(19) Deutsches Patent- und Markenamt

(10) **DE 11 2004 002 259 B4** 2013.06.06

(12) # Patentschrift

(21) Deutsches Aktenzeichen: **11 2004 002 259.2**
(86) PCT-Aktenzeichen: **PCT/US2004/038734**
(87) PCT-Veröffentlichungs-Nr.: **WO 2005/055024**
(86) PCT-Anmeldetag: **07.11.2004**
(87) PCT-Veröffentlichungstag: **16.06.2005**
(43) Veröffentlichungstag der PCT Anmeldung in deutscher Übersetzung: **26.10.2006**
(45) Veröffentlichungstag der Patenterteilung: **06.06.2013**

(51) Int Cl.: ***G06F 12/14*** (2006.01)

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

(30) Unionspriorität:
10/724,321 **26.11.2003** **US**

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

(74) Vertreter:
BOEHMERT & BOEHMERT, 28209, Bremen, DE

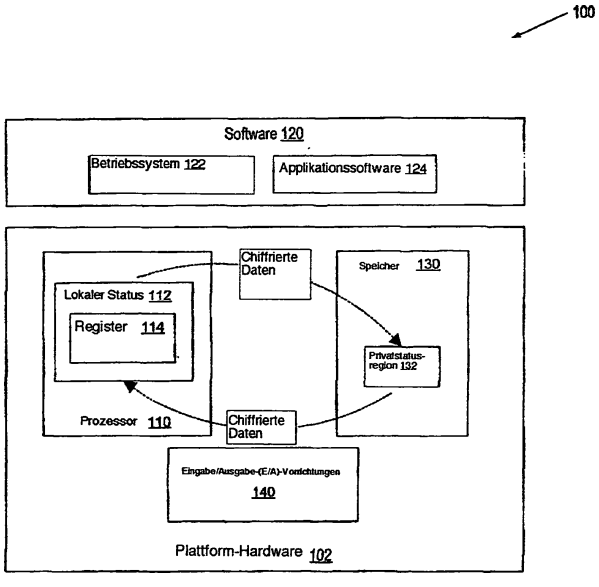
(72) Erfinder:
Robinson, Scott, Portland, Oreg., US; Espinosa, Gustavo, Portland, Oreg., US; Bennett, Steven, Hillsboro, Oreg., US

(56) Für die Beurteilung der Patentfähigkeit in Betracht gezogene Druckschriften:

US	**2001 / 0 018 736**	**A1**
US	**2003 / 0 055 928**	**A1**
US	**2003 / 0 097 551**	**A1**
US	**5 655 125**	**A**
EP	**0 700 002**	**A1**

(54) Bezeichnung: **Zugriff auf private Daten zum Status einer datenverarbeitenden Maschine von einem öffentlich zugänglichen Speicher**

(57) Zusammenfassung: Gemäß einer Ausführungsform der Erfindung ist ein Verfahren zum Betreiben einer Datenverarbeitungsmaschine beschrieben, wobei Daten zum Status der Maschine an einen Ort in einem Speicher geschrieben werden. Dabei handelt es sich um einen Ort, der für Software zugänglich ist, die für die Maschine geschrieben sein kann. Die geschriebenen Statusdaten sind codiert. Diese Statusdaten können aus dem Speicher gemäß einem Decodierungsprozeß abgerufen werden. Andere Ausführungsformen sind ebenfalls beschrieben und beansprucht.



Beschreibung

ALLGEMEINER STAND DER TECHNIK

[0001] Einige der Ausführungsformen der Erfindung betreffen die Art und Weise, wie Prozessoren Statusdaten von einem Speicher eines Computersystems ablesen und in diesen schreiben. Andere Ausführungsformen sind ebenfalls beschrieben.

[0002] Aufgrund verschiedener Designanforderungen können einige Prozessoren Daten mit privatem Status in öffentlich zugängliche Speicher schreiben. Das Format, die Semantik und der Ort dieser Privatdaten können je nach Design-Implementierung unterschiedlich sein. In der Literatur, die den Prozessor beschreibt, sind solche Speicherregionen oft als „RESERVIERT" markiert, was anzeigt, daß ihr Inhalt nicht gelesen oder modifiziert werden sollte, da sie Privatdaten enthalten. Da diese Daten in einen öffentlich zugänglichen Speicher geschrieben sind, können unglücklicherweise Software-Anwendungen, Betriebssysteme oder externe Agenten (z. B. Eingabe-Ausgabe-Vorrichtungen) auf die Speicherregion zugreifen und die Privatstatusdaten, die darin gespeichert sind, in unzulässiger Weise verwenden. Zugriff und Benutzung dieser Privatdaten durch nicht berechtigte Einheiten kann zu irrtümlichen und/oder unerwünschten Auswirkungen auf Prozessor- und Plattformhersteller und Endnutzer führen.

[0003] US 2001/0018736 A1 offenbart einen manipulationssicheren Mikroprozessor, der Kontextinformationen für ein Programm speichert, dessen Ausführung unterbrochen werden soll, wobei die Kontextinformationen einen Ausführungszustand des Programms und die Verschlüsselung des Ausführungscodes für dieses Programm anzeigen. Die Ausführung des Programms kann durch Rückgewinnen des Ausführungszustands des Programms aus der gespeicherten Kontextinformation neu gestartet werden.

[0004] US 5,655,125 offenbart ein Register zur Identifizierung von Prozessoreigenschaften, wie beispielsweise, ob der Prozessor statische logische Vorrichtungen umfasst. Die offenbarte Vorrichtung umfasst des Weiteren eine Übertragungsschaltung, um bei Empfang eines ersten Codes den Inhalt des Registers vom Prozessor zu einem System zu übertragen, das mit dem Prozessor gekoppelt ist.

[0005] Die Aufgabe der Erfindung besteht darin, ein Verfahren sowie eine Vorrichtung bereitzustellen, durch das beziehungsweise durch die in öffentlich zugängliche Speicher geschriebene Privat-Status-daten vor einer unzulässigen Verwendung geschützt sind.

[0006] Diese Aufgabe wird gelöst durch ein Verfahren mit den Merkmalen gemäß Anspruch 1, eine Datenverarbeitungsmaschine mit den Merkmalen gemäß Anspruch 12 und ein System mit den Merkmalen gemäß Anspruch 20.

KURZE BESCHREIBUNG DER FIGUREN

[0007] Die Ausführungsformen der Erfindung sind beispielhaft und nicht begrenzend in den Figuren der begleitenden Zeichnungen illustriert, wobei gleiche Bezugszeichen gleiche Elemente anzeigen. Es ist zu beachten, daß eine Bezugnahme auf „eine" Ausführungsform der Erfindung in dieser Offenbarung nicht notwendigerweise eine Bezugnahme auf dieselbe Ausführung bedeutet, und daß damit wenigstens eine Ausführungsform gemeint ist.

[0008] Fig. 1 zeigt ein Blockdiagramm eines Computersystems, das die öffentliche Speicherung von Privatstatusdaten verschleiern/codieren kann, gemäß einer Ausführungsform der Erfindung.

[0009] Fig. 2 zeigt ein Flußdiagramm, das ein Verfahren zum Lesen eines codierten Privatstatusdatenwerts von einer Privatstatusspeicherregion gemäß einer Ausführungsform der Erfindung darstellt.

[0010] Fig. 3 zeigt ein Flußdiagramm, das ein Verfahren zum Speichern eines codierten Privatstatusdatenwertes in einer Privatstatusspeicherregion gemäß einer Ausführungsform der Erfindung darstellt.

[0011] Fig. 4 zeigt ein Blockdiagramm eines Computersystems, wobei der Prozessor dazu entworfen ist, seine Privatstatusdaten, die in den öffentlichen Speicher geschrieben sind, zu verschleiern/codieren, gemäß einer Ausführungsform der Erfindung.

[0012] Fig. 5 zeigt beispielhafte Funktionskomponenten, die benutzt werden können, um das Verschleiern/Codieren der Privatstatusdaten zu implementieren, gemäß einer Ausführungsform der Erfindung.

[0013] Fig. 6 zeigt ein genaueres logisches Diagramm eines Teils einer beispielhaften Adreß-Verschleierungs/Codierungseinheit gemäß einer Ausführungsform der Erfindung.

GENAUE BESCHREIBUNG

[0014] Der Prozessorstatus, wie er während des Betriebs eines Prozessors in einen Speicher geschrieben wird, umfaßt zwei Arten von Information oder Daten. Eine Art wird hier als Architekturdaten bezeichnet, während die andere als implementierungsspezifische Daten bezeichnet wird (hier auch als „Privatdaten" oder „Privatstatusdaten" bezeichnet).

[0015] Architekturdaten sind Statusinformationen, die allen Prozessoren einer jeweiligen Klasse gemäß der Bestimmung durch den Hersteller gemeinsam sind, d. h. sie weisen im Wesentlichen dieselbe High-Level-Schnittstelle zwischen Hardware und Software auf. Diese Schnittstelle wird als die Instruction Set Architecture (ISA) bezeichnet. Die Benutzung derselben ISA für mehrere unterschiedliche Prozessorimplementierungen unterstützt die Fähigkeit der Software, die ausdrücklich für eine Implementierung geschrieben wurde, unmodifiziert auf späteren Implementierungen zu laufen.

[0016] Eine ISA definiert den Status, der für eine Software verfügbar ist, die auf dem Prozessor läuft, ihr Format und ihre Semantik, sowie verfügbare Betriebsschnittstellen (z. B. Instruktionen, Ereignisse). Ein Teil der ISA-Spezifizierung beschreibt, wie der Prozessor eine oder mehrere Regionen im Speicher der Maschine benutzen kann, um ihren Betrieb zu unterstützen. Diese Speicherregionen können für Software und andere Vorrichtungen der Datenverarbeitungsmaschine zugänglich sein, in der der Prozessor angeordnet ist (z. B. Eingabe-Ausgabe-Vorrichtungen usw.). Der Prozessor kann diese Speicherregionen benutzen, um sowohl Architektur- als auch Privatstatusdaten zu speichern.

[0017] Beispielsweise kann an Prozessoren gedacht werden, die die ISA des Intel® Pentium® Prozessors aufweisen, wie er von der Intel Corporation hergestellt wird, hier bezeichnet als die ISA IA-32. Der Prozessor kann Regionen im Speicher während bestimmter Vorgänge einsetzen. Wenn beispielsweise ein ISA-IA-32-Prozessor in den Systemmanagementmodus übergeht, speichert er verschiedene Werte in einer Speicherregion namens Systemmanagement-(SMM)-Status-Speicherbereich. Verschiedene Architekturdaten (z. B. die verschiedenen Maschinenregister wie z. B. ESI, EBP usw.) werden an Orten und in Formaten gespeichert, die in der Dokumentation für die ISA vorgegeben sind. Zusätzlich werden verschiedene Privatdaten in dem Systemmanagement-Statusspeicherbereich gespeichert. In der Dokumentation für die ISA sind diese Privatstatusbereiche als „reserviert" markiert; der Inhalt, das Format und die Semantik dieser Privatdaten sind nicht in der ISA-Dokumentation vorgegeben. Diese „reservierten" Speicherregionen werden hier als „Privatstatusregionen" bezeichnet.

[0018] Verschiedene Prozessoren können dazu entworfen sein, verschiedene Privatstatusdaten aufzuweisen, die hier auch als „Privatdaten" bezeichnet werden. Dies kann beispielsweise geschehen, um die Leistung zu verbessern oder Herstellungskosten zu senken. Beispielsweise können neue interne Register hinzugefügt werden, einige der alten können anders genutzt werden, und das Format oder der Ort ihres Inhalts, der in den Speicher geschrieben werden soll, können zwecks höherer Effizienz geändert werden. Als Resultat unterscheiden sich die Privatdaten für diese neueren Prozessoren in Inhalt, Format, Semantik oder Ort von denen der älteren Versionen.

[0019] Es kann zu Schwierigkeiten kommen, wenn Privatstatusdaten in öffentlich zugänglichen Bereichen wie z. B. dem Hauptspeicher oder einem anderen Speicher gespeichert werden. Hier ist es für Software wie z. B. das Basic Input Output System (BIOS), Betriebssysteme, den virtuelle Maschinenmanager oder Hypervisor, Gerätetreiber oder Applikationen und Hardware wie E/A-Vorrichtungen oder andere externe Agenten möglich, auf die Privatstatusdaten zuzugreifen (d. h. sie zu lesen und/oder zu schreiben). Die Benutzung dieser Privatstatusdaten durch solche Einheiten kann zu irrtümlichen und/oder unerwünschten Auswirkungen auf Prozessor- und Plattformhersteller und Endnutzer führen. Wenn eine Applikation beispielsweise von bestimmten Privatstatusdaten abhängig ist, die in einer Prozessorimplementierung verfügbar sind, könnte die Applikation fehlerhaft arbeiten, wenn sie auf einer unterschiedlichen Prozessorimplementierung läuft, die die Privatstatusdaten anders (oder gar nicht) implementiert. Software, die von Privatdaten abhängig ist, kann auch aufgrund interner Prozessor-Speicher-Kohärenzverhaltensweisen bzw. -richtlinien ausfallen, die von Implementierung zu Implementierung variieren. Softwareabhängigkeit von Privatstatusdaten kann dem Prozessorhersteller zur Verfügung stehende Implementierungsalternativen hinsichtlich der Privatstatusdatennutzung erschweren und/oder einschränken. Aus diesem Grund dokumentieren Prozessorhersteller oft diese Privatstatusdaten (und ihre Speicherung in einer Speicherregion) als RESERVIERT, wodurch sie anzeigen, daß diese in zukünftigen Implementierungen verändert werden können.

