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

(10) **DE 698 25 350 T2** 2005.07.21

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 871 110 B1**
(21) Deutsches Aktenzeichen: **698 25 350.7**
(96) Europäisches Aktenzeichen: **98 302 633.7**
(96) Europäischer Anmeldetag: **03.04.1998**
(97) Erstveröffentlichung durch das EPA: **14.10.1998**
(97) Veröffentlichungstag
der Patenterteilung beim EPA: **04.08.2004**
(47) Veröffentlichungstag im Patentblatt: **21.07.2005**

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

(30) Unionspriorität:
840080 **09.04.1997** **US**

(73) Patentinhaber:
Hewlett-Packard Development Co., L.P., Houston, Tex., US

(74) Vertreter:
Schoppe, Zimmermann, Stöckeler & Zinkler, 82049 Pullach

(84) Benannte Vertragsstaaten:
DE, FR, GB

(72) Erfinder:
Larson, Douglas V., Santa Clara, US

(54) Bezeichnung: **Verzweigungsvorhersage in Rechnersystem**

Beschreibung

[0001] Die vorliegende Erfindung bezieht sich auf die Ausführung von Computeranweisungen in einem Computersystem. Im Einzelnen bezieht sich die vorliegende Erfindung auf ein Verfahren und eine Vorrichtung zum Erzeugen einer Verzweigungsvoraussage für eine Computerzweiganweisung durch Unterbrechen eines Prozessors und Beobachten einer anstehenden Verzweigungsanweisung.

[0002] Frühe Computersysteme führten Computeranweisungen nacheinander einzeln aus und begannen eine Ausführung einer Anweisung erst, wenn die vorherige Anweisung abgeschlossen war. Mit zunehmendem Fortschritt in der Technik der Computerkonstruktion begannen Computerkonstrukteure, verschiedene Arten von Parallelität in Computersysteme zu integrieren.

[0003] Eine Typus einer Parallelität ist die Pipeline-Betriebsweise bzw. parallele Ausführung der Befehlsabwicklung. Die Pipeline-Betriebsweise gliedert die Ausführung einer Computeranweisung in verschiedene Schritte und verarbeitet mehrere Anweisungen gleichzeitig, indem sie diese Schritte überlappen lässt. Eine weitere Art von Parallelität ist das Superskalieren. Das Superskalieren verwendet mehrere Ausführungseinheiten, um separate Anweisungen gleichzeitig zu verarbeiten.

[0004] Parallelentwurfstechniken stellen für Verzweigungsanweisungen Probleme dar. Oft wurde, wenn eine Verzweigungsanweisung ausgeführt wird, die Bedingung, die die Verzweigungsanweisung testen muss, noch nicht bestimmt. Frühe Computersysteme hielten die Ausführung der Verzweigungsanweisung (und nachfolgender Anweisungen) einfach an, bis die Bedingung bestimmt wurde. Dies beeinträchtigt jedoch die Leistungsfähigkeit. Bei einem mit dem Pipelinebetrieb ausgestatteten Computer muss die Pipeline oft erst geleert werden, bevor die Bedingung bestimmt werden kann, was die durch den Pipelinebetrieb erzielten Vorteile einschränkt.

[0005] Um dieses Problem anzugehen, begannen Computerdesigner, Mechanismen aufzunehmen, die ein Verzweigungsverhalten voraussagen. Wenn eine Verzweigungsanweisung angetroffen wird, wird das Verzweigungsverhalten der Verzweigungsanweisung vorausgesagt. Später, wenn die Bedingung ausgewertet werden kann, wird auch eine Voraussage ausgewertet, um zu bestimmen, ob sie richtig ist. Falls die Voraussage richtig ist, wird die Ausführung fortgeführt, und die durch die parallele Ausführung erzielten Vorteile werden bewahrt. Falls die Voraussage falsch ist, müssen Anweisungen, die vorläufig ausgeführt wurden, aus der Pipeline beseitigt werden, und die Anweisungen von der richtigen Verzweigung müssen ausgeführt werden. Der Nachteil für eine falsche Verzweigung ist üblicherweise jedoch nicht schlimmer als ein Anhalten der Ausführung und ein Warten, bis die Bedingung bestimmt wird.

[0006] Die durch eine Verzweigungsvoraussage erzielten Gewinne bei der Leistungsfähigkeit stehen selbstverständlich stark mit der Genauigkeit der Voraussage in Beziehung. Demgemäß wurden viele Techniken entwickelt, um genaue Verzweigungsvoraussagen zu liefern. Eine der frühesten Techniken bestand darin, einfach vorauszusagen, dass eine Verzweigung immer genommen wird. Statistisch werden die meisten Verzweigungen genommen, so dass sich diese Technik als etwas erfolgreich erwies. Eine ähnliche Technik sagt voraus, dass immer Rückwärtsverzweigungen genommen werden und dass Vorwärtsverzweigungen niemals genommen werden.

[0007] Eine weitere Technik verwendet eine Adresstabelle von Adressen, zu denen sich kürzlich Verzweigungsanweisungen verzweigten. Üblicherweise besteht die Tabelle aus einem Assoziativspeicher, der 4 bis 8 Einträge aufweist. Falls eine Adresse in einer Verzweigungsanweisung ebenfalls in der Tabelle erschien, so wird diese Adresse als vorausgesagter Ausführungspfad verwendet.

[0008] Ein anspruchsvollerer Lösungsansatz wurde von James E. Smith in der US-Patentschrift Nr. 4,370,711 offenbart. Smith offenbarte einen Direktzugriffsspeicher (RAM), der z.B. 16 Einträge aufwies, von denen jeder einen Zwei-Bit-Zählwert enthielt, der in der Lage war, die Werte +1, 0, –1 und –2 anzunehmen. Ein Hash-Mechanismus wandelt die Verzweigungsanweisungsadresse in eine Vier-Bit-Adresse cum, die auf den RAM zugreift. Falls der Wert, der in einem einer Verzweigungsanweisung zugeordneten Eintrag gespeichert ist, +1 oder 0 ist, wird die Verzweigung als genommen vorausgesagt. Andernfalls lautet die Voraussage, dass die Verzweigung nicht genommen wird. Nachdem die Verzweigungsanweisung ausgeführt wird, falls sie genommen wird, wird der Zählwertspeichereintrag auf eine Grenze von +1 inkrementiert. Falls sie nicht genommen wird, wird die Zählwertspeicheradresse auf eine Grenze von –2 dekrementiert. Das durch Smith offenbarte Voraussageschema beinhaltet eine Verzweigungshistorie bei der Formulierung der Verzweigungsvoraussage. Falls die Verzweigung beispielsweise mehrere Male genommen wurde, darf sie nicht zwei Mal hintereinander genommen werden, um die Voraussage zu ändern. Viele Computersysteme verwenden eine Variation dieses Schemas, mit einer Tabelle, die eine Voraussage speichert, und einer Hash-Funktion, die einer Voraussage eine Verzweigungsanweisung zuordnet.

[0009] Ein weiterer Lösungsansatz wird durch Hanan Potash in der US-Patentschrift Nr. 4,435,756 offenbart. Potash offenbart eine Codierung einer Ver-

zweigungsvoraussage in jeder Verzweigungsanweisung auf der Basis dessen, ob es wahrscheinlich ist, dass die Verzweigungsbedingung als wahr oder falsch bewertet wird. Bei einem anderen Ausführungsbeispiel offenbart Potash ein Codieren einer Verzweigungshistorie und einer Verzweigungsvoraussage in einer Verzweigungsanweisung. Bei diesem Ausführungsbeispiel wird, falls sich die Voraussage zwei Mal hintereinander als falsch erweist, die Voraussage geändert, was ein Codieren einer neuen Voraussage in die Verzweigungsanweisung und ein Zurückschreiben der Verzweigungsanweisung in den Speicher erfordert. Man beachte, dass auch die Verzweigungsanweisung immer dann, wenn sich die Verzweigungshistorie ändert, in den Speicher zurückgeschrieben werden muss, auch wenn sich die Voraussage nicht ändert. Dies erzeugt eine große Menge von Schreibdaten, was den I/O-Durchsatz verringert. Beispielsweise muss eine Verzweigungsanweisung, die zwischen zwei Verzweigungspfaden abwechselt, jedes Mal, wenn sie ausgeführt wird, in den Speicher zurückgeschrieben werden.

[0010] In der XP000582805 von Conte T M et al.: „Hardware-Based Profiling, an effective technique for profile-driven optimization", International Journal of Parallel Programming, Vol. 2, 1. April 1996, Seiten 187–206, ist eine Kompilieren-Betreiben-Neukompilieren-Sequenz mit einer Kompilieren-Benutzen-Neukompilieren-Sequenz offenbart, bei der ein Programm mit Instrumentierungsanweisungen ausgeführt wird, die eingefügt werden, um ein Verzweigungsverhalten zu überwachen. Das Programm wird ausgeführt, und das Verzweigungsverhalten wird beobachtet und zusammengetragen und wird verwendet, wenn das Programm neu kompiliert wird. Conte et al. schlagen eine Verbesserung dadurch vor, dass Verzweigungsverhalten-Nachverfolgungsregister verwendet werden, die in vielen modernen CPUs zur Verfügung stehen. Folglich muss das Programm nicht unter Verwendung von Instrumentierungsanweisungen kompiliert werden, und die „erste" Kompilierung des Programmes ist viel schneller. Sowohl der Stand der Technik als auch die durch Conte et al. offenbarte Technik erfordern, dass gesammelte Informationen über ein Verzweigungsverhalten als Eingabe in eine neue Kompilierung des Programms verwendet werden.

