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

(10) **DE 600 06 021 T2** 2004.08.12

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 046 983 B1**
(21) Deutsches Aktenzeichen: **600 06 021.7**
(96) Europäisches Aktenzeichen: **00 108 399.7**
(96) Europäischer Anmeldetag: **17.04.2000**
(97) Erstveröffentlichung durch das EPA: **25.10.2000**
(97) Veröffentlichungstag
der Patenterteilung beim EPA: **22.10.2003**
(47) Veröffentlichungstag im Patentblatt: **12.08.2004**

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

(30) Unionspriorität:
11345999 21.04.1999 JP

(84) Benannte Vertragsstaaten:
DE, FR, GB

(73) Patentinhaber:
Mitsubishi Denki K.K., Tokio/Tokyo, JP

(72) Erfinder:
Suzuki, Hiroaki, Tokyo 100-8310, JP

(74) Vertreter:
PRÜFER & PARTNER GbR, 81545 München

(54) Bezeichnung: **VLIW-Prozessor und Programmkomprimierungsgerät and -verfahren**

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] Die vorliegende Erfindung bezieht sich auf VLIW-(sehr langes Instruktionswort)Prozessoren und insbesondere auf einen VLIW-Prozessor, der einen komprimierten Programmcode ausführt, eine Codekompressionseinrichtung, die einen Programmcode komprimiert, ein Codekompressionsverfahren und ein Medium, auf dem ein Codekompressionsprogramm aufgezeichnet wird.

Beschreibung der Hintergrundstechnik

[0002] Es hat zunehmend der Anforderungen für verbesserte Prozessorleistung auf verschiedenen Gebieten wie Multimediaverarbeitung und Verarbeitung hochaufgelöster Bilder gegeben. Die gegenwärtige LSI-(Integration auf großem Maßstab)Herstellungstechnik weist eine Grenze beim Erzielen höherer Einrichtungsgeschwindigkeit auf. Daher haben VLIW-Prozessoren die Aufmerksamkeit angezogen und sind aktiv untersucht und entwickelt worden.

[0003] Der VLIW-Prozessor, der eine Mehrzahl von Verarbeitungseinheiten enthält, zielt auf die Verbesserung der Leistung eines programmgesteuerten Prozessors durch Ausführen einer Mehrzahl von Instruktionen in dem gleichen Zyklus und somit Vergrößerung der Parallelität. Bei dem VLIW-Prozessor ist eine Verarbeitungseinheit entsprechend einem Instruktionsfeld vorbestimmt, und daher werden nicht alle Instruktionsslots mit gültigen Instruktionen gefüllt. Somit wird eine nop-(keine Tätigkeit)Instruktion als eine ungültige Instruktion in einen Code eingefügt, was in dem wesentlichen Problem niedriger Codedichte resultiert.

[0004] Der VLIW-Prozessor, bei dem zwei Instruktionen simultan ausgegeben werden, kann nicht simultan "load mem (100) → R1" und "add R1, R2 → R3" ausführen, wie in dem Codebeispiel 1 unten gezeigt ist.

Codebeispiel 1:

Codenummer	Instruktionssequenz für linke Unterinstruktionen	Instruktionssequenz für rechte Unterinstruktionen
1	store R0 -> mem (200)	:sub R4, R5 -> R6
2	load mem (100) -> R1	:nop
3	nop	:add R1, R2 -> R3.

[0005] Im folgenden werden die Ausdrücke "Instruktion" und "Unterinstruktion" zum Bezeichnen des gleichen Gegenstandes benutzt, im Gegensatz zu VLIW (siehe unten).

[0006] Da eine von ihnen eine Instruktion zum Laden eines Wertes in ein Register R1 ist und die andere eine Instruktion zum Bezugnehmen auf den Wert des gleichen Registers R1 ist, können diese Instruktionen nicht zur gleichen Zeit ausgeführt werden. In diesem Fall wird eine nop-Instruktion eingeführt, wie in dem Codebeispiel 1 gezeigt ist. In der folgenden Beschreibung wird eine lange Instruktion, die durch Kombinieren zweier Instruktionen gebildet wird, eine VLIW-(sehr langes Instruktionswort)Instruktion genannt, und jede Instruktion, die in der VLIW-Instruktion enthalten ist (load-, nop- und add-Instruktionen in dem Codebeispiel 1) wird eine Unterinstruktion oder eine Instruktion genannt.

