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

(10) **DE 697 29 974 T2** 2005.07.21

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 809 180 B1**
(21) Deutsches Aktenzeichen: **697 29 974.0**
(96) Europäisches Aktenzeichen: **97 108 346.4**
(96) Europäischer Anmeldetag: **22.05.1997**
(97) Erstveröffentlichung durch das EPA: **26.11.1997**
(97) Veröffentlichungstag
der Patenterteilung beim EPA: **28.07.2004**
(47) Veröffentlichungstag im Patentblatt: **21.07.2005**

(51) Int Cl.[7]: **G06F 9/30**
G06F 9/40, G06F 9/312

(30) Unionspriorität:

12754196	**22.05.1996**	**JP**
13592397	**08.05.1997**	**JP**

(84) Benannte Vertragsstaaten:
DE, FR, GB, IT

(73) Patentinhaber:
Seiko Epson Corp., Tokio/Tokyo, JP

(74) Vertreter:
Hoffmann, E., Dipl.-Ing., Pat.-Anw., 82166 Gräfelfing

(72) Erfinder:
Kudo, Makoto, Suwa-shi, Nagano-ken, JP; Kubota, Satoshi, Suwa-shi, Nagano-ken, JP; Miyayama, Yoshiyuki, Suwa-shi, Nagano-ken, JP; Sato, Hisao, Suwa-shi, Nagano-ken, JP

(54) Bezeichnung: **Datenverarbeitungsschaltung, Mikrocomputer und elektronische Einrichtung**

Anmerkung: Innerhalb von neun Monaten nach der Bekanntmachung des Hinweises auf die Erteilung des europäischen Patents kann jedermann beim Europäischen Patentamt gegen das erteilte europäische Patent Einspruch einlegen. Der Einspruch ist schriftlich einzureichen und zu begründen. Er gilt erst als eingelegt, wenn die Einspruchsgebühr entrichtet worden ist (Art. 99 (1) Europäisches Patentübereinkommen).

Die Übersetzung ist gemäß Artikel II § 3 Abs. 1 IntPatÜG 1991 vom Patentinhaber eingereicht worden. Sie wurde vom Deutschen Patent- und Markenamt inhaltlich nicht geprüft.

Beschreibung

HINTERGRUND DER ERFINDUNG

Gebiet der Erfindung

[0001] Diese Erfindung bezieht sich auf eine Datenverarbeitungsschaltung, einen diese Datenverarbeitungsschaltung beinhaltenden Mikrorechner sowie ein elektronisches Gerät, das mit diesem Mikrorechner versehen ist.

Beschreibung einschlägiger Technik

[0002] Im Stand der Technik gibt es einen RISC-Mikrorechner, der 32-Bit Daten verarbeiten kann und dazu Befehle fester Länge benutzt, die 32 Bit breit sind. Die Verwendung von Befehlen fester Länge ermöglicht eine Verringerung der zum Dekodieren der Befehle erforderlichen Zeit im Vergleich zur Verwendung von Befehlen veränderlicher Länge und macht es außerdem möglich, die Größe der Schaltungsanordnung des Mikrorechners zu verkleinern.

[0003] Aber selbst bei einem 32-Bit Mikrorechner müssen nicht immer die vollen 32 Bits benutzt werden. Wenn also alle Befehle als 32-Bit Befehle geschrieben sind, enthalten viele von ihnen unbenutzte Bereiche, so daß die Speichernutzung einen geringen Wirkungsgrad hat.

[0004] Die Erfinder der vorliegenden Erfindung haben den Gedanken eines Mikrorechners untersucht, der Befehle fester Länge mit einer Bitbreite handhabt, die schmaler ist als die der ausführbaren Daten, um dadurch den Nutzungsgrad zu erhöhen, ohne die Steuerschaltungsanordnung komplizierter zu machen.

[0005] Aber einfach die Befehle fester Länge von 32 Bit zusammenzuschneiden, damit sie auf eine feste Länge von beispielsweise 16 Bit passen, ruft die nachfolgend beschriebenen Probleme hervor.

[0006] Ein wichtiges Merkmal eines RISC-Mikrorechners besteht in der Benutzung von allgemeinen Registern, deren Inhalte keine im voraus zugewiesene Bedeutung haben. Aus diesem Grund wird bei Benutzung eines Stapelzeigers eines der allgemeinen Register als Stapelzeiger herangezogen, und ein auf ein allgemeines Register wirkender Befehl wird zum Durchführen der Stapelarbeitsweise benutzt.

[0007] Beim Kodieren eines Prozesses, der Daten hernimmt, die von einer Speicheradresse spezifiziert sind, die durch Addieren eines vorherbestimmten Versatzes zum Stapelzeiger erhalten wurde, und diese auf ein allgemeines Register überträgt, wofür beispielsweise ein Befehl benutzt wird, der auf ein allgemeines Register wirkt, müssen in den Objektcode des Befehls der Versatz, das vorherbestimmte Register spezifizierende Daten sowie Daten eingeschlossen werden, die das als Stapelzeiger zu verwendende Register spezifizieren.

[0008] Wenn also ein auf ein allgemeines Register wirkender Befehl beim Kodieren eines Prozesses benutzt wird, der auf den Stapelzeiger wirkt, muß in dem Objektcode eine große Datenmenge spezifiziert werden, was es schwierig macht, die Einzelheiten des Befehls innerhalb einer auf 16 Bit festgelegten Länge zu kodieren. Eine Erweiterung der Befehlslänge auf beispielsweise 32 Bits würde zu einer großen Zahl an Befehlen führen, die nicht unbedingt die ganzen 32 Bits brauchen, so daß viele Befehle unbenutzte Teile enthalten, womit der Wirkungsgrad der Speichernutzung erniedrigt wird.

[0009] Darüber hinaus führt eine Vergrößerung der Befehlslänge auch zu einer Vergrößerung des zum Speichern derartiger Befehle erforderlichen Speicherraums, so daß es unter dem Gesichtspunkt wirksamer Speichernutzung vorzuziehen ist, nicht nur Befehle fester Länge zu verwenden, sondern darüber hinaus die Befehle so kurz wie möglich zu gestalten.

[0010] Wenn ein Programm ausgeführt wird, das in einer Sprache geschrieben wurde, die einen Speicherbereich für mit dem Stapelzeiger verknüpfte Auto-Variable sicherstellt, beispielsweise die Programmiersprache C, gibt es viele Befehle, die auf den Stapelzeiger wirken, so daß eine effiziente Kodierung und Ausführung von auf den Stapelzeiger wirkenden Befehlen vorzuziehen ist.

[0011] In diesem Fall ist deshalb eine Architektur vorzuziehen, die das Kodieren und die Ausführung von Befehlen ermöglicht, die so kurz wie möglich sind, wenn ein Prozeß ausgeführt wird, der auf den Stapelzeiger wirkt.

[0012] In letzter Zeit ist es üblich geworden, Mehrzweck-Register intern vorzusehen, insbesondere im Fall einer RISC-CPU, um die Fähigkeiten zu erweitern. Die Anordnung einer großen Anzahl interner Register ermöglicht es, viele Prozesse rasch innerhalb der CPU selbst auszuführen, ohne Zugriff auf den Speicher nehmen zu müssen. Wenn eine solche große Anzahl interner Register vorgesehen ist, muß eine große Anzahl Register während des Prozesses der Registersicherung und Wiederherstellung gesichert werden, wenn entweder eine Interrupt verarbeitet oder ein Unterprogramm aufgerufen wird.

[0013] Wenden wir uns nunmehr einem bekannten Beispiel von Befehlen zu, mit denen die Inhalte von Registern gesichert und wiederhergestellt werden, die bei einer Aufnahme und beim Verlassen eines Unterprogramms häutig Anwendung finden, und zwar sogar innerhalb stapelbezogener Befehle.

[0014] Der Befehlssatz eines Mikrorechners hat normalerweise Befehle zum Sichern und Wiederherstellen der Inhalte von Registern in der CPU in einen im Speicher vorgesehenen Stapel und aus diesem. Hierbei handelt es sich entweder um zweckgebundene Befehle oder um Befehle, die Register indirekt adressieren.

[0015] In dem Chip 80386 von Intel stehen für das Schreiben der Inhalte von Registern in den Stapel Push-Befehle "push", "pusha" und "pushad" zur Verfügung, und für die Rückgabe von Daten aus dem Stapel an die Register stehen Pop-Befehle "pop", "popa" und "popad" zur Verfügung.

[0016] Wenn durch den Push-Befehl der Inhalt eines Registers in den Stapel geschrieben wird, muß das Register als Operand spezifiziert werden, beispielsweise als "push EAX". Dieses Beispiel betrifft ein 32-Bit Register EAX. Wenn die Inhalte der Register EAX, ECX, EDX und EBX alle in den Stapel geschrieben werden sollen, muß dieser Push-Befehl wie folgt wiederholt werden:
"push EAX"
"push ECX"
"push EDX"
"push EBX".

[0017] Wenn für jedes der Register solche Push- und Pop-Befehle benutzt werden, nimmt die Größe des Objektcodes zu und es wird eine große Anzahl Programmausführungsschritte nötig, was zu Verzögerungen in der Ausführungszeit oder Verarbeitung des Programms führt.

[0018] Aus diesem Grund wird der "pusha"- oder "pushad"-Befehl benutzt, um den Inhalt aller acht allgemeinen Register des 80386 in den Stapel einzuspeichern. "pusha" arbeitet an den unteren 16 Bits jedes der acht Register, während "pushad" an allen 32 Bits arbeitet. Durch die Verwendung von "pusha" und "pushad" erübrigt sich die Notwendigkeit einer achtmaligen Wiederholung des Push-Befehls.

[0019] Die "pop"-, "popa"- und "popad"-Befehle wirken auf ähnliche Weise.

[0020] Zu den Nachteilen einer Wiederholung des Push-Befehls zählt nicht nur die vergrößerte Länge des Programmcodes, sondern auch die Verzögerungen bei der Ausführung, die dadurch verursacht werden, daß für jeden Befehl ein Abruf ausgeführt wird. Unter diesem Gesichtpunkt bieten die "pusha"- und "pushad"-Befehle eine starke Verbesserung beim Schreiben der Inhalte aller acht Register in den Stapel. Das ist allerdings kein Vorteil, wenn weniger als die acht Register geschrieben werden, beispielsweise vier oder sechs Register.

[0021] Anders ausgedrückt, wenn die "pusha"-, "pushad"-, "popa"-, und "popad"-Befehle des 80386 auf sämtliche Register einwirken, müssen Befehle mit verzögerten Zykluslängen benutzt werden, wenn es nicht nötig ist, die Inhalte aller Register zu sichern und wiederherzustellen. In einem solchen Fall reicht ein Befehl aus; aber das verursacht insofern ein Problem, als der Ausführungszyklus dieses Befehls zu lange dauert.

[0022] Eine ähnliche Sicherung und Wiederherstellung des Programmzählers ist nötig, wenn der Programmfluß sich in ein Unterprogramm verzweigt und von der aufgerufenen Routine zurückkehrt, wofür Befehle, wie der "call"-Befehl und "ret"-Befehl benutzt werden. Bei einer bekannten RISC-CPU werden diese Prozesse mittels Software verwirklicht. Mit anderen Worten, der Programmzähler wird gesichert und wiederhergestellt durch Ausführen von Assembler-Befehlen (Objektcode), welche diese Prozesse erklären. Das führt zu einer Vergrößerung des Objektcodes der "call"- und "ret"-Befehle, und die Ausführung eines Abrufs für jeden Befehl macht die Ausführungsgeschwindigkeit schlapp.

[0023] Das Dokument J. F. WAKERLY: "Micro-computer Architecture and Programming", 1981, JOHN WILEY & SONS, INC. NEW YORK, US XP-002084219 beschreibt die MOTOROLA 68000 Mikroprozessorarchitektur,

die Merkmale entsprechend denen des Oberbegriffs der Ansprüche 1 und 2 umfaßt. WO 9608767 A offenbart ein Mikrocontrollersystem, welches geeignet ist, Daten zwischen einem Speicherstapel und einer Vielzahl in einem Befehl aufgeführter allgemeiner Register in einem einzigen Befehlszyklus zu übertragen. Ein einziger Befehle, die verursachen, daß Daten für eine Vielzahl von Registern auf den Stapel draufgelegt oder vom Stapel abgehoben werden, umfassen unter anderem eine binäre Liste der betroffenen Register.

ZUSAMMENFASSUNG DER ERFINDUNG

[0024] Eine Aufgabe dieser Erfindung ist es, eine Datenverarbeitungsschaltung, einen Mikrorechner und ein elektronisches Gerät zu schaffen, deren Architektur eine wirksame Kodierung von Prozessen ermöglicht, welche auf den Stapelzeiger wirken, innerhalb einer kurzen Befehlslänge, zur Ausführung.

[0025] Eine weitere Aufgabe dieser Erfindung ist es, eine Datenverarbeitungsschaltung, einen Mikrorechner und ein elektronisches Gerät zu schaffen, das eine wirksame Kodierung von Prozessen ermöglicht, welche den Inhalt von Registern sichern und wiederherstellen, mit rascher Verarbeitung von Interrupts und Unterprogrammaufrufen und Rücksprüngen.

[0026] Diese Ziele werden von einer Datenverarbeitungsschaltung gemäß den Ansprüchen 1 und 2 und den in den abhängigen Ansprüchen genannten, bevorzugten Ausführungsbeispielen erzielt.

[0027] Gemäß der Erfindung wird es möglich gemacht, das Sichern des Stapels der Inhalte einer Vielzahl von Registern, spezifiziert durch verknüpfte sequentielle Werte, und das Wiederherstellen von Daten an eine Vielzahl von Registern, spezifiziert durch verknüpfte sequentielle Werte, nur mit einer Zähleinrichtung und einer einfachen Folgesteuerung zu verwirklichen. Da es hierdurch möglich ist, eine Datenverarbeitungsschaltung mit einer kleinen Anzahl Gates vorzusehen, kann diese Erfindung auf Einchip-Mikrorechner oder dergleichen angewandt werden.

[0028] Wenn es eine Vielzahl allgemeiner Register gibt, gibt es üblicherweise Adressen zum Spezifizieren dieser Register. Ein bevorzugtes Ausführungsbeispiel spezifiziert diese Register durch Registernummer in kontinuierlicher Folge von 0 bis n – 1.

[0029] Dieser Aspekt der Erfindung macht es möglich, mindestens ein Push oder Pop von Daten zwischen Speicher und einer Vielzahl kontinuierlicher Register vom Register 0 zum Register mit einer letzten Registernummer auszuführen, indem irgendeine Registernummer als die letzte Registernummer spezifiziert wird. Deshalb kann der Inhalt der Register gesichert und wirksam wiederhergestellt werden während der Ausführung eines Programmes, das eine Struktur hat, die Register in Folge vom Register Nummer 0 benutzt.

[0030] Gemäß einem Ausführungsbeispiel könnte ein Unterprogramm so etwas sein wie eine Interrupt-Verarbeitung, Ausnahmeverarbeitung oder Fehlersuchverarbeitungsroutine. Deshalb könnte ein Befehl zum Abzweigen zu einem Unterprogramm so etwas sein wie ein Befehl, der ein Unterprogramm oder dergleichen aufruft, oder ein Software-Interrupt-Befehl oder Software-Fehlersuch-Interrupt-Befehl zum Verzweigen zu einer Interrupt-Verarbeitung, Ausnahmeverarbeitung oder Fehlersuchverarbeitungsroutine. Ähnlich könnte ein Rückkehrbefehl von einem Unterprogramm ein Rückkehrbefehl, beispielsweise von einer Interrupt-Verarbeitung, Ausnahmeverarbeitung und Fehlersuchverarbeitungsroutine sein.

[0031] Üblicherweise ist es nötig, den Programmzähler zu sichern und wiederherzustellen, wenn zu einem Unterprogramm abgezweigt oder von einem Unterprogramm zurückgekehrt wird.

[0032] Dieses Ausführungsbeispiel macht es möglich, den Programmzähler gleichzeitig mit der Ausführung des Befehls zum Abzweigen zum Unterprogramm und des Befehls zur Rückkehr vom Unterprogramm zu sichern und wiederherzustellen. Mit anderen Worten, die Datenverarbeitungsschaltung gemäß diesem Aspekt der Erfindung hat einen Schaltungsaufbau, der das Sichern und Wiederherstellen des Programmzählers durch einen oder den anderen Befehl innerhalb eines Befehls zum Abzweigen zu einem Unterprogramm und einem Rückkehrbefehl von dem Unterprogramm. Es besteht also kein Bedarf an Befehlen für das Sichern und Wiederherstellen des Programmzählers, was nötig ist, wenn zu einem Unterprogramm abgezweigt und von diesem zurückgekehrt wird, was eine Verringerung der Anzahl Befehle ermöglicht. Das macht es möglich, die Verarbeitungsgeschwindigkeit während einer Abzweigung zu einer anderen Routine, beispielsweise einem Unterprogrammaufruf und Rückkehr zu beschleunigen, ohne Zyklen zu verschwenden.

[0033] Wenn zum Beispiel ein Software-Interrupt-Befehl erzeugt wird, muß das Prozessorzustandsregister,

welches den gegenwärtigen Zustand der CPU oder einer anderen Datenverarbeitungsschaltung hält, gesichert und wiederhergestellt werden. Aus diesem Grund wird vorzugsweise dieses Sichern und Wiederherstellen des Prozessorzustandsregisters gleichzeitig mit dem Ausführen eines Befehls durchgeführt, beispielsweise eines Software-Interrupt-Befehls. Gemäß einem weiteren Ausführungsbeispiel benutzt die Datenverarbeitungsschaltung Befehle eines RISC.

[0034] Eine RISC-Datenverarbeitungsschaltung ist mit den Zielen einer Verkleinerung der Dimensionen der Hardware und einer Vergrößerung der Geschwindigkeit derselben ausgelegt. Aus diesem Grund hat eine RISC-Datenschaltung viele Mehrzweck-Register. Eine Verringerung der Anzahl Befehle wird erreicht durch die Verwendung nur eines Befehlssatzes, der universelle Gültigkeit hat.

[0035] Deshalb ist bei einer RISC-Datenverarbeitungsschaltung der Stapelzeiger einem allgemeinen Register zugeordnet, und die Verarbeitung benutzt einen Befehlssatz, der auf dieses Mehrzweck-Register wirkt, wenn der Stapelzeiger bearbeitet wird. Aber dieses Verfahren führt zu einer größeren Befehlslänge, so daß der Speichernutzungsgrad gering ist.

[0036] Dieser Aspekt der Erfindung macht es möglich, den Wirkungsgrad, mit dem eine RISC-Datenverarbeitungsschaltung den Speicher nutzt, durch Reduzieren der Befehlslänge zu erhöhen.

[0037] Bei einem Vergleich der Benutzung von Befehlen fester Länge mit der Benutzung von Befehlen veränderlicher Länge ist es möglich, die zum Dekodieren der Befehle nötige Zeit zu verkürzen und die Dimensionen der Datenverarbeitungsschaltung selbst zu verkleinern. Um sicherzustellen, daß der Speicher bei Verwendung von Befehlen fester Länge durch das Vermeiden der Schaffung ungenutzter Teile innerhalb des Befehls wirksam ausgenutzt wird, ist es vorteilhaft, eine geringe Schwankung in der Anzahl Bits zu haben, die für jeden Befehl erforderlich sind, so daß diese so kurz wie möglich sind.

[0038] Ein weiteres Ausführungsbeispiel der Erfindung macht es möglich, die Länge von Befehlen zu verringern, die auf den Stapelzeiger wirken und deshalb dazu neigen, zu lang zu werden. Damit ist es möglich, das Entstehen unbenutzter Teile innerhalb von Befehlen zu vermeiden, selbst wenn Befehle fester Länge benutzt werden, so daß Speicher wirkungsvoller genutzt werden kann.

KURZBESCHREIBUNG DER ZEICHNUNGEN

[0039] Fig. 1 ist eine schematische Ansicht des Schaltkreisaufbaus einer CPU gemäß einem ersten Ausführungsbeispiel dieser Erfindung;

[0040] Fig. 2 zeigt einen Registersatz der CPU dieses Ausführungsbeispiels;

[0041] Fig. 3 ist eine Ansicht zur Veranschaulichung des grundlegenden Betriebs des Stapelzeigers;

[0042] Fig. 4 zeigt ein Beispiel der Verwendung eines zweckgebundenen Stapelzeigerbefehls ebenso wie den Verwendungszustand des im Speicher vorgesehenen Stapels und den Zustand des Stapelzeigers;

[0043] Fig. 5 ist eine Ansicht zur Veranschaulichung des Prozesses der Übertragung von Auto-Variablen im Speicher an Register;

[0044] Fig. 6A und Fig. 6B zeigen die Bitfelder eines SP-bezüglichen Ladebefehls und eines Allzweckladebefehls;

[0045] Fig. 7 ist ein Ablaufdiagramm des Betriebs während des SP-bezüglichen Ladebefehls zum Lesen von Wortdaten;

[0046] Fig. 8 ist ein Ablaufdiagramm des Betriebs während des SP-bezüglichen Ladebefehls zum Schreiben von Wortdaten;

[0047] Fig. 9A bis Fig. 9F sind Ansichten des Benutzungszustands des Stapels im Speicher und des Zustands des Stapelzeigers während jeder Routine eines Ausführungsprogramms, das sich über eine Vielzahl von Routinen erstreckt;

[0048] Fig. 10A und Fig. 10B zeigen die Bitfelder eines Stapelzeigerbewegungsbefehls und eines allgemei-

nen Rechenbefehls für unmittelbare Daten;

[0049] Fig. 11 ist ein Ablaufdiagramm des Betriebs während des Stapelzeigerbewegungsbefehls zur Addition;

[0050] Fig. 12 ist ein Ablaufdiagramm des Betriebs während des Stapelzeigerbewegungsbefehls zur Subtraktion;

[0051] Fig. 13 ist eine Ansicht einer Programmausführungssteuerung während der call- und ret-Befehle;

[0052] Fig. 14 zeigt das Feld der Bits eines call-Befehls einer PC-bezüglichen Subroutine;

[0053] Fig. 15 ist ein Ablaufdiagramm des Betriebs während des call-Befehls der PC-bezüglichen Subroutine;

[0054] Fig. 16 ist ein Ablaufdiagramm des Betriebs während des ret-Befehls;

[0055] Fig. 17A und Fig. 17B zeigen schematisch die Aktionen während der Ausführung von Push-Befehlen (push);

[0056] Fig. 18A und Fig. 18B zeigen schematisch die Aktionen während der Ausführung von Pop-Befehlen (pop);

[0057] Fig. 19 zeigt die Bitstruktur der pushn- und popn-Befehle;

[0058] Fig. 20 ist ein Blockschaltbild der zum Ausführen des sequentiellen push-Befehls (pushn) oder des sequentiellen pop-Befehls (popn) benötigten Hardwarekonfiguration;

[0059] Fig. 21 ist ein Ablaufdiagramm des Betriebs während des pushn-Befehls;

[0060] Fig. 22 ist ein Ablaufdiagramm des Betriebs während des popn-Befehls; und

[0061] Fig. 23 ist ein Blockschaltbild der Hardware eines Mikrorechners entsprechend einem zweiten Ausführungsbeispiel dieser Erfindung.

BESCHREIBUNG BEVORZUGTER AUSFÜHRUNGSBEISPIELE

[0062] Ausführungsbeispiele dieser Erfindung werden nachfolgend unter Hinweis auf die beigefügten Zeichnungen beschrieben.

Ausführungsbeispiel 1

(1) Konfiguration der CPU dieses Ausführungsbeispiels

[0063] Die CPU dieses Ausführungsbeispiels führt dank ihrer Pipeline und Lade/Speicherarchitektur im wesentlichen alle ihre Befehle innerhalb eines Zyklus aus. Alle Befehle sind als 16-Bit Befehle fester Länge ausgedrückt, so daß von der CPU dieses Ausführungsbeispiels verarbeitete Befehle innerhalb eines kompakten Objektcodes verwirklicht sind.

[0064] Insbesondere ist diese CPU so gestaltet, daß sie ein zweckgebundenes Stapelzeigerregister hat, um das effiziente Kodieren und Ausführen von Prozessen zu ermöglichen, die auf den Stapelzeiger wirken. Gleichfalls ermöglicht ist das Dekodieren und die Ausführung eines Befehlssatzes einer Gruppe zweckgebundener Stapelzeigerbefehle mit einem Objektcode, der dieses zweckgebundene Stapelzeigerregister als einen implizierten Operanden spezifiziert.

[0065] Eine schematische Ansicht des Schaltkreisaufbaus der CPU dieses Ausführungsbeispiels ist zur Veranschaulichung in Fig. 1 gezeigt.