[0011] Von der Firma Hewlett-Packard hergestellte Computersysteme verwendeten bisher zwei Arten einer Verzweigungsvoraussage; ein hardwarebasiertes Verzweigungsvoraussageschema, das eine Voraussagetabelle verwendet, um dynamisch erzeugte Verzweigungsvoraussagen nahe der CPU zu speichern, und ein softwarebasiertes Verzweigungsvoraussageschema, das statische Verzweigungsvoraussagen in jede Verzweigungsanweisung codiert, wenn ein Computerprogramm kompiliert wird.

[0012] Bei einer softwarebasierten Verzweigungsvoraussage wird die Voraussage auf der Basis der Reihenfolge der Operanden in der Vergleichsfunktion in der Verzweigungsanweisung codiert. Man betrachte beispielsweise die folgenden Vergleichen-Und-Verzweigen-Anweisungen (COMB-Anweisungen, COMB = compare and branch):
COMB, < R5, R3, Adresse
und
COMB, > = R3, R5, Adresse

[0013] Die in der ersten Anweisung codierte Verzweigungsvoraussage ist das Gegenteil der in der zweiten Anweisung codierten Verzweigungsvoraussage, obwohl die Anweisungen logisch identisch sind.

[0014] Um effektive Voraussagen zu erzeugen, ist es notwendig, einen Durchlauf einer „profilbasierten Optimierung" (PBO-Durchlauf, PBO = profile-based optimization) durchzuführen, bei dem das Verzweigungsverhalten beobachtet wird, während eine Anwendung in einer typischen Rechenumgebung ausgeführt wird. Nachdem der PBO-Durchlauf abgeschlossen ist, werden die Anwendungen des Benutzers neu kompiliert, um aktualisierte Verzweigungsvoraussagen auf der Basis eines während des PBO-Durchlaufs beobachteten Verzweigungsverhaltens zu beinhalten.

[0015] Die Vorteile einer softwarebasierten Verzweigungsvoraussage bestehen darin, dass die Voraussage auf einem Verhalten beruhen kann, das über einen ausgedehnten Zeitraum beobachtet wird, nicht nur über die letzte oder die letzten zwei Verzweigungen. Ferner erfordert eine softwarebasierte Verzweigungsvoraussage eine weniger komplexe, kostengünstigere Hardware. Es ist viel einfacher, eine Hardware zu entwerfen, die lediglich eine Verzweigungsvoraussage implementiert, im Vergleich zu einer Hardware, die auch die Genauigkeit von Voraussagen bewerten und Voraussagen entsprechend aktualisieren muss.

[0016] Die Nachteile einer softwarebasierten Verzweigungsvoraussage bestehen darin, dass die Voraussage statisch ist und sich nicht an Änderungen bei Programmdaten oder bei der Rechenumgebung anpasst. Nachdem die Voraussage in die Verzweigungsanweisung kompiliert wird, wird sie nicht mehr verändert. Außerdem führen Kunden den PBO-Durchlauf, der benötigt wird, um eine Verzweigungsvoraussage-Leistungsfähigkeit auf hohem Niveau zu erzielen, nicht gerne durch.

[0017] Die Vorteile einer hardwarebasierten Verzweigungsvoraussage bestehen darin, dass sie für den Benutzer des Computersystems vollständig transparent ist, sich dynamisch an Änderungen der Rechenumgebungen, die sich auf die Verzweigung

auswirken können, anpasst (z.B. Änderungen von in Datenbanken gespeicherten Informationen) und dass sie dazu tendiert, sehr genau zu sein, wenn die Voraussagetabelle groß ist oder jeder Anweisung eine Voraussage zugeordnet werden kann, wie durch Potash offenbart ist.

[0018] Die Nachteile einer hardwarebasierten Verzweigungsvoraussage bestehen darin, dass ihre Implementierung kostspielig ist und dass bis dato nicht viele Computer konfiguriert sind, um eine Hardware-Verzweigungsvoraussage zu nutzen. Um die Effizienz zu erhöhen und die Anzahl von Logikgattern zu verringern, speichern Voraussagetabellen üblicherweise eine begrenzte Anzahl von Verzweigungsanweisungen und speichern oft keine oder lediglich einen Abschnitt der Adresse einer Verzweigungsanweisung. Dies kann bewirken, dass das Voraussageschema durch Aliasing überwältigt wird, wodurch bewirkt wird, dass eine Voraussage erzeugt wird, die nicht auf der tatsächlichen Verzweigungsanweisung, die gerade ausgeführt wird, beruht. Aliasing kann für Programme, die viele häufig ausgeführte Verzweigungsanweisungen aufweisen, z.B. Datenbankprogramme, ein beträchtliches Problem darstellen.

[0019] Die vorliegende Erfindung schafft ein verbessertes Computersystem.

[0020] Gemäß einem Aspekt der vorliegenden Erfindung ist ein Verfahren zum Einstellen einer Verzweigungsvoraussage gemäß Anspruch 1 vorgesehen.

[0021] Gemäß einem weiteren Aspekt der vorliegenden Erfindung ist ein Zentralverarbeitungssystem gemäß der Spezifizierung in Anspruch 9 vorgesehen.

[0022] Das bevorzugte Ausführungsbeispiel kann ein Verfahren und eine Vorrichtung zum dynamischen Einstellen der einer Verzweigungsanweisung zugeordneten Verzweigungsvoraussage liefern, indem die Zentralverarbeitungseinheit eines Computers periodisch unterbrochen wird und eine Voraussageeinstellroutine ausgeführt wird, die eine anstehende Verzweigungsanweisung beobachtet. Falls keine Verzweigungsanweisung ansteht, endet die Voraussageeinstellroutine, und die Ausführung des unterbrochenen Anweisungsstroms wird wieder aufgenommen. Falls eine Verzweigungsanweisung ansteht, wird die Verzweigungsanweisung ausgewertet und mit einer Verzweigungsvoraussage, die der Verzweigungsanweisung zugeordnet ist, verglichen. Falls die Voraussage richtig ist, wird die Ausführung des unterbrochenen Anweisungsstroms wieder aufgenommen. Falls die Voraussage falsch ist, wird die Voraussage ausgewertet, um zu bestimmen, ob sie geändert werden sollte. Bei einem Ausführungsbeispiel wird ferner eine frühere Verzweigungshistorie der Verzweigungsanweisung verwendet, um zu bestimmen, ob die Verzweigungsvoraussage geändert werden sollte. Bei einem anderen Beispiel wird die Verzweigungsvoraussage umgeschaltet, wenn sie falsch ist.

[0023] Unter Bezugnahme auf die beiliegenden Zeichnungen wird nachfolgend ein Ausführungsbeispiel der vorliegenden Erfindung lediglich beispielhaft beschrieben. Es zeigen:

[0024] Fig. 1 ein vereinfachtes Blockdiagramm eines Computersystems;

[0025] Fig. 2 ein Blockdiagramm eines Abschnitts einer GPU, die ein Bestandteil des in Fig. 1 gezeigten Computersystems ist;

[0026] Fig. 3 ein Flussdiagramm eines bevorzugten Ausführungsbeispiels einer auf einer Unterbrechung beruhenden Voraussageeinstellroutine.

[0027] Fig. 1 ist ein vereinfachtes Blockdiagramm eines Computersystems **10**. Das Computersystem **10** umfasst eine Zentralverarbeitungseinheit (CPU) **12**, einen Cache-Speicher **14** einer Ebene **1** (L1), einen Cache-Speicher **16** einer Ebene **2** (L2), einen Hauptspeicher **18**, eine Dauerspeicherplatte **20** und eine Virtueller-Speicher-Speicherplatte **22**. Bei vielen Computersystemen sind die Dauerspeicherplatte **20** und die Virtueller-Speicher-Speicherplatte **22** auf demselben physischen Festplattenlaufwerk verkörpert.

[0028] Fig. 1 veranschaulicht die verschiedenen Stellen, an denen ein Programmcode vor, während und nach der Ausführung gespeichert ist. Wenn ein Programm zum ersten Mal ausgeführt wird, wird der Programmcode von der Dauerspeicherplatte **20** wiedergewonnen und in dem Hauptspeicher **18** gespeichert. Während Abschnitte des Programmcodes ausgeführt werden, werden diese Abschnitte in dem L2-Cache-Speicher **16** und dem L1-Cache-Speicher **14** gespeichert. Wie in der Technik bekannt ist, ist der L1-Cache-Speicher **14** üblicherweise als sehr schneller Speicher implementiert, der sich nahe an der CPU **12** befindet. Oft liegt er auf derselben integrierten Schaltung vor wie die CPU. Der L2-Cache-Speicher **16** ist etwas langsamer und größer. Schließlich ist der Hauptspeicher **18** sehr groß und langsamer als der L2-Cache-Speicher **16**.