[0007] Erfindungen, die in H. Sato, T. Yoshida, M. Matsuo, T. Kengaku und K. Tsuchihashi: "A Dual-Issue RISC Processor for Multimedia Signal Processing", IEICE Trans. Elektron., Bd. E81-C, Nr. 9, Sept. 1998 und der japanischen Patentoffenlegungsschrift 9-84004 offenbart sind, sind zum Lösen des obigen Problemes gerichtet. Zusätzlich zu diesen Instruktionen wird Codekompression durch Trennen einer VLIW-Instruktion, die Instruktionen enthält, die ursprünglich zu der gleichen Zeit auszuführen sind, in zwei Unterinstruktionen und Ausführen der Unterinstruktionen ausgeführt. Mit anderen Worten, "load mem (100) → R1" und "add R1, R2 → R3" werden als eine VLIW-Instruktion beschrieben, und die VLIW-Instruktion wird in zwei Zyklen während der Ausführung unterteilt zum getrennten Ausführen der Unterinstruktionen, wie in dem Codebeispiel 2 unten gezeigt ist.

Codebeispiel 2

Codenummer	Instruktionssequenz für linke Unterinstruktionen	Instruktionssequenz für rechte Unterinstruktionen
1	store R0 -> mem (200)	:sub R4, R5 -> R6
2	load mem (100) -> R1	:add R1, R2 -> R3.

[0008] Für diese Operation werden FM-(Format-spezifierende)Bit zwei niedrigstwertigen Bit der 32-Bit VLIW-Instruktion zugeordnet, und die Instruktion wird durch den Prozessor auf die folgende Weise gemäß dem FM-Bitmuster ausgeführt.

(1) FM = 00 : gleichzeitiges Ausführen zweier Unterinstruktionen
(2) FM = 01 : Ausführen der linken Unterinstruktion und dann Ausführen der rechten Unterinstruktion
(3) FM = 10 : Ausführen der rechten Unterinstruktion und dann Ausführen der linken Unterinstruktion
(4) FM = 11 : Ausführen der zwei Unterinstruktionen durch Betrachten der zwei Unterinstruktionen als eine lange Instruktion.

[0009] Da der Betriebsmodus in dem Fall FM = 11 irrelevant für die vorliegende Erfindung ist, wird er nicht im einzelnen beschrieben. **Fig.** 1 zeigt schematisch Beziehungen zwischen FM-Bit und Instruktionsausführung.
[0010] Das Codebeispiel 1 wird unten als Codebeispiel 3 beschrieben.

Codebeispiel 3:

Codenummer	FM	Instruktionssequenz für linke Unterinstruktionen	Instruktionssequenz für rechte Unterinstruktionen
1	00	store R0 -> mem (200)	:sub R4, R5 -> R6
2	01	load mem (100) -> R1	:add R1, R2 -> R3

[0011] Beim Ausführen des Codebeispiels 3 führt ein Prozessor Instruktionen aus, wie sie in dem Ausführungssequenzbeispiel 1 gezeigt sind.

Ausführungssequenzbespiel 1:

Schrittnummer	Instruktionssequenz für linke Unterinstruktionen	Instruktionssequenz für rechte Unterinstruktionen
1	store R0 -> mem (200)	:sub R4, R5 -> R6
2	load mem (100) -> R1	:
2'		:add R1, R2 -> R3.

