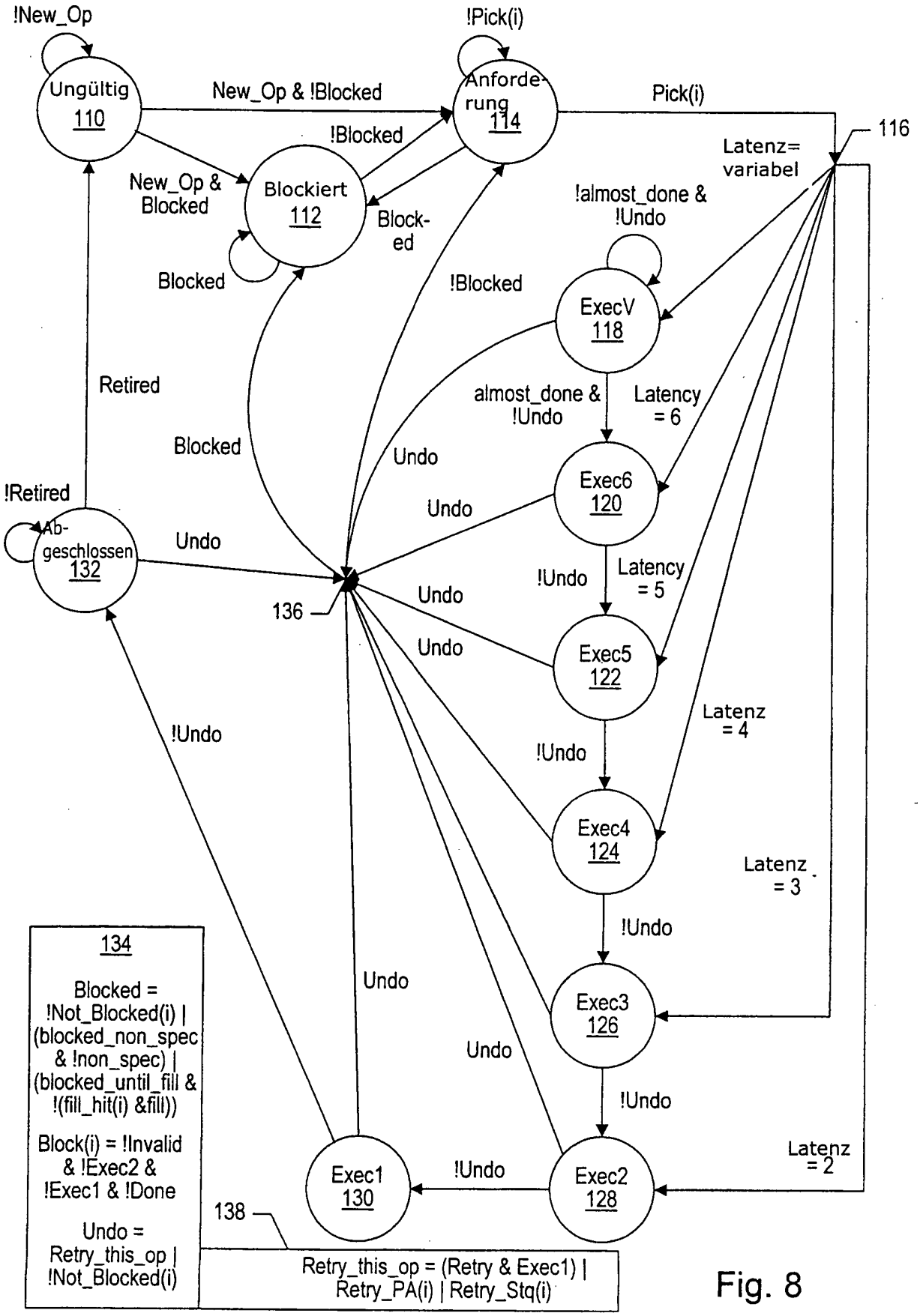

!New_Op
Ungültig
110
New_Op & !Blocked
!Pick(i)
Anforde-
rung
114
Pick(i)
116
Latenz=
variabel
!Blocked
New_Op &
Blocked
Blockiert
112
Block-
ed
Blocked
!almost_done &
!Undo
!Blocked
ExecV
118
Retired
almost_done &
!Undo
Latency
= 6
Blocked
Undo
Exec6
120
!Retired
Undo
Ab-
geschlossen
132
Undo
!Undo
Latency
= 5
136
Undo
Undo
Exec5
122
Latenz
= 4
!Undo
!Undo
Exec4
124
Latenz
= 3
!Undo
Undo
Exec3
126
Undo
!Undo
Latenz
= 2
Exec1
130
!Undo
Exec2
128
138
134
Blocked =
!Not_Blocked(i) |
(blocked_non_spec
& !non_spec) |
(blocked_until_fill &
!(fill_hit(i) &fill))
Block(i) = !Invalid
& !Exec2 &
!Exec1 & !Done
Undo =
Retry_this_op |
!Not_Blocked(i)
Retry_this_op = (Retry & Exec1) |
Retry_PA(i) | Retry_Stq(i)