[0029] Wenn in dem Hauptspeicher **18** Programme und Daten gespeichert werden, kann die Größe der Programme und Daten die physische Größe des Speichers **18** eventuell überschreiten. Wenn dies geschieht, werden Speicherseiten von dem Speicher **18** auf der Virtueller-Speicher-Speicherplatte **22** gespeichert, wodurch in dem Speicher **18** zusätzlicher Speicherplatz zur Verfügung gestellt wird. Wenn ein Programm auf eine Speicherseite Bezug nimmt, die auf

der Platte **22** gespeichert wurde, wird die Speicherseite wiedergewonnen, und, falls nötig, werden andere Seiten zu der Platte **22** umlagert.

[0030] Fig. 1 veranschaulicht eine typische Computerarchitektur, die auf dem Gebiet des Computereinsatzes üblich ist. Das bevorzugte Ausführungsbeispiel wird nachfolgend unter Bezugnahme auf Fig. 1 beschrieben, Fachleute werden jedoch erkennen, dass es bei einer Vielzahl anderer Computerarchitekturen implementiert werden kann, z.B. bei Computersystemen, die weniger oder zusätzliche Cache-Speicher aufweisen, Computer mit mehreren CPUs usw.

[0031] Fig. 2 ist ein Blockdiagramm eines Abschnitts der CPU **12** der Fig. 1. Die CPU **12** umfasst eine Arithmetik-Logik-Einheit (ALE) **24**, einen Programmzähler (PZ) **26**, ein Statusregister (STR) **27** und ein Anweisungsregister (AR) **28**, einen Registerstapel **30**, einen LIFO-Stapel (LIFO = last-in first-out) **32** und eine Unterbrechungseinheit **34**. Die ALE **24** führt verschiedene mathematische Operationen durch, z.B. Addieren, Subtrahieren, Multiplizieren, Verschieben, Vergleichen und dergleichen. Die Register **30** speichern Daten, die durch die ALE **24** verwendet werden. Der PZ **26** speichert eine Adresse, die auf den Speicherplatz der aktuellen Anweisung, die gerade ausgeführt wird, Bezug nimmt, das AR **28** speichert die gerade ausgeführte aktuelle Anweisung, der LIFO-Stapel **32** liefert eine vorübergehende Speicherung an die CPU **12** und die Unterbrechungseinheit **34** verarbeitet Unterbrechungen. Das Statusregister **27** umfasst Statusbits, die verschiedene Modi der CPU **12** steuern und bestimmen.

[0032] Die Unterbrechungseinheit **34** spricht auf Unterbrechungen an und ist ferner in der Lage, Unterbrechungen zu erzeugen. Beispielsweise stellt die ARQ-Leitung **36** extern erzeugte Hardwareunterbrechungen dar, z.B. einen Netzwerkadapter, der eine Hardwareunterbrechung auf einem Bus aktiviert. Eine Pausenanweisung **38** stellt Softwareunterbrechungen dar, z.B. PAUSE- oder FALLE-Anweisungen, die in den Programmcode platziert werden können. Ein Zeitgeber **42** stellt Unterbrechungen dar, die auf der Basis eines Zeitgeberwerts erzeugt werden, z.B. eines Echtzeittaktes, der die CPU **12** in einem periodischen Intervall unterbricht. Schließlich stellt ein Anweisungszähler **40** Unterbrechungen dar, die nach einer bestimmten Anzahl von Anweisungen erzeugt werden. Die durch den Anweisungszähler **40** und den Zeitgeber **42** verwendeten Werte können durch das Betriebssystem geändert werden.

[0033] Wenn die Unterbrechungseinheit **34** eine Unterbrechung verarbeitet, werden die in den Registern **30**, dem PZ **26**, dem STR **27** und dem AR **28** gespeicherten Werte auf dem LIFO-Stapel **32** gespeichert. Dies ist in Fig. 2 gezeigt, indem jeder dieser Werte zusammen mit der Markierung „(Int)" in dem LIFO-Stapel **32** gespeichert gezeigt ist. Nachdem diese Werte auf dem LIFO-Stapel **32** gespeichert wurden, wird eine Unterbrechungsserviceroutine (USR), die der jeweiligen Unterbrechung, die gerade verarbeitet wird, zugeordnet ist, ausgeführt. Nachdem die USR verarbeitet wurde, werden die Werte von dem LIFO-Stapel **32** entfernt und erneut in ihren ursprünglichen Stellen gespeichert.

[0034] Während die USR ausgeführt wird, hat sie Zugriff auf die auf dem LIFO-Stapel **32** gespeicherten Werte. Dementsprechend kann die USR die Anweisung, die gerade ausgeführt wurde, als die Unterbrechung auftrat, die Adresse, bei der diese Anweisung in dem Hauptspeicher gespeichert ist, und die Inhalte der Register zum Zeitpunkt, als die Unterbrechung auftrat, untersuchen.

[0035] Fig. 1 und Fig. 2 zeigen eine CPU-Architektur, die viel einfacher ist als moderne CPU-Architekturen, die auf dem Gebiet des Computereinsatzes bekannt sind. Moderne CPUs umfassen mehrere Ausführungseinheiten, Pipelines, eine Schaltungsanordnung zum Unterstützen einer außerhalb der Reihenfolge erfolgenden Ausführung und dergleichen. Jedoch ist die in den Fig. 1 und Fig. 2 gezeigte Architektur ausreichend, um das bevorzugte Ausführungsbeispiel zu veranschaulichen.

[0036] Obwohl die vorliegende Erfindung nachstehend unter Bezugnahme auf Fig. 1 und Fig. 2 beschrieben wird, werden Fachleute erkennen, dass das beschriebene Ausführungsbeispiel an einer Vielzahl von anderen Computerarchitekturen implementiert werden kann.

[0037] Das bevorzugte Ausführungsbeispiel liefert ein Verfahren und eine Vorrichtung zum dynamischen Einstellen von Verzweigungsvoraussagen. Gemäß seiner Verwendung in dem vorliegenden Dokument bezieht sich der Begriff „Verzweigungsanweisung" allgemein auf Verzweigungsanweisungen, die sich auf der Grundlage einer Bedingung verzweigen. Falls die Verzweigungsanweisung eine bedingungslose Verzweigungsanweisung ist, die sich immer zum selben Ort verzweigt, besteht natürlich kein Erfordernis, vorauszusagen, ob die Verzweigung genommen werden wird. Dagegen verwalten viele Computersysteme bestimmte Aspekte einer Ausführung von bedingungslosen Verzweigungsanweisungen (beispielsweise, die Pipeline voll zu halten) in einem allgemeinen Verzweigungsvoraussagerahmen. Somit wird in dem vorliegenden Dokument ein Voraussagen eines Verzweigungsverhaltens bedingungsloser Verzweigungsanweisungen betrachtet.

[0038] Fig. 3 ist ein Flussdiagramm eines bevorzugten Ausführungsbeispiels einer unterbrechungsbasierten Voraussageeinstellroutine **44**. Obwohl die Routine **44** als USR in Software implementiert sein

kann, kann dieses Ausführungsbeispiel auch teilweise oder vollständig in Hardware implementiert sein.

[0039] Bei Block **46** wird die Routine **44** gestartet. Unter Bezugnahme auf Fig. 2 gibt es mehrere Verfahren, die verwendet werden können, um eine Unterbrechung der Startroutine **44** zu erzeugen. Der Zeitgeber **42** kann konfiguriert sein, um eine Unterbrechung in einem periodischen Intervall zu erzeugen. Alternativ dazu kann der Anweisungszähler **40** konfiguriert sein, um eine Unterbrechung nach einer bestimmten Anzahl von Anweisungen zu erzeugen. Sowohl das Zeitintervall als auch die Anweisungszählung können auf einer willkürlichen Basis variiert werden, um zu gewährleisten, dass der Prozessor an verschiedenen Stellen in dem Programmcode unterbrochen wird.

[0040] Ein Vorteil des Anweisungszählers gegenüber dem Zeitgeber besteht darin, dass der Anweisungszähler bei manchen Computerarchitekturen eventuell eine noch gleichmäßigere Verteilung von Voraussageauswertungen über alle Verzweigungsanweisungen hinweg erzeugt. Bei manchen Implementierungen der PA-RISC-Architektur wird eine Unterbrechung auf einer Ebene, die zur Verwendung bei der vorliegenden Erfindung geeignet ist, beispielsweise verzögert, bis die Anweisung, die gerade ausgeführt wird, abgeschlossen ist, und dann wird die Unterbrechung abgearbeitet. Man betrachte eine Ladeanweisung, die oft einen Cache-Fehltreffer erzeugt und relativ lange braucht, bis sie abgeschlossen ist. Falls ein Zeitgeber die Unterbrechung erzeugt, wird eine Verzweigungsanweisung, die unmittelbar auf eine Ladeanweisung folgt, häufiger ausgewertet als eine Verzweigungsanweisung, die unmittelbar auf eine Addieren-Anweisung folgt, da die Gesamtzeit, die benötigt wird, um sowohl die Verzweigungsanweisung als auch die Ladeanweisung auszuführen, durch den Cache-Fehltreffer dominiert wird. Somit besteht eine erhöhte Wahrscheinlichkeit, dass während eines Cache-Fehltreffers eine Zeitgeberunterbrechung erfolgt und die Unterbrechung bei der nächsten Anweisung verarbeitet wird. Der Anweisungszähler weist dieses Problem nicht auf.