[0066] Diese CPU **10** weist einen Registersatz auf, der Mehrzweck-Register **11**, ein PC-Register **12** zum Halten des Programmzählers, PSR (Prozessorzustandsregister) **13** sowie ein zweckgebundenes Stapelzeigerregister SP **14** umfaßt; einen Befehlsdekodierer **20**, einen Generator unmittelbarer Daten **22**; einen Adressenad-

dierer **30**; ein ALU **40**; einen PC-Inkrementer **44**; verschiedene interne Busse **72**, **74**, **76** und **78**; verschiedene interne Signalleitungen **82**, **84**, **86** und **88**.

[0067] Der Befehlsdekodierer **20** dekodiert den eingegebenen Objektcode, führt die zum Ausführen des resultierenden Befehls notwendige Verarbeitung durch und gibt alle notwendigen Steuersignale aus. Es sei angemerkt, daß dieser Befehlsdekodierer **20** auch als Dekodiereinrichtung zum Dekodieren des Objektcodes des oben erwähnten zweckgebundenen Stapelzeigerbefehls und zur Ausgabe von Steuersignalen auf der Grundlage dieses Befehls dient.

[0068] Der Generator unmittelbarer Daten **22** erzeugt auf der Basis unmittelbarer Daten, die innerhalb des Objektcodes enthalten sind, 32-Bit unmittelbare Daten, die während der Ausführung benutzt werden und erzeugt die konstanten Daten 0, ±1, ±2 und ±4, die für die Ausführung jedes Befehls erforderlich sind. Der PC-Inkrementer **44** aktualisiert den im PC **12** gespeicherten Programmzähler auf der Basis der Ausführungszyklen der Befehle. Der Adressenaddierer **30** addiert die in den verschiedenen Registern gespeicherten Daten und die vom Generator für unmittelbare Daten **22** erzeugten unmittelbaren Daten, um die beim Lesen der Daten aus dem Speicher benötigten Adressendaten zu erzeugen. Das ALU **40** führt numerische Berechnungen und logische Operationen durch.

[0069] Diese CPU weist auch einen internen Bus und Signalleitungen auf. Zu den Funktionen eines PA_BUS **72** und eines PB_BUS **74** gehört die Übertragung eingegebener Signale für das ALU **40**. Zu den Funktionen eines WW_BUS **76** gehört das Abrufen von Rechenergebnissen aus dem ALU **40**, um diese an die allgemeinen Register zu übertragen. Zu den Funktionen eines XA_BUS **78** gehört das Übertragen von Adressendaten, die aus den allgemeinen Registern **11** und dem SP **14** abgerufen wurden. Eine IA-Signalleitung **82** überträgt Adressendaten von den verschiedenen Bauelementen innerhalb der CPU an einen externen I_ADDR_BUS **92**. Eine DA-Signalleitung **84** überträgt Adressendaten von den verschiedenen Bauelementen innerhalb der CPU an einen externen D_ADDR_BUS **96**. Eine DIN-Signalleitung **86** überträgt Daten von einem externen D_DATA_BUS **98** an die verschiedenen Bauelemente innerhalb der CPU. Eine DOUT-Signalleitung **88** überträgt Daten von den verschiedenen Bauelementen innerhalb der CPU an den externen D_DATA_BUS **98**. Ein IA-Multiplexer **83** schaltet zwischen Signalen, die an die IA-Signalleitung **82** auszugeben sind (Signale auf dem PA_BUS **72**, auf dem WW_BUS **76**, vom PC **12** und einem Wert, der PC plus 2 ist). Ein DOUT-Multiplexer **89** schaltet zwischen Signalen, die an die DOUT-Signalleitung **88** auszugeben sind (Signale auf dem PA_BUS **72**, auf dem WW_BUS **76**, vom PC **12**, und dem Wert, der PC plus 2 ist).

[0070] Da die Bauelemente der CPU **10** Befehle auf der Basis von Steuersignalen ausführen, die vom Befehlsdekodierer **20** ausgegeben werden, wirken sie auch als eine Einrichtung zum Ausführen einer Gruppe zweckgebundener Stapelzeigerbefehle anhand der oben beschriebenen Steuersignale und des Inhalts des zweckgebundenen Stapelzeigerregisters.

[0071] Diese CPU **10** überträgt Signale zum und vom Äußeren über einen 16-Bit Befehlsdatenbus (I_DATA_BUS) **94**, einen Befehlsadressenbus (I_ADDR_BUS) **92** für Befehlsdatenzugriff, den 32-Bit Datenbus (D_DATA_BUS) **98**, den Datenadressenbus (D_ADDR_BUS) **96** für Datenzugriff und einen in der Figur nicht gezeigten Steuerbus für Steuersignale.

(2) Registersatz der CPU dieses Ausführungsbeispiels

[0072] Wesentliche Teile der Idee des Registersatzes der CPU dieses Ausführungsbeispiels sollen nun beschrieben werden.

[0073] Der Registersatz der CPU dieses Ausführungsbeispiels ist in **Fig. 2** gezeigt. Diese CPU hat einen Registersatz, der die **16** allgemeinen Register **11**, das PC **12**, das PSR **13**, das SP **14**, ein ALR (arithmetic low register) **15**, sowie ein AHR (arithmetic high register) **16** aufweist.

[0074] Die allgemeinen Register **11** sind 32-Bit Registern funktionsmäßig gleichwertig und mit R0 bis R15 bezeichnet. Diese allgemeinen Register **11** werden während Datenrechnungen und Adressenberechnungen benutzt.

[0075] Das PC **12** ist ein 32-Bit langer Inkrementalrechner, der einen Programmzähler hält, welcher die Adresse des aktuell in Ausführung befindlichen Befehls anzeigt. Im vorliegenden Schriftstück wird "PC" benutzt, wenn es sich auf das Register selbst bezieht und "Programmzähler" ("program counter") wird benutzt, wenn auf den innerhalb des PC gespeicherten Wert Bezug genommen wird.

[0076] Auf das PC **12** kann von Befehlen, wie dem Ladebefehl nicht direkt zugegriffen werden. Bei einem call- oder int-Befehl, oder wenn ein Interrupt oder Ausnahme auftritt, wird der Programmzähler aus dem PC **12** gelesen und zum Stapel gesichert. So kann im PC eine Sprungzieladresse gesetzt werden, wenn ein Verzweigungsbefehl ausgeführt wird. Das geschieht auch bei einer durch einen konditionellen Verzweigungsbefehl spezifizierten Abzweigung. Die Befehlsadresse für das Rückkehrziel wird aus dem Stapel mittels eines ret- oder reti-Befehls gelesen und im PC **12** ersetzt.

[0077] Das PSR (Prozessorzustandsregister) **13** ist ein 32-Bit Register, dem ein Kennzeichen zugeordnet ist und das den aktuellen Zustand der CPU hält. Wenn ein int-Befehl, ein Interrupt, eine Ausnahme oder dergieichen auftritt, wird der Zustand des PSR in diesem Zeitpunkt zum Stapel gesichert, wenn der Fluß zur entsprechenden Verarbeitungsroutine abzweigt. Umgekehrt verursacht die Ausführung des reti-Befehls, daß der gesicherte Wert im PSR ersetzt wird.

[0078] Das SP **14** ist ein 32-Bit zweckgebundenes Stapelzeigerregister, welches den Stapelzeiger enthält, der die höhere Adresse des Stapels anzeigt. Im vorliegenden Schriftstück wird SP benutzt, wenn auf das Register selbst Bezug genommen wird, während Stapelzeiger (stack pointer) benutzt wird, wenn auf den innerhalb des SP gespeicherten Wert Bezug genommen wird. Es sei jedoch angemerkt, daß, da der Stapelzeiger immer auf eine Wortgrenze zeigt, die beiden tiefsten Bits des Stapelzeigers immer Null sind.

[0079] Dieser Stapelzeiger wird bei der Erzeugung einer Falle (trap) oder bei Ausführung eines aus einer Gruppe zweckgebundener Stapelzeigerbefehle aktualisiert, die bei diesem Ausführungsbeispiel geboten werden.

[0080] Beispiele solcher zweckgebundener Stapelzeigerbefehle, mit denen der Stapelzeiger aktualisiert wird, sind Befehle, die zu einer anderen Routine abzweigen, beispielsweise die call- und ret-Befehle, ein Stapelzeigerbewegungsbefehl, ein pushn-Befehl sowie ein popn-Befehl. Bei der Ausführung des call-Befehls beispielsweise wird der Stapelzeiger zunächst um einen Betrag dekrementiert, der der Wortgröße (–4) gleicht, dann wird das PC **12** zum Stapel gesichert. Umgekehrt wird bei Ausführung des ret-Befehls die Zieladresse zur Rückkehr aus dem Stapel in das PC geladen, und der Stapelzeiger wird um einen Betrag inkrementiert, der der Wortgröße (+4) gleicht. Wenn ein int-Befehl ausgeführt wird, oder wenn ein Interrupt oder Ausnahme auftritt, wird der Wert im PC oder PSR auf folgende Weise zum Stapel gesichert:

1. SP = SP – 4
2. Das PC wird an der höheren Adresse des Stapels, angezeigt durch den Stapelzeiger, gesichert.
3. SP = SP – 4
4. Das PSR wird an der höheren Adresse des Stapels, angezeigt durch den Stapelzeiger, gesichert.

[0081] Bei Ausführung des reti-Befehls wird der obige Prozeß umgekehrt, um die CPU in ihren ursprünglichen Zustand zurückzuversetzen. Die Ausführung des call-, ret- oder int-Befehls veranlaßt also, daß der Stapelzeiger so aktualisiert wird, wie es für diese Ausführung angemessen ist. Einzelheiten jedes Befehls werden weiter unten angegeben.

[0082] Eine von diesem Ausführungsbeispiel gebotene trap-Funktion weist Interrupts, die asynchron mit der Ausführung von Befehlen erzeugt werden, sowie Ausnahmen auf, die durch die Ausführung von Befehlen erzeugt werden. Wenn ein trap erzeugt wird, sichert die CPU PC (Programmzähler) und PSR (Prozeßzustandsregister) zum Stapel, liest dann eine Vectortabelle aus einer trap-Tabelle und zweigt zu einer Routine entsprechend diesem trap ab. Gleichzeitig mit der Erzeugung des trap wird ein IE (interrupt enable – Interrupt-Aktivierung) Bit gelöscht und die Erzeugung anschließender maskierbarer Interrupts unterbunden. Maskierbare externe Interrupts werden durch einen Ladebefehl in Bezug auf das PSR zum Schreiben von 1 an die Stelle des IE Bits wieder aktiviert.

[0083] Der reti-Befehl dient zur Rückkehr von der trap-Verarbeitungsroutine zur ursprünglichen Routine. Wenn der reti-Befehl ausgeführt wird, liest die CPU das PSR und das PC in dieser Reihenfolge aus dem Stapel, stellt das PSR mit seinem ursprünglichen Wert wieder her und zweigt gleichfalls zur Rückadresse ab. Es sei erwähnt, daß zu den Ausnahmen Fehlersuchausnahmen, Adressenfehlausrichtungsausnahmen, Überlaufausnahmen sowie Nulldivisionsausnahmen zählen.

[0084] Das ALR (arithmetic low register) **15** und ALH (arithmetic high register) **16** wird hier nicht im einzelnen beschrieben.

[0085] Die Spezialregister PSR **13**, SP **14**, ALR **15** und AHR **16** der CPU können mit Hilfe eines Ladebefehls

Daten zu und von den allgemeinen Registern übertragen. Jedes Register hat eine spezielle Registernummer, und unter dieser Nummer wird auf das Register Zugriff genommen.

Spezialregistername	Spezialregisternummer	Assembler-Mnemonik
Prozeßzustandsregister	0	%PSR
SP	1	%SP
Arthmetic low register	2	%ALR
Arithmetic high register	3	%AHR

(3) Beschreibung des Stapels und Stapelzeigers

[0086] Der Stapel ist ein im Speicher vorgesehener zeitweiliger Speicherbereich. Es handelt sich um einen angrenzenden Bereich, in den Daten von unten nach oben eingespeichert werden, als ob sie in einem Regal gestapelt würden. Der Stapelzeiger zeigt die Adresse der Daten oben im Stapel an, das heißt der als Letztes in den Stapel geschriebenen Daten.

[0087] Der grundlegende Betrieb des Stapelzeigers wird nunmehr unter Hinweis auf Fig. 3 beschrieben.

[0088] Bezugszeichen **100** in Fig. 3 bezeichnet einen im Speicher vorgesehenen Stapelbereich. Ein gestrichelter Teil **102** gibt Daten wieder, die als Letztes in diesem Bereich gespeichert wurden, und eine Speicheradresse **1000** zeigt diese Daten an. Es sei erwähnt, daß ein Bereich **104** unterhalb des gestrichelten Teils **102** Daten enthält, die bereits gespeichert wurden, und ein Teil **106** oberhalb des gestrichelten Teils **102** steht für zukünftiges Speichern von Daten zur Verfügung.

[0089] Der Stapelzeiger zeigt immer eine Wortgrenze an. Das bedeutet, daß beim Schreiben von Daten in den Stapel der Stapelzeiger innerhalb des SP **14** um 4 nach oben bewegt wird, und daß die Daten an dem Ort gespeichert werden, der von diesem Stapelzeiger angezeigt wird. Wenn im Stapel gespeicherte Daten abgerufen werden, sind es die Daten an der vom aktuellen SP **14** angezeigten Adresse, die abgerufen werden, und der im SP **14** gespeicherte Stapelzeiger wird um 4 nach unten bewegt. Auf diese Weise zeigt der Stapelzeiger immer die Speicheradresse derjenigen Daten an, die als Letztes im Stapel gespeichert wurden.

(4) Beschreibung einer Gruppe zweckgebundener Stapelzeigerbefehle

[0090] Eine RISC-CPU arbeitet immer mit einem allgemeinen Register als Stapelzeiger. Dieses Ausführungsbeispiel hat das SP **14**, bei dem es sich um ein zweckgebundenes Stapelzeigerregister handelt, auf das die zuvor erwähnte Gruppe zweckgebundener Stapelzeigerbefehle einwirkt.

[0091] Diese Gruppe zweckgebundener Stapelzeigerbefehle ist ein generischer Name für eine Anzahl von Befehlen, die auf das SP **14** als impliziter Operand Zugriff nehmen und Operationen am SP **14** durchführen. Die Gruppe zweckgebundener Stapelzeigerbefehle weist Befehle auf, die zu anderen Routinen abzweigen, beispielsweise SP-bezügliche Ladebefehle (ld, etc.), Stapelzeigerbewegungsbefehle (add, sub), Befehle zum Abzweigen zu Subroutinen (call, etc.) und Rückkehrbefehle (ret, etc.); einen sequentiellen Push-Befehl (pushn); und einen sequentiellen Pop-Befehl (popn).

[0092] Diesen Befehlen ist gemeinsam, daß sie zweckgebundene Stapelzeigerbefehle sind, so daß es keine Notwendigkeit zur Datenspezifizierung des Stapelzeigers im Objektcode gibt. Ein weiteres gemeinsames Merkmal besteht darin, daß sie das effiziente Kodieren von Prozessen, die den Stapelzeiger nutzen, in kurzen Befehlen ermöglichen.

[0093] Die Anwendung dieser Befehle macht es möglich, in einem im Speicher vorgesehenen Stapel gespeicherte Daten wirksam zu verarbeiten. Wirksam verarbeitet werden können ebenfalls Interrupt- und Subroutine-Abruf/Rückkehr-Prozesse.

[0094] Ein Beispiel für die Anwendung dieser zweckgebundenen Stapelzeigerbefehle bei Abruf einer Subroutine soll nun unter Hinweis auf Fig. 4 beschrieben werden. Diese Figur zeigt auch den Verwendungszustand des im Speicher vorgesehenen Stapels sowie den Zustand des Stapelzeigers. Ein MAIN-Programm **500** und eine Subroutine **520** in Fig. 4 sind als ein von einem C-Kompilierer geschaffener Objektcode geschrieben. Bezugszeichen **540** bezeichnet den Zustand eines Stapels im Speicher. Bezugszeichen **502** zeigt an, daß eine Verarbeitung durchgeführt wird, wofür die allgemeinen Register R0 bis R3 benutzt werden. Bezugszeichen **506**

bezeichnet einen Subroutineaufrufbefehl. Ehe der Subroutineaufrufbefehl vom MAIN-Programm **500** ausgeführt wird, mit anderen Worten, nachdem der mit **504** bezeichnete vorhergehende Befehl ausgeführt worden ist, zeigt der Stapelzeiger (SP) auf eine Adresse <1> im Stapel im Speicher. Wenn dieser Subroutineaufrufbefehl ausgeführt wird, geht die Steuerung zur Subroutine **520** über. In diesem Zeitpunkt erfolgt die Ausführung eines Befehls zum Abzweigen zu einer Subroutine (call-Befehl), bei der es sich um eine der zweckgebundenen Stapelzeigerbefehle dieses Ausführungsbeispiels handelt. Wenn dieser Befehl ausgeführt wird, wird der Wert des Stapelzeigers (SP) automatisch um –4 inkrementiert (bei <2> in Fig. 4), und die Adresse zur Rückkehr zum MAIN-Programm wird in einem Bereich **544** des Stapels gespeichert, der vom Stapelzeiger <2> angezeigt wird.

[0095] Wenn die Subroutine **520** mit dem Ausführen beginnt, überträgt sie zunächst die Werte, die in den vom MAIN-Programm benutzten allgemeinen Registern R0 bis R3 gespeichert sind. Bezugszeichen **524** bezeichnet einen sequentiellen Push-Befehl (pushn), bei dem es sich um einen der zweckgebundenen Stapelzeigerbefehle handelt, der die in den allgemeinen Registern R0 bis R3 gespeicherten Werte zum Stapel sichert. Beim Ausführen dieses Befehls werden die in den allgemeinen Registern R0 bis R3 gespeicherten Werte der Reihe nach zum Stapel übertragen, so daß sie im Stapel **540** so gespeichert werden, wie bei **550** in Fig. 4 gezeigt. Wenn die Ausführung dieses Prozesses endet (**524** <3>), befindet sich der Stapelzeiger (SP) auf **546** (<3> SP).

[0096] Die Subroutine **520** stellt dann einen von der Subroutine zu benutzenden Bereich Auto-Variabler sicher. Ein mit **526** bezeichneter Add-Befehl ist ein Stapelzeigerbewegungsbefehl, bei dem es sich um einen zweckgebundenen Stapelzeigerbefehl handelt, der verursacht, daß der Stapelzeiger nach oben bewegt wird, um einen Stapelbereich sicherzustellen, den die Subroutine **520** benutzen soll. Wenn dieser Befehl ausgeführt wird, bewegt sich der Stapelzeiger (SP) um X Bytes nach oben zu einem Ort **548** (<4> SP), um einen Bereich Auto-Variabler sicherzustellen, der von der Subroutine benutzt werden soll, wie in der Anmerkung 2 ausgedrückt.

[0097] Bezugszeichen **528** bezeichnet eine Verarbeitung, die Auto-Variable und die allgemeinen Register R0 bis R3 innerhalb der Subroutine **520** benutzt. Hierbei ist der Ort des Stapelzeigers mit <5> bezeichnet, und das Laden der Auto-Variablen wird durch Benutzung eines zweckgebundenen SP-Ladebefehls durchgeführt, bei dem es sich um einen der zweckgebundenen Stapelzeigerbefehle handelt.

[0098] Bezugszeichen **529** bezeichnet diesen SP-Ladebefehl, der eine im Speicher gespeicherte Auto-Variable S1 an das Mehrzweck-Register R1 überträgt. Diese Auto-Variable S1 ist an einem Ort gespeichert, der um Y Bytes vom Stapelzeiger (<5> SP) versetzt ist. Der Stapelzeiger bewegt sich während der zuvor genannten Verarbeitung der Subroutine **528** nicht, so daß die Speicheradresse der Auto-Variablen vom Stapelzeiger plus Versatz spezifiziert wird. Das ermöglicht es, mit Hilfe des zweckgebundenen SP-Ladebefehls Daten zu und aus den allgemeinen Registern wirksam abzurufen.

[0099] Ehe die Steuerung von der Subroutine **520** zum MAIN-Programm **500** zurückkehrt, müssen die Werte in den allgemeinen Registern R0 bis R3, die zum Stapel gesichert wurden, in die allgemeinen Register R0 bis R3 umgespeichert und der Stapelzeiger muß zurückgesetzt werden, um den Bereich **544** anzuzeigen, der die Adresse für die Rückkehr zur MAIN-Routine enthält. Hiermit wird der Stapelzeiger, der vom Stapelzeigerbewegungsbefehl **526** bewegt wurde, an seinem ursprünglichen Ort wiederhergestellt. Ein mit **530** bezeichneter sub-Befehl ist ein Stapelzeigerbewegungsbefehl, bei dem es sich um einen zweckgebundenen Stapelzeigerbefehl handelt, der den Stapelzeiger nach unten bewegt. Wenn dieser Befehl ausgeführt wird, bewegt sich der Stapelzeiger zu **546** (<6> SP). Die im Stapel gespeicherten Daten **550** werden zu den allgemeinen Registern R0 bis R3 umgespeichert. Bezugszeichen **532** bezeichnet einen sequentiellen Pop-Befehl (popn), bei dem es sich um einen zweckgebundenen Stapelzeigerbefehl handelt, der Daten im Stapel zu den allgemeinen Registern R0 bis R3 überträgt. Wenn dieser Befehl ausgeführt wird, werden die im Stapel gespeicherten Werte **550** der Reihe nach an die allgemeinen Register R0 bis R3 übertragen, so daß sie im Stapel **540** so gespeichert werden, wie bei **550** in Fig. 4 gezeigt. Wenn die Ausführung dieses Prozesses endet (**532** <7>), zeigt der Stapelzeiger auf **544** (<7> SP).

[0100] Bezugszeichen **534** bezeichnet einen Rückkehrbefehl. Wenn dieser Rückkehrbefehl ausgeführt wird, geht die Steuerung zum MAIN-Programm über. In diesem Fall wird ein Rückkehrbefehl (ret-Befehl), bei dem es sich um einen zweckgebundenen Stapelzeigerbefehl handelt, von diesem Ausführungsbeispiel ausgeführt. Wenn dieser Befehl ausgeführt wird, zweigt die Steuerung zu einem Befehl ab, der die Adresse für die Rückkehr zum MAIN-Programm anzeigt, welches im Stapelbereich mit der Bezeichung <7> SP gespeichert ist. Mit anderen Worten, das MAIN-Programm **500** kehrt zum nächsten Befehl **507** zurück. Der Wert des Stapelzeigers wird automatisch um +4 inkrementiert, um ihn zum oberen Bereich des Stapels zu bewegen, der vom

MAIN-Programm **500** benutzt wird (<8> SP in Fig. 4).

[0101] Die Beschreibung wendet sich nunmehr Einzelheiten jedes der zweckgebundenen Stapelzeigerbefehle ebenso wie dem zum Ausführen dieser Befehle erforderlichen Schaltungsaufbau und dem Betrieb während der Ausführung zu.

(5) Den Stapelzeiger (SP) betreffende Ladebefehle

[0102] Ein C-Kompilierer erzeugt einen Objektcode, der spezifiziert, daß ein Bereich Auto-Variabler mit dem Stapelzeiger verknüpft werden soll. Genauer gesagt, wirkt der C-Kompilierer als ein Mittel, mit dem ein Bereich Auto-Variabler sichergestellt wird, der Y Bytes lang ist, mit einem Versatz von X gegenüber dem Stapelzeiger.

[0103] Eine Ansicht zur Veranschaulichung dieses Prozesses der Übertragung von Auto-Variablen aus dem Speicher an Register ist in Fig. 5 gezeigt. In dieser Figur bedeuten Auto-Variable a Wortdaten, Auto-Variable b und c je Halbwortdaten und Auto-Variable d bis g je Byte-Daten. Der in SP gespeicherte Stapelzeiger zeigt auf **1000**, wobei es sich um die Adresse des obersten Bereichs einer Zone **600** des Stapels handelt, in der Auto-Variable gespeichert sind. Ein Bereich des Stapels an der vom Stapelzeiger angezeigten Speicheradresse wird als ein Bereich sichergestellt, der die Auto-Variable a hält. Bereiche im Stapel an den vom Stapelzeiger plus 2 und dem Stapelzeiger plus 4 angezeigten Speicheradressen sind als Bereiche zum Halten der Auto-Variablen b bzw. c sichergestellt. Ähnlich sind Bereiche im Stapel an den vom Stapelzeiger plus 5, dem Stapelzeiger plus 6, dem Stapelzeiger plus 7 und dem Stapelzeiger plus 8 angezeigten Speicheradressen als Bereiche zum Halten der jeweiligen Auto-Variablen d bis g sichergestellt. Während der Ausführung eines gegebenen Prozesses kann es sich als nötig erweisen, Daten zwischen einem allgemeinen Register und einer Auto-Variablen zu übertragen, die von einer Speicheradresse spezifiziert ist, bei der es sich um den Stapelzeiger plus einem Versatz handelt.