[0012] In dem zweiten Zyklus nimmt eine arithmetische Einheit entsprechend der Addierinstruktion den Nichtbetriebszustand an und führt im wesentlichen die gleiche Tätigkeit wie bei der Ausführung der nop-Instruktion aus. In dem 2'-ten Zyklus nimmt eine arithmetische Einheit entsprechend der Ladeinstruktion den Nichttätigkeitszustand an und führt im wesentlichen die gleiche Tätigkeit wie das Ausführen der nop-Instruktion aus.
[0013] Daher ist das Ausführungssequenzbesipiel 1 äquivalent zu der Ausführung der Codebeispiels 1. Da Instruktionen in einem Instruktionsspeicher in dem Format des Codebeispiels 3 gespeichert sind, werden sie auf 2/3-faches der Kapazität im Vergleich mit dem Codebeispiel 1 komprimiert.
[0014] Bei dem obigen herkömmlichen VLIW-Prozessor kann die Reihenfolge des Beschreibens der Unterinstruktionen freier bestimmt werden, und die Codedichte kann durch Addieren von FM-Bit zu einer VLIW-Instruktion verbessert werden. Da die FM-Bit zu 2 Bit eines Instruktionsfeldes zugeordnet sind, ist jedoch die Zahl der Instruktionstypen begrenzt. Weiterhin müssen die Instruktionen vor der Instruktionsdecodierung abgebildet werden, was die Schaltungsverzögerung vergrößert.
[0015] Da der Austauschabschnitt eine Mehrzahl von Unterinstruktionen austauscht und sie in eine Mehrzahl von Containern eingibt, kann ein komprimierter Code in solch einer Ausführungssequenz ausgeführt werden, die vor der Kompression genommen wird.
[0016] Aus dem US-Patent 5 475 824 kann ein VLIW-Prozessor gemäß dem Oberbegriff des Anspruchs 1 entnommen werden. Wenn eine Operantstörung auftritt, wird zuerst eine mov-Instruktion an eine erste Pipe ausgegeben, während eine nop-Instruktion an eine zweite Pipe ausgegeben wird. Eine ADD-Instruktion wird an die erste Pipe in dem folgenden Zyklus ausgegeben. Eine Instruktion auf diese ADD-Instruktion folgend wird zu der zweiten Pipe ausgegeben. Eine sequentielle Reihenfolge der Schritte wird ausgeführt.

ZUSAMMENFASSUNG DER ERFINDUNG

[0017] Es ist eine Aufgabe der vorliegenden Erfindung, einen VLIW-Prozessor vorzusehen, der ein komprimiertes Programm in solch einer Ausführungssequenz verarbeiten kann, die vor der Kompression genommen wird, ohne daß spezielle Bit einem Instruktionsfeld zugeordnet werden, das bevorzugt dem VLIW-Verfahren von mindestens drei Wegen entspricht, das ein komprimiertes Programm in solch einer Ausführungssequenz ausführen kann, die vor der Kompression genommen wird, und das bevorzugte dem zwei n-Wege-VLIW-Verfahren entspricht, das ein komprimiertes Programm in solch einer Ausführungssequenz verarbeiten kann, die vor der Kompression genommen wird.

[0018] Solch eine Aufgabe wird gelöst durch einen VLIW-Prozessor nach Anspruch 1.
[0019] Bevorzugte Ausgestaltungen der Erfindung sind in den abhängigen Ansprüchen angegeben.
[0020] Die vorangehenden und andere Aufgaben, Merkmale, Aspekte und Vorteile der vorliegenden Erfindung werden ersichtlicher aus der folgenden detaillierten Beschreibung der vorliegenden Erfindung, wenn sie in Zusammenhang mit den begleitenden Zeichnungen genommen wird.

KURZE BESCHREIBUNG DER ZEICHNUNGEN

[0021] **Fig.** 1 zeigt schematisch die Beziehung zwischen FM-Bit, die bei einem herkömmlichen VLIW-Verfahren benutzt werden, und der Instruktionsausführung.
[0022] **Fig.** 2 zeigt schematisch einen VLIW-Prozessor in einer ersten Ausführungsform.
[0023] **Fig.** 3 ist ein Blockschaltbild, das einen schematischen Aufbau des VLIW-Prozessors in der ersten Ausführungsform zeigt.
[0024] **Fig.** 4 ist ein Blockschaltbild, das einen schematischen Aufbau eines VLIW-Prozessors in einer zweiten Ausführungsform zeigt.
[0025] **Fig.** 5 ist ein Blockschaltbild, das einen schematischen Aufbau einer Codekompressionseinrichtung in einer vierten Ausführungsform zeigt.
[0026] **Fig.** 6 ist ein Flußdiagramm zum Beschreiben einer Verarbeitungsprozedur der Codekompressionseinrichtung in der vierten Ausführungsform.
[0027] **Fig.** 7 zeigt ein Beispiel der externen Ansicht der Codekompressionseinrichtung in der vierten Ausführungsform.
[0028] **Fig.** 8 ist ein schematischer Aufbau eines in **Fig.** 7 gezeigten Computers **41**.
[0029] **Fig.** 9 zeigt schematisch einen VLIW-Prozessor in einer sechsten Ausführungsform.
[0030] **Fig.** 10 zeigt schematisch einen VLIW-Prozessor in einer siebten Ausführungsform.
[0031] **Fig.** 11 zeigt einen schematischen Aufbau des VLIW-Prozessors in der siebten Ausführungsform.