[0041] Während relativ häufige Unterbrechungen zu einem höheren Niveau einer Voraussagegenauigkeit führen, bringen häufige Unterbrechungen auch höhere Mehrkosten mit sich. Man stellte fest, dass ein Unterbrechungsintervall von etwa 0,01 Sekunden oder ein Anweisungszählwert von etwa einer Unterbrechung pro 15 Millionen Anweisungen eine relativ hohe Voraussagegenauigkeit erzeugt und dabei minimale Mehrkosten mit sich bringt.

[0042] Ein weiteres Verfahren zum Erzeugen der Unterbrechung, die die Routine **44** einleitet, besteht darin, verschiedene bedingungsbasierte Verzweigungsanweisungen durch PAUSE-Anweisungen zu ersetzen. Wenn die CPU **12** eine PAUSE-Anweisung antrifft, leitet die Unterbrechungseinheit **34** die Routine **44** ein, die die PAUSE-Anweisung durch die richtige Verzweigungsanweisung ersetzt, und wertet die Verzweigungsanweisung aus, wie nachfolgend beschrieben wird. Unter Verwendung dieses Verfahrens kann die Verzweigungsanweisung anfänglich durch eine separate Unterbrechungsabarbeitungsroutine, die Routine **44** oder ein anderes Programm durch eine PAUSE-Anweisung ersetzt werden.

[0043] Schließlich kann die CPU **12** der Fig. 2 bei einem anderen Ausführungsbeispiel konfiguriert sein, um Verzweigungsanweisungen selektiv basierend auf einem Statusflag in dem Statusregister **27** als PAUSE-Anweisungen zu behandeln. Bei diesem Ausführungsbeispiel erzeugt eine Verzweigungsanweisung immer dann, wenn das Statusflag gesetzt ist, eine Softwareunterbrechung, die durch die Routine **44** abgearbeitet wird. Dieses Ausführungsbeispiel kann ferner in Verbindung mit entweder dem Anweisungszähler **40** oder dem Zeitgeber **42** verwendet werden, um das Statusflag nach einem bestimmten Intervall oder einer bestimmten Anzahl von Anweisungen zu setzen, wodurch an der ersten Verzweigungsanweisung, die nach einem bestimmten Zeitintervall oder Anweisungszählwert angetroffen wird, wie oben beschrieben, eine Softwareunterbrechung erzeugt wird.

[0044] Unter erneuter Bezugnahme auf Fig. 3 bestimmt ein Entscheidungsblock **48**, nachdem die Routine **44** bei Block **46** gestartet wird, ob die anstehende Anweisung eine Verzweigungsanweisung ist. Man beachte, dass der Block **48** nicht notwendig ist, falls die CPU **12** konfiguriert ist, um eine Unterbrechung zu erzeugen, wenn eine Verzweigungsanweisung ansteht, wie oben beschrieben wurde. Wenn sie jedoch in einem Computersystem verwendet wird, das nicht in der Lage ist, eine Unterbrechung auf der Basis einer Verzweigungsanweisung zu erzeugen, ist der Schritt **48** notwendig, um zu bestimmen, ob eine bedingungsbasierte Verzweigungsanweisung durch die Unterbrechung „gefangen" wurde. Bei einer üblichen Mischung aus Computeranweisungen machen bedingungsbasierte Verzweigungsanweisungen etwa 15% aller Anweisungen aus.

[0045] Falls die anstehende Anweisung keine bedingungsbasierte Verzweigungsanweisung ist, verzweigt sich die Routine **44** zu Block **50**, wird die unterbrechungsbasierte Voraussageeinstellroutine **44** beendet, und wird die Programmausführung an dem Punkt wieder aufgenommen, an dem der ursprüngliche Programmcode unterbrochen wurde. Falls die anstehende Anweisung jedoch eine Verzweigungsanweisung ist, wertet ein Block **52** die Anweisung aus und bestimmt, ob die Verzweigung genommen wird. Unter Bezugnahme auf Fig. 2 kann dies erfolgen, indem die auf dem LIFO-Stapel **32**, oben beschrieben,

gespeicherten Werte untersucht werden. Selbstverständlich werden Fachleute erkennen, wie eine anstehende Verzweigungsanweisung in einem bestimmten Computersystem auszuwerten ist, wenn dieses Ausführungsbeispiel an eine Verwendung bei jenem Computersystem angepasst ist.

[0046] Bei einem weiteren Ausführungsbeispiel bestimmt der Entscheidungsblock **48** zunächst, ob die anstehende Anweisung eine Verzweigungsanweisung ist. Falls sie eine Verzweigungsanweisung ist, wird die „Ja"-Verzweigung zu Block **52** genommen, wie oben beschrieben ist. Falls sie keine Verzweigungsanweisung ist, führt der Block **48** dann Anweisungen aus, indem er Anweisungen emuliert, bis eine Verzweigungsanweisung erreicht ist. Beim Vergleichen dieses Verfahrens mit dem oben beschriebenen Verfahren kommt es zu Kompromissen, die beim Bestimmen, welches Verfahren effizienter ist, bewertet werden müssen. Wie oben erwähnt wurde, sind etwa 15% der Anweisungen Verzweigungsanweisungen, so dass eine Unterbrechung im Durchschnitt ein Mal pro sechs oder sieben Unterbrechungen eine Verzweigungsanweisung fängt. Dagegen ist die Leistungsfähigkeit der Emulation zwischen 20 und 100 Mal schlechter als eine direkte Anweisungsausführung, und bei einer typischen Anweisungsmischung wird im Durchschnitt ein Mal pro fünf oder sechs Anweisungen eine Verzweigungsanweisung angetroffen. Somit müssen im Durchschnitt mehrere Anweisungen emuliert werden, bevor eine Verzweigungsanweisung erreicht wird. Selbstverständlich führt jede Unterbrechung letzten Endes zur Auswertung einer Verzweigungsanweisung. Man beachte, dass die Verzweigungsanweisung selbst nicht emuliert werden muss, da sie direkt nach Beendigung der Routine **44** ausgeführt werden kann. Ob ein Implementierer entscheidet, Anweisungen zu emulieren, um eine Verzweigungsanweisung zu erreichen, oder zu unterbrechen, bis eine Verzweigungsanweisung „gefangen" wird, hängt von mehreren Faktoren ab, z.B. der Effizienz der Routine **44**, der Effizienz der Emulation und dem Verhältnis von Verzweigungsanweisungen zu anderen Anweisungen in der Anweisungsmischung. In bestimmten Umgebungen kann es wünschenswert sein, beide Verfahren bei demselben System anzuwenden, auf der Grundlage der Mischung von Programmen, die ausgeführt werden, und eine derartige Verwendung wird betrachtet.

[0047] Nachdem die Verzweigungsbedingung bei Block **52** ausgewertet wurde, wird die Verzweigungsvoraussage bei Block **54** ausgewertet. Bei Computersystemen, die Anweisungen gemäß dem Anweisungssatz PA-RISC von Hewlett-Packard ausführen, wird eine Verzweigungsvoraussage in einer Verzweigungsanweisung auf der Basis der Reihenfolge der Operanden codiert.

[0048] Ein Entscheidungsblock **56** entscheidet, ob die Verzweigungsvoraussage richtig ist, indem er die Verzweigungsvoraussage mit der tatsächlichen Verzweigung, die gemäß der bei Block **52** durchgeführten Auswertung der Verzweigungsvoraussage genommen werden muss, vergleicht. Falls die Voraussage richtig ist, verzweigt sich die Routine **56** zu Block **58**, der die Verzweigungshistorie aktualisiert, und dann endet die Routine **56** bei Block **50**. Falls die Verzweigung falsch ist, aktualisiert der Block **60** die Verzweigungshistorie und aktualisiert die Voraussage auf der Basis der Verzweigungshistorie.

[0049] Die Verzweigungshistorie wird verwendet, um eine genauere Verzweigungsvoraussage zu liefern. Sie ist jedoch bei diesem Ausführungsbeispiel nicht erforderlich. Beispielsweise kann die Verzweigungsvoraussage einfach geändert werden, wenn sie falsch ist. Benötigt wird lediglich die Voraussage von dem Block **54** (die in der Verzweigungsanweisung codiert sein kann) und die Auswertung der Verzweigungsbedingung bei Block **52**.