[0104] Um es der CPU dieses Ausführungsbeispiels zu ermöglichen, die obige Übertragungsverarbeitung wirksam mit einem kürzeren Objektcode auszuführen, ist der folgende Befehlssatz als SP-bezügliche Ladebefehle vorgesehen, bei denen es sich um einige der zweckgebundenen Stapelzeigerbefehle handelt:

```
ld.b %Rd, [%sp + imm6] (1)
ld.ub %Rd, [%sp + imm6] (2)
ld.h %Rd, [%sp + imm7] (3)
ld.uh %Rd, [%sp + imm7] (4)
ld.w %Rd, [%sp + imm8] (5)
ld.b [%sp + imm6], %Rs (6)
ld.h [%sp + imm7], %Rs (7)
ld.w [%sp + imm8], %Rs (8)
```

[0105] Die Befehle (1) bis (8) sind Befehlscodes, die ein Assembler erzeugt hat. Mit dem Befehl (1) erfolgt eine Vorzeichenexpansion von Byte-Daten, die vom Stapel zu einem Register übertragen werden, mit dem Befehl (2) erfolgt eine Nullexpansion von Byte-Daten, die vom Stapel zu einem Register übertragen werden, mit dem Befehl (3) erfolgt eine Vorzeichenexpansion von Halbwortdaten, die vom Stapel zu einem Register übertragen werden, mit dem Befehl (4) erfolgt eine Nullexpansion von Halbwortdaten, die vom Stapel zu einem Register übertragen werden, mit dem Befehl (5) werden Wortdaten vom Stapel zu einem Register übertragen, mit dem Befehl (6) werden Byte-Daten von einem Register zum Stapel übertragen, mit dem Befehl (7) werden Halbwortdaten von einem Register zum Stapel übertragen, und mit dem Befehl (8) werden Wortdaten von einem Register zum Stapel übertragen.

[0106] Die Operanden [%sp + imm6], [%sp + imm7] und [%sp + imm8] stellen je unmittelbare Versatzdaten dar. Eine Speicheradresse wird geschaffen, indem ein Versatz, der während der Ausführung des Befehls auf der Basis dieser unmittelbaren Versatzdaten geschaffen wird, und der Wert des im SP **14** gespeicherten Stapelzeigers addiert werden. [%sp + imm6] sind unmittelbare Versatzdaten, die benutzt werden, wenn die Größe der Daten im Speicher Bytes ist, [%sp + imm7] ist das Entsprechende, wenn die Datengröße Halbwörter ist, und [%sp + imm8] ist das Entsprechende, wenn die Datengröße Wörter ist. Gleichgültig welche Datengröße benutzt wird, geschrieben wird sie als 6-Bit unmittelbare Versatzdaten **614** im Objektcode, wie weiter unten unter Hinweis auf Fig. 6A noch beschrieben wird. Wenn imm7 benutzt wird, mit anderen Worten, wenn die Datengröße Halbwörter ist, werden allerdings die unmittelbaren Versatzdaten **614** während der Ausführung des Befehls um ein Bit nach links verschoben, um einen zum Stapelzeiger zu addierenden Versatz zu generieren.

[0107] Wenn imm8 benutzt wird, mit anderen Worten, wenn die Datengröße Wörter ist, werden die unmittel-

baren Versatzdaten **614** ähnlich um zwei Bits nach links verschoben, um einen zum Stapelzeiger zu addierenden Versatz zu generieren.

[0108] In diesem Fall ist der Stapel eine im Speicher vorgesehene vorläufige Speicherzone, und die obigen Speicheradressen können spezifizier werden, um beispielsweise die Orte von Auto-Variablen (a bis g) innerhalb des Stapels zu spezifizieren, wie Fig. 5 zeigt.

[0109] Ein Beispiel eines Bitfeldes **610** dieser SP-betreffenden Ladebefehle (1) bis (8) ist in Fig. 6A gezeigt. Der in Fig. 6A gezeigte SP-betreffende Ladebefehl hat 16 Bits eines Objektcodes mit einem op-Code **612** (6 Bits), die anzeigen, daß die Betriebsfunktion eine Übertragung von Daten zwischen dem Speicher und einem allgemeinen Register ist, unmittelbare Versatzdaten **614** (6 Bits), die von unmittelbaren Daten spezifiziert sind, sowie eine Registernummer **616** (4 Bits), die das Register spezifiziert, das für die Übertragung zu benutzen ist. Zu dem op-Code **612** gehört ein gemeinsamer Code, der anzeigt, daß es sich hierbei um einen auf das SP **14** wirkenden Ladebefehl handelt, sowie unterschiedliche Codes, die je die Datengröße, Vorzeichenexpansion und Nullexpansion spezifizieren. Da es folglich aus dem op-Code klar ist, daß diese Befehle auf den im SP **14** gespeicherten Stapelzeiger wirken, ist es nicht nötig, den Stapelzeiger betreffende Daten in den Operanden des Objektcodes zu spezifizieren. Die unmittelbaren Versatzdaten **614** werden benutzt, um einen Versatz gegenüber dem Stapelzeiger für zu übertragende Daten zu schaffen. Das wird unabhängig von der Datengröße, wie schon gesagt, in 6 Bits spezifiziert. Mit den Befehlen (1) bis (5) enthält die Adresse (Registernummer) **616** des an der Übertragung beteiligten Registers die Nummer des Registers, welches Daten zu enthalten hat, die aus dem Stapel gelesen wurden; mit den Befehlen (6) bis (8) enthält es die Nummer des Registers, das aktuell Daten enthält, die in den Stapel zu schreiben sind.

[0110] Vergleichsweise zeigt Fig. 6B ein Beispiel eines Bitfeldes **620** eines Objektcodes eines Ladebefehls, der auf ein allgemeines Register wirkt, wenn ein anderes allgemeines Register als Stapelzeiger benutzt wird (nachfolgend allgemeiner Ladebefehl genannt).

[0111] Der allgemeine Ladebefehl gemäß Fig. 6B hat 20 Bits eines Objektcodes mit op-Code **622** (6 Bits), die anzeigen, daß die Betriebsfunktion eine Übertragung von Daten zwischen dem Speicher und einem allgemeinen Register ist, unmittelbare Versatzdaten **624** (6 Bits), die durch unmittelbare Daten spezifiziert sind, eine erste Registernummer **620** (4 Bits), die das als Stapelzeiger zu benutzende Mehrzweck-Register spezifiziert, und eine zweite Registernummer **628** (4 Bits), die das für die Übertragung zu benutzende Register spezifiziert. Bei einem Allzweckmikrorechner liegt die Befehlslänge in 8-Bit Einheiten vor, was 24-Bit oder 32-Bit Befehle ergibt.

[0112] Jede der beiden Fig. 6A und Fig. 6B zeigt einen Objektcode für einen Befehl, der bei der Übertragung von Daten zwischen einem Register und einer Speicheradresse benutzt wird, die durch Addieren eines Versatzes zum Stapelzeiger spezifiziert ist. Aus diesen Beispielen geht hervor, daß ein SP-betreffender Ladebefehl mit einem kürzeren Objektcode geschrieben werden kann als ein Allzweckladebefehl.

[0113] Es wird nunmehr die nötige Konfiguration beschrieben, um die obigen Befehle auszuführen, sowie die während der Ausführung auftretenden Operationen. Als Beispiele wird der Befehl (5) genommen, der Wortdaten vom Stapel an ein Register überträgt (da dieser Befehl Wortdaten aus dem Stapel in ein Register liest, wird er nachfolgend als SP-betreffender Ladebefehl zum Lesen von Wortdaten bezeichnet), sowie der Befehl (8), der Wortdaten von einem Register zum Stapel überträgt (da dieser Befehl Wortdaten aus einem Register in den Stapel schreibt, wird er nachfolgend als SP-betreffender Ladebefehl zum Schreiben von Wortdaten bezeichnet).

[0114] Zuerst wird unter Hinweis auf Fig. 1 die Gestaltung in Hardware beschrieben, die für die Ausführung der Befehle nötig ist. Jeder dieser Befehle wird über den I_DATA_BUS **94** von einem externen Speicher (ROM) **52** übertragen und in den Befehlsdekodierer **20** eingegeben. Der Befehl wird vom Befehlsdekodierer **20** dekodiert, der verschiedene, in der Figur nicht gezeigte Signale ausgibt, die zum Ausführen des Befehls erforderlich sind. Der Generator unmittelbarer Daten **22** führt eine nach links gerichtete logische Verschiebung an den unmittelbaren Versatzdaten **614** entsprechend der Datengröße durch, generiert den Versatz, der während der Ausführung der Vorzeichenexpansion oder Nullexpansion benutzt wird, falls nötig, und gibt die Ergebnisse an den PB_BUS **74** aus. Das SP **14** enthält den Stapelzeiger, und dieser Wert kann an den XA_BUS **78** ausgegeben werden, der mit einem Eingang des Adressenaddierers **30** verbunden ist. Ein weiterer Eingang des Adressenaddierers **30** ist mit dem PB_BUS **74** verbunden, der ein Ausgang des Generators unmittelbarer Daten **22** ist. Ein Ausgang (ADDR) des Adressenaddierers **30** ist mit dem externen D_ADDR_BUS **96** über die DA-Signalleitung **84** verbunden.

[0115] BCU **60** (Bussteuereinheit) steuert die Eingabe und Ausgabe von Daten in und aus dem Speicher (RAM und ROM) **50** und **52** (wozu der Stapelbereich gehört) und gibt READ- und WRITE-Steuersignale entsprechend verschiedenen Anforderungssignalen aus, die von der CPU ausgegeben werden (beispielsweise an externe Busse ausgegebene Signale).

[0116] Die Beschreibung betrifft zunächst den Betrieb während der Ausführung eines SP-betreffenden Ladebefehls zum Lesen von Wortdaten.

[0117] Wenn der SP-betreffende Ladebefehl zum Lesen von Wortdaten ausgeführt wird, wird der Wert des Stapelzeigers, der im SP **14** gespeichert ist, zu dem Versatz addiert, den der Generator unmittelbarer Daten **22** auf der Basis der unmittelbaren Versatzdaten **614** geschaffen hat, um eine Speicheradresse zum Lesen zu schaffen. Daten werden aus dem Speicher auf der Grundlage dieser Speicheradresse gelesen und an das Mehrzweck-Register übertragen, welches durch die Registernummer **616** im Objektcode spezifiziert ist.

[0118] Fig. 7 ist ein Ablaufdiagramm, welches die Operation während des SP-betreffenden Ladebefehls zum Lesen von Wortdaten veranschaulicht.

[0119] Nach dem Start der Ausführung dieses Befehls wird der im SP **14** gespeicherte Stapelzeiger an den XA_BUS **78** ausgegeben (Schritt S210). Ein vom Generator für unmittelbare Daten **22** aus den unmittelbaren Versatzdaten geschaffener Versatz imm wird dann an den PB_BUS **74** ausgegeben (Schritt S212). Der Adressenaddierer **30** summiert den Werf auf dem XA_BUS **78** und den Wert auf dem PB_BUS **74**, gibt dann die resultierende Speicherleseadresse (ADDR) über die DA-Signalleitung **84** an den D_ADDR_BUS **96** aus (Schritte S214 und S216). Ein Aufforderungssignal zum Datenlesen von der CPU an die BCU **60** wird aktiv und führt einen externen Speicherlesezyklus aus (Schritt S218). Anders ausgedrückt, die BCU **60** folgt dem Aufforderungssignal, das Lesen von Daten aus dem Speicher unter Verwendung dieser Leseadresse als Speicheradresse und die Ausgabe der Daten an den D_DATA_BUS **98** zu steuern. Die Daten auf dem D_DATA_BUS **98** werden an den WW_BUS **76** über die DIN-Signalleitung **86** ausgegeben (Schritt S220). Der Wert auf dem WW_BUS **76** wird in dem Register (%Rd) gespeichert, welches die Registernummer hat, die von der 4-Bit Adresse **616** des Registers (Ra/Rs) spezifiziert ist, welches im Befehlscode für die Übertragung spezifiziert ist (Schritt S222).

[0120] Nachfolgend wird die Operation während der Ausführung eines SP-betreffenden Ladebefehls zum Schreiben von Wortdaten beschrieben.

[0121] Wenn der SP-betreffende Ladebefehl zum Schreiben von Wortdaten ausgeführt wird, wird der im SP **14** gespeicherte Wert des Stapelzeigers zu dem Versatz addiert, den der Generator unmittelbarer Daten **22** auf der Basis der unmittelbaren Versatzdaten **614** geschaffen hat, um eine Speicheradresse zum Schreiben in den Speicher zu schaffen. Die Daten, die in dem von der Registernummer **616** im Objektcode spezifizierten allgemeinen Register gespeichert sind, werden in die Bereiche im Speicher übertragen, die von dieser Speicheradresse spezifiziert sind.

[0122] Fig. 8 zeigt ein Ablaufdiagramm der Operation während eines SP-betreffenden Ladebefehls zum Schreiben von Wortdaten.

[0123] Beim Start der Ausführung dieses Befehls wird der im SP **14** gespeicherte Stapelzeiger an den XA_BUS **78** ausgegeben (Schritt S230). Dann wird ein vom Generator unmittelbarer Daten **22** anhand der unmittelbaren Versatzdaten geschaffener Versatz imm an den PB_BUS **74** ausgegeben (Schritt S232). Der Adressenaddierer **30** summiert den Wert auf dem XA_BUS **78** und den Wert auf dem PB_BUS **74**, gibt dann die resultierende Speicherschreibadresse (ADDR) über die DA-Signalleitung **84** an den D_ADDR_BUS **96** aus (Schritte S234 und S236). Die Daten, die in dem Register (%Rd) gespeichert sind, das die von der 4-Bit-Adresse **616** des Registers (Rs/Rd) spezifizierte Nummer hat, welches im Befehlscode für die Übertragung spezifiziert ist, werden an den PA_BUS **72** ausgegeben (Schritt **238**). Die Daten auf dem PA_BUS **72** werden über die DOUT-Signalleitung **88** an den D_DATA_BUS **98** ausgegeben (Schritt S240). Ein Aufforderungssignal zum Schreiben von Daten von der CPU an die BCU **60** wird aktiv und führt einen externen Speicherschreibzyklus aus (Schritt S242). Mit anderen Worten, die BCU **60** folgt dem Aufforderungssignal und steuert die Operation des Schreibens der Daten in den Speicher, die an den D_DATA_BUS **98** übertragen wurden, und benutzt dazu diese Schreibadresse als eine Speicheradresse.

(6) Stapelzeigerbewegungsbefehl

[0124] Die Fig. 9A bis Fig. 9F zeigen Ansichten zur Veranschaulichung des Verwendungszustands des Stapels im Speicher und des Zustands des Stapelzeigers während jeder Routine eines Programms, das sich über eine Vielzahl von Routinen erstreckt.

[0125] Der Zustand der Stapelzone im Speicher und des Stapelzeigers (im SP **14** gespeicherter Wert) während der Ausführung eines gegebenen Prozesses 'a' **211** einer MAIN-Routine **210** gemäß Fig. 9A ist in Fig. 9D gezeigt. Bezugszeichen **222** bezeichnet eine Stapelzone, die von der MAIN-Routine **210** benutzt werden soll, und der Stapelzeiger zeigt auf eine höhere Adresse **232** der Zone **222**.

[0126] Eine SUB1-Routine **212** in Fig. 9B ist eine Subroutine, die von der MAIN-Routine **210** aufgerufen und ausgeführt wird. Die Zustände der Stapelzone im Speicher und des Stapelzeigers (im SP **14** gespeicherter Wert) während der Ausführung eines gegebenen Prozesses 'b' **213** der SUB1-Routine **212** sind in Fig. 9E gezeigt. Bezugszeichen **224** bezeichnet eine Stapelzone, die von der SUB1-Routine **212** zu benutzen ist, und der Stapelzeiger zeigt auf eine höhere Adresse **234** der Zone **224**.

[0127] Eine in Fig. 9C gezeigte SUB2-Routine **214** wird von der SUB1-Routine **212** aufgerufen und ausgeführt. Die Zustände der Stapelzone im Speicher und des Stapelzeigers (im SP **14** gespeicherter Wert) während der Ausführung eines gegebenen Prozesses 'c' **215** der SUB2-Routine **214** sind in Fig. 9F gezeigt. Bezugszeichen **226** bezeichnet eine Stapelzone, die von der SUB2-Routine **214** zu benutzen ist, und der Stapelzeiger zeigt auf eine höhere Adresse **236** der Zone **226**.

[0128] Wenn sich die Ausführung auf diese Weise über eine Vielzahl von Subroutinen erstreckt, bewegt sich die von jeder Routine benutzte Stapelzone, so daß der Wert des Stapelzeigers dementsprechend zur Oberseite der von jeder Routine benutzten Stapelzone bewegt wird.

[0129] Um es der CPU bei diesem Ausführungsbeispiel zu ermöglichen, diese Bewegung des Stapelzeigers wirksam mit einem kürzeren Objektcode auszuführen, ist der folgende Befehlssatz als Stapelzeigerbewegungsbefehle vorgesehen, bei denen es sich um einige der zweckgebundenen Stapelzeigerbefehle handelt:
add %sp, imm12 (9)
sub %sp, imm12 (10)

[0130] Die Befehle (9) und (10) sind Befehlscodes, die ein Assembler schafft. Befehl (9) ist ein Additionsbefehl für unmittelbare Daten an den im SP **14** gespeicherten Stapelzeiger, und Befehl (10) ist ein Subtraktionsbefehl für unmittelbare Daten an den selben Stapelzeiger. Bei dem Operanden imm12 handelt es sich um 32-Bit Daten, die durch Verschieben der 10-Bit unmittelbaren Daten um zwei Bits nach links und deren anschließendes Unterwerfen unter eine Nullerweiterung erhalten wurden. Benutzt wird dies für Operationen mit dem im SP **14** gespeicherten Stapelzeiger.

[0131] Ein Bitfeld **630** der Stapelzeigerbewegungsbefehle (9) und (10) ist in Fig. 10A gezeigt. Der Stapelzeigerbewegungsbefehl gemäß Fig. 10A hat 16 Bits eines Objektcodes mit OP-Code **632** (6 Bits), die anzeigen, ob die Bewegungsdaten für den im SP **14** gespeicherten Stapelzeiger eine Addition oder Subtraktion sind, und mit unmittelbaren Daten **634** (10 Bits). Dieser op-Code **632** weist einen gemeinsamen Code, der anzeigt, daß dies ein auf das SP **14** wirkender Stapelzeigerbewegungsbefehl ist, sowie verschiedene Codes auf, die entweder Addition oder Subtraktion spezifizieren. Da es infolgedessen aus dem op-Code klar hervorgeht, daß diese Befehle auf den im SP **14** gespeicherten Stapelzeiger einwirkten, ist es unnötig, den Stapelzeiger betreffende Daten in den Operanden des Objektcodes zu spezifizieren. Die unmittelbaren Versatzdaten **634** werden zum Schaffen eines Versatzes zum Subtrahieren vom Stapelzeiger oder zum Addieren zu demselben benutzt.

[0132] Als Vergleich zeigt Fig. 10B ein Beispiel eines Bitfeldes **640** eines Objektcodes eines add/subtract-Befehls, wenn ein allgemeines Register als Stapelzeiger benutzt wird (nachfolgend als Allzweckrechenbefehl bezeichnet).

[0133] Der Allzweckrechenbefehl gemäß Fig. 10B hat 20 Bits eines Objektcodes mit op-Code (6 Bits), die anzeigen, daß die Betriebsfunktion eine Addition oder Subtraktion unmittelbarer Daten zu oder von einem Wert in einem allgemeinen Register ist, unmittelbare Daten **644** (10 Bits) und eine Registernummer **646** (4 Bits), die das Register spezifiziert, auf das eingewirkt werden soll. Bei einem Allzweckmikrorechner liegt die Befehlslänge in Einheiten von 8 Bits vor, was 24 Bit oder 32 Bit Befehle ergibt.

[0134] Jede der Fig. 10A und Fig. 10B zeigt den Objektcode für einen Befehl, der benutzt wird, wenn unmittelbare Daten zum Stapelzeiger addiert oder von ihm subtrahiert werden. Diese Beispiele zeigen deutlich, daß ein Stapelzeigerbewegungsbefehl mit einem kürzeren Objektcode geschrieben werden kann als ein Allzweckrechenbefehl.

[0135] Es wird nunmehr die Konfiguration beschrieben, die zum Ausführen der obigen Befehle und der während dieser Ausführung passierenden Operationen nötig ist, wofür als Beispiel der Befehl (9), der dem SP unmittelbare Daten hinzufügt (nachfolgend Stapelzeigerbewegungsbefehl zur Addition genannt) und der Befehl (10) genommen wird, der unmittelbare Daten vom SP subtrahiert (nachfolgend Stapelzeigerbewegungsbefehl zur Subtraktion genannt).

[0136] Die zum Ausführen der Befehle erforderliche Hardwarekonfiguration wird zunächst unter Hinweis auf Fig. 1 beschrieben. Jeder dieser Befehle wird über den I_DATA_BUS **94** von einem externen Speicher (ROM) **52** übertragen und in den Befehlsdekodierer **20** eingegeben. Der Befehl wird vom Befehlsdekodierer **20** dekodiert, der verschiedene, in der Figur nicht gezeigte Signale ausgibt, die für die Ausführung des Befehls nötig sind. Der Generator unmittelbarer Daten **22** führt an den 10 Bits der unmittelbaren Daten **634** eine logische Verschiebung um zwei Bits nach links aus, unterzieht sie einer Nullexpansion und gibt sie an den PA_BUS **72** aus. Das SP **14** enthält den Stapelzeiger, und dieser Wert wird an den XA_BUS **78** ausgegeben. Der XA_BUS **78** ist mit dem PB_BUS **74** verbunden, der einen Eingang in das ALU **40** darstellt. Ein weiterer Eingang des ALU **40** ist mit dem PA_BUS **72** verbunden, der ein Ausgang des Generators unmittelbarer Daten **22** ist. Der Ausgang des ALU **40** ist mit dem WW_BUS **76** verbunden. Dieser WW_BUS **76** ist mit dem Eingang des SP verbunden.

[0137] Die Beschreibung behandelt zunächst den Betrieb während der Ausführung des Stapelzeigerbewegungsbefehls zur Addition.

[0138] Wenn der Stapelzeigerbewegungsbefehl zur Addition ausgeführt wird, wird der im SP **14** gespeicherte Wert des Stapelzeigers dem Versatz hinzugefügt, der vom Generator unmittelbarer Daten **22** auf der Basis der unmittelbaren Versatzdaten **634** geschaffen wurde, um einen neuen Stapelzeiger zu erzeugen. Dieser Wert wird im SP **14** gespeichert.

[0139] Fig. 11 zeigt ein Ablaufdiagramm der Operation während des Stapelzeigerbewegungsbefehls zur Addition.

[0140] Beim Start der Ausführung dieses Befehls wird der im SP **14** gespeicherte Stapelzeiger an den XA_BUS **78** ausgegeben (Schritt S250). Die Daten auf dem XA_BUS **78** werden an den PB_BUS **74** ausgegeben (Schritt S252). Vom Generator unmittelbarer Daten **22** anhand der unmittelbaren Daten geschaffene Bewegung unmittelbarer Daten imm wird an den PA_BUS **72** ausgegeben (Schritt S254). Das ALU **40** summiert den Wert auf dem PB_BUS **74** und den Wert auf dem PA_BUS **72** und gibt das Ergebnis an den WW_BUS **76** aus (Schritt S256). Dann wird der Wert auf dem WW_BUS **76** in das SP **14** eingegeben (Schritt S258). Als nächstes wird die Operation während der Ausführung des Stapelzeigerbewegungsbefehls zur Subtraktion beschrieben.

[0141] Wenn der Stapelzeigerbewegungsbefehl zur Subtraktion ausgeführt wird, wird das vom Generator unmittelbarer Daten **22** auf der Basis der unmittelbaren Daten **634** geschaffene 'unmittelbare Daten bewegen' vom Wert des Stapelzeigers subtrahiert, der im SP **14** gespeichert ist, um einen neuen Stapelzeiger zu schaffen. Dieser Wert wird im SP **14** gespeichert.

[0142] Fig. 12 zeigt ein Ablaufdiagramm der Operation während des Stapelzeigerbewegungsbefehls zur Subtraktion.

[0143] Beim Start der Ausführung dieses Befehls wird der im SP **14** gespeicherte Stapelzeiger an den XA_BUS **78** ausgegeben (Schritt S260). Dann werden die Daten auf dem XA_BUS **78** an den PB_BUS **74** ausgegeben (Schritt S262). Von dem Generator unmittelbarer Daten **22** auf der Grundlage der unmittelbaren Daten erzeugtes 'bewege unmittelbare Daten imm' wird an PA_BUS **72** ausgegeben (Schritt S264). Das ALU **40** subtrahiert den Wert auf dem PA_BUS **72** von dem Wert auf dem PB_BUS **74** und gibt das Ergebnis an den WW_BUS **76** aus (Schritt S266). Dann wird der Wert auf dem WW_BUS **76** in das SP **14** eingegeben (Schritt S268).