[0020] Die oben genannte Möglichkeit, alte Software auf neueren Maschinen laufen zu lassen, geht davon aus, daß die alte Software auf die Privatdaten einer Maschine (die bei einer neueren Version der Maschine unterschiedlich sein können) nicht falsch zugegriffen hat. Allerdings wurde festgestellt, daß die Softwareentwickler Applikationen und Betriebssystemprogramme schreiben, die das Gegenteil tun, d. h., sie greifen auf Privatdaten, so wie sie z. B. im Hauptspeicher gespeichert sind zu und verlassen sich auf sie. Dies erzeugt ein Problem, da ältere Software auf einer neueren Maschine möglicherweise nicht korrekt läuft, obwohl die neuere Maschine dieselben Architekturdaten aufweist wie die ältere Maschine und immer noch die älteren Mechanismen zum Zugriff auf gespeicherte Statusdaten (z. B. Lade- und Speicherinstruktionen gemäß der ISA) „verstehen" kann. Dies liegt daran, daß bei der neueren Maschine ein Teil oder die Gesamtheit der Privatdaten geändert worden sein kann, was dazu führt, daß die Software nicht richtig arbeitet. Zusätzlich kann der

Hersteller zögern, zukünftige Versionen seines Prozessors mit Verbesserungen auszustatten, da dies Kompatibilitätsprobleme mit älterer Software verursachen könnte.

[0021] Gemäß einer Ausführungsform der Erfindung sind eine Datenverarbeitungsmaschine und ein Verfahren zum Betreiben beschrieben, die einen Softwareentwickler davon abhalten können, Software zu schreiben, die von Privatstatusdaten (z. B. einem bestimmten Wert, seinem Ort, seiner Semantik oder seinem Format) abhängig ist, die in einer öffentlich zugänglichen Region des Speichers gespeichert sind. Dies kann es zukünftigen Versionen der Maschine erlauben, ein andersartiges Verhalten hinsichtlich privater Daten an den Tag zu legen, was erforderlich sein kann, da sich das interne Hardwaredesign der Maschine weiterentwickelt, während sie immer noch dieselbe ISA aufweisen, die benötigt wird, um ältere Software lauffähig zu machen.

[0022] Einige Ausführungsformen der Erfindung können die Anwendung von Architekturschnittstellen zu Daten fördern, die als Privatstatusdaten gespeichert sind. Beispielsweise können Instruktionen vorgesehen sein, um auf Daten zuzugreifen, die als Privatstatusdaten gespeichert sein können, indem die Identität der Daten spezifiziert wird, auf die zugegriffen werden soll, anstelle des Speicherortes der Daten in der Privatstatusdatenregion. Dies erlaubt dahingehend eine Implementierungsfreiheit, wie die Daten gespeichert sind (z. B. innerhalb der Privatstatusregion), während ein Architekturmechanismus zum Zugriff auf die Daten ausgebildet ist. Es wird beispielsweise angenommen, daß ein Datenelement, das in dem Systemmanagement-Statusspeicherbereich der ISA IA-32 (wie oben beschrieben) gespeichert ist, der Wert der CS-Segment-Basisadresse ist. Der Speicherort dieses Datenelements innerhalb des Statusspeicherbereichs ist nicht in der ISA-Spezifikation angegeben. Statt dessen kann von der ISA eine Instruktion bereitgestellt werden, welche die Daten indirekt adressiert. Das Datenelement kann in jeder möglichen Art und Weise in dem Statusspeicherbereich codiert und gespeichert sein, die für eine Prozessorimplementierung gewünscht wird (oder es muß überhaupt nicht in dem Statusspeicherbereich gespeichert sein und kann statt dessen beispielsweise in einem speziellen Register oder einem Ort im Prozessor angeordnet sein).

[0023] Die Erfindung erlaubt es, daß Privatstatusdaten so codiert werden, daß eine schnelle Software-Decodierung der Daten vereitelt wird, im Gegensatz zu den vorgegebenen Architekturschnittstellen. Ausführungsformen der Erfindung können die Codierungskomplexität je nach Zielprozessor oder -plattform variieren. Sobald der Zielprozessor bekannt ist, kann ein Fachmann eine Ausführungsform der Erfindung wählen, die sicherstellt, daß softwarebasierte Verfahren zum Decodieren der gewählten Codierung mehr Zeit benötigen als das Benutzen der vorgegebenen Schnittstellen (z. B. Instruktionen). Beispielsweise können nicht vorgegebene Softwareverfahren dazu in der Lage sein, bestimmte Privatstatusdaten in **400** Takten zu decodieren (z. B. durch die Benutzung bestimmter Instruktionen und Algorithmen), während von der Architektur vorgegebene Instruktionen und Verfahren nur einen Bruchteil dieser Zeit benötigen würden. Eine Ausführungsform der Erfindung liegt in der Benutzung bestimmter Meßwerte, um die Kosten der Privatstatus-Decodierung zu messen, darunter beispielsweise die Meßwerte der Zeit (Geschwindigkeit) und des Leistungsverbrauchs.

[0024] Der hier verwendete Begriff „Codieren" umfaßt Konzepte wie Verschlüsselung, Chiffrierung, Formatierung oder die Zuweisung oder Interpretation spezifischer Bitmuster. Von Codierungen durch Ausführungsformen dieser Erfindung wird hier als die Privatdaten „verschleiernd" gesprochen.

[0025] Bezugnehmend auf Fig. 1 ist ein Blockdiagramm eines Computersystems dargestellt. Software **120** läuft auf einer Plattform-Hardware **102**. Die Plattform-Hardware **102** kann ein Personal-Computer (PC), ein Großrechner, ein tragbares Gerät, ein tragbarer Computer, ein Digitalempfänger oder jedes andere Rechnersystem sein. Die Plattform-Hardware **102** umfaßt einen Prozessor **110**, Speicher **130** und kann eine oder mehrere Eingabe-Ausgabe-(E/A)-Vorrichtungen **140** aufweisen.

[0026] Der Prozessor **110** kann jede Art von Prozessor sein, der dazu in der Lage ist, Software auszuführen, wie z. B. ein Mikroprozessor, ein Digitalsignalprozessor, ein Mikrocontroller oder ähnliches. Der Prozessor **110** kann einen Mikrocode, eine programmierbare Logik oder eine fest programmierte Logik zum Durchführen der Ausführung bestimmter Verfahrensausführungsformen der vorliegenden Erfindung aufweisen. Obwohl Fig. 1 nur einen solchen Prozessor **110** zeigt, kann mehr als ein Prozessor in dem System vorgesehen sein.

[0027] Die E/A-Vorrichtung oder E/A-Vorrichtungen **140** können beispielsweise Netzwerkschnittstellenkarten, Kommunikationsports, Videosteuerungen, Festplattensteuerungen, Systembusse und -steuerungen (z. B. PCI, ISA, AGP) oder Vorrichtungen sein, die in die Plattform-Chipsatzlogik oder den Prozessor integriert sind (z. B. Echtzeittaktuhren, programmierbare Taktgeber, Leistungszähler). Einige oder alle E/A-Vorrichtungen **140** können eine Direktspeicherzugriff-(DMA)-Fähigkeit besitzen, was es ihnen ermöglicht, den Speicher **130** unabhängig von der Steuerung durch den Prozessor **110** oder die Software **120** zu lesen und/oder zu beschreiben.

[0028] Der Speicher **130** kann eine Festplatte, eine Diskette, ein Schreib-Lese-Speicher (RAM), ein Zwischenspeicher, ein Lese-Speicher (ROM), ein Flash-Speicher, ein statischer Schreib-Lese-Speicher (SRAM), eine Kombination der genannten Vorrichtungen oder jede andere Art von Speichermedium sein, auf die der Prozessor **110** zugreifen kann. Der Speicher **130** kann Instruktionen und/oder Daten zum Durchführen der Verfahrensausführungsformen der vorliegenden Erfindung speichern. Der Speicher **130** kann ein öffentlich zugänglicher Bereich einer Registerdatei des Prozessors sein, oder er kann ein Bereich außerhalb des Prozessors sein, wie z. B. der Hauptspeicher.

[0029] Daten zum Status der Maschine **112**, wie z. B. der Inhalt bestimmter interner Register **114**, wird in einen Privatstatusbereich **132** im Speicher **130** geschrieben, wo die geschriebenen Statusdaten „codiert" oder „verschleiert" werden. Obwohl also der Ort, an den die Statusdaten geschrieben werden, öffentlich ist, indem von den E/A-Vorrichtungen oder der Software **120** (z. B. dem Betriebssystem **122** oder Applikationssoftware **124**), die auf der Plattform-Hardware **102** läuft, auf ihn zugegriffen werden kann, macht es die Codierung schwierig, die Statusdaten in einem angemessenen Zeitrahmen zurückzuentwickeln (d. h. zu decodieren). Wenn die Statusdaten aus dem Speicher **130** abgerufen werden sollen, wird ein vorgegebener Decodierungsprozeß, z. B. ein vom Prozessor initiierter Decodierungsprozeß, der von dem Prozessorhersteller definiert wird, angewandt. Die Kontrolle über den Decodierungsprozeß kann mit bestimmten Prozessorfunktionen verbunden sein, wie z. B. spezifischen Instruktionen und Steuersignalen, wie weiter unten erläutert werden soll. Nicht vorgegebene Verfahren (alternative Softwareinstruktionen und Algorithmen) zum Zugreifen auf Statusdaten würden diese Steuerungen nicht aktivieren und könnten deshalb kostenintensiver sein.

[0030] Die abgerufenen Daten können dann in den lokalen Status **112** versetzt werden, der für Software **120** oder die E/A-Vorrichtungen **140** zugänglich sein kann oder nicht. Der lokale Status **112** können beispielsweise eine Region in einem internen Zwischenspeicher oder Register sein, die nicht für unkontrollierten Zugriff über die Instruction Set Architecture (ISA) zur Verfügung stehen. In einigen Fällen kann über Software oder andere externe Agenten (z. B. E/A-Vorrichtungen) nicht auf den lokalen Status zugegriffen werden. In einigen Fällen ist der lokale Status ganz oder teilweise für Software oder andere externe Agenten zugänglich. In anderen Fällen kann auf den lokalen Status indirekt über spezielle Schnittstellen (z. B. Instruktionen) zugegriffen werden. Da der lokale Status intern im Prozessor und im „öffentlichen" Speicher angeordnet ist, kann der Prozessor den Zugriff auf den lokalen Status strikt diktieren.

[0031] Obwohl die Statusdaten, die in den öffentlich zugänglichen Bereich des Speichers **130** geschrieben sind, in einer codierten Form vorliegen, kann von der Software eine herstellerdefinierte Instruktion benutzt werden, die Teil der ISA für den Prozessor sein kann, um die Daten aus dem Speicher **130** abzurufen. Die Codierung sollte stark genug sein, um Softwareentwickler davon abzuhalten, eine solche Instruktion zu umgeben, wenn sie Zugriff auf die Statusdaten suchen. Ein Beispiel der internen Logik, die zum Lesen oder Abrufen der Statusdaten unter Benutzung eines Mikrooperations- oder Hardware-Steuersignals aus dem Speicher benötigt wird, soll weiter unten unter Bezugnahme auf **Fig. 5** beschrieben werden.

[0032] In einer Ausführungsform muß der benutzte Codierungsprozeß nur stark genug sein, um einen Autor der Software **120** dazu zu veranlassen, beim Schreiben der Software ein Verfahren anzuwenden, das von einem Hersteller des Prozessors zum Zugreifen auf die Statusdaten von dem Speicher vorgegeben ist, anstatt das Verfahren zu umgehen. In anderen Fällen kann die Codierung stärker sein, wenn der Hersteller es dem Softwareentwickler erschweren möchte, auf die in dem Speicher vorhandenen Statusdaten (einschließlich bestimmter Werte, ihres Ortes, ihrer Semantik oder ihres Formats) zuzugreifen und sich auf diese zu verlassen.

[0033] Steuersignale, die benutzt werden, um die Codierung und Decodierung der Privatstatusdaten zu steuern, können an Hardware-Statusmaschinen, Prozessorinstruktionen (auch bekannt als Makroinstruktionen), Betriebsmodi (z. B. PAL-Modi) oder Mode-Bits oder Instruktionsoperationsgruppen, Mikrocodes oder Mikrocode-Operationen (UOPs) und Hardware-Steuersignale oder Ereignisse gekoppelt oder für diese zugänglich sein.