[0050] Ein einfacher Verzweigungshistorienalgorithmus ändert eine Voraussage einfach, wenn die Voraussage nach zwei aufeinander folgenden Auswertungen falsch ist. Um zu veranschaulichen, wie die Verzweigungshistorie die Verzweigungsvoraussage erhöhen kann, betrachte man eine Programmschleife, die 100 Mal durchlaufen wird und dann nicht durchlaufen wird. Ohne eine Verzweigungshistorie besteht eine Chance von 1 zu 100, dass die Schleifendurchlaufverzweigungsanweisung bei der letzten Iteration der Schleife ausgewertet wird, wodurch die Voraussage von „Verzweigung" zu „keine Verzweigung" geändert wird, was keine gute Voraussage für die Verzweigungsanweisung ist. Bei einem Verzweigungshistorienalgorithmus, wie er unmittelbar vorstehend beschrieben wurde, muss die Schleifendurchlaufverzweigungsanweisung bei der letzten Iteration der Schleife zwei Mal hintereinander ausgewertet werden, um die Voraussage zu ändern. Die Wahrscheinlichkeit, dass dies eintritt, ist 1 zu 10.000. Somit wird die Genauigkeit der Voraussage, die der Schleifendurchlaufverzweigungsanweisung zugeordnet ist, durch eine Verzweigungshistorie erhöht.

[0051] Um eine Verzweigungshistorie aufzunehmen, wenn eine Voraussage modifiziert wird, muss die Verzweigungshistorie natürlich gesichert werden. Ein Codieren einer Verzweigungshistorie in der Verzweigungsanweisung wurde von Hanan Potash in der US-Patentschrift Nr. 4,435,756 offenbart. Wenn eine Verzweigungshistorie jedoch in der Verzweigungsanweisung gespeichert ist, muss die Verzweigungsanweisung immer dann, wenn die Verzweigung ausgewertet wird, in den Speicher zurückgeschrieben werden, auch wenn die Voraussage richtig ist.

[0052] Obwohl der Anweisungssatz PA-RISC von Hewlett-Packard ein Verfahren zum Codieren einer Verzweigungsvoraussage in einer Anweisung definiert, definiert er kein Verfahren zum Codieren einer Verzweigungshistorie innerhalb einer Verzweigungsanweisung. Bei einem Ausführungsbeispiel werden Verzweigungsvoraussagen in einer Historientabelle in einem Programmspeicher gespeichert, der 32 K Bits aufweist und adressiert wird, indem auf die Adresse der Verzweigungsanweisung eine Hash-Funktion angewendet wird, um die Stelle in der Tabelle zu bestimmen, die eine bestimmte Verzweigungsanweisung darstellt. Bei hardwarebasierten Verzweigungsvoraussageschemata des Standes der Technik ist es üblich, Historientabellen aufzuweisen, die eine Größe von etwa 0,5 bis 2,0 K aufweisen. Einer der Vorteile besteht darin, dass, da sie in Software implementiert sein kann, es leicht ist, die Größe der Historientabelle einzustellen, um eine maximale Leistungsfähigkeit zu erzielen.

[0053] Eine Verzweigungshistorie kann auf verschiedene Weise codiert werden. Beispielsweise kann das Bit in der Historientabelle, auf das Bezug genommen wird, so definiert sein, dass es ein erster Wert ist, falls die Verzweigung genommen wurde, und dass es ein zweiter Wert ist, falls die Verzweigung nicht genommen wurde. Alternativ dazu kann das Bit, auf das Bezug genommen wurde, so definiert sein, dass es ein erster Wert ist, falls sich die vorherige Voraussage als richtig erwies, und dass es ein zweiter Wert ist, falls sich die vorherige Voraussage als falsch erwies. Letzteres Codierungsschema weist den Vorteil auf, dass es Probleme, die mit Aliasing verbunden sind, verringert. Man betrachte zwei Verzweigungsanweisungen, die auf Grund von Aliasing demselben Eintrag in einer Verzweigungshistorientabelle zugeordnet sind. Da die meisten Verzweigungsvoraussagen richtig sind, besteht eine hohe Wahrscheinlichkeit, dass beide Verzweigungsanweisungen den Tabelleneintrag als „vorherige Voraussage richtig" codieren. Wenn dagegen das erstgenannte Codierungsschema verwendet wird und eine Anweisung die Voraussage „genommen" aufweist und die andere die Voraussage „nicht genommen" aufweist, ist es viel wahrscheinlicher, dass die zwei Verzweigungsanweisungen einander stören, wenn auf die Verzweigungshistorientabelle zugegriffen wird. Ferner liefert das zuletzt genannte Schema einen Vorteil beim ersten Initialisieren der Verzweigungshistorientabelle, da alle Einträge der Tabelle auf „vorherige Voraussage richtig" initialisiert werden können.

[0054] Ferner wird in Betracht gezogen, höher entwickelte Verzweigungshistorienalgorithmen zu verwenden. Beispielsweise kann eine Verzweigungshistorientabelle definiert sein, einen laufenden Durchschnitt eines Verzweigungsverhaltens für eine bestimmte Verzweigungsanweisung zu halten und die Voraussage auf der Basis des laufenden Durchschnitts zu aktualisieren. Fachleute werden erkennen, dass eine Vielzahl von Algorithmen verwendet werden können, um eine Verzweigungshistorie nachzuverfolgen und eine Verzweigungsvoraussage auf der Basis der Verzweigungshistorie einzustellen. Da das System in Software implementiert sein kann, ist es für einen Programmierer leicht, verschiedene Algorithmen auszuprobieren, um zu bestimmen, welcher in einer bestimmten Umgebung am besten funktioniert. Ferner ist es möglich, komplexe Voraussagealgorithmen zu verwenden, deren Implementierung in Hardware nicht praktisch wäre.

[0055] Wie zuvor erwähnt wurde, aktualisiert der Block **58** die Historientabelle, wenn die Voraussage richtig ist, und der Block **60** aktualisiert die Verzweigungshistorientabelle und aktualisiert die Voraussage auf der Basis der aktualisierten Verzweigungshistorientabelle, wenn die Voraussage falsch ist. Bei der überwältigenden Mehrheit von Voraussage-/Historienalgorithmen würde man niemals eine Voraussage ändern, wenn sich die Voraussage als richtig erweist, und es wird nicht in Betracht gezogen, dass jemand dies beim Implementieren der vorliegenden Erfindung zu tun wünscht. Jedoch wird in Betracht gezogen, auch die Voraussage in Block **58** zu ändern, falls dies durch einen bestimmten Algorithmus gefordert wird.

[0056] Der Block **60** aktualisiert eine Verzweigungshistorie und ändert eventuell auch die Voraussage. Falls, wie oben beschrieben wurde, die Voraussage beispielsweise falsch ist, jedoch das vorherige Mal, als die Verzweigungsanweisung ausgeführt wurde, richtig war, wird die Voraussage eventuell nicht geändert. Ein weiterer Grund, warum es vorteilhaft ist, eine Verzweigungshistorie zu verwenden, um zu bestimmen, ob eine Verzweigungsvoraussage geändert werden sollte, liegt darin, dass dies die Häufigkeit von Verzweigungsvoraussagenänderungen tendenziell verringert. Beim Verwenden des Anweisungssatzes PA-RISC von Hewlett-Packard muss jedes Mal, wenn eine Voraussage geändert wird, die Verzweigungsanweisung in den Speicher zurückgeschrieben werden. Somit ist es wünschenswert, die Häufigkeit von Voraussageänderungen auf das Ausmaß zu minimieren, bei dem die Gesamtgenauigkeit von Voraussagen nicht wesentlich beeinflusst wird.

[0057] Falls die Voraussage aktualisiert werden muss, aktualisiert der Block **60** auch die Voraussage. Während der Anweisungssatz PA-RISC eine Voraussage innerhalb einer Verzweigungsanweisung umfasst, wird auch erwogen, Voraussagen unter Verwendung beliebiger anderer Verfahren, die in der Technik bekannt sind, z.B. einer Voraussagetabelle, zu codieren.

[0058] Beim Aktualisieren einer in eine Anweisung codierten Voraussage muss der Anweisungs-Opera-

tionscode aktualisiert werden, und die Anweisung muss in den Speicher zurückgeschrieben werden. Unter Bezugnahme auf Fig. 1 kann eine Verzweigungsanweisung in dem L1-Cache-Speicher **14**, dem L2-Cache-Speicher **16**, dem Hauptspeicher **18**, der Virtueller-Speicher-Speicherplatte **22** und der Dauerspeicherplatte **20** gespeichert werden. Obwohl ein Programmierer eventuell nicht wünscht, die Verzweigungsanweisung in jede dieser Speichervorrichtungen zurückzuschreiben, liegt es sicherlich innerhalb des Schutzumfangs der Erfindung, dies zu tun.

[0059] Bei einem Ausführungsbeispiel werden Verzweigungsanweisungen in den L1-Cache-Speicher **14**, den L2-Cache-Speicher **16** und den Hauptspeicher **18** zurückgeschrieben, jedoch nicht in die Dauerspeicherplatte **20** oder die Virtueller-Speicher-Speicherplatte **22**. Ein Zurückschreiben von Daten in die Dauerspeicherplatte **20** führt zu einem minimalen Anstieg der Leistungsfähigkeit und kann eine Vielzahl von Problemen bezüglich eines Verwaltens eines ausführbaren Codes mit sich bringen. Ferner ist es üblich, dass mehrere Benutzer Programmdateien gemeinsam verwenden, so dass es nicht möglich wäre, dass man jeden Benutzer eine Verzweigungshistorie in gemeinsamen Dateien speichern lässt. Jedoch kann es in bestimmten Situationen vorteilhaft sein, aktualisierte Verzweigungsvoraussagen auf die Platte **20** zurückzuspeichern.