(7) Verzweigungsbefehl

[0144] Eine Ansicht, die die Steuerung der Programmausführung während der call- und ret-Befehle veranschaulicht, ist in Fig. 13 gezeigt. Während einer in Fig. 13 dargestellten MAIN-Routine **300** wird ein call-Befehl (**302**) ausgeführt, um eine Abzweigung zu einer Subroutine SUB **310** vorzunehmen, so daß die Steuerung auf die Subroutine SUB **310** übergeht. Ein ret-Befehl (**312**) ist am Ende der Subroutine angeordnet. Wenn dieser Befehl ausgeführt wird, kehrt der Fluß zum nächsten, dem Befehl (**304**) nach dem call-Befehl (**302**) der MAIN-Routine **300** zurück. Das Programm wird also in der Folge <1>, <2> und dann <3> in Fig. 13 ausgeführt. Am Ende der Ausführung dieser Subroutine SUB **310** kehrt die Steuerung zur MAIN-Routine **300** zurück, und die Ausführung soll vom nächsten Befehl (**304**) nach dem call-Befehl (**302**) fortgesetzt werden. Das macht es nötig, die Adresse des Rückkehrziels irgendwo aufzuzeichnen, wenn eine Abzweigung zur Subroutine SUB **310** passiert. Aus diesem Grund wird die Rückadresse zum Stapel gesichert, wie in Fig. 4 gezeigt, während ein Verzweigungsbefehl ausgeführt wird, der zu einer Subroutine abzweigt, beispielsweise der call-Befehl. Dann wird die Rückadresse vom Stapel zum Programmzähler zurückgegeben, während ein Verzweigungsbefehl ausgeführt wird, der von einer Subroutine zurückkehrt, beispielsweise dem ret-Befehl (dieser Prozeß wird nachfolgend als Sichern und Wiederherstellen des Programmzählers bezeichnet).

[0145] In einer bekannten RISC-CPU wird der Prozeß des Sicherns und Wiederherstellens des Programmzählers mittels Software erledigt, so daß ein Objektcode vorgesehen werden muß (Assembler-Befehle), um diesen Prozeß während der Ausführung eines Verzweigungsbefehls, beispielsweise des call-Befehls auszuführen. So ist es beispielsweise nötig, einen Objektcode (Assembler-Befehle) vorzusehen, um den Stapelzähler um ein Maß zu dekrementieren, welches der Wortgröße (4) während der Ausführung des call-Befehls gleichwertig ist, um die Adresse des nächsten Befehls nach dem call-Befehl im Stapel auf die Basis des Wertes im Programmzähler zu bringen.

[0146] Die CPU gemäß diesem Ausführungsbeispiel hat allerdings eine Hardwarekonfiguration, die das Sichern und Wiederherstellen des Programmzählers ermöglicht, wenn der call- oder ret-Befehl ausgeführt wird. Deshalb ist es nicht nötig, einen Objektcode (Assembler-Befehle) vorzusehen, mit dem der Programmzähler getrennt vom Objektcode des call- und ret-Befehls gesichert und wiederhergestellt wird.

[0147] Damit die CPU gemäß diesem Ausführungsbeispiel diesen Prozeß des Sicherns und Wiederherstellens des Programmzählers mit einem einzigen Befehl ausführen kann, wird der folgende Befehlssatz als Verzweigungsbefehle vorgesehen, bei denen es sich um einige der zweckgebundenen Stapelzeigerbefehle handelt:
call sign9 (11)
call %Rb (12)
ret (13)
reti (14)
retd (15)
int imm2 (16)
brk (17)

[0148] Die Befehle (11) bis (17) sind von einem Assembler geschaffene Befehlscodes. Der Befehl (11) ist ein Aufrufbefehl für eine PC-betreffende Subroutine, der den Programmzähler (PC) als Grundadresse benutzt, um auf eine relative Weise zu einer Zweigzieladresse auf der Basis der als Operand spezifizierten Verschiebedaten sign9 abzuzweigen. Der Befehl (12) ist ein Aufrufbefehl für eine indirekte Registersubroutine, der zur Zweigzieladresse abzweigt, die innerhalb des als Operand spezifizierten Registers enthalten ist. Der Befehl (13) ist ein Befehl zur Rückkehr von einer Subroutine. Der Befehl (14) ist ein Befehl zur Rückkehr von einer Interrupt- oder Ausnahmeverarbeitungsroutine. Der Befehl (15) ist ein Befehl zur Rückkehr von einer Fehlersuchverarbeitungsroutine. Der Befehl (16) ist ein Software-Interrupt-Befehl. Der Befehl (17) ist ein Software/Fehlersuch-Interrupt-Befehl.

[0149] Ein Beispiel eines Bitfeldes **650** des Aufrufbefehls (11) für die PC-betreffende Subroutine ist in Fig. 14 gezeigt. Dieser in Fig. 14 gezeigte Aufrufbefehl hat 16 Bits eines Objektcodes mit einem op-Code **652** (8 Bits), der anzeigt, daß die Betriebsfunktion ein Aufrufbefehl ist, der zu einer Subroutine mit einer Zweigzieladresse abzweigt, die relativ spezifiziert ist, wozu der Programmzähler als Basisadresse benutzt ist, und durch unmittelbare Daten spezifizierte Verschiebedaten sign9 (8 Bits) **654**. Während der Ausführung dieses Befehls werden die 8-Bit unmittelbaren Daten um ein Bit logisch nach links verschoben und dann einer Vorzeichenexpansion unterzogen.

[0150] Die CPU gemäß diesem Ausführungsbeispiel kann den Programmzähler für den Stapel bei der Ausführung des Aufrufbefehls allein mit dem in Fig. 14 gezeigten Objektcode sichern.

[0151] Als nächstes wird die Konfiguration beschrieben, die zum Ausführen der obigen Befehle erforderlich ist, sowie die Operationen, die während dieser Ausführung geschehen, und dazu werden als Beispiele der Aufrufbefehl (11) für die PC-betreffende Subroutine als ein Befehl zum Abzweigen zu einer Subroutine und der Rückkehrbefehl (13) als ein Befehl zum Zurückkehren von der Subroutine herangezogen.

[0152] Zuerst soll die zum Ausführen der Befehle erforderliche Hardwarekonfiguration unter Hinweis auf Fig. 1 beschrieben werden. Jeder dieser Befehle wird über den I_DATA_BUS **94** vom externen Speicher (ROM) **52** übertragen und in den Befehlsdekodierer **20** eingegeben. Der Befehl wird vom Befehlsdekodierer **20** dekodiert, der verschiedene, in der Figur nicht gezeigte Signale ausgibt, die zum Ausführen des Befehls benötigt werden. Der Generator unmittelbarer Daten **22** führt eine logische Verschiebung um ein Bit nach links an den Verschiebungsdaten **654** durch, unterzieht sie einer Vorzeichenexpansion, um eine 32-Bit Datenverschiebung imm zu schaffen, und gibt dann das Ergebnis an den PB_BUS **74** aus. Das SP **14** enthält den Stapelzeiger, und dieser Wert wird an den XA_BUS **78** ausgegeben, der mit einem Eingang des Adressenaddierers **30** verbunden ist. Ein weiterer Eingang des Adressenaddierers **30** ist mit dem PB_BUS **74** verbunden, der einen Ausgang des Generators für unmittelbare Daten **22** darstellt. Ein Ausgang (ADDR) des Adressenaddierers **30** ist mit dem externen I_ADDR_BUS **92** über die IA-Signalleitung **82** verbunden.

[0153] Der I_ADDR_BUS **92** und der I_DATA_BUS **94** ist mit dem ROM **52** verbunden, in dem der Befehlsobjektcode gespeichert ist. Die Bussteuereinheit (BCU) **60** gibt READ-Steuersignale aus, die diesen Befehlsobjektcode aus dem Speicher (ROM) **52** gemäß verschiedener Aufforderungssignale lesen, die von der CPU ausgegeben werden (zum Beispiel Signale, die an externe Busse ausgegeben werden).

[0154] Die folgende Beschreibung betrifft zunächst den Betrieb während der Ausführung eines Aufrufbefehls einer PC-betreffenden Subroutine.

[0155] Wenn der Aufrufbefehl der PC-betreffenden Subroutine ausgeführt wird, wird der im PC **12** gespeicherte Wert des Programmzählers zum Stapel gesichert, wie unter Hinweis auf Fig. 4 beschrieben, und der im SP **14** gespeicherte Wert des Stapelzeigers wird um die Wortgröße (4) dekrementiert. Im PC **12** wird eine Zweigzieladresse gesetzt, die durch Addieren des Programmzählers und der 32-Bit Verschiebung unmittelbarer Daten erhalten wurde.

[0156] Fig. 15 zeigt ein Ablaufdiagramm der Operation während des Aufrufbefehls der PC-betreffenden Subroutine.

[0157] Beim Start der Ausführung dieses Befehls wird der im SP **14** gespeicherte Stapelzeiger an den XA_BUS **78** ausgegeben (Schritt S270). Dies bildet eine Eingabe in den Adressenaddierer, und konstante Daten (–4), die vom Generator unmittelbarer Daten **22** geschaffen werden, bilden die andere Eingabe in den Adressenaddierer **30**. Die Konstanten –4 werden dem Wert des Stapelzeigers auf dem XA_BUS **78** vom Adressenaddierer **30** hinzugefügt, um eine Schreibadresse an den Stapel zum Speichern der Rückadresse zu schaffen, und das Ergebnis wird an den WW_BUS **76** ausgegeben. Diese Schreibadresse wird an den D_ADDR_BUS **96** ausgegeben (Schritt S272). Der Wert des im PC **12** gespeicherten Programmzählers wird vom PC-Inkrementer **44** um 2 inkrementiert, um die Rückadresse zu schaffen, und das Ergebnis wird über die DOUT-Signalleitung (**88**) an den D_DATA_BUS **98** ausgegeben (Schritt S274). Ein Aufforderungssignal zum Datenschreiben von der CPU an die BCU **60** wird aktiv und führt einen externen Speicherschreibzyklus aus (Schritt S276). Mit anderen Worten, die BCU **60** befolgt das Aufforderungssignal und speichert diese Rückadresse in dem im Speicher vorgesehenen Stapel unter Verwendung dieser Schreibadresse als Speicheradresse.

[0158] Der Wert auf dem WW_BUS **76** wird dann an das SP **14** ausgegeben (Schritt S278). Anders ausgedrückt, der Wert des Stapelzeigers wird aktualisiert, indem er um 4 dekrementiert wird.

[0159] Dann wird der im PC **12** gespeicherte Programmzähler an den XA_BUS **78** ausgegeben (Schritt S280). Der Generator unmittelbarer Daten **22** führt eine logische Verschiebung um ein Bit nach links an den Verschiebungsdaten **654** aus, unterzieht diese einer Vorzeichenexpansion, um eine 32-Bit Verschiebung unmittelbarer Daten imm zu schaffen und gibt das Ergebnis dann an den PB_BUS **74** aus. Der Adressenaddierer **30** summiert den Programmzähler auf dem XA_BUS **78** und die Verschiebung unmittelbarer Daten imm auf dem PB_BUS **74**, um eine Zweigadresse (ADDR) zu schaffen, gibt diese Adresse dann über die IA-Signallei-

tung **82** an den I_ADDR_BUS **92** aus (Schritt S282). Ein Aufforderungssignal zum Befehlelesen von der CPU an die BCU **60** wird aktiv und führt einen Lesezyklus hinsichtlich des externen Speichers (ROM) **52** aus, um den Objektcode des Abzweigezielbefehls abzurufen (Schritt S284).

[0160] Anschließend wird der Betrieb während der Ausführung des ret-Befehls beschrieben.

[0161] Wenn der ret-Befehl ausgeführt wird, wird der Wert des Programmzählers, der zum Stapel gesichert wurde, zum PC **12** umgespeichert, und der im SP **14** gespeicherte Wert des Stapelzeigers wird um die Wortgröße (4) inkrementiert, wie unter Hinweis auf Fig. 4 beschrieben.

[0162] Fig. 16 zeigt ein Ablaufdiagramm der Operation während des ret-Befehls.

[0163] Ehe dieser Befehl ausgeführt wird, zeigt der im SP **14** gespeicherte Stapelzeiger die Adresse im Stapel an, welche die Zieladresse zur Rückkehr von der aufgerufenen Routine enthält.

[0164] Beim Start der Ausführung dieses Befehls wird der im SP **14** gespeicherte Stapelzeiger an den XA_BUS **78** ausgegeben (Schritt S290). Der Stapelzeiger auf dem XA_BUS **78** wird dann an den D_ADDR_BUS **96** ausgegeben (Schritt S292). Ein Aufforderungssignal zum Datenlesen von der CPU an die BCU **60** wird aktiv und führt einen externen Speicherlesezyklus aus. Mit anderen Worten, die BCU **60** folgt dem Aufforderungssignal, die im Speicher vorgesehene Rückadresse vom Stapel zu lesen, wozu der Stapelzeiger als Speicheradresse benutzt wird. Diese Rückadresse, die aus dem Speicher (RAM) **50** gelesen wurde, wird innerhalb der CPU vom D_DATA_BUS **94** über die DIN-Signalleitung **86** abgerufen, dann von der DIN-Signalleitung **86** über die IA-Signalleitung **82** an den I-ADDR_BUS **92** ausgegeben (Schritt S294). Ein Aufforderungssignal zum Befehlelesen von der CPU an die BCU **60** wird aktiv und führt einen Lesezyklus hinsichtlich des externen Speichers (ROM) **52** aus, um den Objektcode des Rückkehrzielbefehls abzurufen (Schritt S296).

[0165] Der Wert des Stapelzeigers auf dem XA_BUS **78** ist eine Eingabe des Adressenaddierers **30**, und vom Generator unmittelbarer Daten **22** erzeugte konstante Daten (+4) bilden die andere Eingabe des Adressenaddierers **30**. Die konstanten +4 werden dem Wert des Stapelzeigers auf dem XA_BUS **78** vom Adressenaddierer **30** hinzugefügt, um die Adresse des oberen Bereichs der Stapelzone zu schaffen, die von der Rückkehrzielroutine sichergestellt ist, und das Ergebnis wird an den WW_BUS **76** ausgegeben (Schritt S298). Die Adresse auf dem WW_BUS **76** (die Adresse des oberen Bereichs der Stapelzone, die von der Rückkehrzielroutine sichergestellt ist) wird dann an das SP **14** ausgegeben (Schritt S300).

(8) Beschreibung eines sequentiellen Push-Befehls (pushn) und eines sequentiellen Pop-Befehls (popn)

[0166] Wie schon gesagt, ist es vor kurzem üblich geworden, eine Konfiguration mit einer großen Anzahl interner allgemeiner Register zu haben, insbesondere in RISC-Zentraleinheiten, um ein rasches Ablaufen von Prozessen innerhalb der Zentraleinheit selbst ohne Zugriff auf den Speicher zu ermöglichen. Die CPU des vorliegenden Ausführungsbeispiels beinhaltet 16 Mehrzweck-Register, die so ausgelegt sind, daß sie die Geschwindigkeit der Verarbeitung erhöhen. Wenn es jedoch so viele interne Register gibt, müssen die Inhalte einer großen Anzahl von Registern während einer Verarbeitung gesichert werden, mit der Register gesichert und wiederhergestellt werden, wenn es zu einem Interrupt kommt oder eine Subroutine aufgerufen wird.

[0167] Zum Sichern oder Wiederherstellen des Inhalts dieser Register wird im Stand der Technik ein Push-Befehl zum Speichern des von einer Adresse spezifizierten Inhalts oder ein Pop-Befehl zum Abrufen des Inhalts des Stapels an ein Register benutzt. Die Aktionen dieser gewöhnlichen Push- und Pop-Befehle werden nachfolgend beschrieben.

[0168] Die während der Ausführung von Push-Befehlen durchgeführten Aktionen sind schematisch in den Fig. 17A und Fig. 17B gezeigt, während die Aktionen während der Ausführung von Pop-Befehlen schematisch in den Fig. 18A und Fig. 18B gezeigt sind. Unter Hinweis auf die Fig. 17A, Fig. 17B, Fig. 18A und Fig. 18B sollen nun die Aktionen beschrieben werden, die durchgeführt werden, wenn Daten zwischen einer Vielzahl von allgemeinen Registern und dem Stapel übertragen werden.

[0169] Fig. 17A zeigt die Aktionen, wenn ein 'push R1'-Befehl ausgeführt wird, um den Inhalt eines allgemeinen Registers R1 in den Stapel zu schreiben. Während der Ausführung dieses Befehls wird der Inhalt des SP **14** dadurch aktualisiert, daß vom aktuellen Wert 4 subtrahiert wird (**1000** in Fig. 17A wird zu 996 aktualisiert). Der Inhalt 'a' des allgemeinen Registers R1 wird unter **996** eingespeichert, nämlich der vom aktualisierten Stapelzeiger im SP **14** angezeigten Speicheradresse.

[0170] Fig. 17B zeigt die Aktionen, wenn ein 'push R2'-Befehl ausgeführt wird, um ferner den Inhalt eines allgemeinen Registers R2 in den Stapel zu schreiben. Während der Ausführung dieses Befehls wird der Inhalt des SP **14** dadurch aktualisiert, daß vom aktuellen Wert 4 subtrahiert wird (**996** in Fig. 17B wird zu 992 aktualisiert). Der Inhalt 'b' des allgemeinen Registers R2 wird unter 992 eingespeichert, nämlich der vom aktualisierten Stapelzeiger im SP **14** angezeigten Speicheradresse.

[0171] Fig. 18A zeigt die Aktionen, wenn ein 'pop R2'-Befehl ausgeführt wird, um den Inhalt des Stapels des allgemeinen Registers R2 abzurufen. Während der Ausführung dieses Befehls wird der Inhalt 'b', der an der vom Stapelzeiger im SP **14** angezeigten Speicheradresse 992 gespeichert ist, abgerufen und im allgemeinen Register R2 gespeichert. Der Inhalt des SP **14** wird durch Addition von 4 zum aktuellen Wert aktualisiert (der in Fig. 18A gezeigte Wert 992 von vor der Ausführung wird zu 996 aktualisiert).

[0172] Schließlich zeigt die Fig. 18B die Aktionen, wenn ein 'pop R1' ausgeführt wird, um auch den Inhalt des Stapels zum allgemeinen Register R1 abzurufen. Während der Ausführung dieses Befehls wird der Inhalt 'a', der an der Speicheradresse 996 gespeichert ist, die vom Stapelzeiger im SP **14** angezeigt ist, abgerufen und im allgemeinen Register R1 gespeichert. Der Inhalt des SP **14** wird durch Addition von 4 zum aktuellen Wert aktualisiert (der in Fig. 18B gezeigte Wert 996 von vor der Ausführung wird zu 1000 aktualisiert).

[0173] Wenn Daten zwischen einer Vielzahl allgemeiner Register und dem Stapel bei diesem Beispiel aus dem Stand der Technik übertragen werden, ist es nötig, Push- oder Pop-Befehle mehrmals auszuführen. Das liegt daran, daß jede Ausführung eines Push- oder Pop-Befehls nur auf ein einziges Register einwirken kann.

[0174] Bei einer Verarbeitung zum Sichern von Daten aus einer Anzahl von Registern zum Stapel oder der Wiederherstellung von Daten aus dem Stapel zu einer Anzahl von Registern führt aus diesem Grund die entstehende Zunahme der Anzahl von Befehlen zu einer Zunahme der Größe des Objektcodes. Die Anzahl der Ausführungsschritte im Programm nimmt gleichfalls zu, was Verzögerungen der Ausführungszeit oder der Verarbeitung des Programms zur Folge hat.

[0175] Für die CPU des vorliegenden Ausführungsbeispiels ist folgender Befehlssatz vorgesehen:
pushn %Rs (18)
popn %Rd (19)

[0176] Befehl (18) ist ein von einem Assembler geschaffener sequentieller Push-Befehl. Mit ihm wird der Inhalt von n allgemeinen Registern (n = eine ganze Zahl von 1 bis 16) von %Rs bis R0 der Reihe nach auf den Stapel draufgelegt. Befehl (19) ist ein von einem Assembler geschaffener sequentieller Pop-Befehl. Mit ihm werden n Posten Wortdaten (n = eine ganze Zahl von 1 bis 16) der Reihe nach vom Stapel abgehoben und auf Mehrzweck-Register %Rd bis R0 draufgelegt. Jeder der pushn- und popn-Befehle besteht aus einem op-Code und einem Operanden. Der Operand %Rs des pushn-Befehls zeigt die Registerzahl von Rs an, wenn Daten aus den Registern %Rs bis R0 in den Stapel eingespeist werden sollen. Der Operand %Rd des popn-Befehls zeigt die Registernummer von Rd an, wenn Daten aus dem Stapel entnommen und in die Register R0 bis %Rd eingespeist werden sollen.

[0177] Die Bitstruktur der pushn- und popn-Befehle ist in Fig. 19 gezeigt. Das unterste 4-Bit Feld enthält einen Code, der %Rs oder %Rd anzeigt; jedes beliebige der 16 allgemeinen Register kann spezifiziert werden. Wenn Daten zwischen einem Spezialregister und dem Stapel transferiert werden sollen, werden sie über ein allgemeines Register geschickt.

[0178] **Fig.** 10 ist ein Blockschaltbild der Hardwarekonfiguration, die zum Ausführen des sequentiellen Push-Befehls (pushn) oder des sequentiellen Pop-Befehls (popn) erforderlich ist. Für die Beschreibung des sequentiellen Push-Befehls (pushn) und des sequentiellen Pop-Befehls (popn) nötige Teile werden aus Fig. 1 entnommen; und weiter benötigte Teile werden hinzugefügt. Teile, die denen in Fig. 1 gleichen, sind mit den gleichen Bezugsziffern versehen. In dieser Figur sind 16 Mehrzweck-Register, die R0 bis R15 genannt sind, mit Bezugszeichen **11** versehen. Diese allgemeinen Register **11** übertragen Daten zu und von dem D_DATA_BUS **98**. Ein Registerauswahladressensignal **54** zum Auswählen eines Registers kommt von einem Steuerschaltungsblock **45**. Der Stapelzeiger ist im SP **14** gespeichert. Der Wert des SP **14** kann an den D_ADDR_BUS **96** und den internen Adressenbus (XA_BUS) **78** ausgegegeben werden, der mit einem Eingang des 32-Bit Adressenaddierers **30** verbunden ist. Ein weiterer Eingang des Adressenaddierers **30** ist mit einem Versatzsignal **24** verbunden, welches vom Steuerschaltungsblock **45** ausgegeben wird. Eine Ausgabe des Adressenaddierers **30** wird von einem Adressenspeicher (Add_LT) **32** gehalten und dann weiter ausgegeben an den XA_BUS **78** oder den WW_BUS **76**. Der WW_BUS **76** ist mit dem Eingang des SP **14** verbunden.

Im Steuerschaltungsblock **45** gibt es einen 4-Bitzähler (countx) **46**, um die Anzahl der an der Datenübertragung beteiligten Register zu zählen. Ferner ist der Steuerschaltungsblock **45** mit einem in der Figur nicht gezeigten Befehlsregister versehen, welches den Operanden %Rs oder %Rd des pushn- oder popn-Befehls hält und auch Steuersignale entsprechend Befehlen vom Befehlsdekodierer ausgibt. In dieser Figur bezeichnet Bezugszeichen **60** eine Bussteuereinheit (BCU), die die Eingabe und Ausgabe von Daten in und aus einem externen Speicher (RAM) **50** steuert, der den Stapelbereich umfaßt und READ- und WRITE-Steuersignale ausgibt.

[0179] Zunächst soll jetzt die Operation während der Ausführung eines sequentiellen Push-Befehls (pushn) beschrieben werden.

[0180] Wenn der sequentielle Push-Befehl (pushn) ausgeführt wird, wird der in den allgemeinen Registern gespeicherte Inhalt der Reihe nach auf den Stapel draufgelegt, und zwar beginnend mit dem Register mit der Registernummer %Rs zum allgemeinen Register R0, wie schon unter Hinweis auf Fig. 4 beschrieben.

[0181] Fig. 21 zeigt ein Ablaufdiagramm der Operation während des pushn-Befehls.

[0182] Beim Start der Ausführung des pushn-Befehls ist der Versatz eines Versatzsignals **24** –4. Der Zähler (countx) **46** wird auf Null gelöscht und der im Speicher SP **14** gespeicherte Stapelzeiger an den XA_BUS **78** ausgegeben (Schritt S100).