BESCHREIBUNG DER BEVORZUGTEN AUSFÜHRUNGSFORMEN

Erste Ausführungsform

[0032] Ein VLIW-Prozessor in einer ersten Ausführungsform der vorliegenden Erfindung wird im folgenden beschrieben. In dieser Ausführungsform benutzte Begriffe werden zuerst beschrieben. Ein "Hauptzyklus" bezieht sich auf einen Zyklus, wenn eine Instruktion ausgegeben ist, das heißt ein Zyklus, in dem Schrittnummern nur durch Nummern in einer Ausführungssequenz bezeichnet sind. Ein "Unterzyklus" bezieht sich auf einen Zyklus, der einen Zyklus nach dem Hauptzyklus ausgeführt wird, das heißt, ein Zyklus, in dem die Schrittnummern durch Nummern mit Apostroph (') in einer Ausführungssequenz bezeichnet sind. Der Unterzyklus wird nicht immer nach dem Hauptzyklus ausgeführt.
[0033] Ein "linker Container" bezieht sich auf den linken von Instruktionspuffern, in denen VLIW-Instruktionen angeordnet sind. Ein "linker Instruktionsdecoder" bezieht sich auf einen Decoder, der eine Instruktion decodiert, die in dem linken Container angeordnet ist. Eine "linke Verarbeitungseinheit" bezieht sich auf eine Verarbeitungseinheit, die die Instruktion ausführt, die durch den linken Instruktionsdecoder decodiert ist. Eine "linke Unterinstruktion" (oder "linke Instruktion") gehört zu einer Instruktionssequenz für linke Unterinstruktionen (oder linke Instruktionen), das heißt eine Instruktion, die vorbestimmt ist, daß sie in der linken Verarbeitungseinheit zu verarbeiten ist.
[0034] Ein "rechter Container" bezieht sich auf den rechten von Instruktionspuffern, in denen VLIW-Instruktionen angeordnet sind. Ein "rechter Instruktionsdecoder" bezieht sich auf einen Decoder, der eine Instruktion decodiert, die in dem rechten Container angeordnet ist. Eine "rechte Verarbeitungseinheit" bezieht sich auf eine Verarbeitungseinheit, die die Instruktion ausführt, die von dem rechten Instruktionsdecoder decodiert ist. Eine "rechte Unterinstruktion" (oder "rechte Instruktion") gehört zu einer Instruktionssequenz für rechte Unterinstruktionen (oder rechte Instruktionen), das heißt eine Instruktion, die vorbestimmt ist, daß sie in der rechten Verarbeitungseinheit zu verarbeiten ist.
[0035] **Fig.** 2 zeigt schematisch einen VLIW-Prozessor in dieser Ausführungsform. Der VLIW-Prozessor enthält einen Instruktionsspeicher **1**, einen linken Container **2**, einen rechten Container **3**, einen linken Instruktionsdecoder **4**, einen rechten Instruktionsdecoder **5**, eine linke Verarbeitungseinheit **6** und eine rechte Verarbeitungseinheit **7**. Wie in **Fig.** 2 gezeigt ist, führen der linke und der rechte Container **2** und **3** eine Austauschtätigkeit zum Austauschen ihrer Unterinstruktionen aus.
[0036] Die Verarbeitungsregel von VLIW-Instruktionen wird im folgenden beschrieben.

(1) Wenn eine linke Unterinstruktion **1** zu dem linken Container ausgegeben wird und eine rechte Unterinstruktion **2** zu dem rechten Container ausgegeben wird, werden diese zwei Unterinstruktionen simultan zu entsprechenden Instruktionsdecodern übertragen. In dem Hauptzyklus werden die Unterinstruktionen

durch die Decoder decodiert und simultan durch entsprechende Verarbeitungseinheiten ausgeführt. In diesem Fall wird ein Unterzyklus nicht verursacht.

(2) Wenn eine linke Unterinstruktion **3** zu dem linken Container ausgegeben wird und eine linke Unterinstruktion **4** zu dem rechten Container ausgegeben wird, werden diese zwei Unterinstruktionen simultan zu entsprechenden Instruktionsdecodern übertragen. In dem Hauptzyklus wird die linke Unterinstruktion **3** durch den linken Decoder decodiert und in der linken Verarbeitungseinheit ausgeführt. Weiterhin versagt der rechte Decoder, die linke Unterinstruktion **4** zu decodieren und behandelt sie als eine nop-Instruktion, da sie ungültig ist, und die linke Unterinstruktion **4** wird nicht in der rechten Verarbeitungseinheit ausgeführt. Zu der gleichen Zeit wird das Vorhandensein der Instruktion, die mit der Instruktionssequenz für linke Unterinstruktionen verknüpft ist, in dem rechten Container erfaßt, und die Inhalte des linken und des rechten Containers werden ausgetauscht.