[0034] Es können verschiedene Arten von Codierungsprozessen benutzt werden. Die Daten, die in die Privatstatusregion des Speichers geschrieben werden, können vor der Speicherung verändert werden. Diese Art des Codierungsprozesses wird als Datencodierung bezeichnet. Alternativ können die Adressen verändert werden, die benutzt werden, um auf Privatstatusdaten in Privatstatusregionen zuzugreifen. Diese Art Codierungsprozeß wird als Adreßverschleierung bezeichnet, und die Umwandlung von der Originaladresse in die verschleierte Adresse wird als Adressenzuordnung bezeichnet. Datencodierung und Adreßverschleierung werden im folgenden beschrieben.

[0035] Codierungsprozesse können entweder statisch oder dynamisch sein. Statische Codierungen verändern sich nicht im Laufe der Zeit, während die Maschine läuft (und den Codierungsprozeß durchführt). (Statische Codierungen können während der

Initialisierung bzw. der Zurücksetzung des Prozessors oder während der Startphase neu konfiguriert werden, jedoch nicht später während des Betriebs.) Ein Prozeß, der statische Codierungen erzeugt, wird als eine statische Verschleierung bezeichnet.

[0036] Beispielsweise kann sich ein Speicherformat des Inhalts eines bestimmten Elements von Privatstatusdaten, wie es in den Speicher **130** geschrieben ist, verändern, während die Maschine ausführt. Dies wird als dynamische Verschleierung bezeichnet. Beispielsweise kann das Format immer dann nach einer zufälligen oder pseudozufälligen Sequenz (die der Prozessor erzeugt und verfolgt) zwischen Big-Endian und Little-Endian wechseln, wenn die Statusdaten in den Speicher geschrieben werden müssen; dieser Wechsel kann nur die Speicherregion(en) beeinflussen, aus der oder denen die Privatdaten abgelesen oder in die sie hineingeschrieben werden. Wiederum besteht hier die Absicht, das schnelle Rückentwickeln und Decodieren der Statusdaten aus einer Speicherregion in Speicher **130** zu erschweren, die öffentlich zugänglich ist.

[0037] In einer Ausführungsform werden Privatstatusdaten, wenn sie in einen Speicher geschrieben werden sollen, in eine Speicherregion (z. B. den Hauptspeicher) geschrieben, der zusammenhängende Adressen aufweist. In anderen Ausführungsformen ist der Privatdatenbereich nicht zusammenhängend und besteht aus mehr als einer gesonderten Speicherregion. Es ist nicht nötig, daß die Codierung die Privatstatusregion vollständig besetzt; d. h., einige Bits oder Bytes bleiben ungenutzt. Es kann ein gewisses Maß an Freiheit beim Entwerfen der Codierungs- und/oder Verschleierungsfunktionen erreicht werden, indem die Größe der Privatstatusregion geändert wird, indem sie beispielsweise größer ausgebildet wird, als dies strenggenommen zum Speichern der Privatdaten notwendig wäre. (Dies würde beispielsweise, wie später beschrieben, die Benutzung größerer Polynome im MISR (Multiple Input Shift Register) erlauben.)

[0038] In einer Ausführungsform wird ein Multibyte-Wert (z. B. eine 32-Bit „lange” ganze Zahl) von Statusdaten in mehrere Teile aufgeteilt, die dann statt in einer Sequenz an nicht zusammenhängenden Orten gespeichert werden. Auf diese Weise kann ein 4-Byte-Wert in vier 1-Byte-Werte aufgeteilt werden, die an nicht zusammenhängenden Orten in einer Privatstatusregion gespeichert werden. Die Orte, an denen die vier 1-Byte-Werte gespeichert sind, können einem dynamischen und zufälligen Wechsel unterliegen, während die Maschine arbeitet. Natürlich sollten die Ausführungsformen auch dazu in der Lage sein, solche Daten zu lokalisieren und zu decodieren. Es ist zu beachten, daß die ISA bestimmte Anforderungen hinsichtlich der Atomarität der Zugriffe aufstellen kann, für den Fall, daß einzelne Datenwerte unter Benutzung multipler Speicherzugriffe gespeichert oder geladen werden.

[0039] In einer Ausführungsform der Erfindung werden die Adreßbits codiert, die zum Speicherzugriff benutzt werden. Diese Codierung von Adreßbits kann die Reihenfolge der Adreßbits (oder der Adreßbitgruppen) verändern. Ein Beispiel dafür kann das Umstellen vom Little-Endian-Format auf ein Big-Endian-Format innerhalb eines bestimmten Speicherbereichs sein. Andere Adreßmischzuordnungen sind möglich, wobei einige komplexere Umwandlungen umfassen.

[0040] Eine andere Art Adreßcodierung ordnet einen Satz von einzigartigen Adressen K einem anderen Satz von einzigartigen Adressen K zu; das heißt, daß das Zuordnen mathematisch gesehen bijektiv ist (sowohl injektiv (eins zu eins) als auch surjektiv (zu)). Hier können die oberen Adreßbits unverändert bleiben, während die unteren Adreßbits modifiziert werden. In solchen Fällen ist es möglich, Zuordnungen zu bilden, die einen bestimmten Speicherbereich sich selbst zuordnen. Das heißt, die Basis-Adreßverschiebung des Speicherabschnitts ist dieselbe, und die Größe der Speicherregion ist dieselbe. Dies ist eine attraktive Lösung, da nur die Daten innerhalb des Speicherabschnitts „verschleiert” werden. Das heißt, nur die Adreßbits innerhalb des Speicherabschnitts werden gemischt. Fig. 4 und Fig. 6 stellen Beispiele einer solchen Zuordnung und zugehöriger Adreßmischmechanismen dar.

[0041] Adreßverschleierungsmechanismen sind leichter benutzbar, wenn die Privatstatusregionen eine Größe oder eine Basisadresse aufweisen, die Potenzen der zugrundeliegenden N-ären Logik sind. Die meisten gegenwärtigen Prozessoren benutzen Binärlogik, weshalb Privatstatusregionen mit einer Größe oder Basisadresse, die eine Potenz von 2 ist, zu bevorzugen sind. (Es werden hier Binärlogik und -arithmetik behandelt, doch können bei Bedarf auch N-äre Logik und Arithmetik benutzt werden und werden im Allgemeinfall angenommen.) Filter und andere Mechanismen können benutzt werden, um Privatstatusregionen mit einer Größe oder Basisadresse zu verwalten, die nicht Potenzen der N-ären Logik sind. Solche Adreßbit-Manipulationen können neben verschiedenen Speicheranordnungen und virtuellen Speicherverfahren (z. B. Seitenorientierung, Segmentierung usw.) existieren.

[0042] Adreßverschleierungsmechanismen können die Datenanordnung im Speicher verändern und dazu dienen, die Daten zu mischen, bisweilen jedoch nur mit einer Granularität des Speichers. Bei den meisten gegenwärtigen Prozessoren ist der Hauptspeicher Byteadressierbar, weshalb der Ort individueller Bytes eines Datenelements innerhalb der Privatstatusregion neu angeordnet werden kann, wo-

bei jedoch die Datenbits innerhalb individueller Bytes nicht durch die Adreßverschleierung verändert werden (obwohl sie durch Datencodierungsmechanismen verändert werden können).

[0043] Bei diesen Ausführungsformen der Adreßzuordnung können die ursprünglichen Adreßzuordnungen durch einen Decodierungsprozeß extrahiert werden. Diese Extraktion ist die Anwendung einer Umkehrfunktion der Adreßzuordnungsfunktion. Die Auswahl der Zuordnungsfunktion kann unter Berücksichtigung dieser Anforderung erfolgen, da nicht alle Adreßzuordnungsfunktionen umkehrbar sind.

[0044] Eine Ausführungsform der Erfindung codiert die Datenbits, die in den Speicher geschrieben werden. Diese Datencodierungen können die Daten umformen, die in der Privatstatusregion gespeichert werden, ohne notwendigerweise von Adressierbarkeitsbeschränkungen eingeschränkt zu werden, wie z. B. der Größe des adressierbaren Speichers. Datensegmente können mit anderen Datensegmenten vertauscht werden. Beispielsweise können zwei Halbbytes (d. h. 4-Bit-Segmente in einem Byte) innerhalb jedes Byte vertauscht werden. Datencodierungen können bitweise exklusiv-OR innerhalb einer konstanten XOR-Maske erfolgen. Daten können auch bitweise exklusiv-OR mit der Ausgabe eines MISR (Multiple Input Feedback Shift Register) codiert werden. Datencodierungen können unter Benutzung einer kryptographischen Funktion erfolgen. In diesen Ausführungsformen können die Originaldaten durch einen Decodierungsprozeß extrahiert werden. Dieser Decodierungsprozeß sollte sicherstellen, daß er schneller ist als Decodierungsverfahren, die der Software zur Verfügung stehen, die auf der Plattform läuft (z. B. Benutzung von ISA-definierten Lade- und Speicheroperationen, mathematische Operationen usw.). Die Tabellen **470** und **480** aus Fig. 4 zeigen beispielsweise die Benutzung eines Vigenère-artigen Chiffre, der auf Daten (Bytes) innerhalb eines bestimmten 16-Byte-Abschnitts der Speicheradressen angewandt wird.

[0045] Einige der oben genannten Ausführungsformen können mit statischen Zuordnungen implementiert werden. Das heißt, sie verändern sich während der Betriebszeit von Prozessor oder Plattform nicht. Geeignete Zuordnungen können zum Zeitpunkt des Entwurfs, während der Herstellung, nach der Herstellung oder zu einem frühen Zeitpunkt des Systembetriebs (z. B. während des Systemstarts, beim Einschalten des Systems, bei Zurücksetzen des Prozessors) eingestellt werden. Verschiedene Prozessoren können, müssen aber nicht mit denselben statischen Zuordnungen konfiguriert sein. Wenn Zuordnungen nicht festgelegt sind, bis das System betriebsbereit ist (z. B. zum Systemstart), ist es möglich, bei jedem Prozessorstart eine neue Zuordnung zu wählen. In einer Ausführungsform können verschiedene Steuerungsgruppen (z. B. Betriebsmodi, Gruppen von Instruktionen) jeweils eine unterschiedliche Zuordnungskonfiguration benutzen. Innerhalb einer Steuerungsgruppe bleiben die Zuordnungen konstant. Zwischen Instruktionsgruppen oder Modi können die Zuordnungen jedoch unterschiedlich sein (müssen dies aber nicht).

[0046] Andere Ausführungsformen können mit dynamischen Zuordnungen implementiert sein, die sich verändern, während der Prozessor arbeitet. In einer Ausführungsform können sich Zuordnungskonfigurationen nur dann ändern, wenn gegenwärtig keine verbleibenden codierten Daten in einer Privatstatusregion des Speichers gespeichert sind. Diese Ausführungsform kann einen Zähler benutzen, der erhöht wird, wenn codierte Daten in eine Privatstatusregion des Speichers geschrieben werden, sodaß diese aktiviert wird. Der Zähler wird herabgesetzt, wenn die Privatstatusregion nicht länger als aktiv betrachtet wird. Wenn der Zähler null ist, kann die Zuordnungskonfiguration geändert werden. In einer Ausführungsform wird die Zuordnungskonfiguration für jede Privatstatusregion in einem Zuordnungsdeskriptor gespeichert. Der Zuordnungsdeskriptor kann an einem bekannten, nicht codierten Ort innerhalb der Privatstatusregion selbst gespeichert oder separat durch eine Verfolgungsstruktur wie z. B. eine Warteschlange oder eine Suchtabelle verwaltet werden, die innerhalb oder außerhalb des Prozessors vorgesehen sein kann. In einer Ausführungsform sind unterschiedliche Zuordnungen für jede Privatstatusregion möglich.

[0047] Fig. 2 zeigt Prozeß **200** zum Lesen eines codierten Privatstatus-Datenwerts aus einer Privatstatus-Speicherregion gemäß einer Ausführungsform der Erfindung. Der Prozeß kann durch eine Verarbeitungslogik durchgeführt werden, die Hardware (z. B. Schaltungen, dedizierte Logik, programmierbare Logik, Mikrocode usw.), Software (wie sie beispielsweise auf einem allgemein einsetzbaren Computersystem oder einer dedizierten Maschine läuft), oder eine Kombination der beiden umfassen. In einer Ausführungsform ist die Verarbeitungslogik in Prozessor **110** aus Fig. 1 implementiert.

[0048] Bezugnehmend auf Fig. 2 beginnt Prozeß **200** damit, daß die Verarbeitungslogik eine Adresse für das Datenelement bestimmt (Verarbeitungsblock **202**). Als nächstes bestimmt die Prozeßlogik, ob das Datenelement in codierter Form in einer Privatstatusregion des Speichers gespeichert ist (Verarbeitungsblock **204**).

[0049] Eine Ausführungsform der Erfindung benutzt ein Mikrocode-erzeugtes oder Hardware-erzeugtes Steuersignal, das der Verarbeitungslogik anzeigt, daß das abgefragte Datenelement einer Decodierung bedarf. Eine Abwesenheit dieses Signals führt dazu, daß der NEIN-Pfad zu Block **250** genommen wird.