[0183] Der Adressenaddierer **30** addiert dann –4 zu dem Wert auf dem XA_BUS **78** und gibt das Ergebnis in den Adressenspeicher (Add_LT) **32** ein (Schritt S101).

[0184] Der Wert im Adressenspeicher (Add_LT) **32** wird dann an den D_ADDR_BUS **96** ausgegeben. Der Unterschied zwischen dem Wert von %Rs, der in den niedrigsten 4 Bits des Befehlsregisters gehalten ist, und dem Wert im Zähler (countx) **46** wird vom Steuerschaltungsblock **45** berechnet und an das Registerauswahladressensignal **54** ausgegeben. Das von diesem Signal **54** ausgewählte Register wird mit dem D_DATA_BUS **98** verknüpft und ein Schreibvorgang in Bezug auf den externen Speicher (RAM) **50** durchgeführt (Schritt S102).

[0185] In einem Schritt S103 wird der Zähler (countx) **46** mit %Rs verglichen. Wenn die Werte zusammenpassen, ist das Einschreiben der Register in den externen Speicher vollendet worden, und die Ausführung des pushn-Befehls endet damit, daß der Wert im Adressenspeicher (Add_LT) **32** über den WW_BUS **76** in das SP **94** geschrieben wird (Schritt S104).

[0186] Wenn die Werte nicht zusammenpassen, wird der Zähler (countx) um 1 inkrementiert, der Wert im Adressenspeicher (Add_LT) **32** an den XA_BUS **78** ausgegeben und die Verarbeitung ab dem Schritt S101 wiederholt (Schritt S105).

[0187] Als nächstes wird die Operation während der Ausführung eines sequentiellen Pop-Befehls (popn) beschrieben.

[0188] Wenn ein sequentieller Pop-Befehl (popn) ausgeführt wird, wird der Inhalt des Stapels der Reihe nach aus dem allgemeinen Register R0 auf das Mehrzweck-Register mit der Registernummer %Rd draufgelegt, wie schon im Zusammenhang mit Fig. 4 beschrieben.

[0189] Fig. 22 ist ein Ablaufdiagramm, welches den Betrieb während des popn-Befehls zeigt.

[0190] Beim Start der Ausführung des popn-Befehls ist der Versatz eines Versatzsignals **24** –4. Der Zähler (countx) **46** wird auf Null gelöscht und der im SP **14** gespeicherte Stapelzeiger wird an den XA_BUS **78** und den D_ADDR_BUS **96** ausgegeben (Schritt S110).

[0191] Der Adressenaddierer **30** addiert dann 4 zum Wert im XA_BUS **78** und gibt das Ergebnis in den Adressenspeicher (Add_LT) **32** ein (Schritt S111).

[0192] Dann wird ein externer Speicherlesezyklus durchgeführt. Die so gelesenen Daten werden über den D_DATA_BUS **98** in die allgemeinen Register **11** geschrieben. Währenddessen wird der Zähler (countx) **46** vom Steuerschaltungsblock **45** an das Registerauswahladressensignal **54** ausgegeben (Schritt S112).

[0193] Im einem Schritt S113 wird der Zähler (countx) **46** mit %Rs verglichen. Wenn die Werte zusammenpassen, ist das Lesen aus dem externen Speicher in die Register vollendet, und die Ausführung des popn-Befehls endet mit dem Schreiben des Wertes im Adressenspeicher (Add_LT) **32** über den WW_BUS **76** in das SP **14** (Schritt S114).

[0194] Wenn die Werte nicht zueinanderpassen, wird der Zähler (countx) um 1 inkrementiert, der Wert im Adressenspeicher (Add_LT) **32** an den XA_BUS **78** und den D_ADDR_BUS **96** ausgegeben und die Verarbeitung ab dem Schritt S111 wiederholt (Schritt S115).

[0195] So ermöglicht es der 'pushn %Rs'-Befehl, Daten aus den Registern %Rs bis R0 in den Stapel zu schreiben, und der 'popn %Rd'-Befehl ermöglicht es, die nötige Anzahl Datenposten aus dem Stapel zu den Registern R0 bis %Rd zurückzuführen. So können beispielsweise Daten aus den Registern R3 bis R0 durch die Ausführung von 'pushn %Rs' draufgelegt werden.

[0196] Eine weitere Erhöhung des Wirkungsgrads wird in diesem Fall dadurch erzielt, daß die Art und Weise, in der Register benutzt werden, eingeschränkt wird. Anders ausgedrückt, das Sichern und die Wiederherstellung von Registern ist besonders nötig, wenn ein Programm zu einer anderen Routine abzweigt, beispielsweise während einer Interrupt-Verarbeitung oder bei Aufruf einer Subroutine. Jede in einem solchen Fall aufgerufene Routine, beispielsweise eine weitere Interrupt-Routine oder Subroutine sollte vorzugsweise die Register in Folge ab R0 benutzen. Damit wird sichergestellt, daß der Inhalt der Register wirksam gesichert oder wiederhergestellt werden kann, wenn der pushn-Befehl oder der popn-Befehl angewandt wird, ohne daß die Register von R0 bis Rd oder Rs eingeschlossen werden müssen, welche Daten enthalten, die nicht gesichert zu werden brauchen. Alle Register in diesem Ausführungsbeispiel haben die gleichen Funktionen, und es gibt weiter keine Einschränkungen der Benutzung der Register und des Befehls, so daß diese Einschränkung das obige Problem löst, ohne zu weiteren Problemen Anlaß zu geben.

[0197] Die Anwendung des pushn-Befehls und des popn-Befehls bei diesem Ausführungsbeispiel ermöglicht also eine wirkungsvolle Verwirklichung des Sicherns von Daten aus Registern zum Stapel in einem Speicher und des Umspeicherns von Daten aus dem Stapel zu Registern mittels eines einzigen Befehls, eines pushn- oder popn-Befehls. Das minimiert nicht nur die Objektcodegröße, sondern auch die Anzahl der Schritte zur Programmausführung und bedeutet, daß die Anzahl der Ausführungszyklen auf einen einzigen Befehlsabruf und die nötige Anzahl Datenübertragungen verringert werden kann, womit eine Minimierung der Anzahl Zyklen möglich wird. Hierdurch kann die Geschwindigkeit der Interrupt-Verarbeitung von Routinen und Subroutinen erhöht werden.

[0198] Da die für diese Ausführungen benötigten strukturellen Elemente eine Zähleinrichtung und eine einfache Folgesteuerung sind, kann eine Datenverarbeitungsschaltung verwirklicht werden, die eine kleine Anzahl Gates umfaßt, was die Erfindung für einen Einchip-Mikrorechner anwendbar macht.

[0199] Die obige Beschreibung betrifft die Konfiguration, die zum Ausführen des sequentiellen Auflegebefehls (pushn) und des sequentiellen Pop-Befehls (popn) benötigt wird, wenn ein zweckgebundenes Stapelzeigerregister als SP **14** verwendet wird. Es sollte aber für den Fachmann offensichtlich sein, daß die vorliegende Erfindung auch bei einem Aufbau anwendbar ist, bei dem ein beliebiges allgemeines Register als Stapelzeiger verwendet ist.

Ausführungsbeispiel 2

[0200] Ein Blockschaltbild der Hardware eines Mikrorechners gemäß diesem Ausführungsbeispiel ist in Fig. 23 gezeigt.

[0201] Dieser Mikrorechner **2** ist ein 32-Bit Mikrocontroller, der folgendes aufweist: eine CPU **10**; ROM **52**; RAM **50**, eine hochfrequente Oszillatorschaltung **910**; eine niederfrequente Oszillatorschaltung **920**; eine Rückstellschaltung **930**; einen Prescaler **940**; einen 16-Bit programmierbaren Zeitgeber **950**; einen 8-Bit programmierbaren Zeitgeber **960**; einen Taktzeitgeber **970**; einen intelligenten direkten Speicherzugriff DMA **980**; einen Hochgeschwindigkeits-DMA **990**; einen Interrupt-Controller **800**; eine serielle Schnittstelle **810**; eine Bussteuereinheit BCU **60**; einen A/D-Umsetzer **830**; einen D/A-Umsetzer **840**; einen Eingangsanschluß **850**; einen Ausgangsanschluß **860**; und einen I/O-Anschluß **870**; ebenso wie verschiedene Busse **92**, **94**, **96** und **98**, die mit diesen Bauelementen verbunden sind, sowie verschiedene Stifte **890**.

[0202] Die CPU **10** hat ein SP, bei dem es sich um ein zweckgebundenes Stapelzeigerregister handelt; und

sie dekodiert und führt die zweckgebundenen Stapelzeigerbefehle aus, wie vorstehend beschrieben. Diese CPU **10** hat die gleiche Konfiguration wie das zuvor beschriebene Ausführungsbeispiel 1 und wirkt als eine Dekodiereinrichtung und Ausführungseinrichtung.

[0203] Der Mikrorechner gemäß diesem Ausführungsbeispiel kann deshalb mit kurzen Befehlen eine wirksame Kodierung ermöglichen und Prozesse ausführen, die auf den Stapelzeiger wirken.

[0204] Auch die Prozesse zum Sichern und Wiederherstellen von Registern können wirksam kodiert werden, so daß eine Interrupt-Verarbeitung und Aufruf/Rückkehr von Subroutinen rasch durchgeführt werden können.

[0205] Der Mikrorechner gemäß dieser Erfindung kann für periphere Geräte für PC's, beispielsweise Drucker oder elektronische Geräte, wie tragbare Geräte verwendet werden. Derartige Anwendungen der Erfindung machen es möglich, ein kostengünstiges und trotzdem hochentwickeltes elektronisches Gerät bereitzustellen. Erleichtert wird dies durch eine Datenverarbeitungsschaltung, die eine hohe Verarbeitungsgeschwindigkeit hat und eine höchst wirksame Nutzung des Speichers bietet.

[0206] Es sei darauf hingewiesen, daß die Erfindung nicht als auf die oben beschriebenen Ausführungsbeispiele beschränkt aufzufassen ist; sie kann auf verschiedene andere Art und Weise verkörpert werden.

Patentansprüche

1. Datenverarbeitungsschaltung, aufweisend:

ein zweckgebundenes Stapelzeigerregister, das nur für den Stapelzeiger (**14**) benutzt wird;
eine Dekodiereinrichtung (**20**) zum Dekodieren von Objektcode einer Gruppe zweckgebundener Stapelzeigerbefehle, die einen Objektcode haben, der das zweckgebundene Stapelzeigerregister als einen implizierten Operanden spezifiziert, und sich auf das Verarbeiten aufgrund des zweckgebundenen Stapelzeigerregisters beziehen, sowie zum Ausgeben eines Steuersignals aufgrund des Objektcodes,
eine Ausführungseinrichtung (**10**) zum Ausführen der Gruppe zweckgebundener Stapelzeigerbefehle aufgrund des Steuersignals und des Inhalts des zweckgebundenen Stapelzeigerregisters; und
eine Vielzahl von Registern (**11**), die in einer aneinandergrenzenden Folge vorgesehen sind, wobei die Gruppe zweckgebundener Stapelzeigerbefehle mindestens einen von zwei Befehlen, nämlich einen sequentiellen Push-Befehl und einen sequentiellen Pop-Befehl, aufweist, die Daten zum Spezifizieren einer Vielzahl von Registern in ihrem Objektcode haben;
wobei die Dekodiereinrichtung (**20**) mindestens einen der beiden Befehle, nämlich den sequentiellen Push-Befehl und den sequentiellen Pop-Befehl, dekodiert; und
die Ausführungseinrichtung (**10**), während der Ausführung mindestens eines der beiden Befehle, des sequentiellen Push-Befehls und des sequentiellen Pop-Befehls, mindestens einen von zwei Prozessen, nämlich einen Prozeß der Ausführung einer Vielzahl sequentieller Pushs von Daten aus der Vielzahl von Registern auf einen im Speicher vorgesehenen Stapel und einen Prozeß der Ausführung einer Vielzahl sequentieller Pops von Daten von dem Stapel an die Vielzahl von Registern, aufgrund des Inhalts einer von dem zweckgebundenen Stapelzeigerregister (**14**) spezifizierten Speicheradresse und der Daten zum Spezifizieren einer Vielzahl von Registern, durchführt;
dadurch gekennzeichnet, daß die Ausführungseinrichtung (**10**) folgendes aufweist:
eine Schreibeinrichtung zum Schreiben des Inhalts eines gegebenen Registers, welches eines der Vielzahl von Registern (**11**) ist, in einen im Speicher (**50B**) vorgesehenen Stapel aufgrund einer von dem zweckgebundenen Stapelzeigerregister (**14**) spezifizierten Speicheradresse;
eine Schreibfrequenz-Zähleinrichtung (**46**), die zählt, wie oft die Schreibeinrichtung geschrieben hat; und
eine Vergleichseinrichtung (**45**), die die von der Zähleinrichtung gezählte Schreibfrequenz mit dem Wert der Daten zum Spezifizieren einer Vielzahl von Registern vergleicht;
wobei die Schreibeinrichtung folgendes aufweist:
eine Schreibspeicheradressen-Generatoreinrichtung (**30**, **32**), die eine erste Eingabe und eine zweite Eingabe durch einen Addierer summiert und eine Schreibspeicheradresse generiert, um ein Schreibziel zu spezifizieren;
eine erste Eingabesteuereinrichtung (**45**), die eine derartige Steuerung bietet, daß die erste Eingabe des Addierers der Inhalt des zweckgebundenen Stapelzeigerregisters beim Start der Ausführung eines sequentiellen zweckgebundenen Befehls ist und eine Schreibadresse ist, die anschließend von einer Schreibadressengeneratoreinrichtung generiert wird;
eine zweite Eingabesteuereinrichtung (**45**), die an den zweiten Eingang des Addierers einen Versatz ausgibt, der beim Schreiben eines Wortes an den Stapel benutzt wird; und
eine Schreibeinrichtung (**45**), die in den Stapel den Inhalt eines durch Subtraktionsverarbeitung spezifizierten

Registers schreibt, für die die Daten zum Spezifizieren einer Vielzahl von Registern und die Schreibfrequenz aufgrund der Schreibspeicheradresse benutzt wird;
wodurch das Schreiben des Inhalts der Vielzahl von Registern in den Stapel und das Beenden des Schreibens aufgrund des Vergleichsergebnisses der Vergleichseinrichtung gesteuert werden.

2. Datenverarbeitungsschaltung, aufweisend:
ein zweckgebundenes Stapelzeigerregister, das nur für den Stapelzeiger (**14**) benutzt wird;
eine Dekodiereinrichtung (**20**) zum Dekodieren von Objektcode einer Gruppe zweckgebundener Stapelzeigerbefehle, die einen Objektcode haben, der das zweckgebundene Stapelzeigerregister als einen implizierten Operanden spezifiziert, und sich auf das Verarbeiten aufgrund des zweckgebundenen Stapelzeigerregisters beziehen, sowie zum Ausgeben eines Steuersignals aufgrund des Objektcodes,
eine Ausführungseinrichtung (**10**) zum Ausführen der Gruppe zweckgebundener Stapelzeigerbefehle aufgrund des Steuersignals und des Inhalts des zweckgebundenen Stapelzeigerregisters; und
eine Vielzahl von Registern (**11**), die in einer aneinandergrenzenden Folge vorgesehen sind,
wobei die Gruppe zweckgebundener Stapelzeigerbefehle mindestens einen von zwei Befehlen, nämlich einen sequentiellen Push-Befehl und einen sequentiellen Pop-Befehl, aufweist, die Daten zum spezifizieren einer Vielzahl von Registern in ihrem Objektcode haben;
wobei die Dekodiereinrichtung (**20**) mindestens einen der beiden Befehle, nämlich den sequentiellen Push-Befehl und den sequentiellen Pop-Befehl, dekodiert; und
die Ausführungseinrichtung (**10**), während der Ausführung mindestens eines der beiden Befehle, des sequentiellen Push-Befehls und des sequentiellen Pop-Befehls, mindestens einen von zwei Prozessen, nämlich einen Prozeß der Ausführung einer Vielzahl sequentieller Pushs von Daten aus der Vielzahl von Registern auf einen im Speicher vorgesehenen Stapel und einen Prozeß der Ausführung einer Vielzahl sequentieller Pops von Daten von dem Stapel an die Vielzahl von Registern, aufgrund des Inhalts einer von dem zweckgebundenen Stapelzeigerregister (**14**) spezifizierten Speicheradresse und der Daten zum Spezifizieren einer Vielzahl von Registern durchführt;
dadurch gekennzeichnet, daß die Ausführungseinrichtung (**10**) folgendes aufweist:
eine Leseeinrichtung zum Lesen des Inhalts eines im Speicher vorgesehenen Stapels aufgrund einer von dem zweckgebundenen Stapelzeigerregister spezifizierten Speicheradresse und zum Speichern des Inhalts in einem gegebenen Register der Vielzahl von Registern;
eine Lesefrequenz-Zähleinrichtung, die zählt, wie oft die Leseeinrichtung gelesen hat; und
eine Vergleichseinrichtung (**45**), die die von der Zähleinrichtung gezählte Lesefrequenz mit dem Wert der Daten zum Spezifizieren einer Vielzahl von Registern vergleicht;
wobei die Leseeinrichtung folgendes aufweist:
eine erste Eingabesteuereinrichtung (**45**), die eine derartige Steuerung bietet, daß die erste Eingabe eines Addierers der Inhalt des zweckgebundenen Stapelzeigerregisters beim Start der Ausführung eines sequentiellen zweckgebundenen Befehls ist und eine Lesespeicheradresse ist, die anschließend von einer Leseadressengeneratoreinrichtung generiert wird;
eine zweite Eingabesteuereinrichtung (**45**), die an den zweiten Eingabe des Addierers einen Versatz ausgibt, der beim Lesen eines Wortes von dem Stapel benutzt wird; und
eine Leseeinrichtung (**45**), die den Inhalt des Stapels aufgrund der Lesespeicheradresse liest und den Inhalt in einem Register speichert, das aufgrund der Lesefrequenz spezifiziert wurde,
wodurch das Lesen des Inhalts des Stapels und das Beenden des Lesens aufgrund des Vergleichsergebnisses der Vergleichseinrichtung gesteuert werden.

3. Datenverarbeitungsschaltung nach Anspruch 1 oder 2, ferner aufweisend:
n Mehrzweck-Register (**11**), die durch Registernummern 0 bis n – 1 spezifiziert sind;
wobei der Objektcode mindestens eines der beiden Befehle, des sequentiellen Push-Befehls und des sequentiellen Pop-Befehls, als Daten zum Spezifizieren einer Vielzahl von Registern eine letzte Registernummer umfaßt, auf die eine der Registernummern spezifiziert ist; und
die Ausführungseinrichtung (**10**) mindestens einen von zwei Prozessen, nämlich einen Prozeß der Ausführung einer Vielzahl sequentieller Pushs von Daten auf einen im Speicher vorgesehenen Stapel von einer Vielzahl von Registern, beginnend mit dem Register 0 bis zu einem von der letzten Registernummer spezifizieren Register, und einen Prozeß zur Ausführung einer Vielzahl sequentieller Pops von Daten aus dem Stapel zu einer Vielzahl von Registern, beginnend mit dem Register 0 bis zu einem von der letzten Registernummer spezifizierten Register, aufgrund des Inhalts einer Speicheradresse durchführt, die von dem zweckgebundenen Stapelzeigerregister spezifiziert ist.

4. Datenverarbeitungsschaltung nach einem der Ansprüche 1 bis 3, ferner aufweisend:
ein Programmzählerregister (**12**), welches nur für den Programmzähler verwendet wird;

wobei die Gruppe zweckgebundener Stapelzeigerbefehle Verzweigungsbefehle aufweist, die ein Befehl zum Abzweigen zu einer Subroutine und ein Rückkehrbefehl von der Subroutine sind;
die Dekodiereinrichtung (**20**) die Verzweigungsbefehle dekodiert; und
die Ausführungseinrichtung (**10**) folgendes aufweist:
eine Einrichtung zur Ausführung mindestens eines von zwei Prozessen, nämlich eines Prozesses der Sicherung des Inhalts des Programmzählerregisters (**12**) zu einem gegebenen zweiten Bereich des im Speicher vorgesehenen Stapels und eines Prozesses zum Wiederherstellung des Inhalts des zweiten Bereiches in dem Programmzählerregister (**12**) aufgrund einer von dem zweckgebundenen Stapelzeigerregister spezifizierten Speicheradresse während der Ausführung des Verzweigungsbefehls; und
eine Einrichtung zum Aktualisieren des Inhalts des zweckgebundenen Stapelzeigerregisters (**12**) aufgrund der Sicherung und Wiederherstellung.

5. Datenverarbeitungsschaltung nach einem der Ansprüche 1 bis 4, bei der die Schaltung Befehle eines reduzierten Befehlssatzrechners benutzt.

6. Datenverarbeitungsschaltung nach einem der Ansprüche 1 bis 5, bei der ein Befehl fester Länge dekodiert wird und die Ausführung auf dem Befehl fester Länge beruht.

7. Mikrorechner mit der Datenverarbeitungsschaltung nach einem der Ansprüche 1 bis 6, einer Speichereinrichtung und Eingabe- und Ausgabeeinrichtungen zur Eingabe von Daten von und zur Ausgabe von Daten an externe Vorrichtungen.

8. Mikrorechner nach Anspruch 7, bei dem ein Programm, das von ihm ausgeführt wird, eine Programmsprache benutzt, die eine Speicherzone für Auto-Variable durch Benutzung des Stapelzeigers sicherstellt.