[0060] Obwohl ein Zurückschreiben von Verzweigungsanweisungen in die Virtueller-Speicher-Speicherplatte **22** mit weniger Problemen verbunden ist, führt dies zu einem geringen Anstieg der Leistungsfähigkeit und bringt beträchtliche Mehrkosten mit sich. Dagegen führt ein Zurückschreiben von Verzweigungsanweisungen in den L1-Cache-Speicher **14**, den L2-Cache-Speicher **16** und den Hauptspeicher **18** zu einem beträchtlichen Anstieg der Leistungsfähigkeit.

[0061] Verschiedene Computersysteme verwenden verschiedene Techniken, um einen Speicher zu verwalten. Beispielsweise ist es üblich, eine Speichermarke einem Block eines Speichers in einem Cache-Speicher zuzuordnen, um anzugeben, ob dieser Block einfach aus dem Cache-Speicher verworfen werden kann, wenn er nicht mehr benötigt wird, oder in den Hauptspeicher zurückgeschrieben werden muss, um eine Änderung in dem Speicherblock widerzuspiegeln. Fachleute werden erkennen, wie diese Techniken zu verwenden sind, um die Erfindung nach Wunsch zu implementieren. Falls beispielsweise gewünscht wird, geänderte Verzweigungsanweisungen lediglich in den L1-Cache-Speicher **14** und den L2-Cache-Speicher **16**, jedoch nicht in den Hauptspeicher **18** zu schreiben, kann das Dirty-Bit eines Blocks, der eine geänderte Verzweigungsanweisung enthält, auf dem L2-Cache-Speicher **16** leer gelassen werden, um anzugeben, dass der Block verworfen werden kann und nicht in den Hauptspeicher **18** zurückgeschrieben werden muss. Falls, alternativ dazu, das Dirty-Bit gesetzt wird, muss der Block (und die geänderte Verzweigungsanweisung mit der aktualisierten Voraussage) in den Hauptspeicher **18** zurückgeschrieben werden, bevor der Speicherblock, der die Verzweigungsanweisung enthält, verschoben wird, wodurch die aktualisierte Verzweigungsvoraussage bewahrt wird, jedoch zusätzliche Rechenressourcen verbraucht werden.

[0062] Es kann auch wünschenswert sein, verschiedene Codetypen auf verschiedene Weise zu behandeln. Beispielsweise kann es wünschenswert sein, Verzweigungsvoraussagen in Bezug auf einen Kerncode, jedoch nicht einen Benutzercode, einzustellen. Desgleichen kann es sein, dass ein Programmierer eine Verzweigungsvoraussage abschalten möchte, während er bei einem Programm eine Fehlersuche durchführt, um zu verhindern, dass sich Verzweigungsanweisungen während der Programmausführung ändern. Fachleute werden Situationen erkennen, in denen es wünschenswert ist, das System zu benutzen oder nicht zu benutzen, und werden in der Lage sein, Programmierern und Computerbenutzern entsprechende Schnittstellen zur Durchführung bereitzustellen.

[0063] Nachdem der Block **60** die Verzweigungshistorie aktualisiert hat und möglicherweise die Verzweigungsvoraussage aktualisiert hat, verzweigt sich die Routine **44** zu Block **50**, und die Routine endet.

[0064] Wie aus dem Vorstehenden hervorgeht, können die beschriebenen Ausführungsbeispiele die folgenden Merkmale liefern:
ein System und eine Vorrichtung zum dynamischen Einstellen von Verzweigungsvoraussagen durch Unterbrechen einer CPU und Untersuchen einer anstehenden Verzweigungsvoraussage;
die Vorteile von Software- und Hardwaretechniken;
sie können unter Verwendung von Software implementiert werden, und sie können auch in Hardware oder Mikrocode implementiert werden. Wenn sie in Software implementiert sind, sind sie leicht zu konfigurieren. Parameter wie z.B. Unterbrechungsintervall und Historientabellengröße können ohne weiteres eingestellt, und die Änderungen der Leistungsfähigkeit können beobachtet werden, um zu ermöglichen, dass das System auf eine optimale Leistungsfähigkeit abgestimmt wird. In der Regel ist dies bei einer hardwarebasierten Verzweigungsvoraussage des Standes der Technik nicht möglich;
sie können auf Systemen implementiert sein, die entworfen sind, um eine statische softwarecodierte Verzweigungsvoraussage zu interpretieren, wodurch derartigen Systemen eine dynamische Verzweigungsvoraussage bereitgestellt wird;
sie können größere Historien- und Voraussagespeicher bereitstellen als Verzweigungsvoraussagesche-

mata des Standes der Technik, da diese Speicher als Softwaredatenstrukturen definiert sein können;
obwohl sie Voraussagen über die Zeit dynamisch einstellen können, können sie dies bei einer viel niedrigeren Häufigkeit tun als Voraussageschemata des Standes der Technik. Im Stand der Technik tendierte eine Verzweigungsvoraussage dazu, sich entweder bei einem Extrem oder bei einem anderen zu befinden. Eine softwarebasierte Verzweigungsvoraussage des Standes der Technik erzeugte einmalig Voraussagen und verriegelte sie dann in die Codierung der Anweisungen, wo die Voraussage während der Ausführung nicht geändert wurde. Dagegen wertete eine hardwarebasierte Verzweigungsvoraussage des Standes der Technik eine Verzweigungsvoraussage jedes Mal, wenn eine Verzweigungsanweisung ausgeführt wurde, aus. Es ist ein beträchtlicher Umfang an Hardware erforderlich, um diese Funktion auszuführen, ohne die Leistungsfähigkeit zu verringern. Ferner ist, falls die Voraussage tendenziell stabil ist (und das sind die meisten Voraussagen), wenig gewonnen, indem die Voraussage jedes Mal, wenn die Verzweigungsanweisung ausgeführt wird, ausgewertet wird, und falls die Voraussage oft abwechselt, so bietet auch ein ständiges Ändern der Voraussage keinen beträchtlichen Anstieg der Leistungsfähigkeit;
sie können ein Gleichgewicht zwischen den beiden Polen schaffen, die darin bestehen, eine Voraussage niemals zu aktualisieren, und eine Aussage jedes Mal, wenn eine Verzweigungsanweisung ausgeführt wird, zu aktualisieren. Bei einem der erörterten Ausführungsbeispiele wird die CPU lediglich einmal pro 15 Millionen Anweisungen unterbrochen. Da die Verzweigungsvoraussage einer Verzweigungsanweisung so selten ausgewertet wird, ist es möglich, zusätzliche Zeit darauf zu verwenden, die Voraussage zu aktualisieren, ohne beträchtliche Mehrkosten zu verursachen;
eine relativ seltene Auswertung kann auch ein Vorteil sein, wenn eine Implementierung in Hardware vorliegt. Eine derartige Implementierung könnte viel einfacher sein als bekannte Hardware-Verzweigungsvoraussagetechniken, da ein Auswerten und Aktualisieren der Voraussage beträchtlich länger dauern kann, ohne die Gesamtleistungsfähigkeit des Systems zu beeinflussen. Da weniger Hardware für die Verzweigungsvoraussage reserviert ist, steht mehr Hardware dafür zur Verfügung, andere Funktionen der CPU zu beschleunigen;
sie können eine beträchtliche Verbesserung der Leistungsfähigkeit gegenüber bekannten Verzweigungsvoraussageschemata liefern. Simulationen ergaben eine 4- bis 8%ige Verbesserung gegenüber bekannten Hardware-Voraussageschemata und eine 10- bis 15%ige Verbesserung gegenüber bekannten Software-Voraussageschemata.

[0065] Die Offenbarungen in der US-Patentanmeldung Nr. 08/840,080, deren Priorität die vorliegende Anmeldung beansprucht, und in der Zusammenfassung, die dieser Anmeldung beiliegt, sind durch Bezugnahme in das vorliegende Dokument aufgenommen.

Patentansprüche

1. Ein Verfahren zum Einstellen einer Verzweigungsvoraussage, die einer Verzweigungsanweisung, die in einem Segment eines Programmcodes enthalten ist, zugeordnet ist, während der Programmcode durch eine Zentralverarbeitungseinheit ausgeführt wird, wobei das Verfahren folgende Schritte umfasst: wiederkehrendes Unterbrechen der Ausführung des Programmcodes, wenn die Ausführung der Verzweigungsanweisung durch die Zentralverarbeitungseinheit ansteht; Ausführen einer Voraussageeinstellroutine (**46**), die die Verzweigungsvoraussage aktualisiert; und Wiederaufnehmen der Ausführung des Programmcodes.

2. Ein Verfahren gemäß Anspruch 1, bei dem das Unterbrechen der Ausführung des Programmcodes, wenn eine Ausführung der Verzweigungsanweisung durch die Zentralverarbeitungseinheit ansteht, folgende Schritte umfasst: Initialisieren eines Zeitgebers auf ein Zeitintervall oder eines Anweisungszählers auf einen Anweisungszählwert; Warten, bis das Zeitintervall abläuft, oder ein Zählen von Anweisungen, während sie ausgeführt werden; und Unterbrechen der Ausführung des Programmcodes, wenn das Zeitintervall abgelaufen ist oder wenn die Anzahl ausgeführter Anweisungen gleich dem Anweisungszählwert ist.