Fig. 8

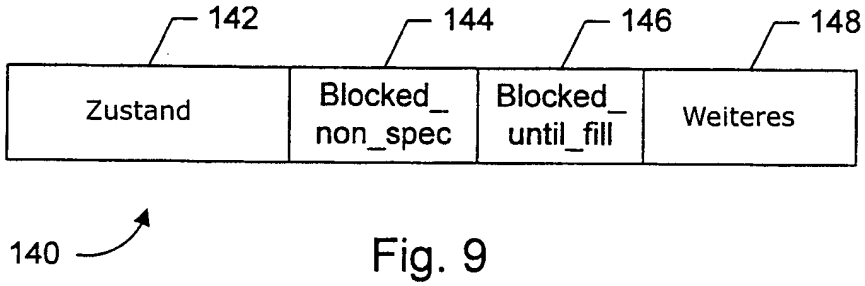
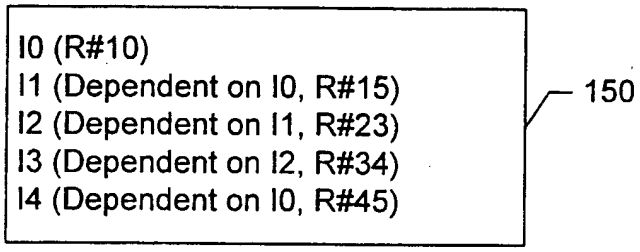

142
144
146
148
Zustand
Blocked_ non_spec
Blocked_ until_fill
Weiteres
140

Fig. 9


I0 (R#10)
I1 (Dependent on I0, R#15)
I2 (Dependent on I1, R#23)
I3 (Dependent on I2, R#34)
I4 (Dependent on I0, R#45)
150

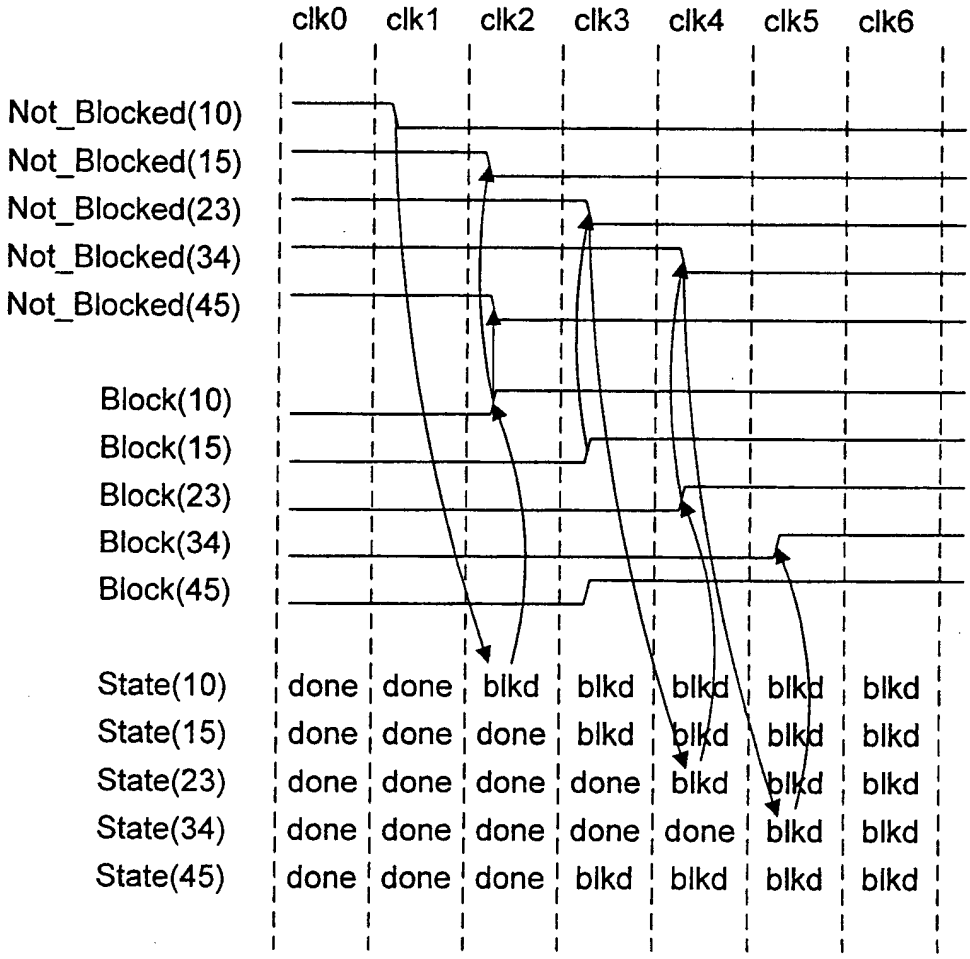

clk0 clk1 clk2 clk3 clk4 clk5 clk6
Not_Blocked(10)
Not_Blocked(15)
Not_Blocked(23)
Not_Blocked(34)
Not_Blocked(45)
Block(10)
Block(15)
Block(23)
Block(34)
Block(45)
State(10) done done blkd blkd blkd blkd blkd
State(15) done done done blkd blkd blkd blkd
State(23) done done done done blkd blkd blkd
State(34) done done done done done blkd blkd
State(45) done done done blkd blkd blkd blkd