9. Elektronisches Gerät, welches den in Anspruch 7 oder 8 beschriebenen Mikrorechner aufweist.

Es folgen 24 Blatt Zeichnungen

Anhängende Zeichnungen

FIG. 1

FIG. 1A	FIG. 1B

FIG. 1A

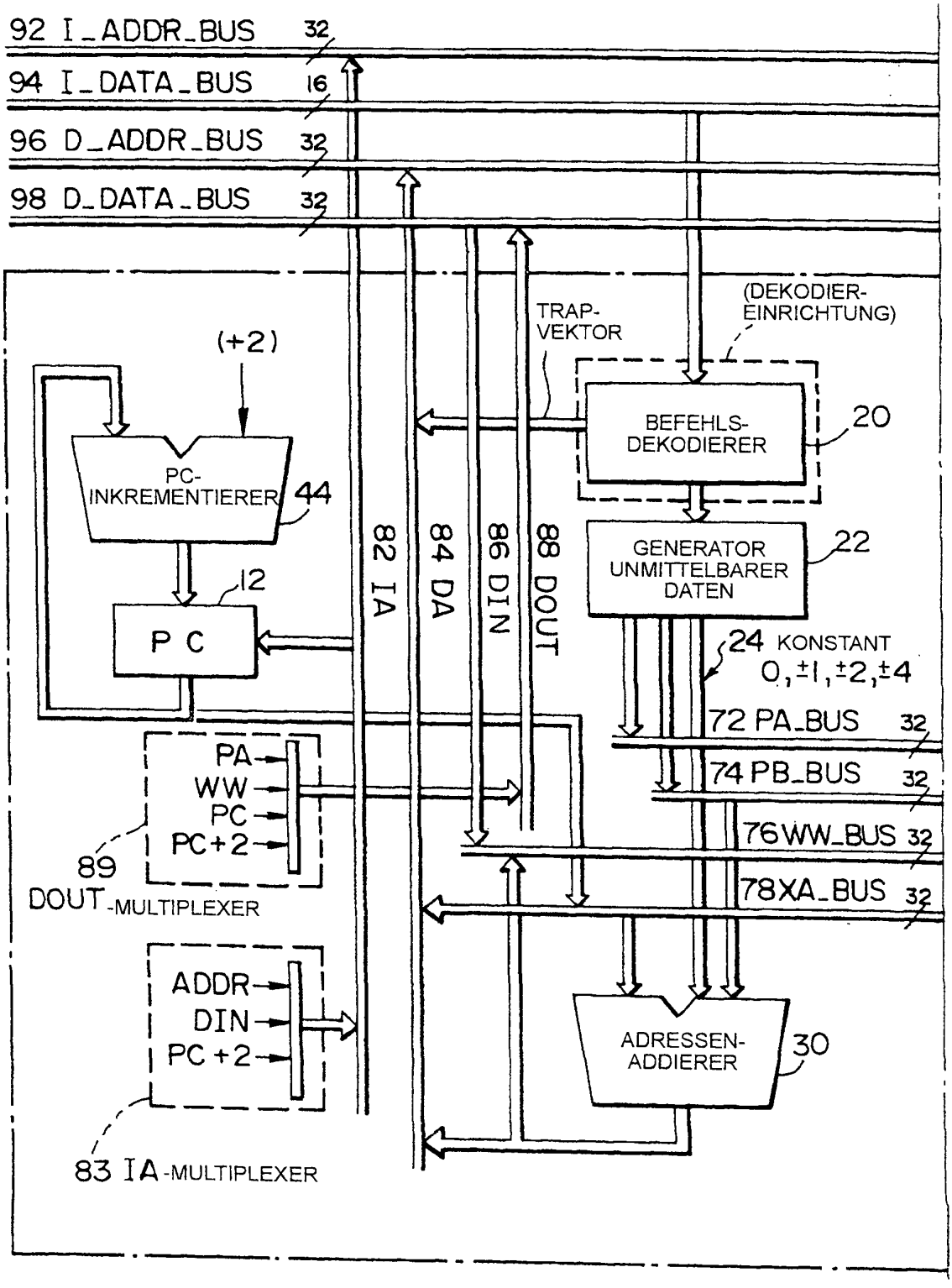


FIG. 1B

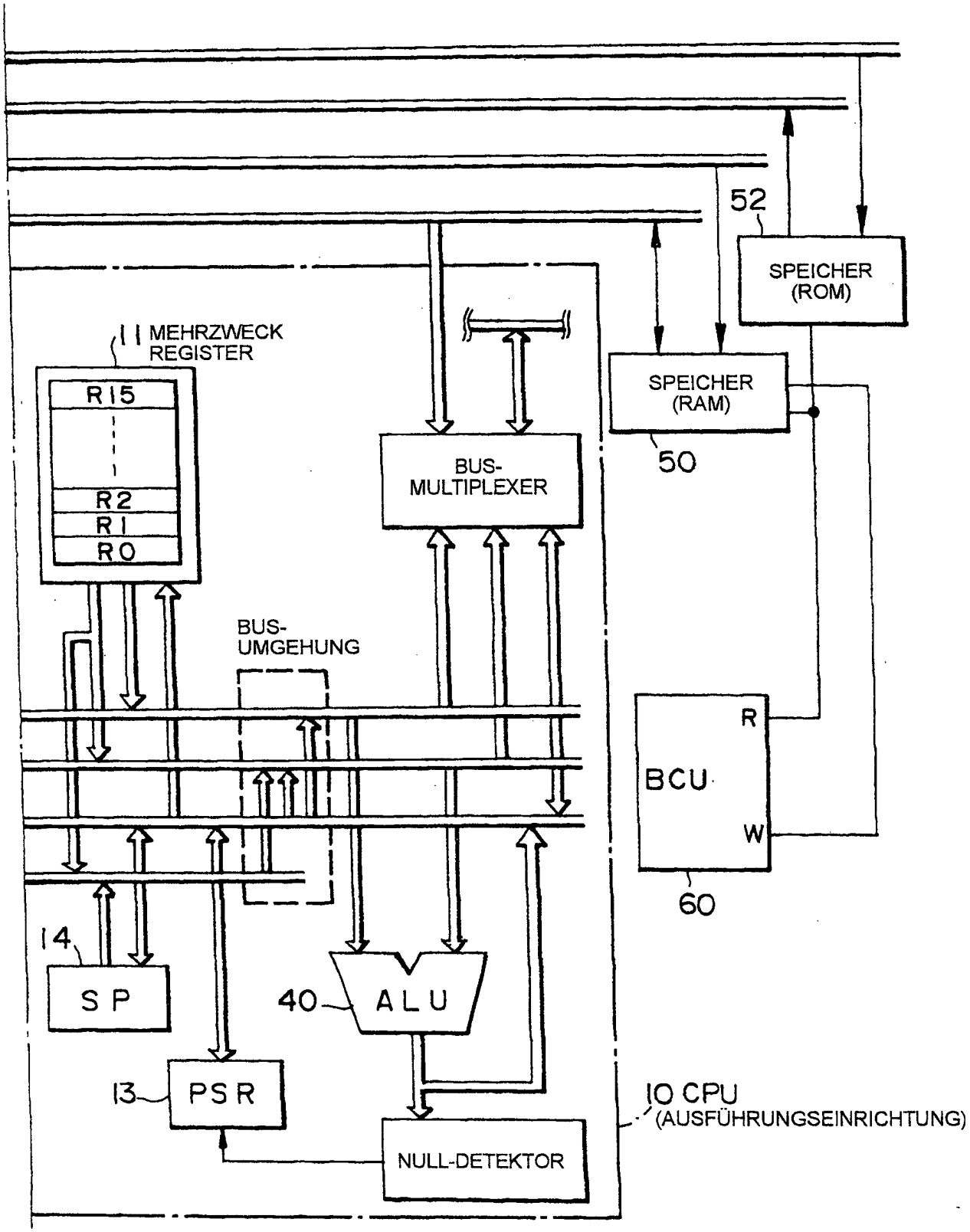

52
SPEICHER (ROM)
SPEICHER (RAM)
50
11 MEHRZWECK REGISTER
R15
R2
R1
R0
BUS-MULTIPLEXER
BUS-UMGEHUNG
R
BCU
W
60
14
SP
40
ALU
13
PSR
NULL-DETEKTOR
10 CPU
(AUSFÜHRUNGSEINRICHTUNG)

FIG.2

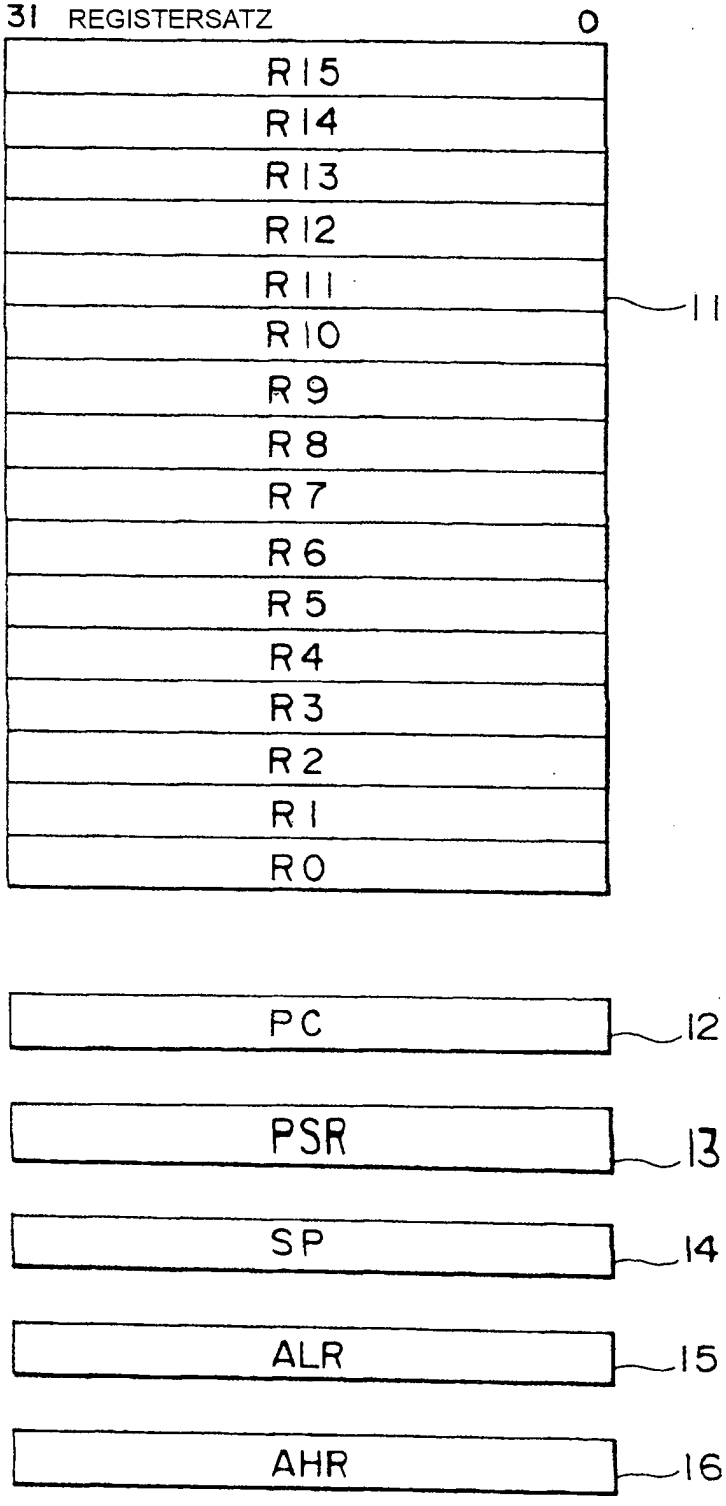

31 REGISTERSATZ 0
R15
R14
R13
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
11
PC
12
PSR
13
SP
14
ALR
15
AHR
16

FIG. 3

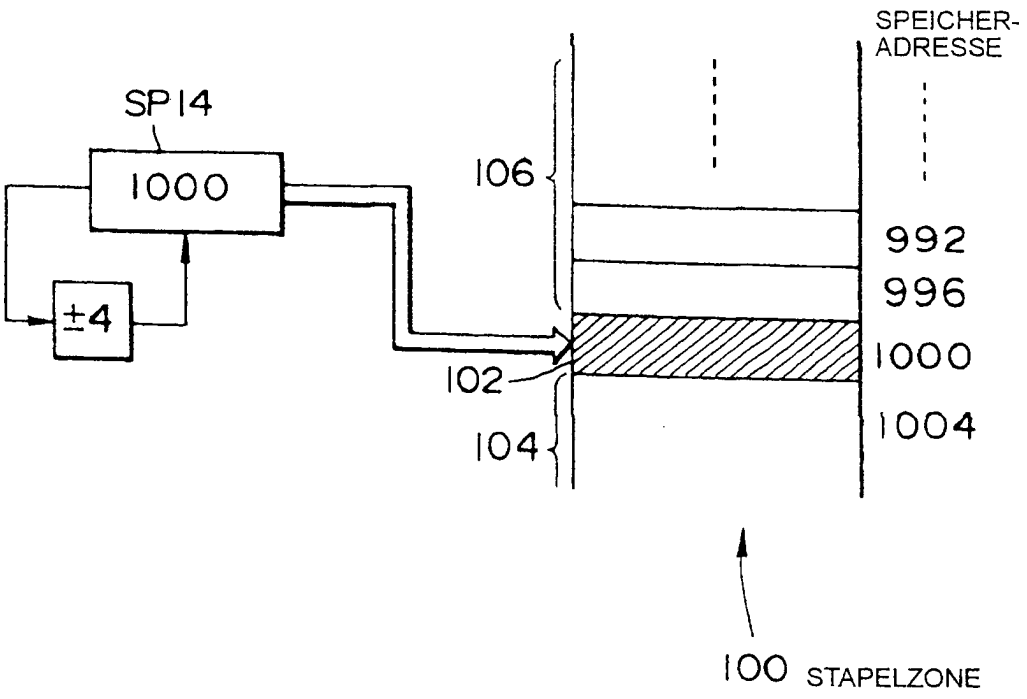
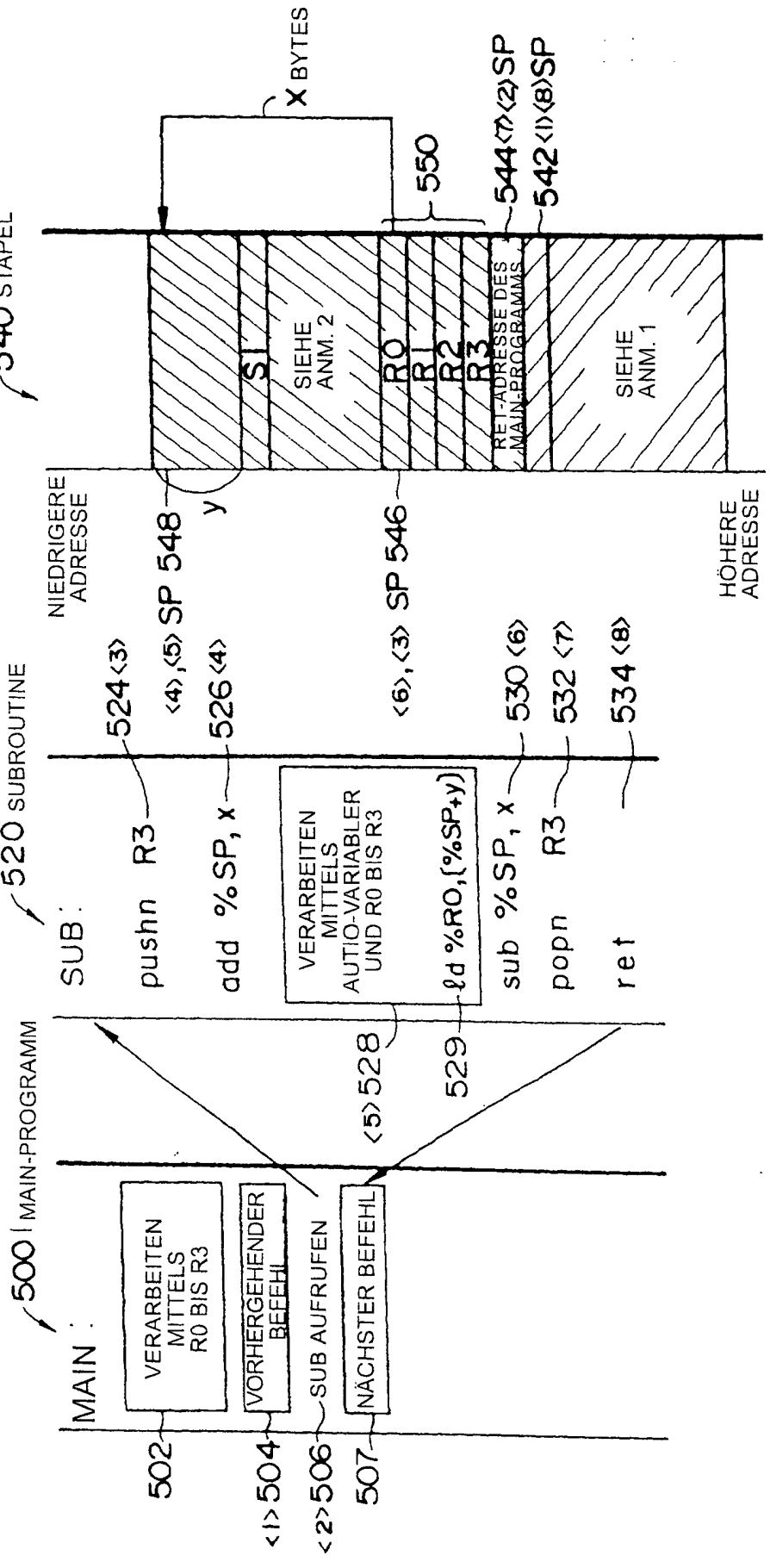

SPEICHER-
ADRESSE
SP14
1000
±4
106
992
996
1000
1004
102
104
100 STAPELZONE

FIG. 4



ANM. 1: STAPELZONE VON MAIN-PROGRAMM BENUTZT

ANM. 2: ZONE AUTO-VARIABLER VON DER SUBROUTINE BENUTZT

FIG. 5

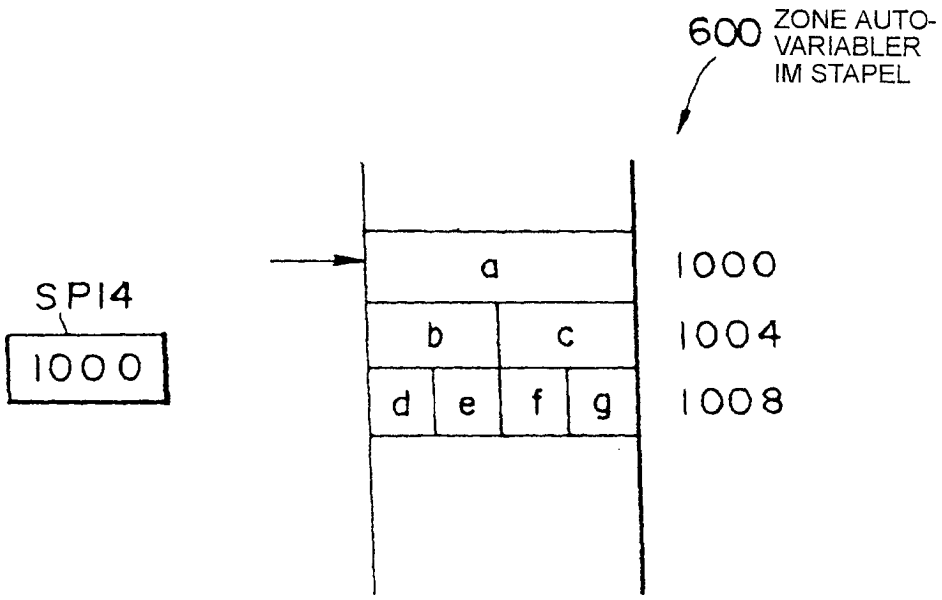

600
ZONE AUTO-
VARIABLER
IM STAPEL
SP14
1000
a
b
c
d
e
f
g
1000
1004
1008

FIG. 6A

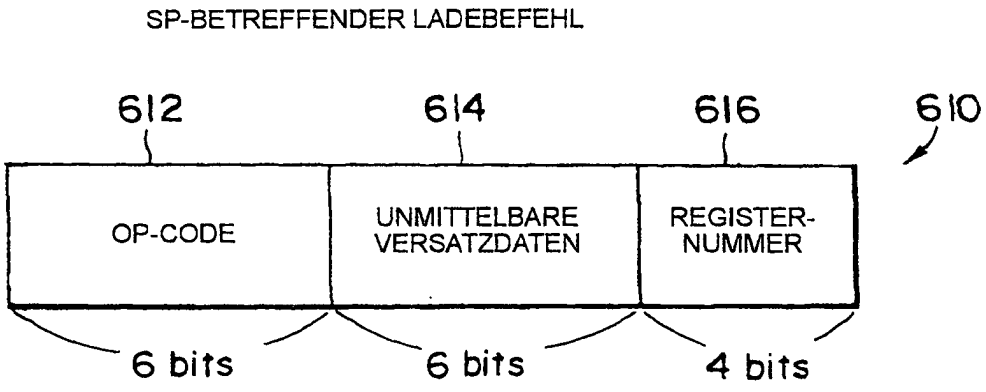

SP-BETREFFENDER LADEBEFEHL
612
614
616
610
OP-CODE
UNMITTELBARE VERSATZDATEN
REGISTER-NUMMER
6 bits
6 bits
4 bits

FIG. 6B

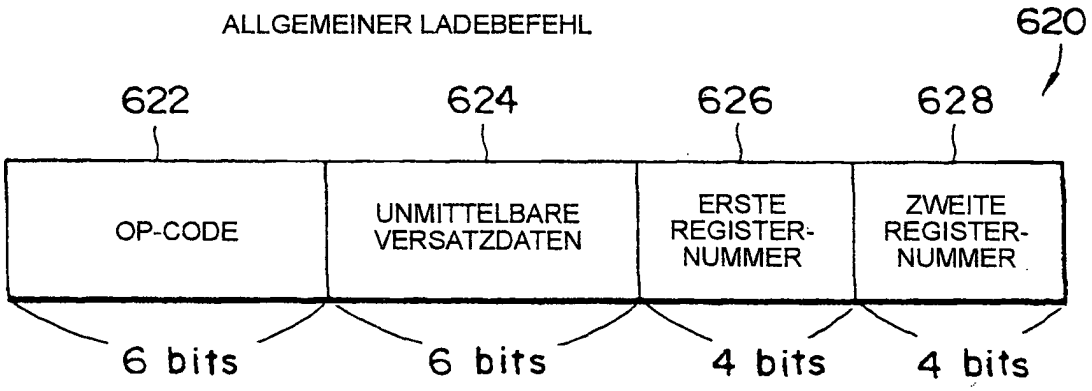

ALLGEMEINER LADEBEFEHL
620
622
624
626
628
OP-CODE
UNMITTELBARE VERSATZDATEN
ERSTE REGISTER-NUMMER
ZWEITE REGISTER-NUMMER
6 bits
6 bits
4 bits
4 bits

FIG. 7

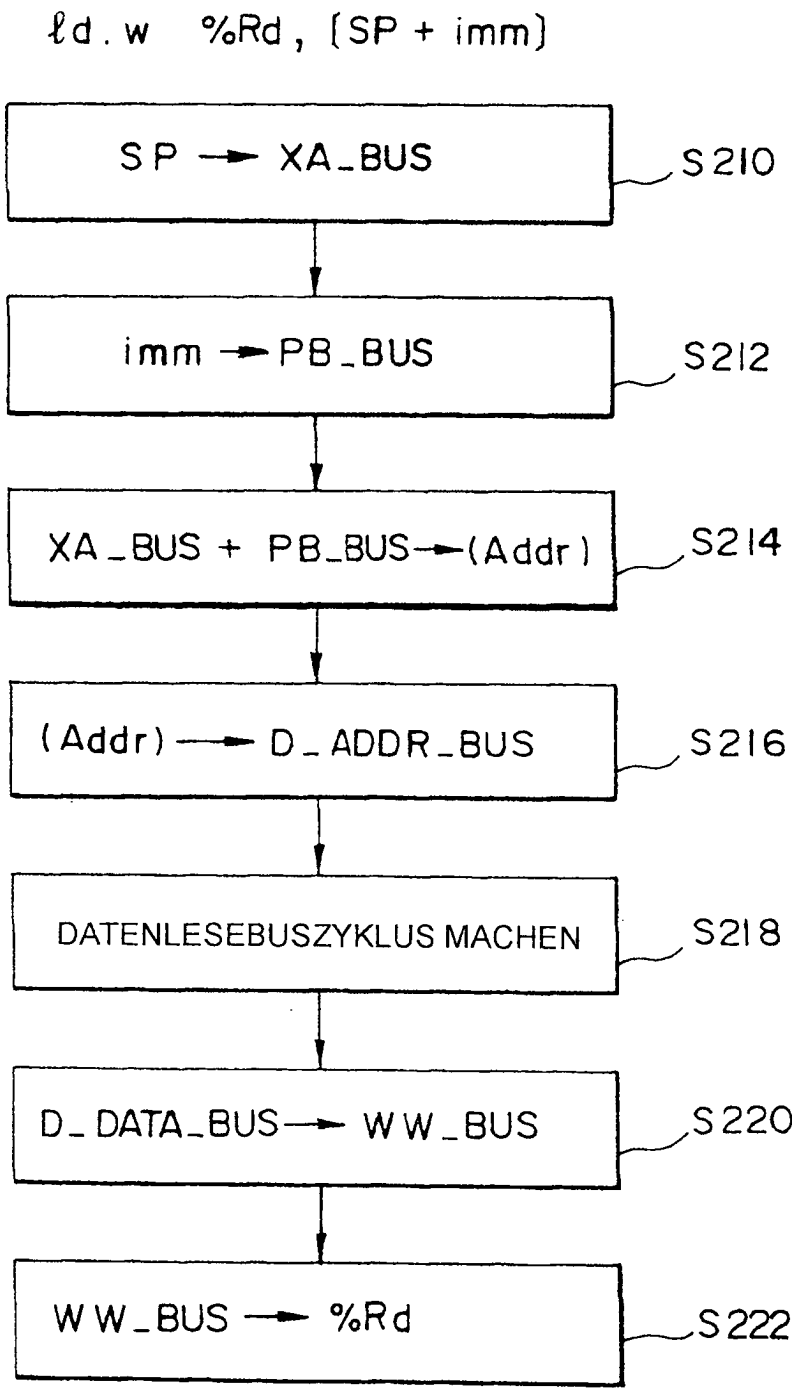
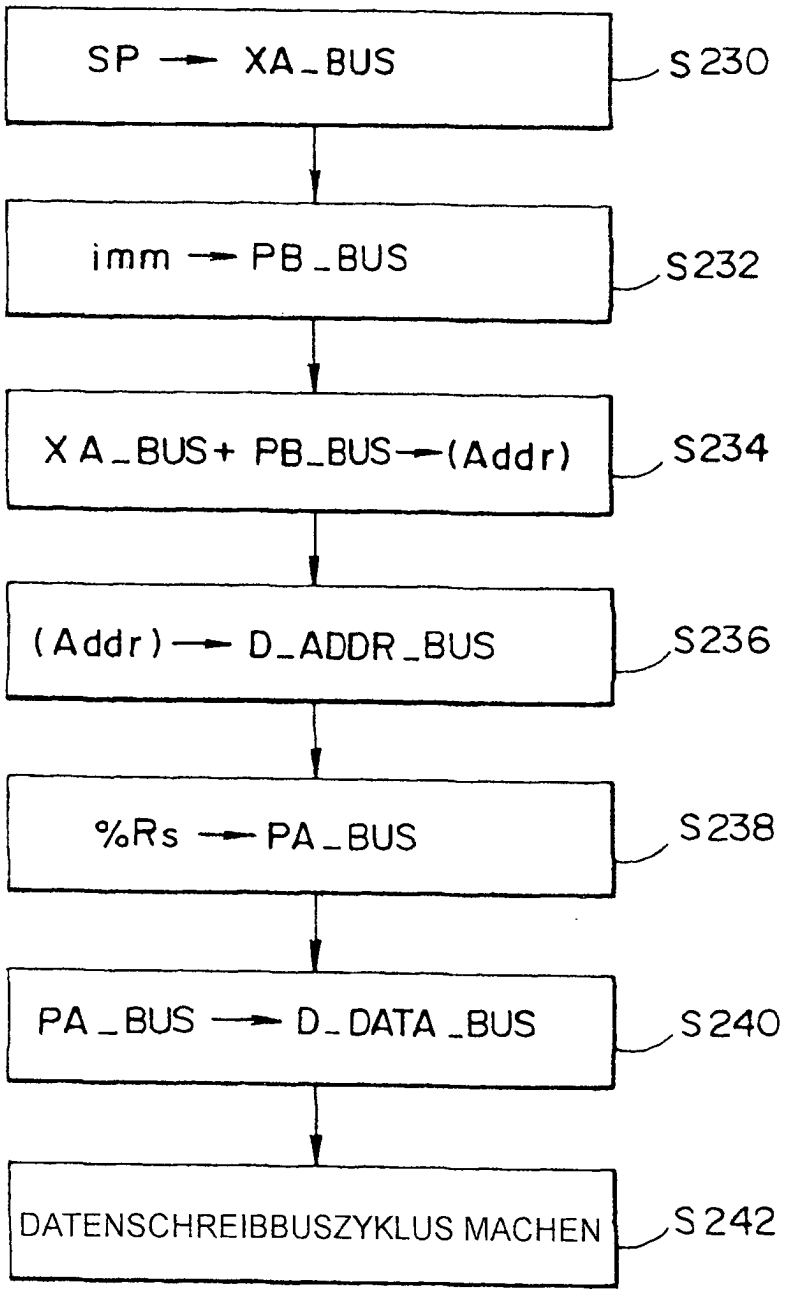

ld.w %Rd, [SP + imm]
SP → XA_BUS
S210
imm → PB_BUS
S212
XA_BUS + PB_BUS → (Addr)
S214
(Addr) → D_ADDR_BUS
S216
DATENLESEBUSZYKLUS MACHEN
S218
D_DATA_BUS → WW_BUS
S220
WW_BUS → %Rd
S222