[0050] Wenn das Datenelement nicht decodiert werden soll, geht die Verarbeitungslogik zu Verarbeitungsblock **250** über, wo sie ein Datenelement aus dem Speicher unter der Adresse lädt, die im Verarbeitungsblock **202** bestimmt wurde. Der Prozeß kann dann enden. Die geladenen Daten werden nicht decodiert; das heißt, keine Adreß- oder Datendecodierung wird durchgeführt. Es ist zu beachten, daß die Daten, die auf diesem Pfad gelesen werden, gewöhnlicher Art sein können (d. h. keine Privatstatusdaten sind), oder Privatstatusdaten in codierter Form sein können (auf die aber in einer nicht vorgegebenen Art und Weise zugegriffen wurde).

[0051] Wenn allerdings das Datenelement decodiert werden soll, bestimmt die Verarbeitungslogik als nächstes anhand der Adresse, die in Verarbeitungsblock **202** bestimmt wurde, die Adresse, unter der es gespeichert ist (die Adresse kann verschleiert sein) (Verarbeitungsblock **210**). Die Verarbeitungslogik lädt als nächstes das codierte Datenelement aus dem Speicher unter der Adresse, die in Verarbeitungsblock **210** bestimmt wurde (Verarbeitungsblock **220**). Die Verarbeitungslogik decodiert dann das Datenelement, das in Verarbeitungsschritt **220** aus der Privatstatusregion des Speichers geladen wurde (Verarbeitungsblock **230**). Der decodierte Wert ist ein Ergebnis von Prozeß **200**. Der Prozeß kann dann enden. Oft wird der decodierte Zustand in einem Privatstatus-Zwischenspeicher oder dem privaten lokalen Zustand des Prozessors abgelegt.

[0052] Fig. 3 zeigt Prozeß **300** zum Speichern eines Privatstatus-Datenwerts in einer Privatstatusregion des Speichers gemäß einer Ausführungsform der Erfindung. Der Prozeß kann durch eine Verarbeitungslogik durchgeführt werden, die Hardware (z. B. Schaltungen, dedizierte Logik, programmierbare Logik, Mikrocode usw.), Software (wie sie beispielsweise auf einem allgemein einsetzbaren Computersystem oder einer dedizierten Maschine läuft), oder eine Kombination der beiden umfassen. In einer Ausführungsform ist die Verarbeitungslogik in Prozessor **110** aus Fig. 1 implementiert.

[0053] Bezugnehmend auf Fig. 3 beginnt Prozeß **300** damit, daß die Verarbeitungslogik einen Datenwert und eine Speicheradresse eines Datenelements bestimmt (Verarbeitungsblock **302**). Als nächstes bestimmt die Verarbeitungslogik, ob das zu speichernde Datenelement ein Privatstatuselement ist, das in codierter Form in einer Privatstatusregion des Speichers gespeichert werden soll (Verarbeitungsblock **304**).

[0054] Eine Ausführungsform der Erfindung benutzt ein Mikrocode-erzeugtes oder Hardware-erzeugtes Steuersignal, um der Verarbeitungslogik anzuzeigen, daß das geschriebene Datenelement einer Codierung bedarf. Eine Abwesenheit dieses Signals führt dazu, daß der NEIN-Pfad zu Block **350** eingeschlagen wird.

[0055] Wenn das Datenelement nicht in codierter Form in einer Privatstatusregion gespeichert werden soll, geht die Verarbeitungslogik zu Verarbeitungsblock **350** über, wo sie das Datenelement in uncodierter (unmodifizierter) Form im Speicher unter der Adresse speichert, die im Verarbeitungsblock **302** bestimmt wurde. Der Prozeß kann dann enden. Die geschriebenen Daten werden nicht codiert.

[0056] Wenn allerdings das Datenelement in codierter Form in einer Privatstatusregion gespeichert werden soll, codiert die Verarbeitungslogik als nächstes das Datenelement (Verarbeitungsblock **310**) und bestimmt eine verschleierte Adresse, unter der das Datenelement zu speichern ist (Verarbeitungsblock **320**). Die Verarbeitungslogik speichert dann das nun codierte Datenelement im Speicher unter der Adresse, die in Verarbeitungsblock **320** bestimmt wurde (Verarbeitungsblock **330**). Der Prozeß kann dann enden.

[0057] Es ist zu beachten, daß die Verarbeitung, die in Verarbeitungsblock **310** und Verarbeitungsblock **320** durchgeführt wird, auch in umgekehrter Reihenfolge durchgeführt werden kann, d. h. die Verschleierung des Adreßwertes erfolgt vor der Codierung der Daten. Einige Ausführungsformen führen nur einen und nicht beide Verarbeitungsblöcke durch. Einige Ausführungsformen können die Verarbeitungsblöcke parallel ausführen.

[0058] Bezugnehmend auf Fig. 4 ist ein Computersystem **402** in Form eines Blockdiagramms dargestellt. Dieses System **402** weist einen Prozessor **404** auf, der dazu entworfen ist, die Methodologie zu unterstützen, die oben zum Verschleiern der Privatstatusdaten im Speicher beschrieben wurde. Der Prozessor **404** weist einen Standardzwischenspeicher **410** und einen Privatzwischenspeicher **416** auf, wobei auf den letzteren auf dem System **402** ausgeführte Software nicht zugreifen kann und er dazu benutzt wird, die Privatstatusdaten in einer nicht codierten (nicht verschleierten Form) zu speichern. In dieser Ausführungsform ist ein System-Chipsatz **406** vorgesehen, um es dem Prozessor **404** zu ermöglichen, mit dem Speicher **408** zu kommunizieren. Der Chipsatz **406** kann eine Speichersteuerung (nicht dargestellt) sowie sonstige Logik umfassen, die benötigt wird, um eine Schnittstelle zu den Peripheriegeräten eines Computers (ebenfalls nicht dargestellt) herzustellen. In einigen Ausführungsformen kann die Funktionalität des Chipsatzes **406** oder ein funktionaler Untersatz im Prozessor **404** implementiert sein.

[0059] In Fig. 4 ist der Speicher **408** gezeigt, wie er in einer öffentlich zugänglichen Region **418** die codierten Privatstatusdaten des Prozessors **404** spei-

chert. Dies ist ein Beispiel, wobei eine Chiffre auf die Werte des internen Prozessorstatus des Prozessors **404** angewandt wurde, sodaß die tatsächlichen Werte nicht durch einfaches Überwachen und Lesen des Speichers **408** leicht abgerufen oder zurückentwickelt werden können.

[0060] Wie oben beschrieben, kann die Verschleierung von Daten, die in der codierten Privatstatusregion **418** gespeichert sind, auf verschiedenen Wegen erzielt werden. Fig. 4 zeigt ein Beispiel eines solchen Mechanismus, wobei sowohl die Datenwerte codiert werden als auch das Datenlayout codiert/verschleiert wird. Erste Datenwerte in Tabelle **470** sind mit Hilfe einer Vigenère-Chiffre codiert, wodurch sich die Datenwerte ergeben, die in Tabelle **480** (im Folgenden beschrieben) gezeigt sind. Dann wird eine spezielle Zuordnung von logischen Adreßwerten der Privatstatusdaten zu physikalischen Adreßwerten angewandet, wobei die Zuordnungsresultate in Tabelle **490** dargestellt sind. Die physikalischen Adressen geben vor, wo die Privatstatusdaten tatsächlich im Speicher gespeichert werden. Von den physikalischen Adressen kann also gesagt werden, daß sie aus einer Codierung der logischen Adressen resultieren.

[0061] Die Tabelle **470** in Fig. 4 namens „dechiffrierte Adreßdaten" weist eine Liste von Beispielen logischer Adressen und ihrer zugehörigen Privatstatusdatenwerte auf, die in nicht codierter Form in dem Zwischenspeicher **416** abgelegt sind. Hier wurden nur Null-Datenwerte ausgewählt, um die resultierende Codierung zu demonstrieren. Es ist zu beachten, daß ein „X" die nicht codierten oberen Bits der virtuellen und physikalischen Adressen der Statusdaten repräsentiert. Die Tabelle **490** namens „Privatstatus-Speicheradreßkarte" zeigt ein Beispiel für die Zuordnung zwischen nicht codierten und codierten Adressen. Hier sind nur die unteren 4 Bits codiert.

[0062] Fig. 6 zeigt eine Ausführungsform einer programmierbaren (parametrisierten) Adreßzuordnungsfunktion, die in dem System aus Fig. 4 benutzt werden kann. In Fig. 6 würde das Polynom-Steuerregister **604** mit $P_0 = 1$, $P_1 = 1$, $P_2 = 0$, $P_3 = 0$ geladen, um das primitive Polynom $x^4 + x^1 + x^0$ zu implementieren und das optionale Maskenregister **610** mit allen Nullwerten zu laden. Diese Logik ist eine Anwendung der Gleichungen, die allgemeine MISRs mit einer Bitbreite w bestimmen, und kann benutzt werden, um verschiedene Arten von Adreßcodierungs-Kombinationslogik zu konstruieren. Die parametrisierten MISR-Statusgleichungen sind:

$$S_i(t + 1) = S_{i-1}(t) + I_i + (P_i \cdot S_{w-1}(t)),$$

$$1 \leq i \leq w - 1$$

$$S_0(t + 1) = I_0 + (P_0 \cdot S_{w-1}(t))$$

[0063] Hier repräsentiert der Operator „+" eine Modulo-2-Addition (XOR) und „·" repräsentiert eine Modulo-2-Multiplikation (AND). Parameter „t" repräsentiert Zeit (Takte), S, den Status des i-ten Flipflop, I_i, das i-te Inputvektor-Bit, und P_i den i-sten Polynomkoeffizienten. Der Koeffizient P_w ist implizit 1. Um die Adreßmischausführungsform aus Fig. 4 zu erreichen, sind alle $S_i(t)$ mit den entsprechenden Adreßwerten A_i und $S_i(t + 1)$ mit Ausgabe O_i zu ersetzen. Andere Ausführungsformen sind möglich.

[0064] Primitive Polynome der Ordnung w sind insofern nützlich, als sie eine „maximale Sequenz" erzeugen können; das heißt, sie können alle Binärkombinationen oder -muster mit der Bitbreite w erzeugen. Es können primitive Polynome von bis zu **300** (Bitbreite **300**) oder sogar höherer Ordnung benutzt werden.

[0065] Um unter Benutzung von Fig. 4 die oben gezeigte Funktion zum Zugreifen auf Daten an der logischen Adreßverschiebung 0001 (wie in Eintrag **471** gezeigt) zu erläutern, wird auf den physikalischen Speicher an Ort 0010 zugegriffen (wie Eintrag **491** zeigt). Der zu dieser Adresse gehörende nicht codierte Inhaltswert (siehe Eintrag **471**) besteht in diesem Fall ausschließlich aus Nullen. Wenn er allerdings, wie in Eintrag **481** gezeigt, in codierter Form gespeichert ist, erscheint eine Nicht-Null-Reihe (d. h. 11110101) in der öffentlichen Region **418** des Speichers **408**. (Dieser Codierungschiffre wird an späterer Stelle genauer beschrieben.) Obwohl aus praktischen Gründen beschränkte Bitbreiten gezeigt sind, kann das Verfahren auf breitere oder parallele Bitscheibendaten angewandt werden.

[0066] Speicherung und Abruf codierter Privatstatusdaten in Speicher **408**, wie sie in Fig. 4 gezeigt sind, können unter Benutzung der logischen Blöcke aus Fig. 5 implementiert werden.

[0067] Für dieses Beispiel wurde eine spezielle Mikrooperation (d. h. ein Steuersignal) für den Prozessor festgelegt, die (oder das) benutzt wird, wenn Privatstatusdaten im Speicher gespeichert oder aus diesem abgerufen werden.

[0068] Eine Adreßerzeugungseinheit (AGU) **504** empfängt eine spezielle Mikrooperation und berechnet in dieser Ausführungsform eine logische Adresse mit einer hohen und einer niedrigen Komponente. In einer Ausführungsform ist die logische Adresse eine virtuelle Adresse. In einer anderen Ausführungsform, wie in Fig. 5 gezeigt, ist die logische Adresse eine lineare Adresse, wie sie in Intel® Pentium® Prozessoren zu finden ist. In einer weiteren Ausführungsform ist die logische Adresse eine physische Adresse, und es ist keine Übersetzung der hohen Adreßbits nötig. In Fig. 5 wird die hohe Komponente der Adresse einem linear-physikalischen Adreßübersetzungsblock (auch bezeichnet als Übersetzungspuffer

oder TLB) **508** zugeführt, der diese hohe Komponente der linearen Adresse (die eine virtuelle Seitenzahl sein kann) in einen Teil einer physikalischen Adresse **509** übersetzt.

[0069] Eine Adreßverschleierungs- bzw. -codierungseinheit **514** soll in dieser Ausführungsform den niedrigen Teil des linearen Adreßwerts empfangen, der den jeweiligen Privatstatusdaten des Prozessors zugeordnet ist. In Reaktion darauf übersetzt die Adreßverschleierungseinheit **514** diese niedrige Komponente der linearen Adresse, um einen anderen Abschnitt der physikalischen Adresse **509** bereitzustellen. Der Wert dieses Abschnitts der physikalischen Adresse ist eine gemischte oder codierte Version der linearen Adresse, wie beispielsweise oben unter Bezugnahme auf Fig. 1 und Fig. 4 beschrieben.