Fig. 10

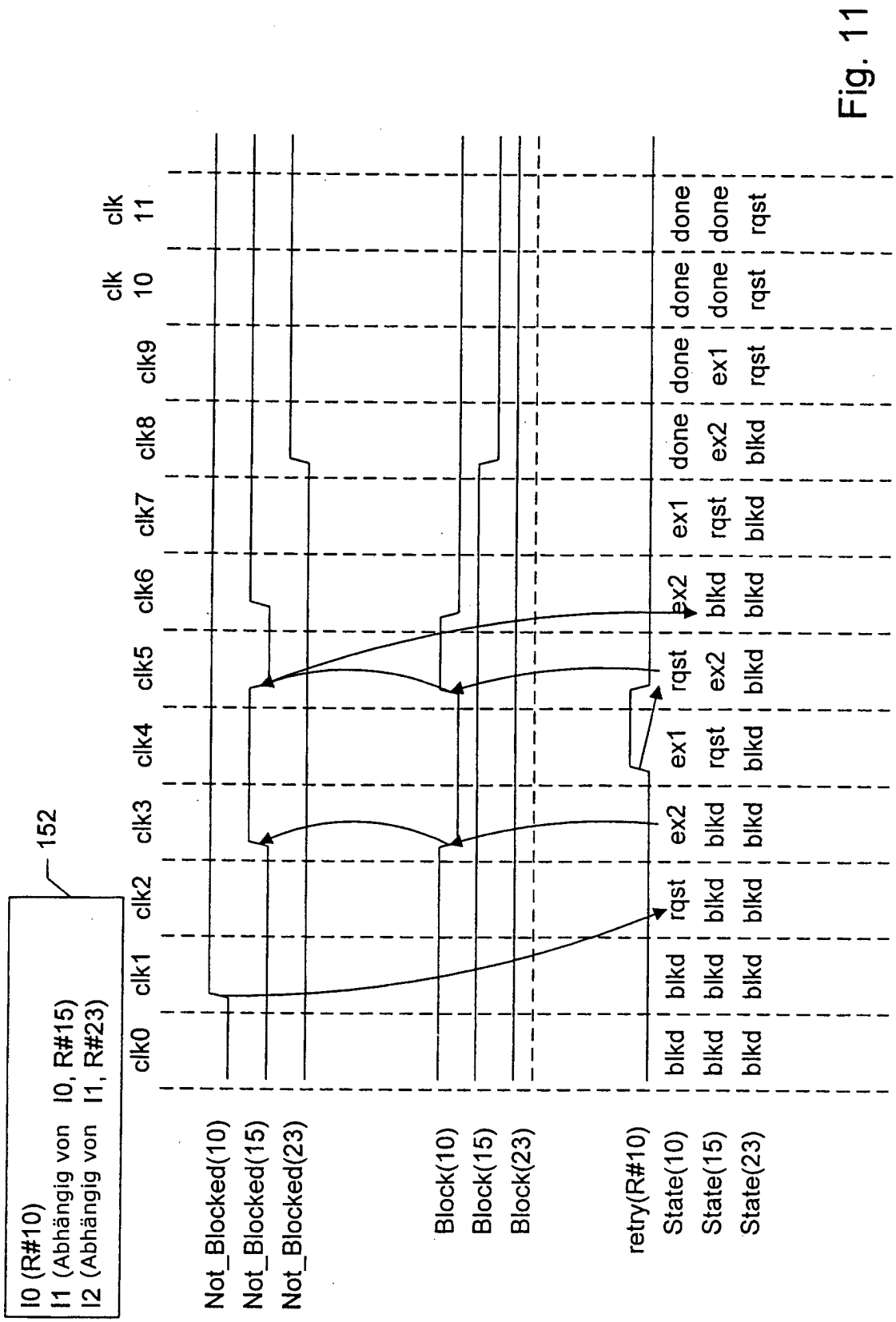

I0 (R#10)
I1 (Abhängig von I0, R#15)
I2 (Abhängig von I1, R#23)
152
clk0 clk1 clk2 clk3 clk4 clk5 clk6 clk7 clk8 clk9 clk 10 clk 11
Not_Blocked(10)
Not_Blocked(15)
Not_Blocked(23)
Block(10)
Block(15)
Block(23)
retry(R#10)
State(10)
State(15)
State(23)
blkd rqst ex2 ex1 rqst ex2 ex1 done done done done
blkd blkd blkd blkd blkd blkd blkd blkd blkd ex1 ex2 rqst
blkd blkd blkd blkd blkd blkd blkd blkd ex2 rqst done
blkd blkd blkd blkd blkd blkd blkd ex2 ex1 done
blkd blkd blkd blkd blkd blkd rqst done done
blkd blkd blkd blkd blkd rqst done done
blkd blkd blkd blkd rqst done done
blkd blkd blkd rqst done done
blkd blkd rqst done done
blkd rqst done done
rqst done done