FIG. 8

ld.w [SP + imm], %Rs



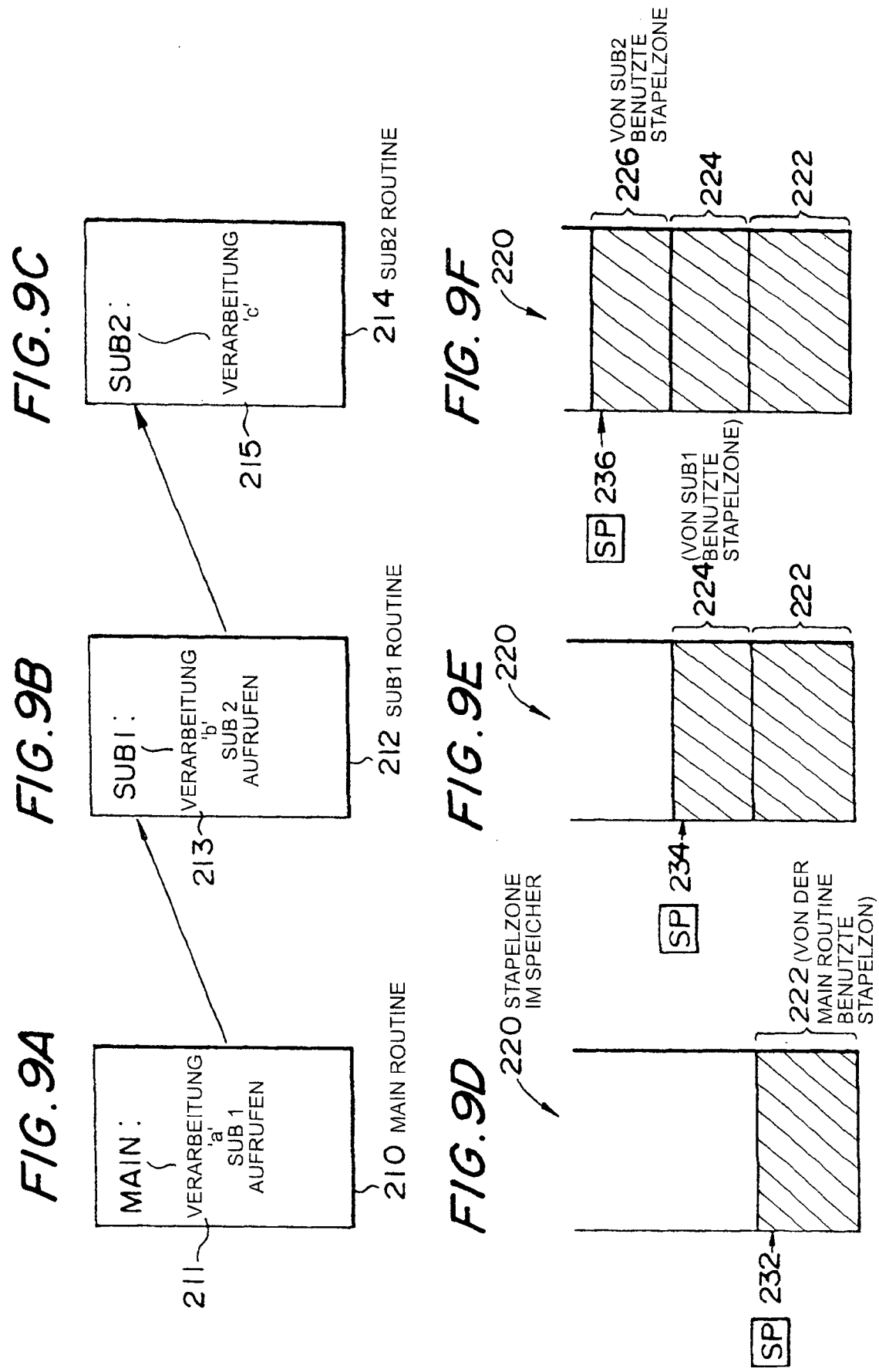


FIG. 9A
MAIN :
VERARBEITUNG 'a'
SUB 1 AUFRUFEN
211
210 MAIN ROUTINE
FIG. 9B
SUB1 :
VERARBEITUNG 'b'
SUB 2 AUFRUFEN
213
212 SUB1 ROUTINE
FIG. 9C
SUB2 :
VERARBEITUNG 'c'
215
214 SUB2 ROUTINE
FIG. 9D
220 STAPELZONE IM SPEICHER
222 (VON DER MAIN ROUTINE BENUTZTE STAPELZON)
SP
232
FIG. 9E
220
224 (VON SUB1 BENUTZTE STAPELZONE)
222
SP
234
FIG. 9F
220
226 VON SUB2 BENUTZTE STAPELZONE
224
222
SP
236

FIG. 10A

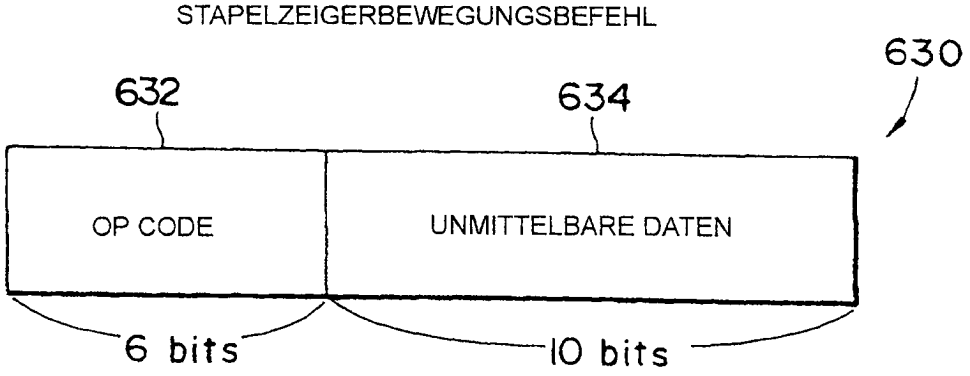


FIG. 10B

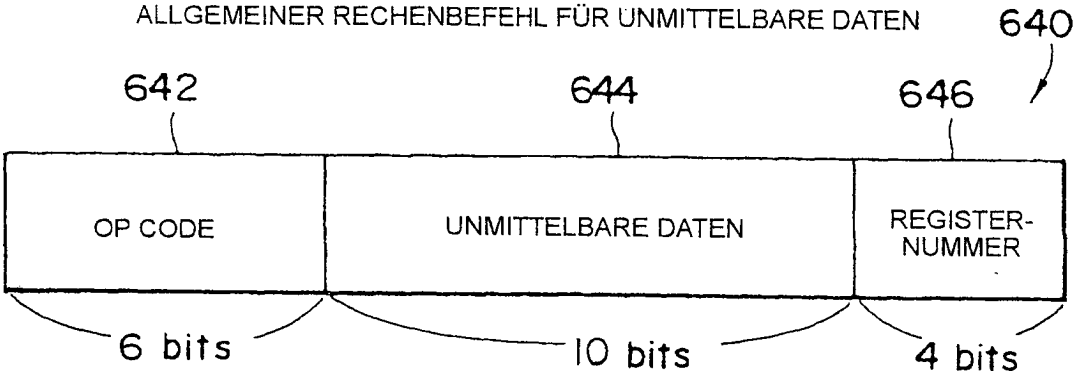


FIG. 11

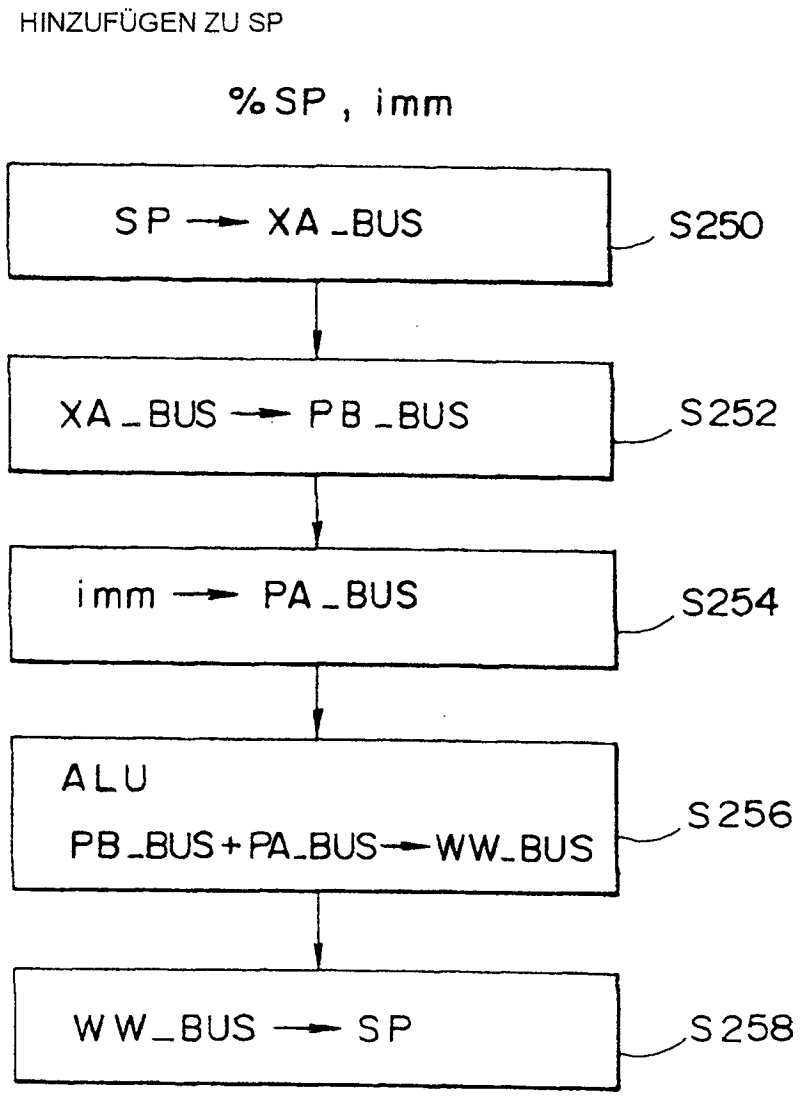

HINZUFÜGEN ZU SP
%SP, imm
SP → XA_BUS
S250
XA_BUS → PB_BUS
S252
imm → PA_BUS
S254
ALU
PB_BUS + PA_BUS → WW_BUS
S256
WW_BUS → SP
S258

FIG. 12

sub %sp imm

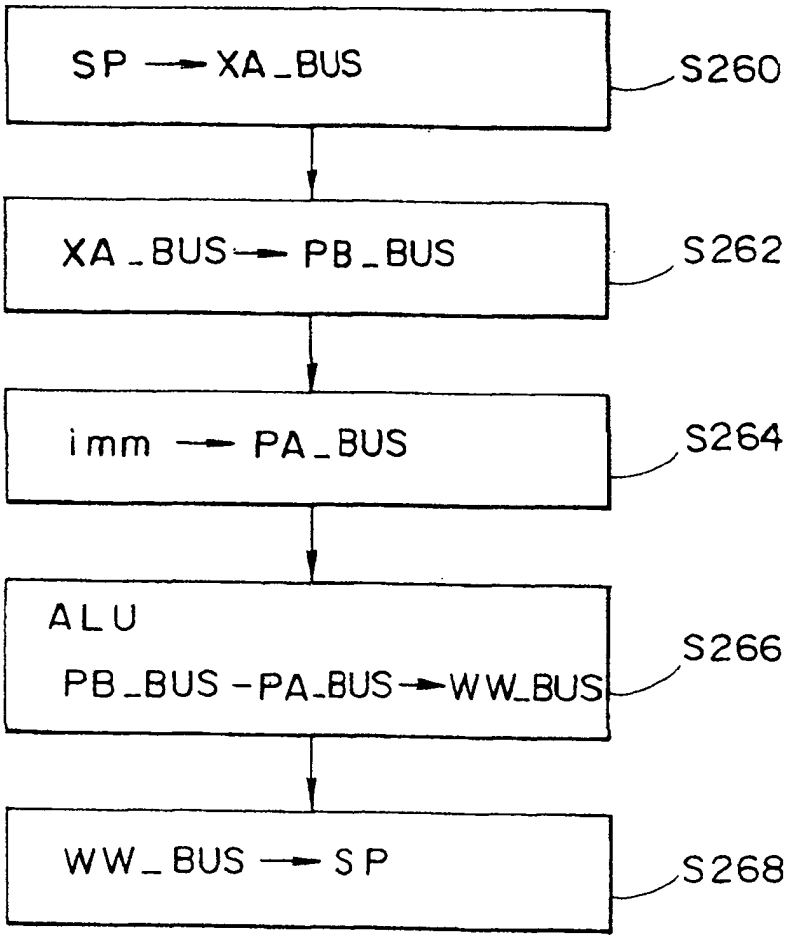


FIG. 13

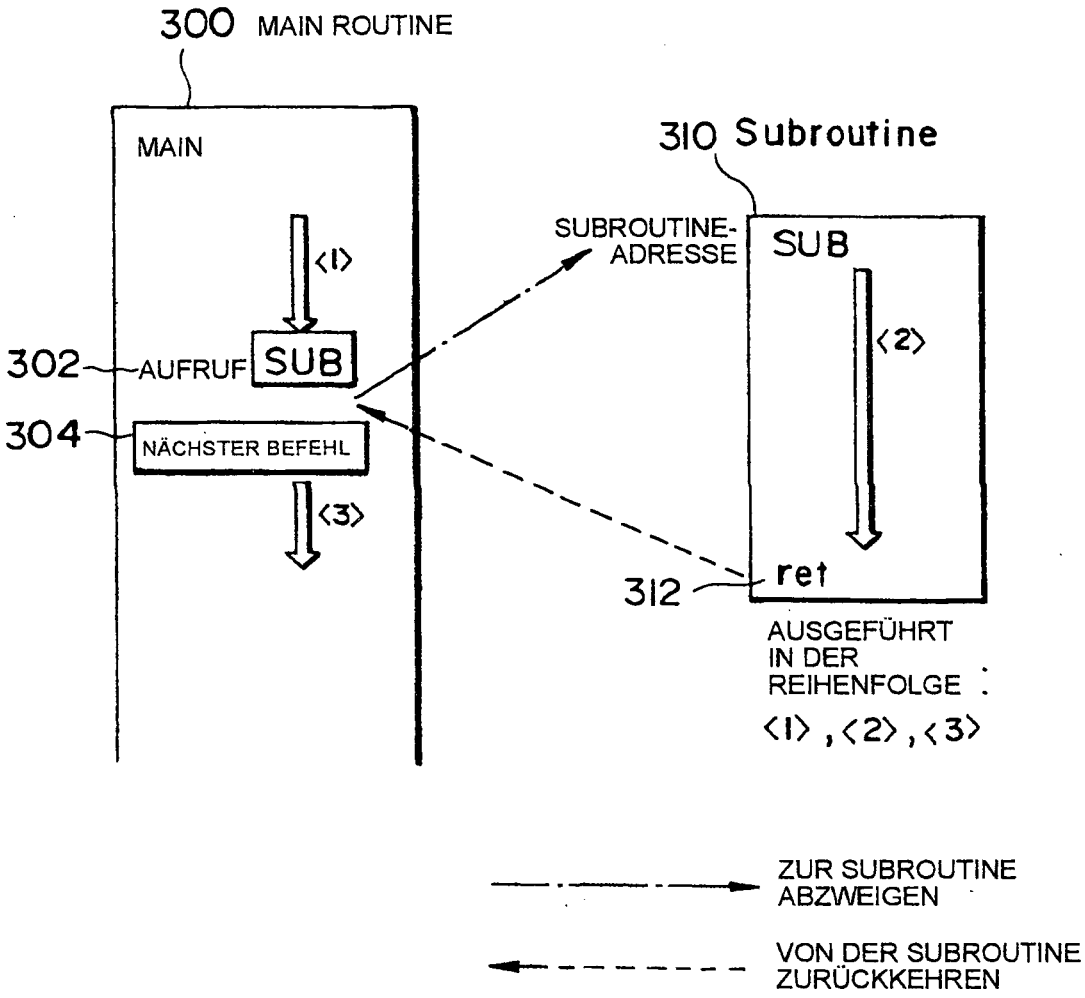

300 MAIN ROUTINE
MAIN
310 Subroutine
SUBROUTINE-ADRESSE
SUB
<1>
<2>
302
AUFRUF SUB
304
NÄCHSTER BEFEHL
<3>
312
ret
AUSGEFÜHRT IN DER REIHENFOLGE :
<1>, <2>, <3>
ZUR SUBROUTINE ABZWEIGEN
VON DER SUBROUTINE ZURÜCKKEHREN

FIG. 14

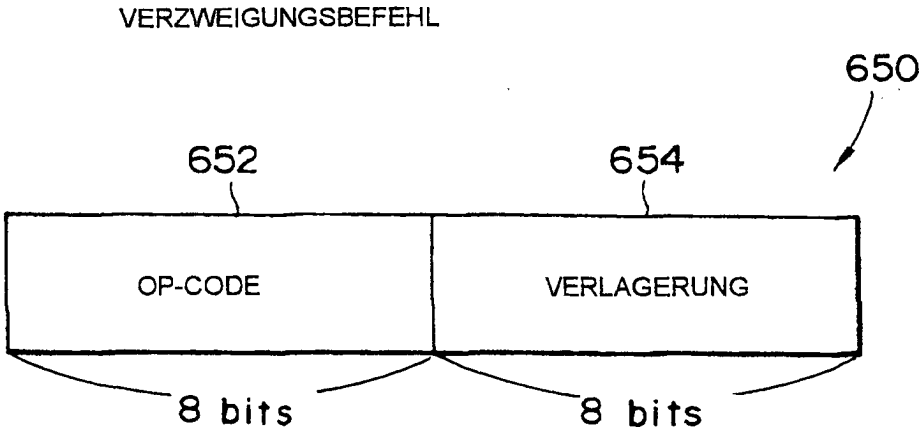

VERZWEIGUNGSBEFEHL
650
652
654
OP-CODE
VERLAGERUNG
8 bits
8 bits

FIG. 15

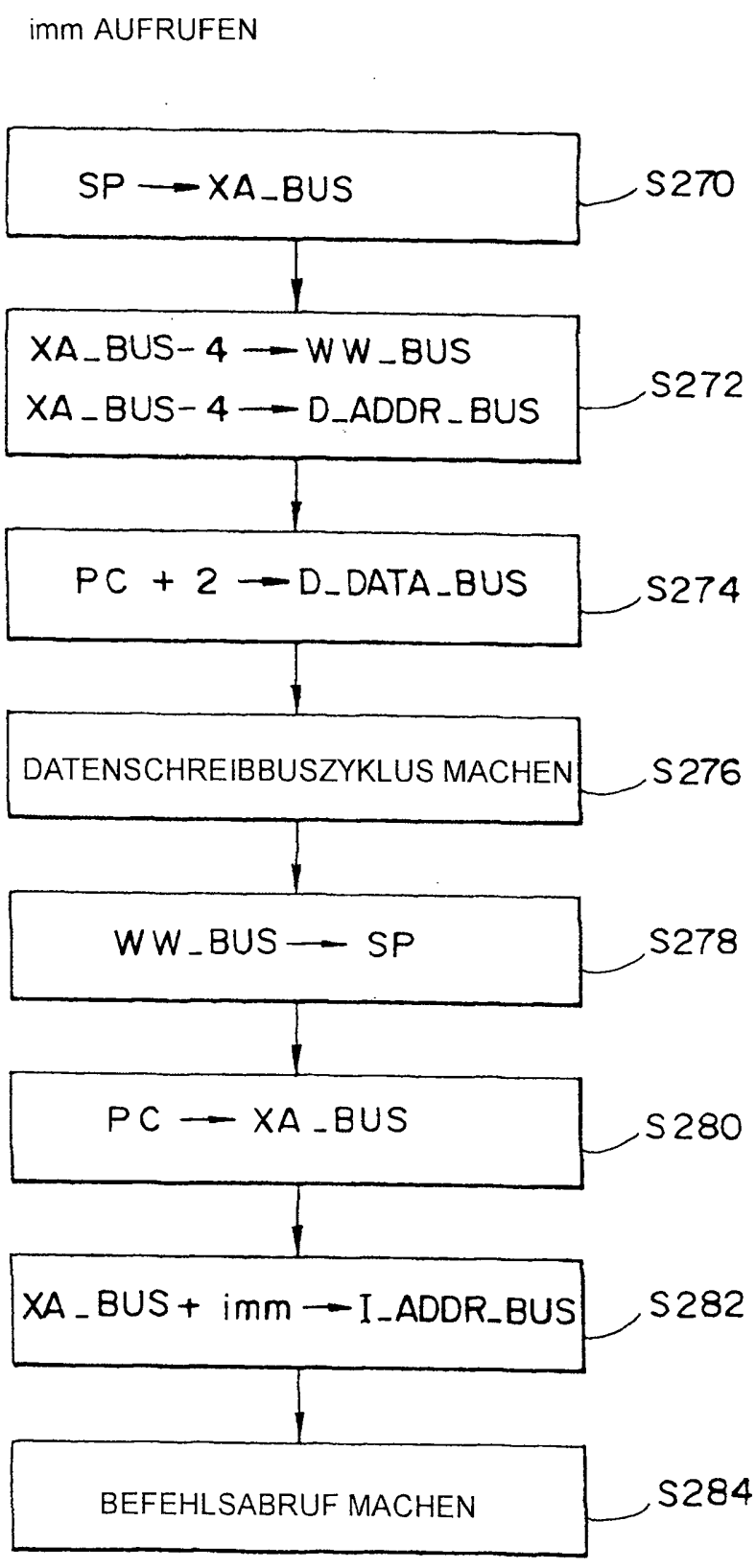

imm AUFRUFEN
SP → XA_BUS
S270
XA_BUS-4 → WW_BUS
XA_BUS-4 → D_ADDR_BUS
S272
PC + 2 → D_DATA_BUS
S274
DATENSCHREIBBUSZYKLUS MACHEN
S276
WW_BUS → SP
S278
PC → XA_BUS
S280
XA_BUS + imm → I_ADDR_BUS
S282
BEFEHLSABRUF MACHEN
S284