[0070] In einer Ausführungsform bestimmen die spezielle Mikrooperation oder das UOP-Signal (Steuersignal), ob die Adreßcodierungseinheit **514** die Adreßbits niedriger Ordnung codieren soll. Wenn das Steuersignal nicht bestätigt wird, können die Adreßbits niedriger Ordnung in nicht codierter Form durch die Einheit **514** oder an der Einheit **514** vorbei gelangen. Auch wenn das Codierungssteuersignal (oder die Signale) bestätigt werden, können einige Adreßbits uncodiert passieren. Dies kann geschehen, wenn beispielsweise nur eine Untergruppe der Adreßbits codiert werden muß, wenn die Privatstatus-Speicherregion kleiner ist als die Adreßraumgröße, die von den Bits niedriger Ordnung adressierbar ist. Andere Ausführungsformen existieren, wobei eine Adreßcodierung nach der linear-physikalischen Adreßübersetzung erfolgt und deshalb die Codierung von Adreßräumen ermöglicht wird, die größer als eine virtuelle Speicherseite sind. Ein Vorteil der in Fig. 5 gezeigten Ausführungsform ist, daß die linear-physikalische Übersetzung parallel zum Codierungsvorgang abläuft und nicht seriell, so daß sie potentiell schneller ist. Auch sind Codierungen oft nur für Privatstatus-Speicherregionen nötig, die kleiner sind als die virtuelle Speicherseitengröße.

[0071] Die hohe Komponente der physikalischen Adresse (erzeugt von TLB **508**) und die niedrige Komponente der physikalischen Adresse (erzeugt von der Adreßverschleierungs- bzw. -codierungseinheit **514**) erzeugen, wenn sie verknüpft werden, eine physikalische Adresse **509**, die auf den tatsächlichen Ort im Speicher **408** hinweist, wo die jeweiligen Statusdaten gespeichert sind. Die physikalische Adresse **509** wird in dieser Ausführungsform zunächst auf den Zwischenspeicher **410** angewandt, und wenn dies zu einem Fehlschlag führt, wird der Inhalt des Orts aus Speicher **408** abgerufen oder in Speicher **408** gespeichert (je nachdem, ob der Vorgang ein Lade- oder ein Speichervorgang ist). Andere Anordnungen der Speicherhierarchie sind möglich.

[0072] Es ist zu beachten, daß in dieser Ausführungsform eine Region im Speicher **408**, die für die Speicherung der Privatstatusdaten bestimmt ist, nur einen Abschnitt der Seite einnehmen kann und am Rand einer virtuellen Speicherseite ausgerichtet sein kann. In diesem Fall passiert nur der Seitenverschiebungsabschnitt der linearen Adresse (also der niedrige Abschnitt der linearen Adresse) die Adreßverschleierungs- bzw. -codierungseinheit **514**, um die codierte physikalische Seitenverschiebung zu erzeugen. Andere Implementierungen sind möglich. Zusätzlich kann die Adreßverschleierungs- bzw. -codierungseinheit **514** eine Bereichswahllogik enthalten, sodaß nur Adressen innerhalb spezifischer Regionen des Speichers codiert werden. Mit dieser Logik muß die Speicherregion nicht unbedingt am Rand einer virtuellen Speicherseite ausgerichtet sein oder eine Größe einer Potenz von 2 aufweisen. Intern kann die Adreßverschleierungs- bzw. -codierungseinheit **514** mit Hilfe von Mikrocode, Software, Suchtabellen, festverdrahteter Funktionslogik, programmierbarer Logik oder jeder Kombination dieser Verfahren implementiert werden (siehe Fig. 6 zu einem Schlüsselelement einer solchen Implementierung).

[0073] Immer noch bezugnehmend auf Fig. 5 ist zu beachten, daß der Standardzwischenspeicher **410** des Prozessors in dieser Ausführungsform benutzt wird, um die codierten oder verschleierten Privatstatusdaten zu speichern. Wenn codierter Inhalt **510** entweder vom Zwischenspeicher **410** oder dem Speicher **408** zu liefern ist, kann er mit Hilfe einer Datencodierungs- und -decodierungseinheit **524** decodiert werden. Der decodierte Inhaltswert **520** wird dann in dieser Ausführungsform in dem Privatstatusbereich **516** des Prozessors gespeichert. Wie zuvor sind Zwischenspeicher **410** und Speicher **408** öffentlich zugänglich (z. B. durch das Betriebssystem), während der Privatstatusbereich nur den inneren Vorgängen des Prozessors zugänglich ist. Die Datencodierungs- und -decodierungseinheit **524** kann auch umgekehrt benutzt werden, wenn die Privatstatusdaten in codierter Form in den Speicher geschrieben werden. In einer solchen Ausführungsform würde die Einheit **524** einen Inhaltswert codieren, der aus dem Privatstatusbereich **516** stammen kann.

[0074] In einigen Ausführungsformen können in der ISA des Prozessors spezielle Instruktionen zum Zugriff auf einige oder alle Privatstatusdaten vorgesehen sein. Diese Instruktionen können, wenn sie ausgeführt werden, zum Transfer nicht codierter Daten von dem Privatstatusbereich **516** (siehe Fig. 5) führen, oder sie können spezielle Mikrooperationen oder Hardware-Steuersignale zum Zugreifen auf Region **418** in Speicher **408** (siehe Fig. 4) versenden, wo die Privatstatusdaten in codierter Form gespeichert sind. Während andere Instruktionen der ISA (z. B. normale Lade- und Speicherinstruktionen) auch dazu in der Lage sein können, auf die öffentlichen Regionen des

Speichers **408** und/oder des Zwischenspeichers **410** zuzugreifen, resultieren aus einem solchen Lesezugriff Privatstatusdatenwerte, deren Adreßwerte entweder verschleiert sind und/oder deren Dateninhalt codiert ist. Entsprechend ist es ohne spezielle Hardwareunterstützung nicht möglich, die Privatstatusdaten innerhalb eines akzeptablen Zeitrahmens zurückzuentwickeln oder auf andere Weise wiederherzustellen.

[0075] Obwohl der oben beschriebene Mechanismus logische Komponenten aufweist, die innerhalb einer Prozessorvorrichtung implementiert sind, sind andere Anordnungen möglich, wobei ein Teil der Logik oder die gesamte Logik beispielsweise im System-Chipsatz implementiert ist. Zusätzlich können spezielle Buszyklen für den Zugriff auf die Privatstatusregion **418** des Speichers **408** (Fig. 4) definiert sein.

[0076] Unter Hinwendung auf Fig. 6 ist ein genaueres Design eines Beispiels eines programmierbaren 4-Bit-Adreßbit-Verschleierungs-(Codierungs)-Mechanismus gezeigt. Dieses Design kann für die Adreßverschleierungs- bzw. -codierungseinheit **514** aus Fig. 5 benutzt werden, und um die logisch-physikalische Adreßzuordnung (für die Bits niedriger Ordnung) in Fig. 4 zu erzeugen.

[0077] Das logische Diagramm aus Fig. 6 ist eine Ausführungsform eines Kombinationslogikabschnitts eines 4 Bit breiten, MISR (Multiinput Linear Feedback Shift Register) mit einem Polynom vierter Ordnung unter Verwendung des oben beschriebenen Verfahrens. Diese Kombinationslogik wird von dem Polynomsteuerregister **604**, dem optionalen Maskenregister **610** und der Eingabeadreßquelle **606** zugeführt. Es ist zu beachten, daß diese Logik kein vollständiges MISR ist, sondern die Zuordnungseigenschaften eines MISR ausnutzt.

[0078] In Fig. 6 wird das Polynomkontrollregister **604** mit den Binärkoeffizienten eines Polynoms geladen. Um beispielsweise die Schaltung aus Fig. 6 zu konfigurieren, um die in Fig. 4 dargestellte logisch-physikalische Adreßzuordnung mit dem primitiven Polynom $x^4 + x^1 + x^0$ zu implementieren, würde das Polynomkontrollregister **604** mit binären Koeffizienten $P_0 = 1$, $P_1 = 1$, $P_2 = 0$, $P_3 = 0$ geladen. Der Adreßbitvektor 0000 ist 0000 zugeordnet, wenn das optionale Maskenregister **610** auf 0000 eingestellt ist. Die Eingabeadreßquelle **606** repräsentiert die zu codierende logische 4-Bit-Adresse. Die optionale Maskenquelle **610** (z. B. das Steuerregister) erlaubt die Konstruktion verschiedener Zuordnungen.

[0079] Wie oben beschrieben, können das Maskenregister **610** und das Polynomkontrollregister **604** während des Betriebs geändert werden. Beispielsweise können die Werte, die geladen werden, aus einer pseudozufälligen Datenquelle während der Einschalt-Rücksetzverarbeitung hergeleitet sein. Dies kann Versuche des Zugriffs auf Privatstatusdaten oder der Umgehung vorgegebener Zugriffsverfahren (wie z. B. der oben beschriebenen speziellen ISA-Instruktionen) verhindern. Fig. 6 ist eine Ausführungsform, die relativ effizient ist und bei moderatem Hardwareaufwand eine Programmierbarkeit mit Binärkoeffizient- und Maskenwerten mit verhältnismäßig wenigen Gatterverzögerungen erlaubt. Andere Logikentwürfe zum Implementieren der Adreßverschleierungs- bzw. -codierungseinheit **514** sind möglich. Zusätzliche Logik oder Assoziativspeicher (CAMs) können zur weiteren Beschränkung des Bereichs der von dem Adreßbit-Codiermechanismus modifizierten Adressen benutzt werden. Zusätzlich kann eine komplexere Logik für die Codierungs- und Decodierungsprozesse entworfen werden, um beispielsweise die Codierung zu verstärken (falls nötig).

[0080] Die Codierung der Inhaltswerte der Privatstatusdaten kann in ähnlicher Weise erreicht werden, wie oben für die Adreßverschleierung beschrieben wurde. Ein Ansatz ist es, die logischen Adreßverschiebungen (für ausgerichtete Regionen von Privatstatusdaten) oder einige konstante Abszissenwerte mit XOR zu verknüpfen, wobei die Inhalte eines gegebenen Priviatstatuselements zu kodieren sind. Ein anspruchsvollerer Codierungsmechanismus kann auf einen Strom von Privatstatusdatenwerten angewandt werden. Eine Variante eines rückgekoppelten Schieberegisterverfahrens (linear, nicht-linear, multi-Input usw.) kann mit einem Initialsamen benutzt werden. Der Initialsamen ist als der Ausgangsstatus definiert, der in das rückgekoppelte Schieberegister geladen wird. Für jeden Datenwert in Folge kann das rückgekoppelte Schieberegister vorgeschoben und sein Inhalt bitweise mit dem Inhalt des internen Registers XOR-verknüpft werden. Dies wird als eine Vigenère-Chiffre bezeichnet, und ein Beispiel hierfür ist in Tabellen **470**, **480** aus Fig. 4 oben gezeigt, wobei jeder nicht codierte Inhalts-(Daten)-Wert in **470** null ist (z. B. Eintrag **471**), aber nicht als solcher erscheint, wenn er in codierter Form in **480** (z. B. Eintrag **481**) gespeichert ist im Speicher **408**. Mit dieser Chiffre wird das Schieberegister benutzt, um eine pseudozufällige Sequenz von bitweisen XOR-Masken zu erzeugen. In diesem Fall wird jede pseudozufällige Maske mit Byte-Breite, während sie von einem MISR (siehe **480**) erzeugt wird, bitweise mit dem nächsten Datenwert in der Adreßsequenz XOR-verknüpft. Nur das Polynom und der Schieberegister-Initialsamenwert werden benötigt, um exakt dieselbe Sequenz wieder zu erzeugen. In einer Ausführungsform kann die Konfigurierungsinformation (z. B. Polynom und Initialsamen) der Codierungs- und/oder Decodierungseinheit zusammen mit der codierten Statusregion in Speicher **408** gespeichert sein. Um den Privatstatus zu decodieren, würde die Konfigurierungsinformation (z. B. Polynom und Initialsa-

men) abgerufen (und möglicherweise unter Benutzung eines anderen festgelegten Codierungsverfahrens decodiert) und dann benutzt. Solange jede Maske in der Sequenz auf die entsprechenden Daten in derselben Reihenfolge angewandt wird (also z. B. eine Maske pro adressierbare Einheit angewandt wird), produziert (decodiert) die bitweise XOR-Maskierung die ursprünglichen Daten. Wie zuvor erläutert, können das Polynom und der MISR-Initialsamenwert mit Hilfe verschiedener Verfahren oder Beschränkungen geändert werden (z. B. beim Starten, während des Betriebs usw.). Um die ursprünglichen Daten wiederherzustellen, müssen das oder die Decodierungsverfahren angewandt werden, die für das oder die ursprünglich benutzten Codierungsverfahren geeignet sind, um nämlich die Codierung rückgängig zu machen. Vigenère-Chiffren sind nur ein Beispiel für Codierungsmechanismen für Privatstatus-Datenwerte, das effizient ist und eine Programmierbarkeit mit einfachen Binärkoeffizientenlisten, Samen usw. zuläßt, sowie einen moderaten Umfang der Hardware und nur einige wenige Gatterverzögerungen. Andere Ausführungsformen sind ebenfalls möglich.