Fig. 11

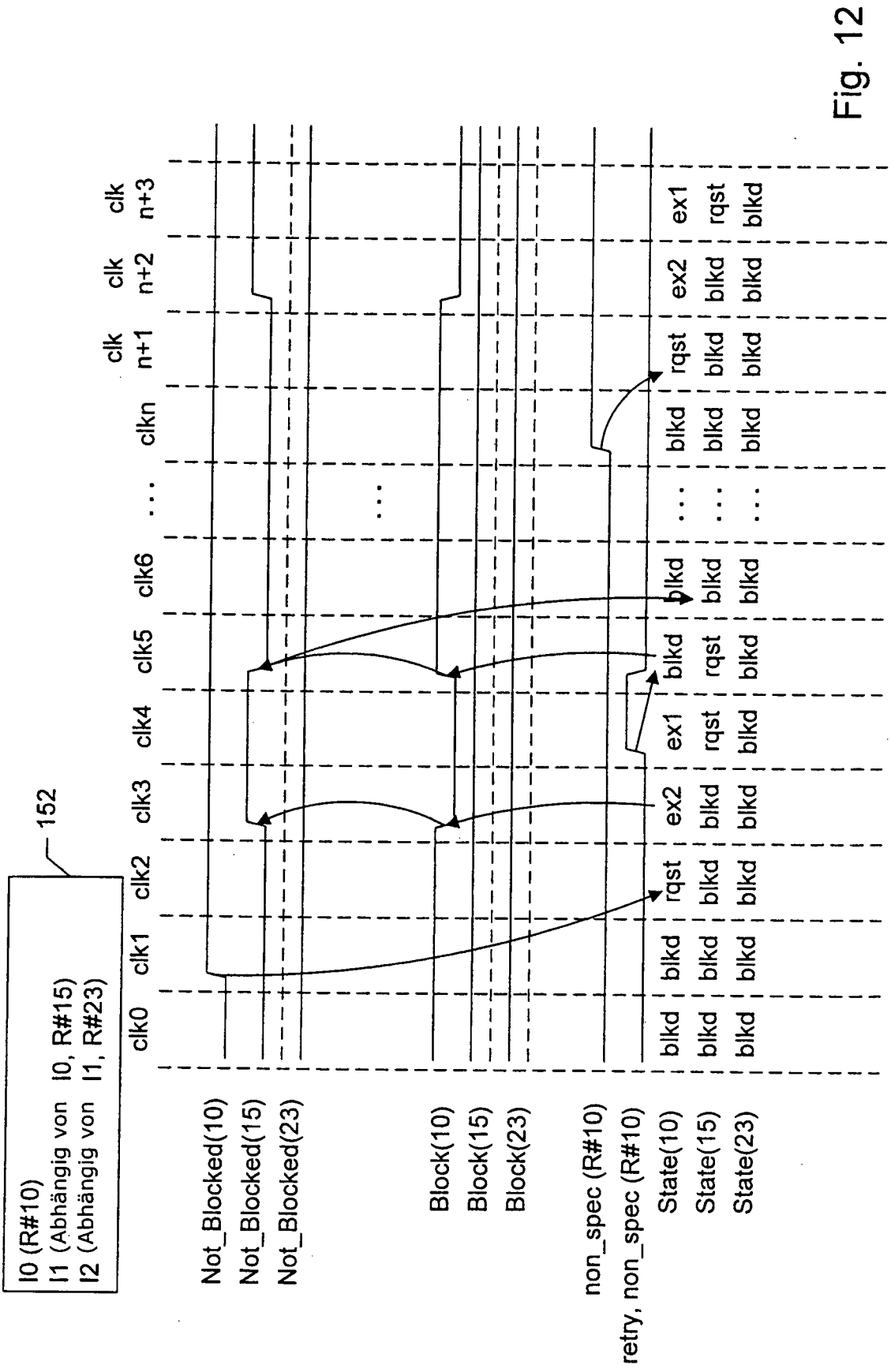

I0 (R#10)
I1 (Abhängig von I0, R#15)
I2 (Abhängig von I1, R#23)
152
clk0 clk1 clk2 clk3 clk4 clk5 clk6 ... clkn clk n+1 clk n+2 clk n+3
Not_Blocked(10)
Not_Blocked(15)
Not_Blocked(23)
Block(10)
Block(15)
Block(23)
non_spec (R#10)
retry, non_spec (R#10)
State(10)
State(15)
State(23)
blkd
rqst
ex1
ex2