In dem Unterzyklus wird die linke Unterinstruktion **4** durch den linken Decoder decodiert und in der linken Verarbeitungseinheit ausgeführt. Weiterhin versagt der rechte Decoder zum Decodieren der linken Unterinstruktion **3** und behandelt sie als eine nop-Instruktion, da sie ungültig ist, und die linke Unterinstruktion **3** wird nicht in der rechten Verarbeitungseinheit ausgeführt. Das heißt, eine VLIW-Instruktion, die eine linke Unterinstruktion **3** und eine rechte nop-Instruktion enthält, wird in dem Hauptzyklus ausgeführt, und eine VLIW-Instruktion, die eine linke Unterinstruktion **4** und eine rechte nop-Instruktion enthält, wird in dem Unterzyklus ausgeführt.

(3) Wenn eine rechte Unterinstruktion **5** zu dem linken Container ausgegeben wird und eine rechte Unterinstruktion **6** zu dem rechten Container ausgegeben wird, werden diese zwei Unterinstruktionen simultan zu entsprechenden Instruktionsdecodern übertragen. In dem Hauptzyklus wird die rechte Unterinstruktion **6** durch den rechten Decoder decodiert und in der rechten Verarbeitungseinheit ausgeführt. Weiterhin versagt der linke Decoder zum Decodieren der rechten Unterinstruktion **5** und behandelt sie als eine nop-Instruktion, da sie ungültig ist, und die rechte Unterinstruktion **5** wird nicht in der linken Verarbeitungseinheit verarbeitet. Zu der gleichen Zeit wird das Vorhandensein der Instruktion, die mit der Instruktionssequenz für die rechten Unterinstruktionen verknüpft ist, in dem linken Container erfaßt, und die Inhalte des linken und des rechten Containers werden ausgetauscht.

In dem Unterzyklus wird die rechte Unterinstruktion **5** durch den rechten Decoder decodiert und in der rechten Verarbeitungseinheit ausgeführt. Weiterhin versagt der linke Decoder zum Decodieren der rechten Unterinstruktion **6** und behandelt sie als eine nop-Instruktion, da sie ungültig ist, und die rechte Unterinstruktion **6** wird nicht in der linken Verarbeitungseinheit verarbeitet. Das heißt, eine VLIW-Instruktion, die eine rechte Unterinstruktion **6** und eine linke nop-Instruktion enthält, wird in dem Hauptzyklus ausgeführt, und eine VLIW-Instruktion, die eine rechte Unterinstruktion **5** und eine linke nop-Instruktion enthält, wird in dem Unterzyklus ausgeführt.

(4) Wenn eine rechte Unterinstruktion **7** zu dem linken Container ausgegeben wird und eine linke Unterinstruktion **8** zu dem rechten Container ausgegeben wird, werden diese zwei Unterinstruktionen simultan zu den entsprechenden Instruktionsdecodern übertragen. In dem Hauptzyklus versagt der linke Decoder zum Decodieren der rechten Unterinstruktion **7** und behandelt sie als eine nop-Instruktion, da sie ungültig ist, und die rechte Unterinstruktion **7** wird nicht in der linken Verarbeitungseinheit verarbeitet. Weiterhin versagt der rechte Decoder zum Decodieren der linken Unterinstruktion **8** und behandelt sie als eine nop-Instruktion, da sie ungültig ist, und die linke Unterinstruktion **8** wird nicht in der rechten Verarbeitungseinheit verarbeitet. Zu der gleichen Zeit wird das Vorhandensein der Instruktion, die mit der Instruktionssequenz für die rechte Unterinstruktionen verknüpft ist, in dem linken Container erfaßt, und das Vorhandensein der Instruktion, die mit der Instruktionssequenz für die linken Unterinstruktionen verknüpft ist, wird in dem rechten Container erfaßt. Somit werden die Inhalte des linken und des rechten Containers ausgetauscht.