[0081] In einer Ausführungsform der Erfindung kann der Prozessor die Privatstatusregion **132** im Speicher (siehe Fig. 1) an Übergängen zwischen Betriebsmodi des Prozessors nutzen. Beispielsweise kann der Prozessor auf die Privatstatusregion zugreifen, wenn er wie oben beschrieben in den Systemmanagement-Modus (SMM) eintritt. Diese Übergänge zwischen Betriebsmodi werden hier als Moduswechsel bezeichnet. Moduswechsel umfassen beispielsweise eine Bewegung zwischen Normalmodus und Systemmanagementmodus, zwischen einer virtuellen Maschine (VM) und einem virtuellen Maschinenmonitor (VMM) in einem virtuellen Maschinensystem, zwischen einem Betriebssystemprozeß auf Nutzerebene und dem Betriebssystemkern usw.

[0082] In einer Ausführungsform der Erfindung kann der Prozessor die Privatstatusregion **132** im Speicher jederzeit nach Zuweisung der Privatstatusregion nutzen. Beispielsweise kann in einem virtuellen Maschinensystem der VMM eine Region im Speicher zur Prozessornutzung während des virtuellen Maschinenbetriebs vorsehen. Der VMM kann dem Prozessor den Ort der Privatstatusregion anzeigen (z. B. durch Ausführen einer in der ISA definierten Instruktion). Nachdem der Prozessor die Anzeige erhalten hat, steht es ihm frei, die Privatstatusregion in geeigneter Weise zu nutzen. Beispielsweise kann der Prozessor während Übergängen zwischen einer VM und dem VMM (d. h. an Moduswechselpunkten) auf die Privatstatusregion zugreifen. Außerdem kann der Prozessor während des Betriebs einer VM oder des VMM auf die Region zugreifen. Beispielsweise kann der Prozessor auf Steuerinformation aus der Privatstatusregion zugreifen, oder der Prozessor kann vorläufige Werte in der Privatstatusregion speichern.

[0083] Die ISA kann außerdem einen Mechanismus vorsehen, wodurch der VMM bestimmen kann, daß eine Privatstatusregion nicht länger benutzt werden soll (z. B. durch Ausführen einer Instruktion). In anderen Ausführungsformen können Privatstatusregionen unter Benutzung anderer Verfahren zugeteilt werden. Beispielsweise kann eine Privatstatusregion durch Schreiben von modellspezifischen Registern (MSRs), durch Ausführen von Instruktionen in der ISA, durch Schreiben von Speicherorten usw. zugeteilt werden.

[0084] Obwohl die oben genannten Beispiele Ausführungsformen der vorliegenden Erfindung im Kontext von Ausführungseinheiten und logischen Schaltungen beschreiben, lassen sich andere Ausführungsformen der vorliegenden Erfindung mit Hilfe von Software erzielen. Beispielsweise kann die vorliegende Erfindung in einigen Ausführungsformen als ein Computerprogrammprodukt oder als Software vorgesehen sein, die eine Maschine oder ein computerlesbares Medium umfaßt, worauf Instruktionen gespeichert sind, die zum Programmieren eines Computers (oder anderer elektronischer Vorrichtungen) benutzt werden können, um einen Prozeß gemäß einer Ausführungsform der Erfindung durchzuführen. In anderen Ausführungsformen können Vorgänge von spezifischen Hardwarekomponenten durchgeführt werden, die Mikrocode, festverkabelte Logik oder irgendeine andere Kombination programmierter Computerkomponenten und maßgefertigter Hardwarekomponenten aufweisen.

[0085] So kann ein maschinenlesbares Medium einen Mechanismus zum Speichern oder Übertragen von Information in einer von einer Maschine (z. B. einem Computer) lesbaren Form aufweisen, ist aber nicht beschränkt auf, Disketten, Compact Disks, Lesespeicher (CDROMs) und magnetooptische Disks, Lesespeicher (ROMS), Schreib-Lese-Speicher (RAM), löschbare programmierbare Lesespeicher (EPROM), elektrisch löschbare programmierbare Lesespeicher (EEPROM), magnetische oder optische Karten, Flash-Speicher, eine Übertragung über das Internet, elektrische, optische, akustische oder andere Formen von übertragenen Signalen (z. B. Trägerwellen, Infrarotsignale, Digitalsignale usw.) oder ähnliches.

[0086] Ferner kann ein Entwurf mehrere Stufen durchlaufen, von der Erzeugung bis zur Simulation hin zur Herstellung. Daten, die einen Entwurf repräsentieren, können den Entwurf auf unterschiedliche Art und Weise repräsentieren. Zunächst, wie es in Simulationen nützlich ist, kann die Hardware unter Benutzung einer Hardware-Beschreibungssprache oder einer anderen funktionalen Beschreibungssprache repräsentiert werden. Außerdem kann auf einigen Stufen des Entwurfsprozesses ein Schaltungslevelmodell mit Logik- und/oder Transistorgattern erzeugt

werden. Auch erreichen die meisten Entwürfe auf einer Stufe ein Datenlevel, das die physikalische Anordnung verschiedener Vorrichtungen im Hardwaremodell repräsentiert. Für den Fall, daß übliche Halbleiterherstellungsverfahren benutzt werden, können die Daten, die ein Hardwaremodell repräsentieren, die Daten sein, die das Vorhandensein oder die Abwesenheit verschiedener Merkmale auf verschiedenen Maskenschichten spezifizieren, die benutzt werden, um den integrierten Schaltkreis herzustellen. Bei jeder Repräsentation des Entwurfs können die Daten auf jeder Art maschinenlesbaren Mediums gespeichert werden. Eine optische oder elektrische Welle, die moduliert oder anders erzeugt wird, um solche Information zu übertragen, ein Speicher oder ein magnetischer oder optischer Speicher wie z. B. eine Disk können ein maschinenlesbares Medium sein. Jedes dieser Medien kann den Entwurf oder die Softwareinformation „tragen" oder „anzeigen". Wenn eine elektrische Trägerwelle, die den Code oder den Entwurf anzeigt oder trägt, übertragen wird, sodaß ein Kopieren, Puffern oder eine Neuübertragung des elektrischen Signals durchgeführt wird, wird eine neue Kopie erstellt. So kann ein Kommunikationsanbieter oder ein Netzwerkanbieter Kopien eines Artikels (einer Trägerwelle) anfertigen, die Verfahren der vorliegenden Erfindung verkörpert.

[0087] In der vorangegangenen Beschreibung wurde die Erfindung unter Bezugnahme auf verschiedene Verfahren zum Zugreifen auf Daten zum Status einer Datenverarbeitungsmaschine von einem öffentlich zugänglichen Speicher beschrieben. Man wird jedoch verstehen, daß verschiedene Modifizierungen und Änderungen vorgenommen werden können, ohne vom breiteren Geist und Umfang der Ausführungsformen der Erfindung abzuweichen, die in den beiliegenden Ansprüchen definiert ist. Die Beschreibung und die Figuren sind dementsprechend als erläuternd und nicht als begrenzend zu verstehen.

Patentansprüche

1. Verfahren zum Betreiben einer Datenverarbeitungsmaschine mit den folgenden Schritten:
a) Anwenden eines Codierungsprozesses auf Privat-Statusdaten durch einen Prozessor (**110**), wobei die Privat-Statusdaten einen Status des Prozessors erfassen,
b) Schreiben der codierten Privat-Statusdaten an einen Ort in einem Speicher (**130**), der für Software durch Ausführen eines ersten Befehls aus einer Mehrzahl von Befehlen in der Befehlssatzarchitektur des Prozessors zugänglich ist, und
c) Erlangen des Status des Prozessors durch die Software, durch Ausführen eines zweiten Befehls der Mehrzahl von Befehlen in der Befehlssatzarchitektur des Prozessors, wobei das Ausführen des zweiten Befehls bewirkt, dass der Prozessor (**110**) die kodierten Privat-Statusdaten aus dem Speicher liest, die kodierten Privat-Statusdaten dekodiert, um dekodierte Privat-Statusdaten zu erzeugen, und die dekodierten Privat-Statusdaten im Prozessor speichert.

2. Verfahren nach Anspruch 1, wobei die Privat-Statusdaten entweder geschrieben werden
a) an einem frei zugänglichen Ort in einer Registerdatei des Prozessors,
b) in einen Zwischenspeicher oder
c) in einen Speicher.

3. Verfahren nach Anspruch 1, wobei der Codierungsprozess ein solcher ist, bei dem der Ort des geschriebenen Inhalts eines jeweiligen internen Registers (**114**) des Prozessors sich wenigstens einmal zufallsgesteuert ändert, während die Schritte a) bis b) wiederholt werden.

4. Verfahren nach Anspruch 1, wobei die den Status der Maschine betreffenden Privat-Daten entweder ein Registerwert oder ein Wert aus dem Speicher (**408**) sind.

5. Verfahren nach Anspruch 1, wobei der Codierungsprozess ein solcher ist, bei dem das Speicherformat des geschriebenen Inhalts eines jeweiligen internen Registers (**114**) des Prozessors sich wenigstens einmal zwischen Big-Endian und Little-Endian zufallsgesteuert ändert, während die Schritte a) bis b) wiederholt werden.

6. Verfahren nach Anspruch 1, wobei das Codieren ein Chiffrieren eines Werts der Privat-Daten umfasst, um die codierten Privat-Daten zu erhalten.

7. Verfahren nach Anspruch 1, wobei das Codieren eine Adreßcodierung umfasst, um den Adreßwert der Privat-Daten zu verschleiern.

8. Verfahren nach Anspruch 1, wobei der Codierungsprozess ein solcher ist, bei dem eine Chiffre auf den Inhalt eines jeweiligen Registers (**114**) angewandt wird, um einen codierten Wert zu erzeugen, der dann an den Ort im Speicher (**130**) geschrieben wird.

9. Verfahren nach Anspruch 1, der des weiteren ein Speichern der dekodierten Privat-Statusdaten in einem Privat-Speicher des Prozessors umfasst.

10. Verfahren nach Anspruch 1, wobei das Erlangen umfasst:
Lesen mehrerer Werte aus dem Speicher (**408**); und Kombinieren der gelesenen mehreren Werte, um einen einzelnen nicht codierten Wert der Privat-Daten zu bilden.

11. Verfahren nach Anspruch 1, wobei das Erlangen umfasst:

Lesen mehrerer Werte aus einem oder mehreren nicht zusammenhängenden Speicherorten;
Kombinieren der gelesenen Werte, um einen einzigen Wert zu bilden; und
Decodieren des einzigen Werts, um einen einzigen nicht codierten Wert der Privat-Daten zu bilden.

12. Datenverarbeitungsmaschine mit einem Prozessor (**110**) und einem Speicher, wobei die Datenverarbeitungsmaschine eingerichtet ist,
a) einen Codierungsprozess auf Privat-Statusdaten durch den Prozessor (**110**) anzuwenden, wobei die Privat-Statusdaten einen Status des Prozessors erfassen,
b) die codierten Privat-Statusdaten an einen Ort in einem Speicher (**130**) zu schreiben, der für Software durch Ausführen eines ersten Befehls aus einer Mehrzahl von Befehlen in der Befehlssatzarchitektur des Prozessors zugänglich ist, und
c) den Status des Prozessors durch die Software, durch Ausführen eines zweiten Befehls der Mehrzahl von Befehlen in der Befehlssatzarchitektur des Prozessors zu erlangen, wobei das Ausführen des zweiten Befehls bewirkt, dass der Prozessor (**110**) die kodierten Privat-Statusdaten aus dem Speicher liest, die kodierten Privat-Statusdaten dekodiert, um dekodierte Privat-Statusdaten zu erzeugen, und die dekodierten Privat-Statusdaten im Prozessor speichert.

13. Datenverarbeitungsmaschine nach Anspruch 12, wobei der Prozessor (**110**) eine spezielle Lesemikrooperation aufweist, die benutzt wird, wenn der Prozessor (**110**) die Statusdaten aus der Speichereinheit liest.

14. Datenverarbeitungsmaschine nach Anspruch 13, wobei der Prozessor (**110**) ferner einen internen Zwischenspeicher (**410**) aufweist und die codierten Statusdaten an einen frei zugänglichen Ort in dem Zwischenspeicher (**410**) schreibt.

15. Datenverarbeitungsmaschine nach Anspruch 13, wobei der Prozessor (**110**) die Statusdaten wiedererlangt und die wiedererlangten Statusdaten an einen privaten Ort in der Datenverarbeitungsmaschine schreibt

16. Datenverarbeitungsmaschine nach Anspruch 13, wobei der Prozessor (**110**) die Statusdaten wiedererlangt und sich selbst in Vorbereitung auf die Wiederaufnahme der Ausführung einer unterbrochenen Aufgabe mit den wiedererlangten Statusdaten konfiguriert.

17. Datenverarbeitungsmaschine nach Anspruch 13, wobei für den Prozessor (**110**) eine vom Hersteller definierte Instruktion vorliegt, die bei Ausführung durch den Prozessor (**110**) die Statusdaten aus der Speichereinheit dekodiert und liest.