Fig. 12

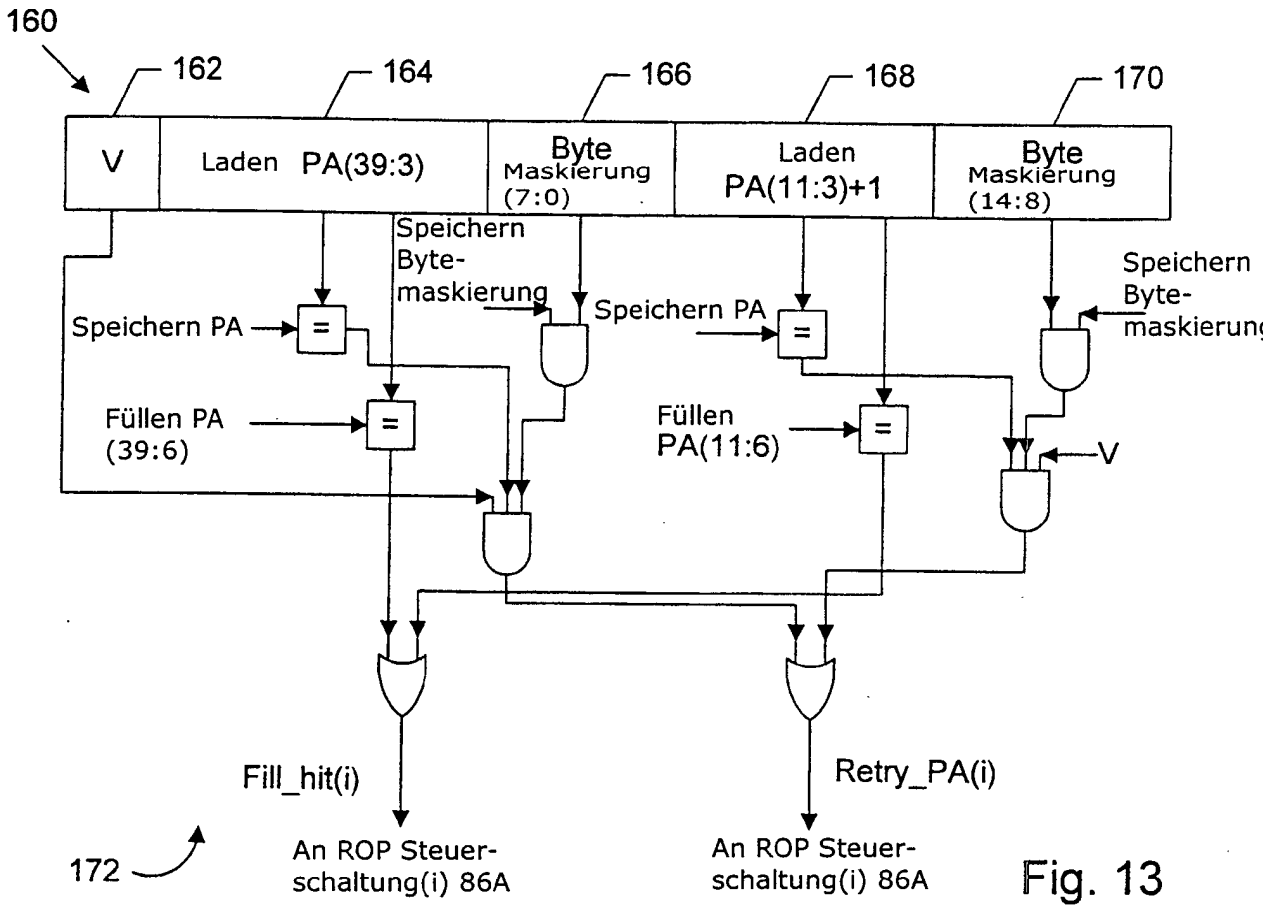

160
162
164
166
168
170
V
Laden PA(39:3)
Byte Maskierung (7:0)
Laden PA(11:3)+1
Byte Maskierung (14:8)
Speichern Byte-maskierung
Speichern PA
Füllen PA (39:6)
Füllen PA(11:6)
V
Fill_hit(i)
Retry_PA(i)
172
An ROP Steuer-schaltung(i) 86A
An ROP Steuer-schaltung(i) 86A