In dem Unterzyklus wird die rechte Unterinstruktion **7** durch den rechten Decoder decodiert und in der rechten Verarbeitungseinheit ausgeführt. Weiterhin wird die linke Unterinstruktion **8** durch den linken Decoder decodiert und in der linken Verarbeitungseinheit ausgeführt. Das heißt, eine VLIW-Instruktion, die eine rechte nop-Instruktion und eine linke nop-Instruktion enthält, wird in dem Hauptzyklus ausgeführt, und eine VLIW-Instruktion, die eine rechte Unterinstruktion **7** und eine linke Unterinstruktion **8** enthält, wird in dem Unterzyklus ausgeführt.

[0037] **Fig.** 3 ist ein Blockschaltbild, das einen schematischen Aufbau eds VLIW-Prozessors in dieser Ausführungsform zeigt. Der VLIW-Prozessor enthält einen Instruktionsspeicher **1**, einen linken Container **2**, einen rechten Container **3**, einen linken Instruktionsdecoder **4**, einen rechten Instruktionsdecoder **5**, eine linke Verarbeitungseinheit **6**, eine rechte Verarbeitungseinheit **7**, einen Austauschabschnitt **8**, der die Inhalte des linken und des rechten Containers **2** und **3** austauscht, einen Programmzähler **9**, Verzweigungszielregister (BTR) **10**, Wähler **11** und **12**, einen Inkrementierer **13** und ein Steuersystem **14**, das den gesamten Prozessor steuert. Der Instruktionsspeicher enthält eine Mehrzahl von Speicherzellen **15** und einen Adreßdecoder **16**, der die Inhalte des Programmzählers zum Auswählen einer Speicherzelle **15** decodiert. Der Austauschabschnitt **18** ent-

hält ein Register **17**, das die Inhalte des linken Containers **2** hält, ein Register **19**, das die Inhalte des rechten Containers **3** hält, und Wähler **19** und **20**.

[0038] Der linke Container **2**, der rechte Container **3**, der Programmzähler **9** und das Verzweigungszielregister **10** halten Eingangssignale an einem Anstieg eines Taktsignales, das nicht gezeigt ist. Das Register **17** und **18** halten Eingangssignale an dem Abfall des Taktsignales. Wenn der linke und der rechte Container **2** und **3** Unterinstruktionen von dem Instruktionsspeicher **1** in einem Zyklus halten, können sie daher die Unterinstruktionen in dem nächsten Zyklus austauschen und empfangen.

[0039] Für die Hauptzyklussteuerung steuert das Steuersystem **14** so, daß der Wähler **11** die Ausgabe des Inkrementierers **13** auswählt, der Wähler **12** wählt die Ausgabe des Wählers **11**, und die Wähler **19** und **20** wählen die Ausgaben des Instruktionsspeichers **1**. Für die Unterzyklusverarbeitung steuert das Steuersystem **14** so, daß der Wähler **11** die Ausgaben des Programmzählers **9** auswählt, der Wähler **12** wählt die Ausgabe des Wählers **11**, und die Wähler **19** und **20** wählen die Ausgaben der Register **18** bzw. **17**.

[0040] Bei der Schreibtätigkeit des in **Fig.** 3 gezeigten Programmzählers **9** wird er so gesteuert, daß der Ausgabe des Verzweigungszielregisters **10** Priorität über die Ausgabe des Inkrementierers **13** gegeben wird.

[0041] Bei der Schreibtätigkeit in dem in **Fig.** 3 gezeigten Programmzähler **9** wird er so gesteuert, daß der Ausgabe des Verzweigungszielregisters **10** Priorität über die Ausgabe des Inkrementierers **13** gegeben wird. Dieses ist so, da Programmierer Instruktionen codierten, während eine nop-Instruktion eingefügt ist, wie in dem Codebeispiel **1**, das im Stand der Technik beschrieben wurde, und daher, wenn Verzweigung vor dem Zyklus der Codenummer **3** gefunden wird, muß der Verzweigung Priorität gegeben werden, ohne daß "add R1, R2 → R3" als ein Unterzyklus ausgeführt wird.

[0042] Es sei als ein Beispiel angenommen, daß ein Programmierer das Programm von dem Codebeispiel 4 unten hergestellt hat.

Codebeispiel 4:

Codenummer	Instruktionssequenz für linke Unterinstruktionen	Instruktionssequenz für rechte Unterinstruktionen
1	store R0 -> mem (200)	:nop
2	load mem (100) -> R1	:nop
3	shift R4, R5 -> R6	:add R1, R2 -> R3.

[0043] Dieses Programm des Codebeispiels 4 wird in das Codebeispiel 5 unten durch einen Codekompressionsalgorithmus umgewandelt, der unten beschrieben wird.