18. Datenverarbeitungsmaschine nach Anspruch 12,
wobei für den Prozessor (**110**) eine spezielle Mikrooperation definiert ist, um auf die codierten Statusdaten aus der Speichereinheit zuzugreifen, und
wobei der Prozessor (**110**) eine Adreßverschleierungseinheit aufweist, um einen Adreßwert zu empfangen, der jeweiligen Statusdaten des Prozessors zugeordnet ist, wobei der Adreßwert von einer Meldung der speziellen Mikrooperation abgeleitet wurde, und die Verschleierungseinheit einen codierten physikalischen Adreßwert bereitstellt, der auf den tatsächlichen Ort in der Speichereinheit hinweist, an dem die jeweiligen Statusdaten gespeichert sind.

19. Datenverarbeitungsmaschine nach Anspruch 12, wobei für den Prozessor (**110**) ein Hardware-Steuersignal zum Zugreifen auf die codierten Daten aus der Speichereinheit definiert ist, und wobei der Prozessor (**110**) einen internen Zwischenspeicher und eine Datenumwandlungseinheit aufweist, die Datenwerte von dem Zwischenspeicher als Resultat eines aus dem Hardwaresteuersignal abgeleiteten Zwischenspeicherzugriffs empfängt, und den Datenwert in die tatsächlichen Statusdaten des Prozessors decodiert.

20. Computersystem, mit
einem Prozessor (**404**), und
einem mit dem Prozessor (**404**) kommunizierend verbundenen Hauptspeicher (**408**), der einen öffentlichen Bereich (**416**) aufweist, der dazu bestimmt ist, die Privat-Statusdaten des Prozessors (**404**) in codierter Form zu speichern,
wobei der Prozessor (**404**) eingerichtet ist,
a) einen Codierungsprozess auf Privat-Statusdaten durch den Prozessor (**110**) anzuwenden, wobei die Privat-Statusdaten einen Status des Prozessors erfassen,
b) die codierten Privat-Statusdaten an einen Ort in einem Speicher (**130**) zu schreiben, der für Software durch Ausführen eines ersten Befehls aus einer Mehrzahl von Befehlen in der Befehlssatzarchitektur des Prozessors zugänglich ist, und
c) den Status des Prozessors durch die Software, durch Ausführen eines zweiten Befehls der Mehrzahl von Befehlen in der Befehlssatzarchitektur des Prozessors zu erlangen, wobei das Ausführen des zweiten Befehls bewirkt, dass der Prozessor (**110**) die kodierten Privat-Statusdaten aus dem Speicher liest, die kodierten Privat-Statusdaten dekodiert, um dekodierte Privat-Statusdaten zu erzeugen, und die dekodierten Privat-Statusdaten im Prozessor speichert.

21. System nach Anspruch 20, wobei der Prozessor (**404**) die Privat-Statusdaten vor dem Speichern im öffentlichen Bereich codiert.

22. System nach Anspruch 20, wobei der Prozessor (**404**) einen Wert, der aus dem öffentlichen Be-

reich abgelesen wurde, vor einer Benutzung decodiert.

23. System nach Anspruch 20, wobei der Prozessor (**404**) ferner eine interne Speichereinheit (**428**) aufweist, in der ein öffentlicher Bereich bestimmt ist, um eine Kopie der Privat-Statusdaten in codierter Form zu speichern.

24. System nach Anspruch 23, wobei die interne Speichereinheit (**428**) entweder ein Zwischenspeicher oder eine Registerdatei ist.

25. System nach Anspruch 23, wobei ein privater Bereich in der internen Speichereinheit (**416**) bestimmt ist, um die Privat-Statusdaten in nicht codierter Form zu speichern.

26. System nach Anspruch 20, das ferner einen Systemchipsatz (**406**) aufweist, der den Prozessor (**404**) mit dem Hauptspeicher kommunizierend verbindet.

Es folgen 6 Blatt Zeichnungen

Anhängende Zeichnungen

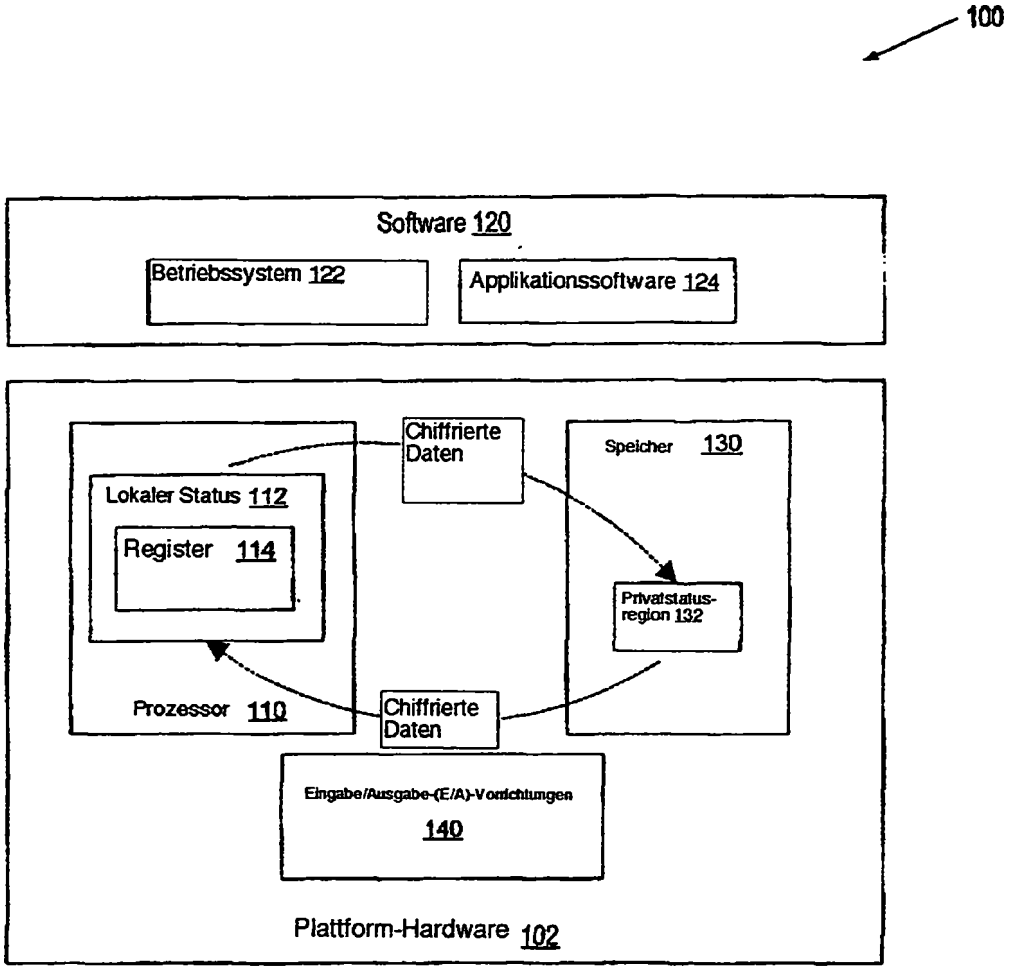


FIG. 1

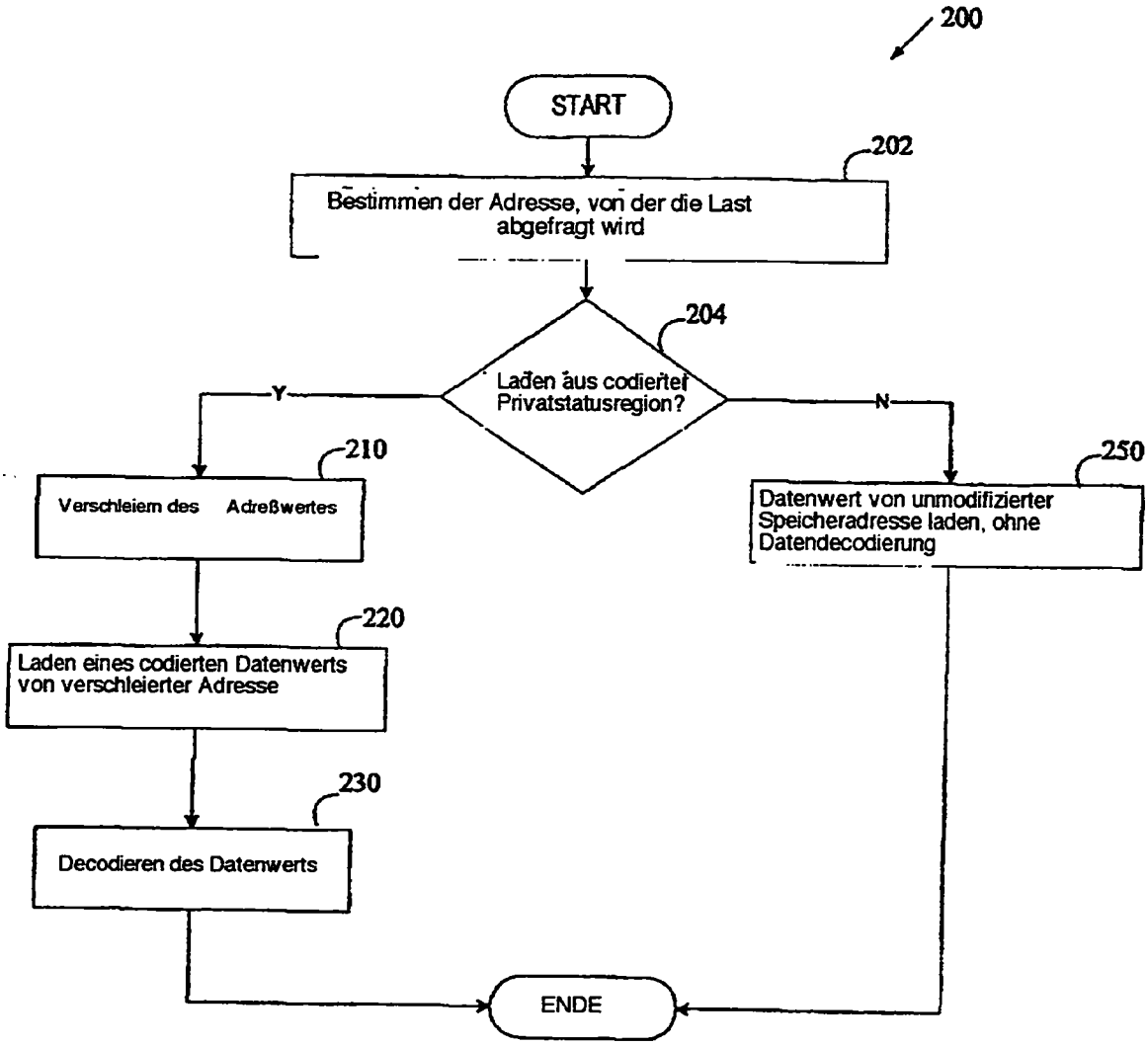

200
START
202
Bestimmen der Adresse, von der die Last abgefragt wird
204
Laden aus codierter Privatstatusregion?
Y
N
210
Verschleiern des Adreßwertes
250
Datenwert von unmodifizierter Speicheradresse laden, ohne Datendecodierung
220
Laden eines codierten Datenwerts von verschleierter Adresse
230
Decodieren des Datenwerts
ENDE