Fig. 13

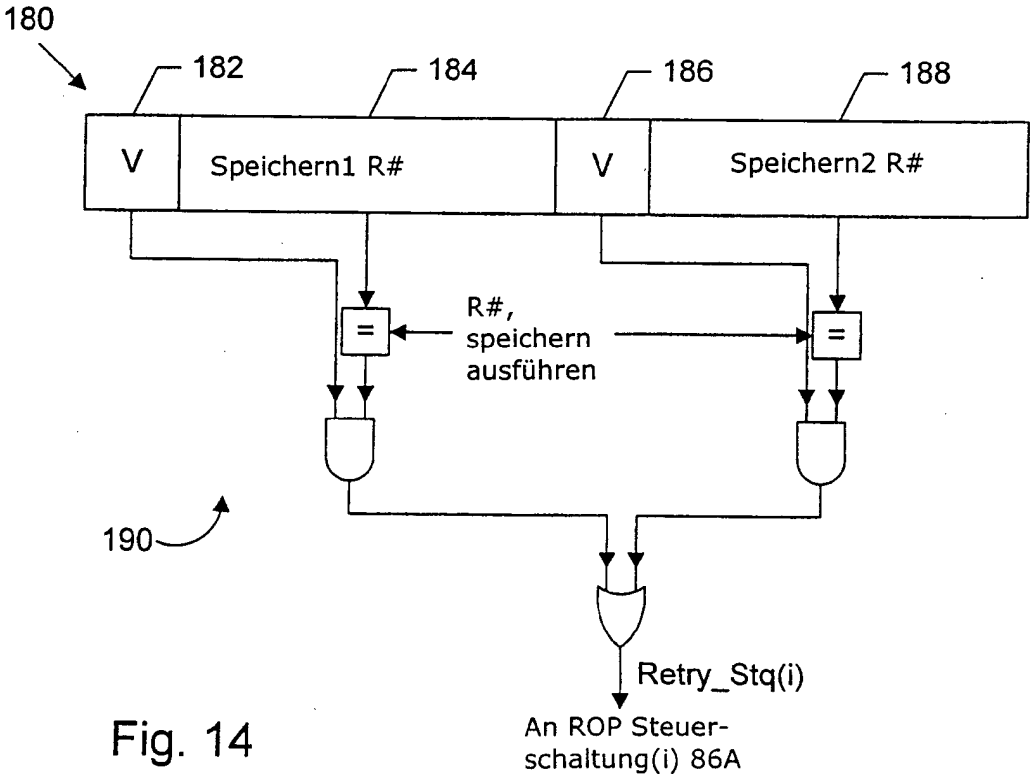

180
182
184
186
188
V
Speichern1 R#
V
Speichern2 R#
R#, speichern ausführen
190
Retry_Stq(i)
An ROP Steuer-schaltung(i) 86A