Codebeispiel 5:

Codenummer	Instruktionssequenz für linke Unterinstruktionen	Instruktionssequenz für rechte Unterinstruktionen
1	store R0 -> mem (200)	:load mem (100) -> R1
2	shift R4, R5 -> R6	:add R1, R2 -> R3.

[0044] Wenn der VLIW-Prozessor in dieser Ausführungsform das Programm des Codebeispiels 5 ausführt, ist die Ausführungssequenz wie folgt.

Ausführungssequenzbeispiel 2:

Schrittnummer	Instruktionssequenz für linke Unterinstruktionen	Instruktionssequenz für rechte Unterinstruktionen
1	store R0 -> mem (200)	:nop [load mem (100) -> R1]
1'	load mem (100) -> R1	:nop [store R0 -> mem (200)]
2	shift R4, R5 -> R6	:add R1, R2 -> R3.

[0045] Hier bedeutet "nop [load mem (100) → R1]", daß, wenn eine linke Unterinstruktion "load mem (100) → R1" zu der Instruktionssequenz für die rechten Unterinstruktionen ausgegeben wird, die Instruktion als eine nop-Instruktion verarbeitet wird. Hier im folgenden bedeutet die Beschreibung von "nop [ ]", daß, wenn eine rechte Unterinstruktion zu der Instruktionssequenz für linke Unterinstruktionen ausgegeben wird oder wenn eine linke Unterinstruktion zu der Instruktionssequenz für rechte Unterinstruktionen ausgegeben wird, die Instruktion als nop-Instruktion verarbeitet wird.

[0046] Wie oben beschrieben wurde, werden gemäß dem VLIW-Prozessor in dieser Ausführungsform, wenn eine rechte Unterinstruktion zu der Instruktionssequenz für linke Unterinstruktionen ausgegeben wird oder wenn eine linke Unterinstruktion zu der Instruktionssequenz für rechte Unterinstruktionen ausgegeben wird, die Unterinstruktionen ausgetauscht und in dem Unterzyklus verarbeitet, und daher kann eine Code-kompri-

miertes Programm in solch einer Ausführungssequenz- verarbeitet werden, daß es vor der Kompression genommen wird.

Zweite Ausführungsform

[0047] **Fig.** 4 ist ein Blockschaltbild, das einen schematischen Aufbau eines VLIW-Prozessors in einer zweiten Ausführungsform der vorliegenden Erfindung zeigt. Der einzige Unterschied von dem in **Fig.** 3 gezeigten VLIW-Prozessor in der ersten Ausführungsform ist der Aufbau des Austauschabschnittes **8**. Daher wird die detaillierte Beschreibung überlappender Aufbauten und Funktionen nicht wiederholt. In dieser Ausführungsform wird das Bezugszeichen **8'** für den Austauschabschnitt genommen.

[0048] Der Austauschabschnitt **8'** enthält Wähler **21** und **22**. In dem Hauptzyklus steuert das Steuersystem **14** den Wähler 21 zum Auswählen einer Unterinstruktion, die zu der Instruktionssequenz für die linken Unterinstruktionen gehört, von dem Instruktionsspeicher **1**, und den Wähler 22 zum Auswählen einer Unterinstruktion, die zu der Instruktionssequenz für die rechten Unterinstruktionen gehört, von dem Instruktionsspeicher **1**. In dem Unterzyklus steuert das Steuersystem **14** den Wähler zum Auswählen einer Unterinstruktion, die zu der Instruktionssequenz für die rechten Unterinstruktionen gehört, von dem Instruktionsspeicher **1**, und den Wähler **22** zum Auswählen einer Unterinstruktion, die zu der Instruktionssequenz für linke Unterinstruktionen gehört, von dem Instruktionsspeicher **1**.

[0049] Der in **Fig.** 3 gezeigte VLIW-Prozessor in der ersten Ausführungsform erfaßt einen Übergang zu dem Unterzyklus an der decodierten Stufe und bewirkt, daß die Unterinstruktion in dem linken und rechten Container **2** und **3** gehalten wird, so daß sie zurückgeführt und ausgetauscht werden. An der Decodierstufe ist der Wert des Programmzählers **9** bereits erhöht worden, und eine aus dem Instruktionsspeicher **1** gelesene VLIW-Instruktion ist bereits für die Instruktion der nächsten Stufe aktualisiert. Somit sind Register **17** und **18** notwendig zum Zurückführen und Austauschen von Unterinstruktionen in dem linken und rechten Container **2** und **3**. Da der VLIW-Prozessor in dieser Ausführungsform nicht die Erhöhung des Wertes des Programmzählers **9** durch Erfassen eines Überganges auf den Unterzyklus an der Befehlshohlstufe steuern kann, gibt es keine Notwendigkeit zum Zurückführen, wodurch die Notwendigkeit für Schaltungen entsprechend den Registern **17** und **18** beseitigt wird. In einem Prozessor, in dem ein Instruktionshohlzyklus ein kritischer Pfad ist, kann die Taktrate, d. h. die Verarbeitungsgeschwindigkeit gesenkt werden.