3. Ein Verfahren gemäß Anspruch 2, das den Schritt des Wartens auf die Verzweigungsanweisung, nachdem das Zeitintervall abgelaufen ist oder nachdem der Anweisungszählwert erreicht wurde, umfasst.

4. Ein Verfahren gemäß Anspruch 2 oder 3, bei dem der Zeitgeber auf ein zufälliges Zeitintervall initialisiert wird oder der Anweisungszähler auf einen willkürlichen Anweisungszählwert initialisiert wird.

5. Ein Verfahren gemäß einem der vorhergehenden Ansprüche, bei dem das Unterbrechen der Ausführung des Programmcodes, wenn eine Ausführung der Verzweigungsanweisung durch die Zentralverarbeitungseinheit ansteht, folgende Schritte umfasst:
a) Ausführen einer Pauseanweisung, die eine Ausführung des Programmcodes unterbricht; und Ersetzen der Pauseanweisung durch die Verzweigungsanweisung, oder
b) Setzen eines Flags, das bewirkt, dass die Verzweigungsanweisung eine Unterbrechung erzeugt; und Unterbrechen der Ausführung des Programmcodes, wenn die erste Pauseanweisung nach dem Setzen des Flags angetroffen wird.

6. Ein Verfahren gemäß einem der vorhergehenden Ansprüche, bei dem das Ausführen einer Voraussageeinstellroutine, die die Verzweigungsvoraussage aktualisiert, folgende Schritte umfasst: Auswerten einer Verzweigungsbedingung der Verzweigungsanweisung, um zu bestimmen, ob sich die Verzweigungsanweisung verzweigen wird; Auswerten einer der Verzweigungsanweisung zugeordneten Verzweigungsvoraussage; Bestimmen, ob die Verzweigungsvoraussage richtig oder falsch ist; und
a) Umschalten der Verzweigungsvoraussage, falls die Verzweigungsvoraussage falsch ist, oder
b) Sichern, ob die Verzweigungsvoraussage richtig oder falsch ist, als Verzweigungshistorie;
Aktualisieren der Verzweigungsvoraussage auf der Grundlage sowohl der Verzweigungshistorie als auch der Ergebnisse des Schrittes des Bestimmens, ob die Verzweigungsvoraussage richtig oder falsch ist.

7. Ein Verfahren gemäß Anspruch 6, bei dem der Teilschritt b) ein Umschalten der Verzweigungsvoraussage umfasst, falls die Voraussage falsch ist, wenn sie zweimal hintereinander ausgewertet wird.

8. Ein Verfahren gemäß Anspruch 1, bei dem das Ausführen einer Voraussageeinstellroutine, die die Verzweigungsvoraussage aktualisiert, folgende Schritte umfasst: Untersuchen einer anstehenden Anweisung, um zu bestimmen, ob die anstehende Anweisung die Verzweigungsanweisung ist; und
a) Wiederaufnehmen der Ausführung des Programmcodes, falls die anstehende Anweisung nicht die Verzweigungsanweisung ist, oder
b) Emulieren einer Ausführung von Anweisungen von dem Programmcode, falls die anstehende Anweisung nicht die Verzweigungsanweisung ist, bis die Verzweigungsanweisung erreicht ist.

9. Eine Zentralverarbeitungseinheit, die entworfen ist, um Computeranweisungen auszuführen, und die eine Unterbrechungseinheit (**34**) umfasst, die wirksam ist, um gemäß dem Verfahren eines der anstehenden Ansprüche eine Verzweigungsvoraussageeinstellroutine einzuleiten, die ansprechend auf eine Unterbrechung, die eine anstehende Verzweigungsanweisung auswertet und eine Verzweigungsvoraussage, die der anstehenden Verzweigungsanweisung zugeordnet ist, auf der Grundlage dessen aktualisiert, ob die Verzweigungsvoraussage richtig ist.

10. Eine Zentralverarbeitungseinheit gemäß Anspruch 9, bei der die Unterbrechungseinheit (**34**) einen Zeitgeber (**42**), der wirksam ist, um ein Zeitintervall zu messen und die Unterbrechung zu erzeugen, wenn das Zeitintervall abgelaufen ist, und/oder einen Anweisungszähler (**40**) umfasst, der auf einen Anweisungszählwert initialisiert werden kann und wirksam ist, um die Unterbrechung zu erzeugen, nachdem eine Anzahl von Anweisungen, die ausgeführt wurden, gleich dem Anweisungszählwert ist.

11. Eine Zentralverarbeitungseinheit gemäß Anspruch 9 oder 10, die folgendes Merkmal aufweist:
ein Statusflag (**27**), das wirksam ist, um die Unterbrechungseinheit zu veranlassen, die Unterbrechung zu erzeugen, wenn die Verzweigungsanweisung ansteht.