Fig. 14

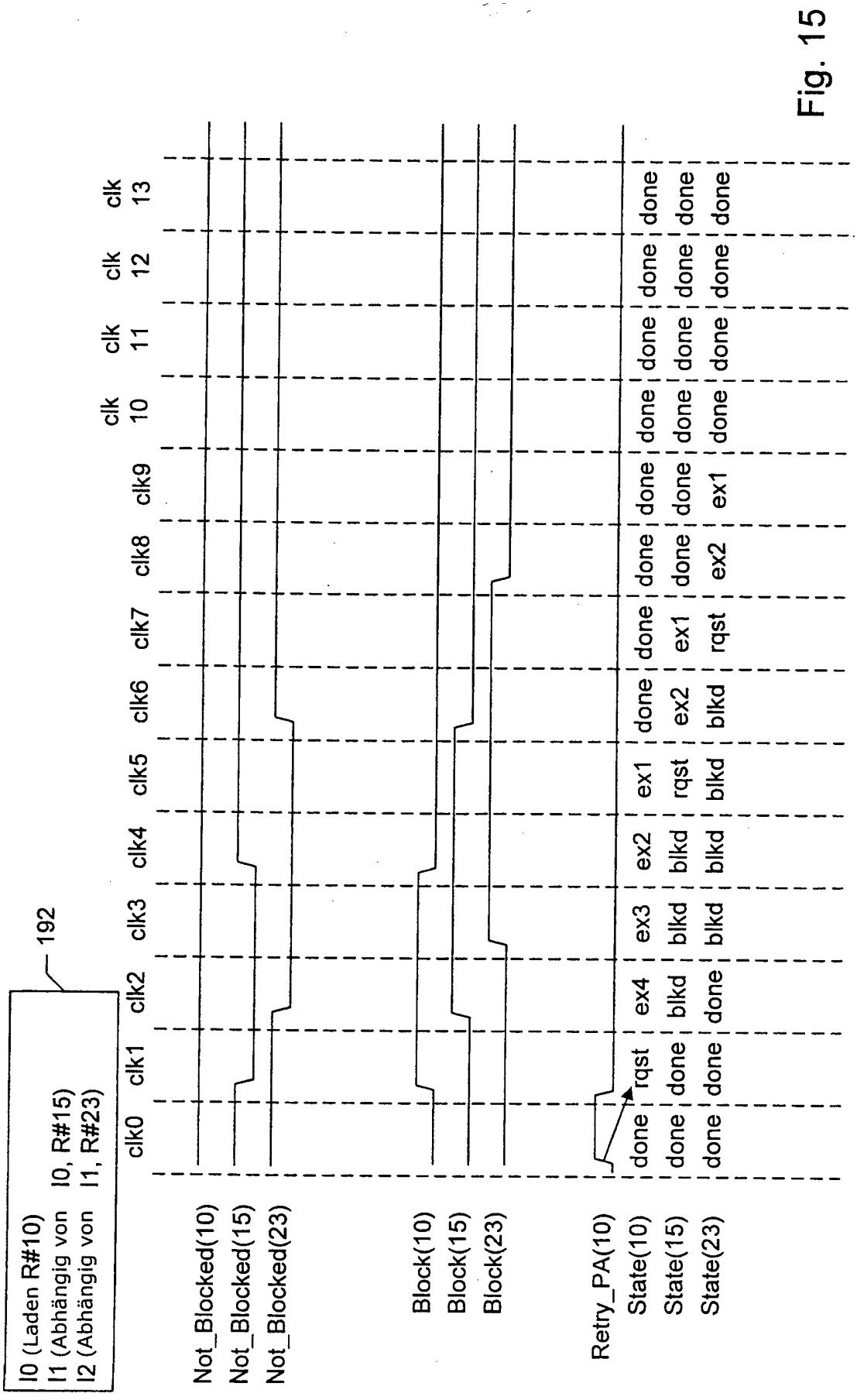

I0 (Laden R#10)
I1 (Abhängig von I0, R#15)
I2 (Abhängig von I1, R#23)
192
clk0 clk1 clk2 clk3 clk4 clk5 clk6 clk7 clk8 clk9 clk 10 clk 11 clk 12 clk 13
Not_Blocked(10)
Not_Blocked(15)
Not_Blocked(23)
Block(10)
Block(15)
Block(23)
Retry_PA(10)
State(10)
State(15)
State(23)
done rqst ex4 ex3 ex2 ex1 done done done done done done done
done done blkd blkd blkd rqst ex2 ex1 done done done done done
done done done blkd blkd blkd blkd rqst ex2 ex1 done done done

Fig. 15

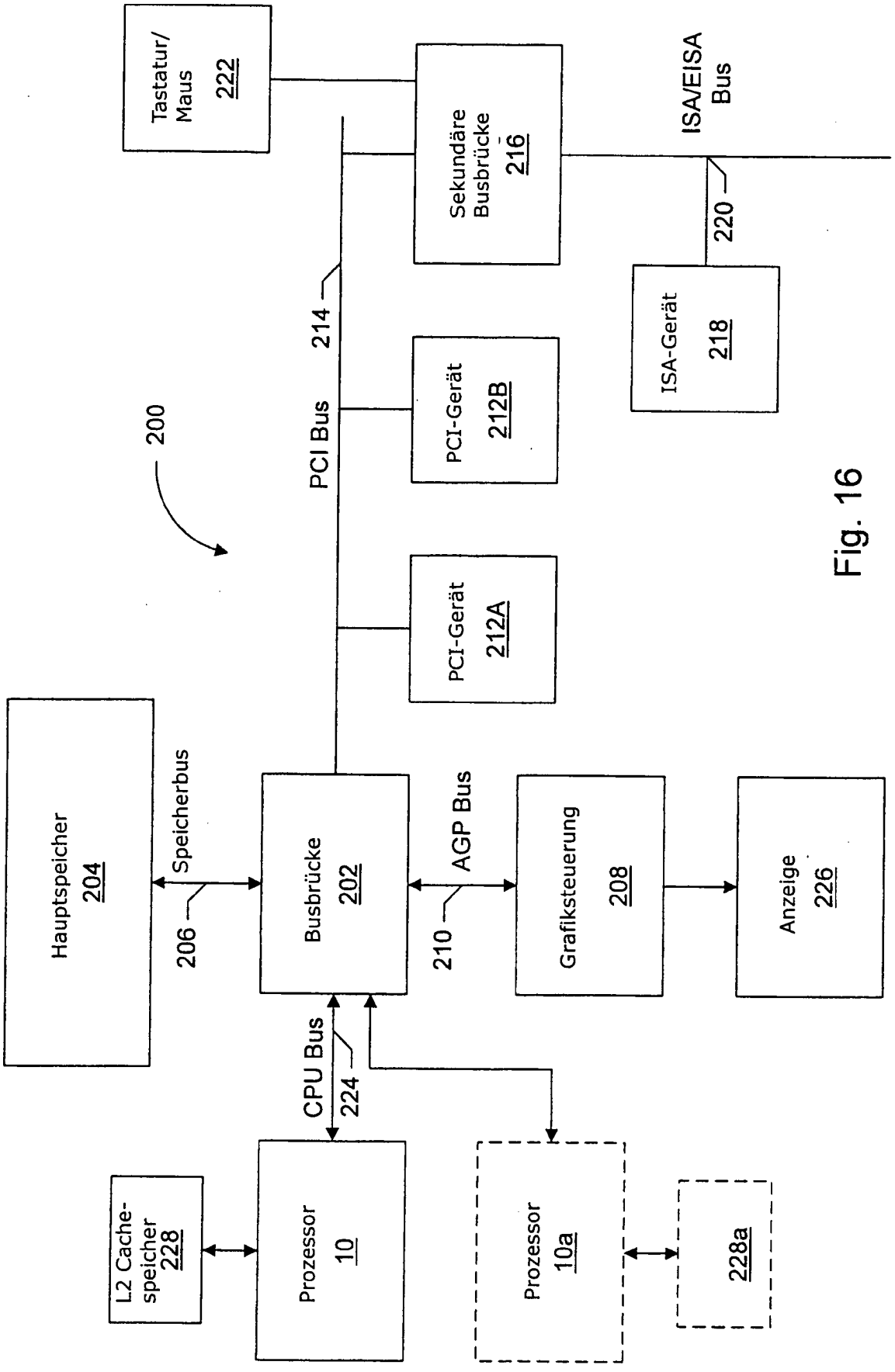

Tastatur/
Maus
222
Sekundäre
Busbrücke
216
ISA/EISA
Bus
220
ISA-Gerät
218
PCI Bus
214
PCI-Gerät
212B
PCI-Gerät
212A
200
Hauptspeicher
204
Speicherbus
206
Busbrücke
202
AGP Bus
210
Grafiksteuerung
208
Anzeige
226
CPU Bus
224
L2 Cache-
speicher
228
Prozessor
10
Prozessor
10a
228a