FIG. 16

ret

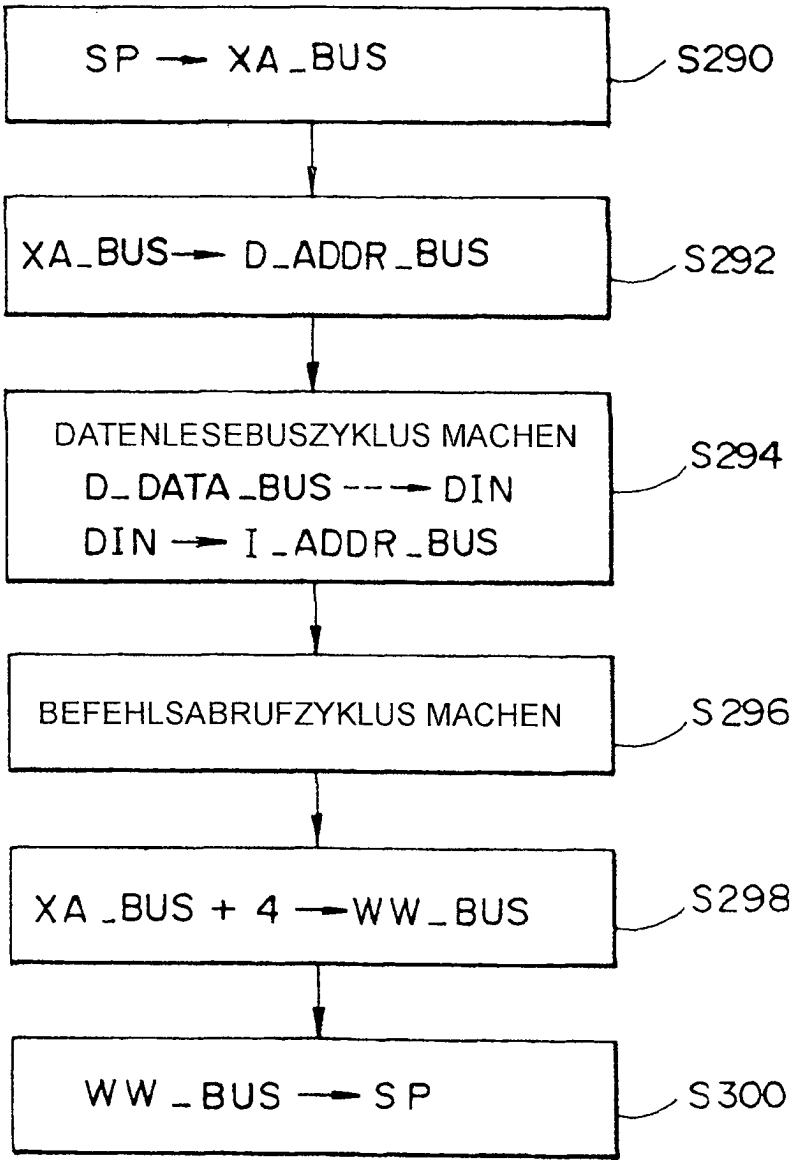

SP → XA_BUS
S290
XA_BUS → D_ADDR_BUS
S292
DATENLESEBUSZYKLUS MACHEN
D_DATA_BUS --→ DIN
DIN → I_ADDR_BUS
S294
BEFEHLSABRUFZYKLUS MACHEN
S296
XA_BUS + 4 → WW_BUS
S298
WW_BUS → SP
S300

FIG. 17A

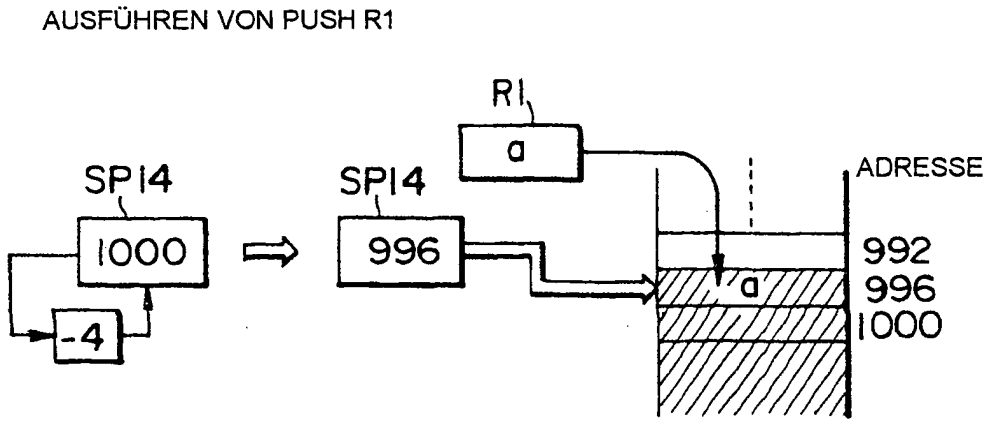


FIG. 17B

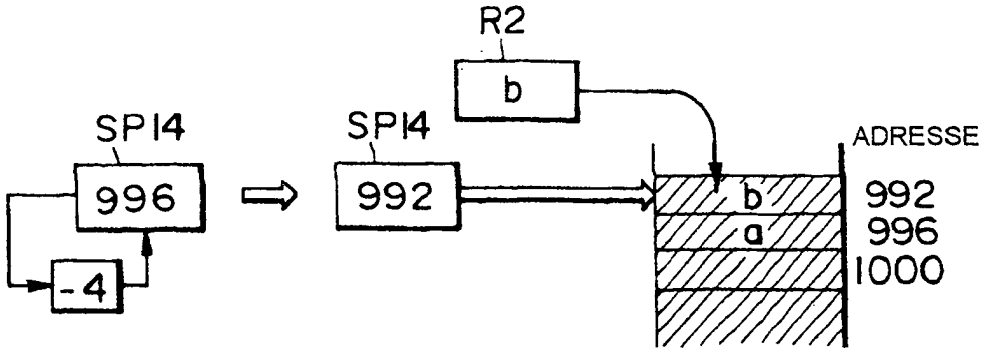


FIG. 18A

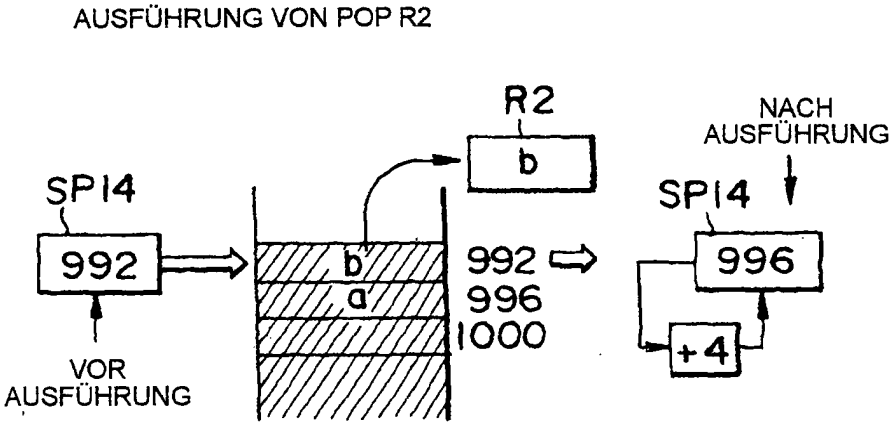


FIG. 18B

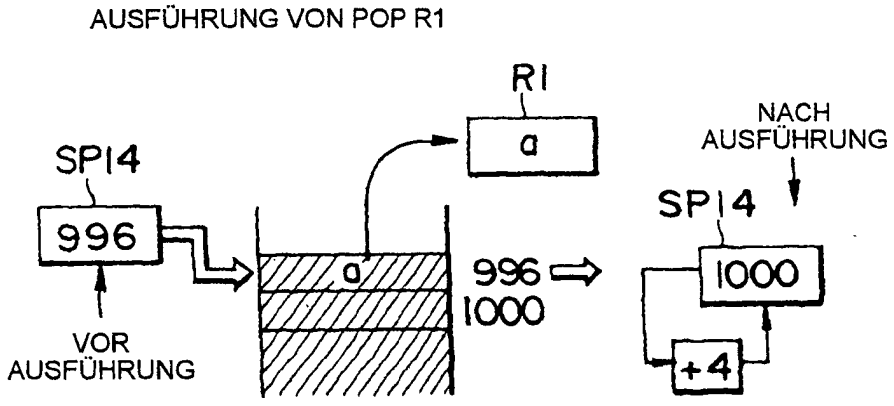
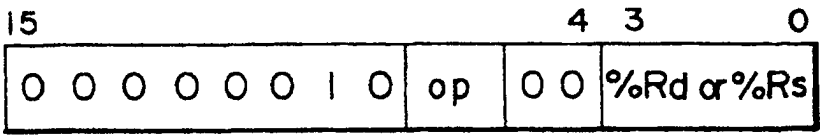


FIG. 19

push- UND popn-BEFEHLE



BEFEHL	op
pushn	0 0
popn	0 1

FIG. 20

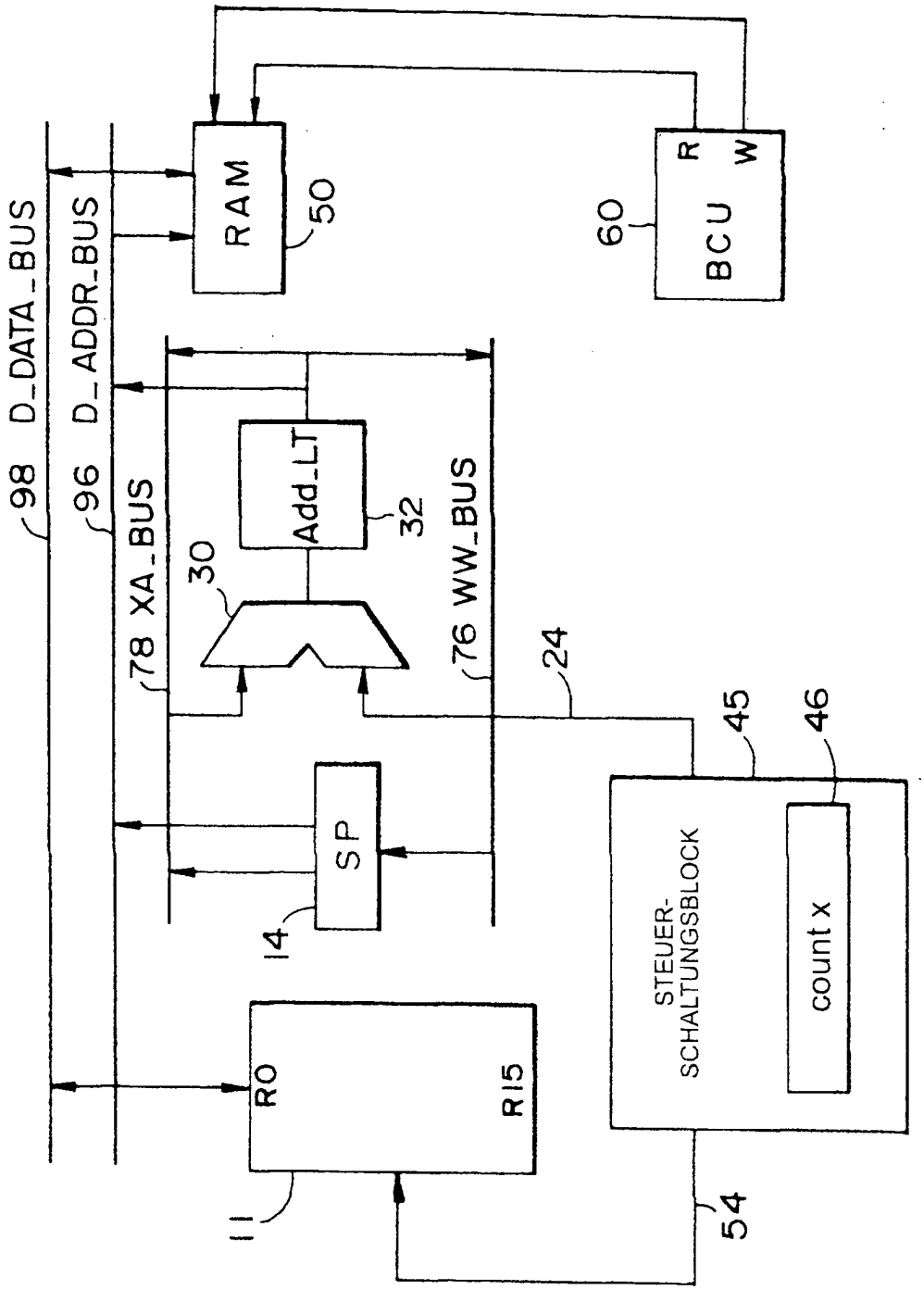

98 D_DATA_BUS
96 D_ADDR_BUS
78 XA_BUS
76 WW_BUS
RAM
50
60
BCU
R
W
Add_LT
32
30
24
45
46
SP
14
STEUER-
SCHALTUNGSBLOCK
count x
R0
R15
11
54

FIG. 21

ABLAUFDIAGRAMM DER AUSFÜHRUNG VON pushn

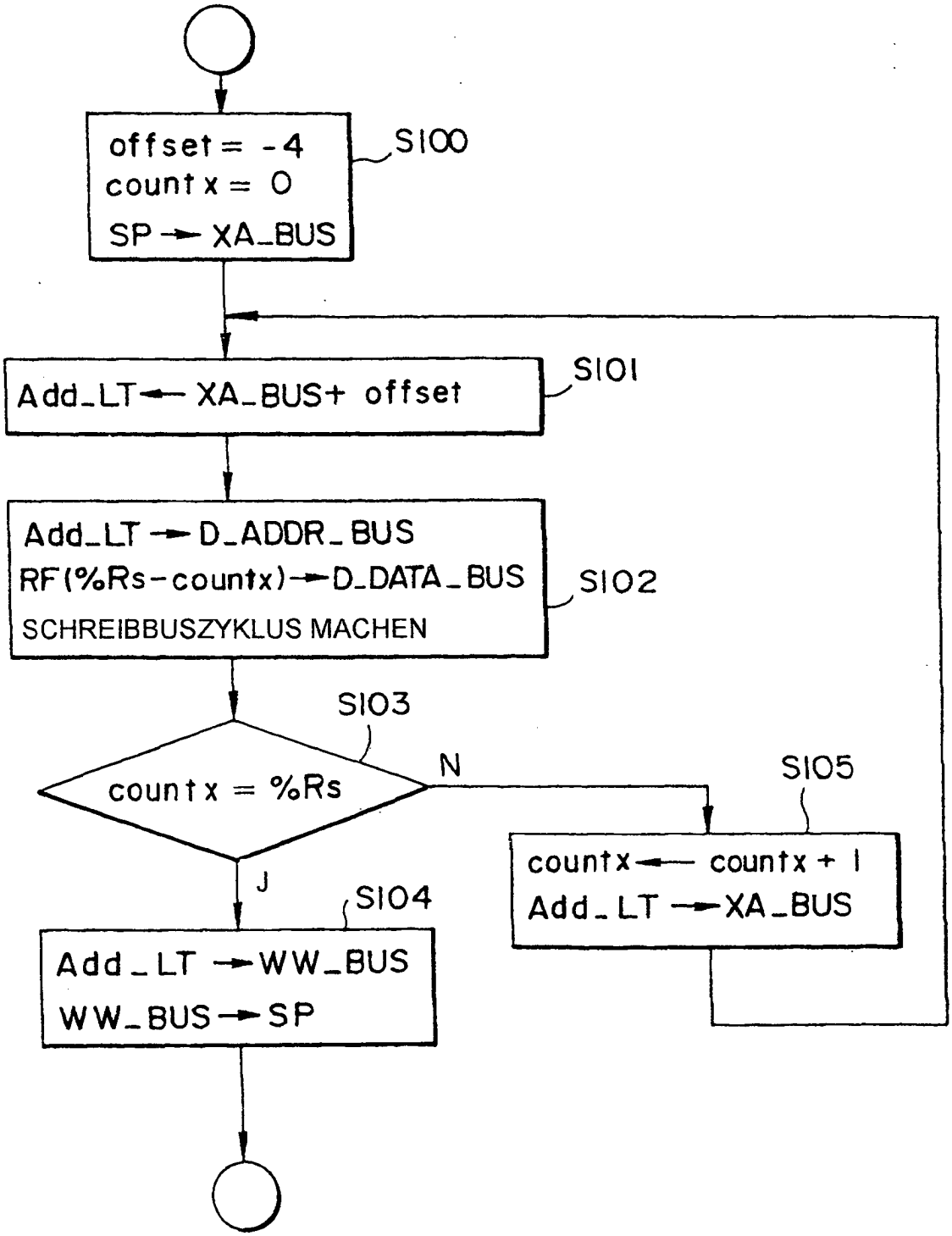


FIG.22

AUFLAUFDIAGRAMM DER AUSFÜHRUNG VON popn

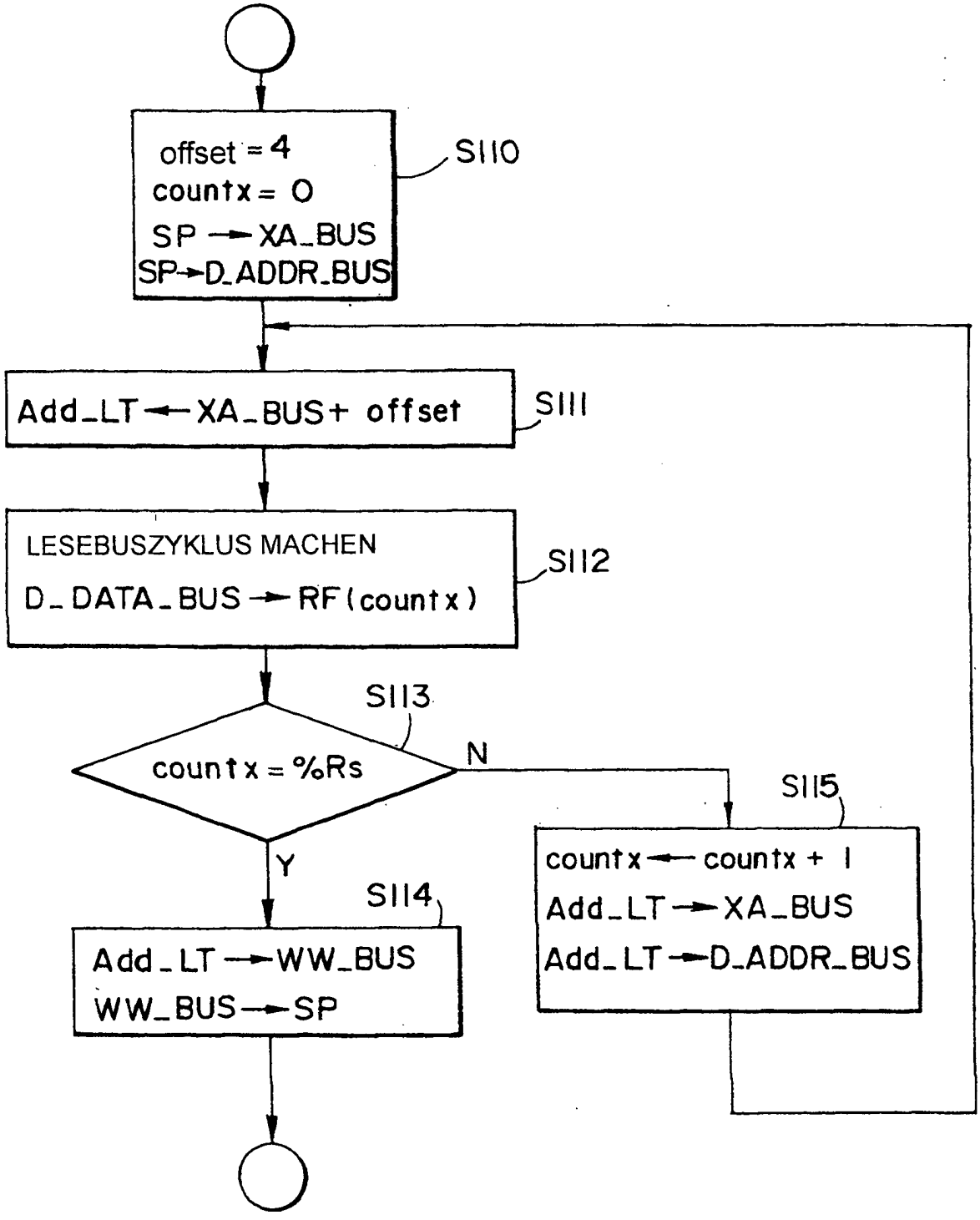
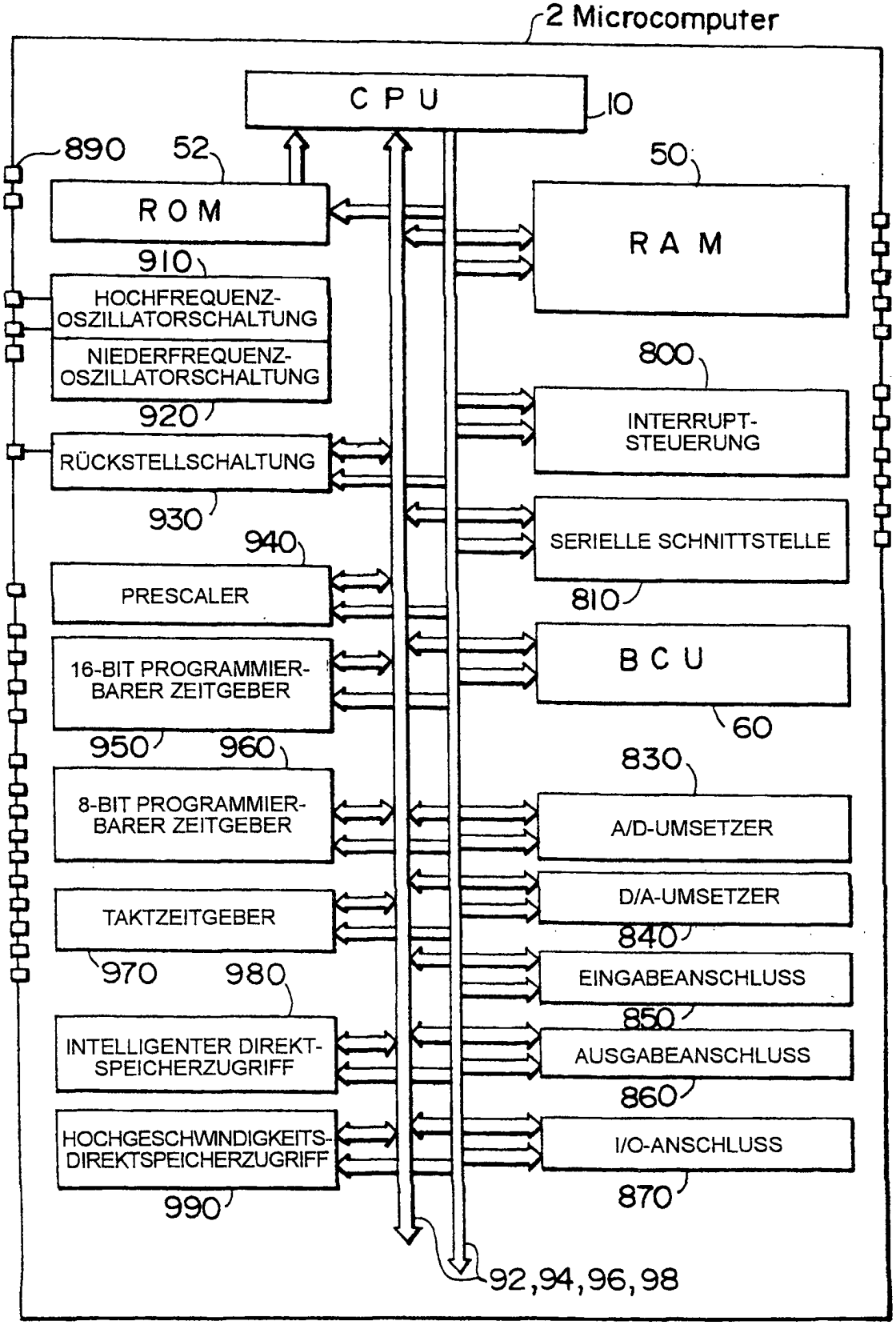


FIG. 23


2 Microcomputer
CPU
10
890
52
ROM
910
HOCHFREQUENZ-OSZILLATORSCHALTUNG
NIEDERFREQUENZ-OSZILLATORSCHALTUNG
920
RÜCKSTELLSCHALTUNG
930
940
PRESCALER
16-BIT PROGRAMMIER-BARER ZEITGEBER
950
960
8-BIT PROGRAMMIER-BARER ZEITGEBER
TAKTZEITGEBER
970
980
INTELLIGENTER DIREKT-SPEICHERZUGRIFF
HOCHGESCHWINDIGKEITS-DIREKTSPEICHERZUGRIFF
990
50
RAM
800
INTERRUPT-STEUERUNG
SERIELLE SCHNITTSTELLE
810
BCU
60
830
A/D-UMSETZER
D/A-UMSETZER
840
EINGABEANSCHLUSS
850
AUSGABEANSCHLUSS
860
I/O-ANSCHLUSS
870
92,94,96,98