Es folgen 3 Blatt Zeichnungen

Anhängende Zeichnungen

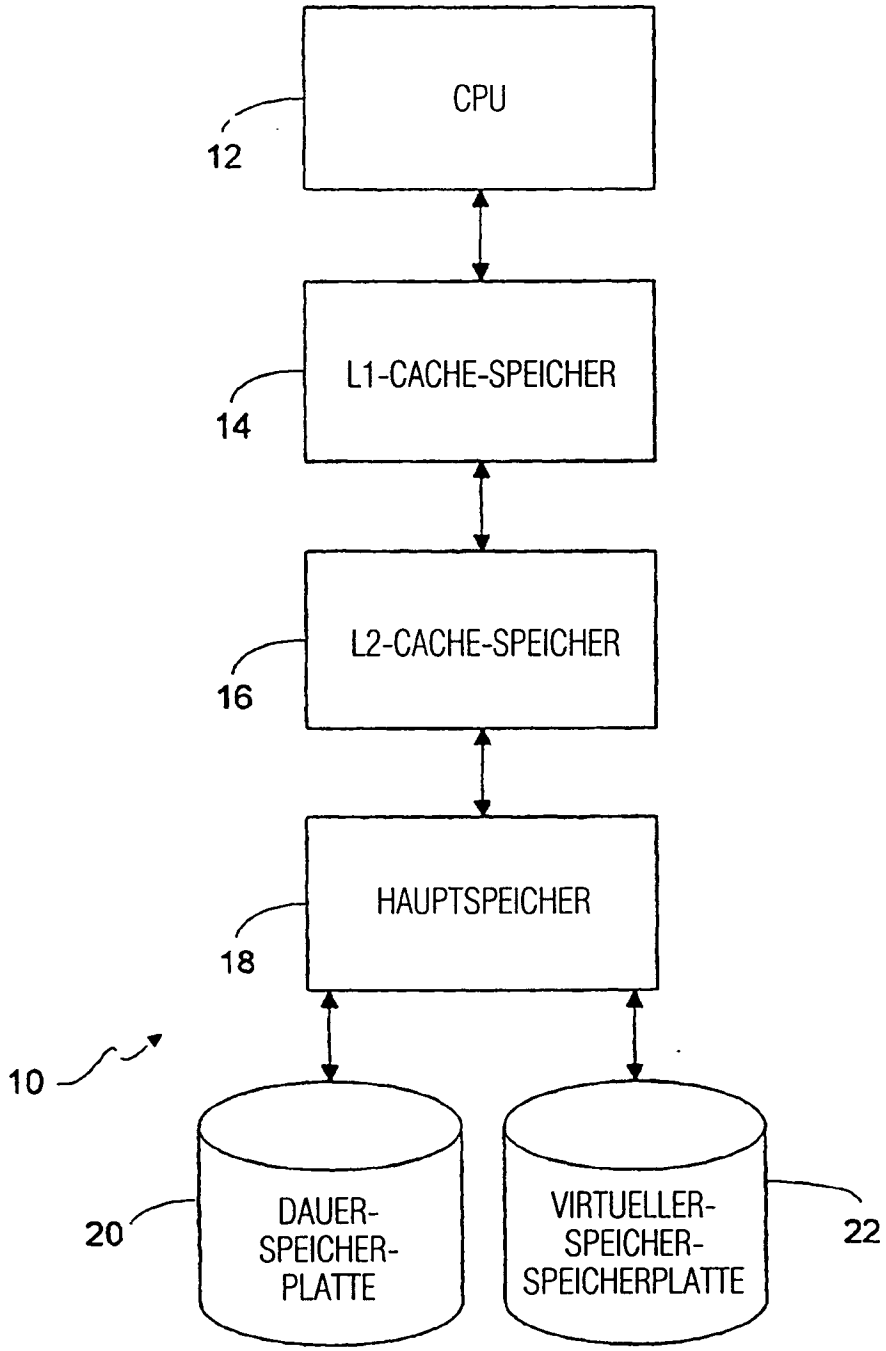


FIG. 1

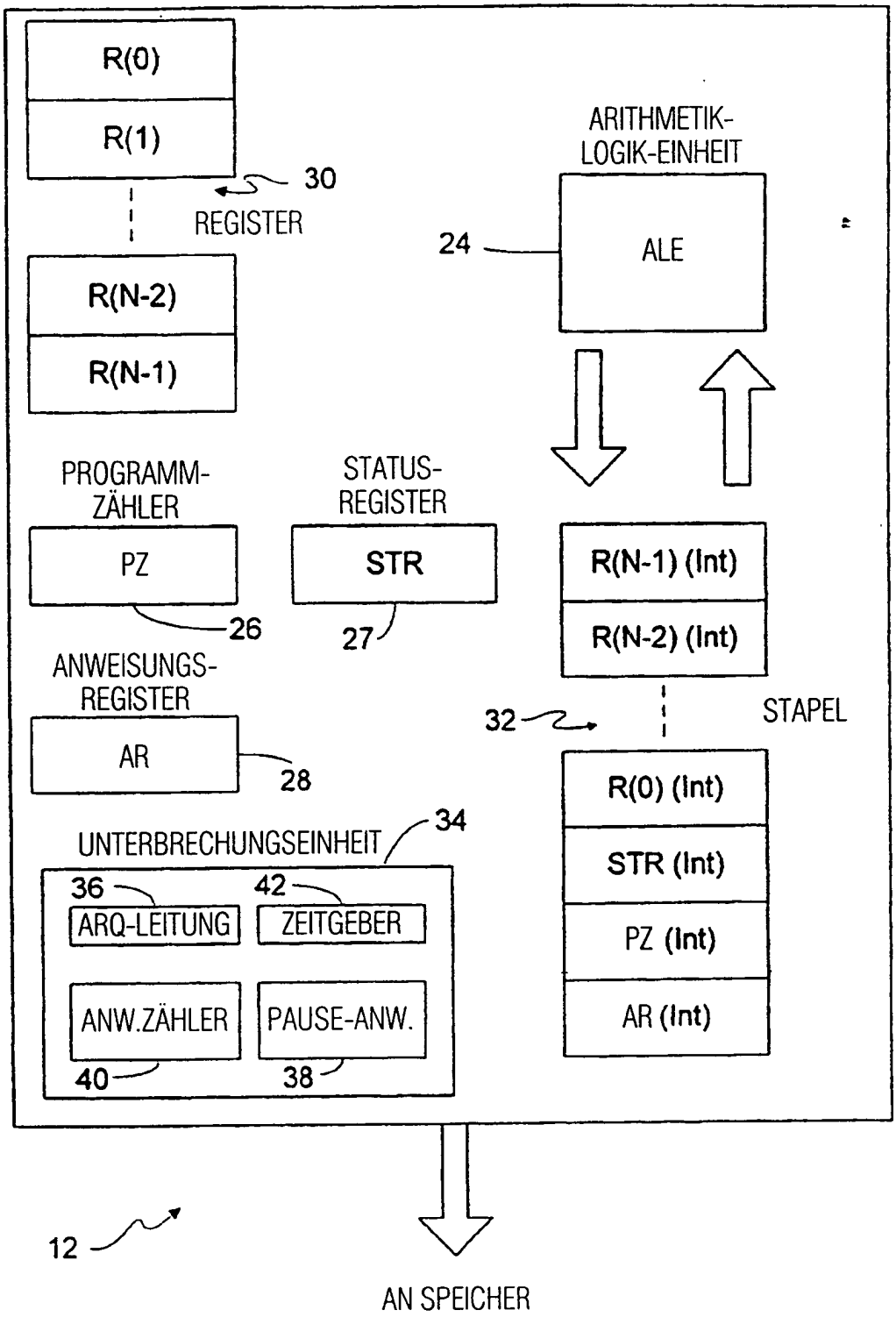
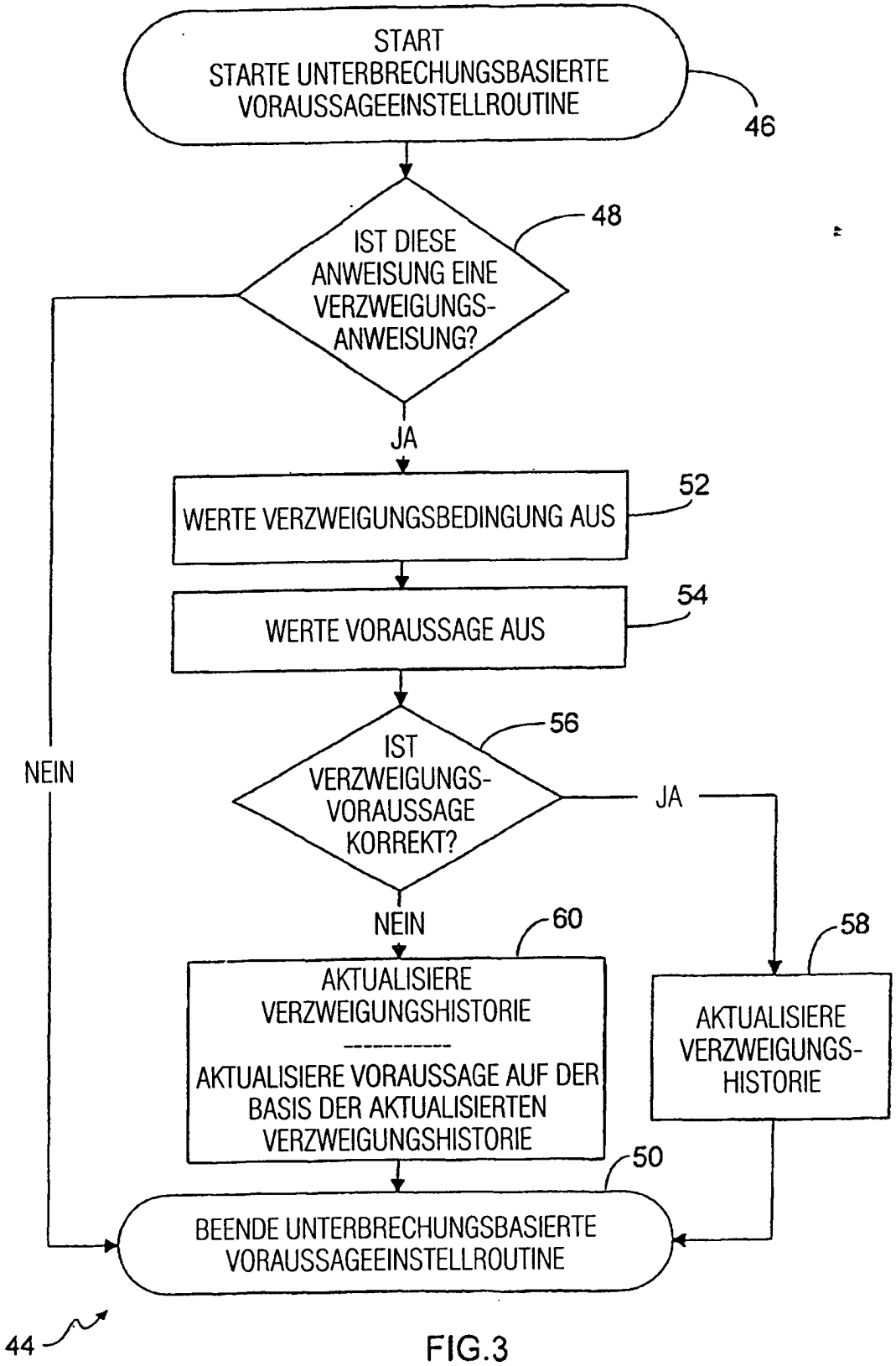

R(0)
R(1)
30
REGISTER
R(N-2)
R(N-1)
ARITHMETIK-
LOGIK-EINHEIT
24
ALE
PROGRAMM-
ZÄHLER
PZ
26
STATUS-
REGISTER
STR
27
R(N-1) (Int)
R(N-2) (Int)
ANWEISUNGS-
REGISTER
AR
28
32
STAPEL
R(0) (Int)
STR (Int)
PZ (Int)
AR (Int)
UNTERBRECHUNGSEINHEIT
34
36
ARQ-LEITUNG
42
ZEITGEBER
ANW.ZÄHLER
40
PAUSE-ANW.
38
12
AN SPEICHER

FIG. 2


START
STARTE UNTERBRECHUNGSBASIERTE VORAUSSAGEEINSTELLROUTINE
46
IST DIESE ANWEISUNG EINE VERZWEIGUNGS-ANWEISUNG?
48
JA
NEIN
WERTE VERZWEIGUNGSBEDINGUNG AUS
52
WERTE VORAUSSAGE AUS
54
IST VERZWEIGUNGS-VORAUSSAGE KORREKT?
56
JA
NEIN
AKTUALISIERE VERZWEIGUNGSHISTORIE

AKTUALISIERE VORAUSSAGE AUF DER BASIS DER AKTUALISIERTEN VERZWEIGUNGSHISTORIE
60
AKTUALISIERE VERZWEIGUNGS-HISTORIE
58
BEENDE UNTERBRECHUNGSBASIERTE VORAUSSAGEEINSTELLROUTINE
50
44

FIG.3