Fig. 16

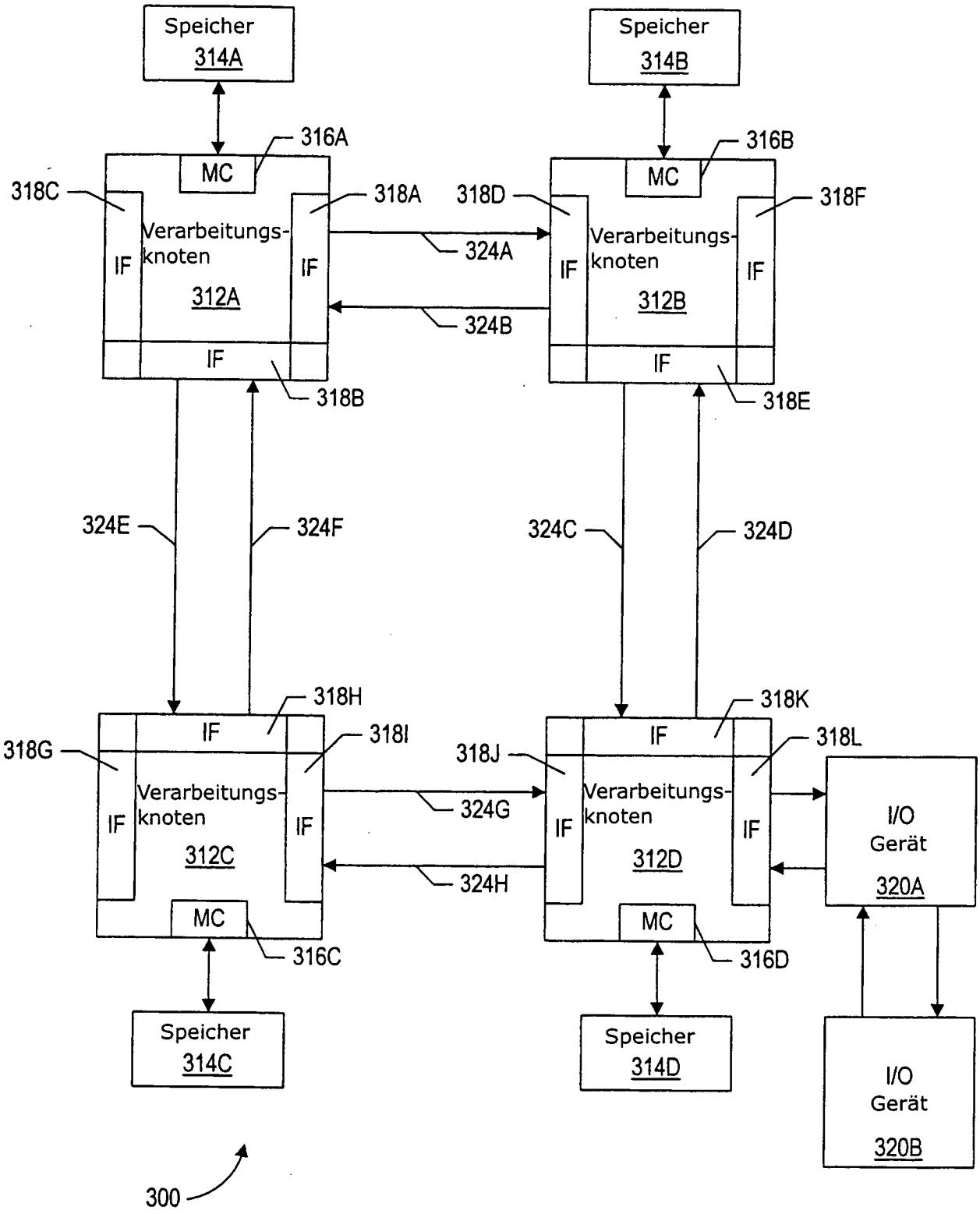

Speicher
314A
Speicher
314B
316A
316B
MC
MC
318C
318A
318D
318F
Verarbeitungs-
knoten
312A
Verarbeitungs-
knoten
312B
IF
IF
IF
IF
IF
IF
324A
324B
318B
318E
324E
324F
324C
324D
318H
318K
318G
318I
318J
318L
Verarbeitungs-
knoten
312C
Verarbeitungs-
knoten
312D
324G
324H
I/O
Gerät
320A
MC
MC
316C
316D
Speicher
314C
Speicher
314D
I/O
Gerät
320B
300

Fig. 17