FIG. 2

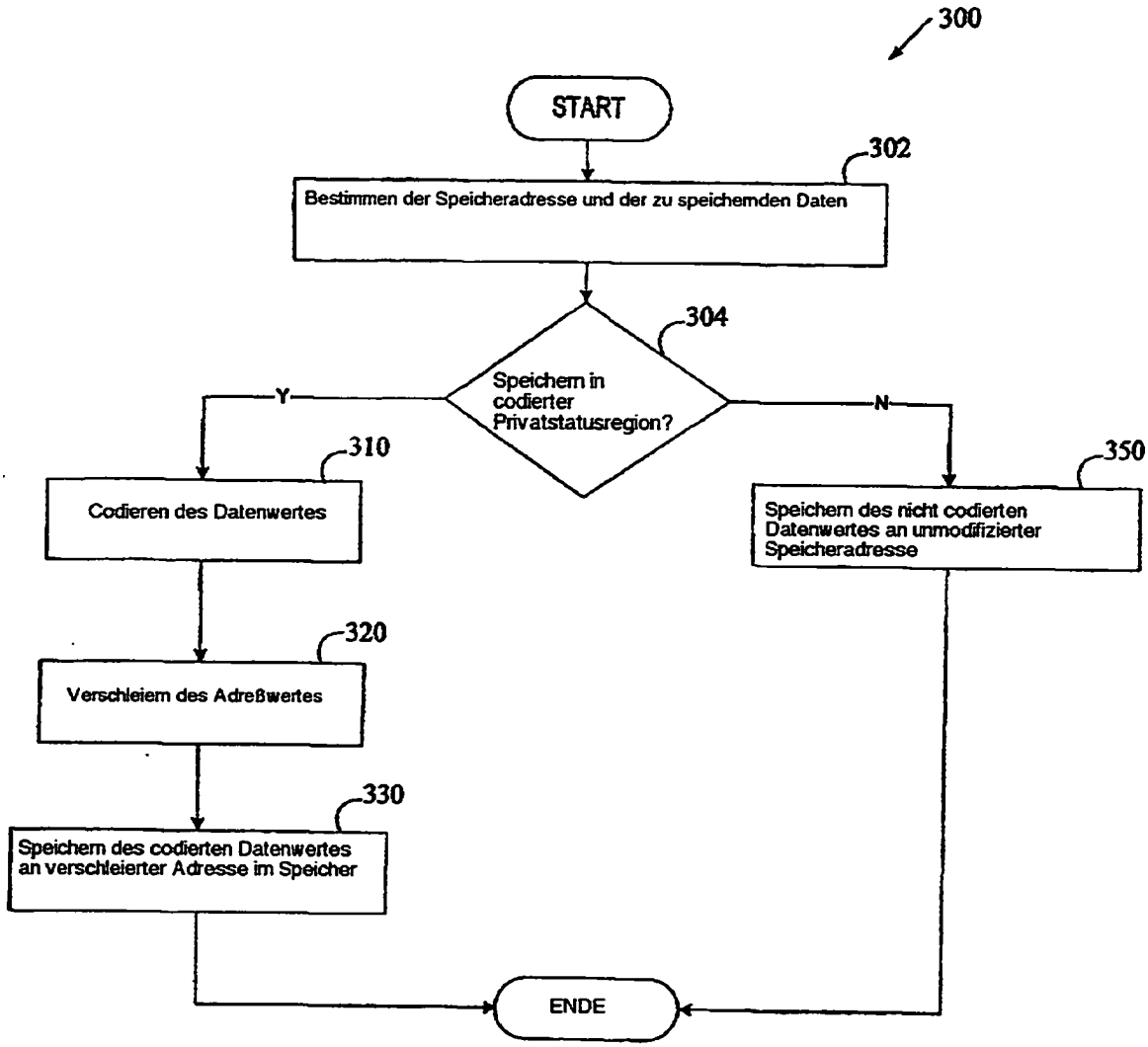

300
START
302
Bestimmen der Speicheradresse und der zu speichernden Daten
304
Speichern in codierter Privatstatusregion?
Y
N
310
Codieren des Datenwertes
320
Verschleiern des Adreßwertes
330
Speichern des codierten Datenwertes an verschleierter Adresse im Speicher
350
Speichern des nicht codierten Datenwertes an unmodifizierter Speicheradresse
ENDE

FIG. 3

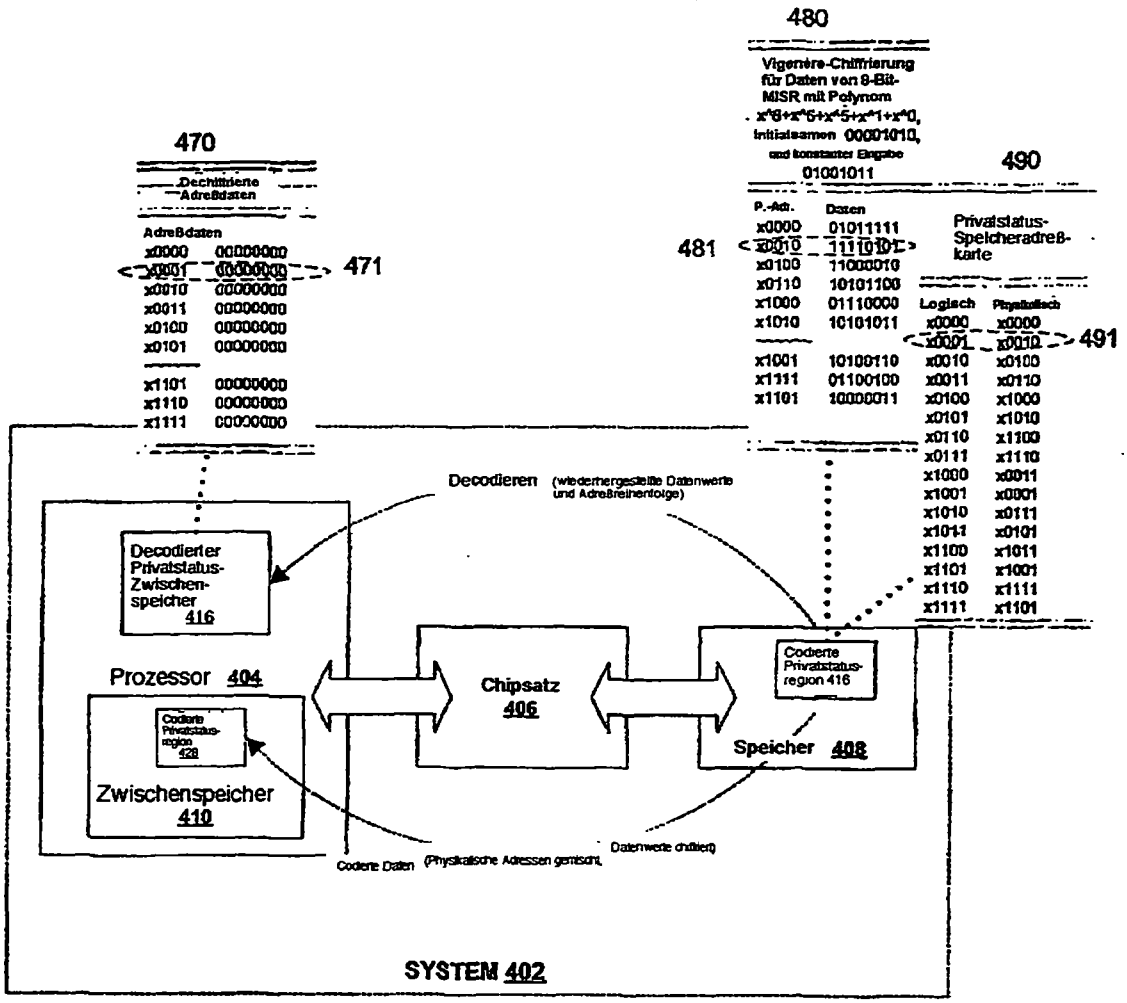

480
Vigenère-Chiffrierung für Daten von 8-Bit-MISR mit Polynom
Initialsamen 00001010,
und konstanter Eingabe
01001011
490
470
Dechiffrierte Adreßdaten
Adreßdaten
471
481
Privatstatus-Speicheradreß-karte
Logisch
Physikalisch
491
Decodieren (wiederhergestellte Datenwerte und Adreßreihenfolge)
Decodierter Privatstatus-Zwischen-speicher 416
Prozessor 404
Codierte Privatstatus-region 428
Zwischenspeicher 410
Chipsatz 406
Codierte Privatstatus-region 416
Speicher 408
Codierte Daten (Physikalische Adressen gemischt, Datenwerte chiffriert)
SYSTEM 402

FIG. 4

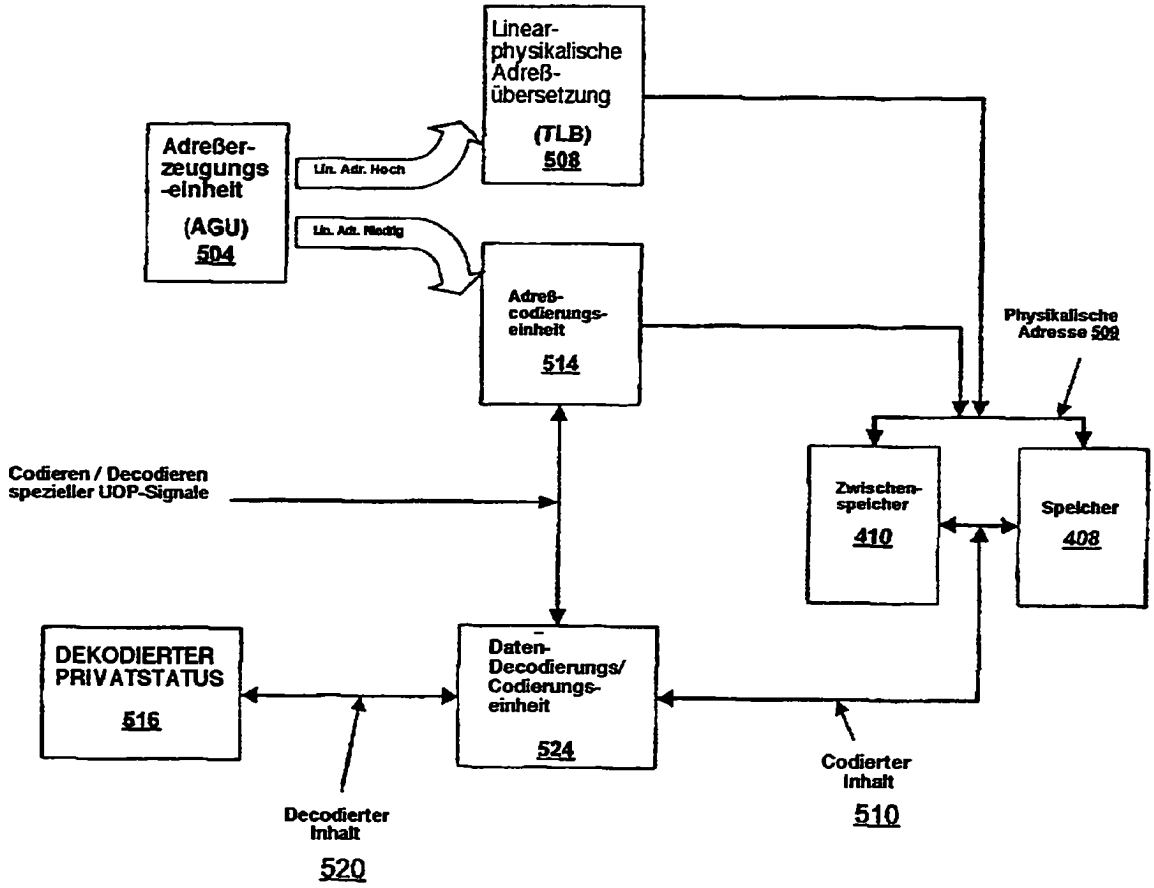

Linear-physikalische Adreß-übersetzung
(TLB)
508
Adreßer-zeugungs-einheit
(AGU)
504
Lin. Adr. Hoch
Lin. Adr. Niedrig
Adreß-codierungs-einheit
514
Physikalische Adresse 509
Codieren / Decodieren spezieller UOP-Signale
Zwischen-speicher
410
Speicher
408
DEKODIERTER PRIVATSTATUS
516
Daten-Decodierungs/ Codierungs-einheit
524
Codierter Inhalt
510
Decodierter Inhalt
520

FIG. 5

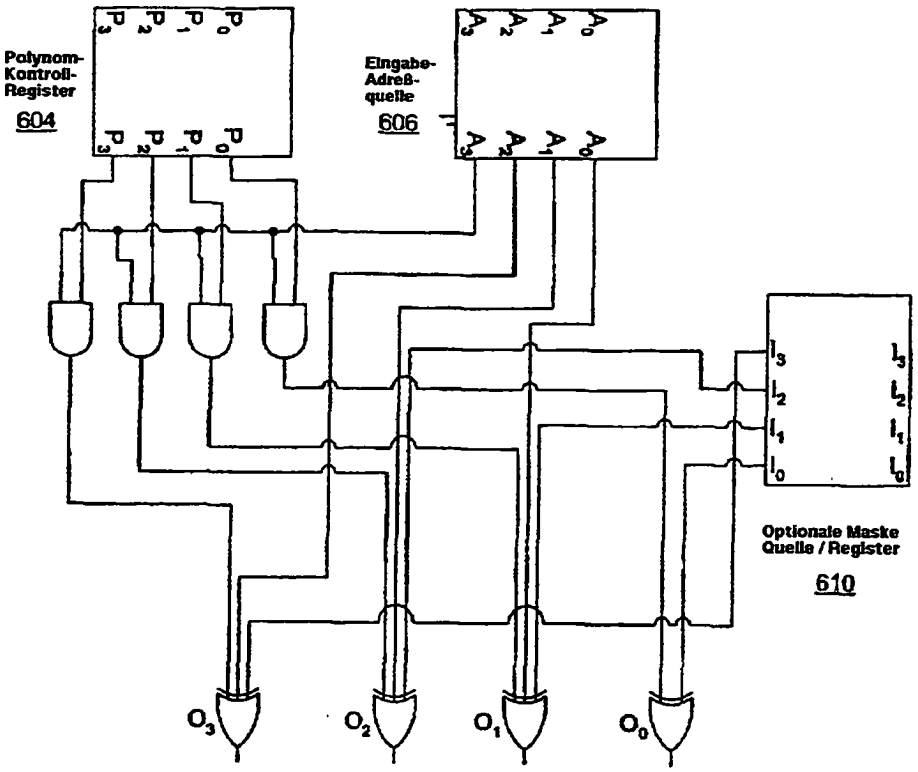

Polynom-Kontroll-Register
604
Eingabe-Adreß-quelle
606
Optionale Maske Quelle / Register
610

FIG. 6