[0050] Wie oben beschrieben wurde kann gemäß dem VLIW-Prozessor in dieser Ausführungsform ein einfacher Schaltungsaufbau und eine Kostenverringerung erzielt werden im Vergleich mit dem VLIW-Prozessor in der ersten Ausführungsform.

Dritte Ausführungsform

[0051] Ein VLIW-Prozessor in einer dritten Ausführungsform der vorliegenden Erfindung ist der gleiche wie die VLIW-Prozessoraufbauten in der ersten und zweiten Ausführungsform. Der einzige Unterschied ist der, daß die nop-Instruktion einer linken Unterinstruktion und die nop-Instruktion einer rechten Unterinstruktion in dieser Ausführungsform unterstützt werden. Daher wird eine detaillierte Beschreibung der überlappenden Aufbauten und Funktionen nicht wiederholt.

[0052] Hier ist die nop-Instruktion für eine linke Unterinstruktion eine Lnop-Instruktion, und die nop-Instruktion einer rechten Unterinstruktion ist eine Rnop-Instruktion. Es sei als ein Beispiel angenommen, daß ein komprimierter Code wie in dem Codebeispiel 6 unten beschrieben ist.

Codebeispiel 6:

```
Codenummer     Instruktionssequenz für        Instruktionssequenz für
               linke Unterinstruktionen       rechte Unterinstruktionen
     1         Lnop                           :Lnop.
```

[0053] Wenn der VLIW-Prozessor in dieser Ausführungsform das Programm des Codebeispiels 6 ausführt, wird das Programm ausgeführt, wie in dem Ausführungssequenzbeispiel 3 unten gezeigt ist.

Ausführungssequenzbeispiel 3:

```
Schrittnummer  Instruktionssequenz für        Instruktionssequenz für
               linke Unterinstruktionen       rechte Unterinstruktionen
     1         Lnop                           :nop [Lnop]
     1'        Lnop                           :nop [Lnop].
```

[0054] Wie oben beschrieben wurde, kann gemäß dem VLIW-Prozessor in dieser Ausführungsform die Aus-

führung entsprechend zu zwei Zyklen von nop-Instruktionen möglich gemacht werden durch eine VLIW-Instruktion, und daher kann Codekompression effektiv durchgeführt werden.

Vierte Ausführungsform

[0055] Eine vierte Ausführungsform der vorliegenden Erfindung bezieht sich auf eine Codekompressionseinrichtung (Verknüpfer), der die oben beschriebene Codekompression durchführt. **Fig.** 5 ist ein Blockschaltbild, das einen schematischen Aufbau der Codekompressionseinrichtung zeigt. Die Codekompressionseinrichtung enthält einen Speicherabschnitt **31** für unkomprimierten Sourcecode, der einen unkomprimierten Sourcecode speichern kann, einen Codeextraktionsabschnitt **32**, der zwei aufeinanderfolgende Zeilen von VLIW-Instruktionen von dem Sourcecode extrahiert, einen Codekompressionsabschnitt **33**, der den von dem Codeextraktionsabschnitt **32** extrahierten Code komprimiert, einen Codeausgabeabschnitt **34**, der den komprimierten Code ausgibt, und einen Speicherabschnitt **35** für komprimierten Sourcecode, der den von dem Codeausgabeabschnitt **34** ausgegebenen Code speichert.
[0056] **Fig.** 6 ist ein Flußdiagramm zum Beschreiben der Verarbeitungsprozedur der Codekompression. Zuerst liest der Codeextraktionsabschnitt **32** einen zu komprimierenden Sourcecode aus dem Speicherabschnitt **31** für unkomprimierten Sourcecode (S1), und "1" wird auf eine Variable i gesetzt (S2). Dann extrahiert der Codeextraktionsabschnitt **32** Codes des i-ten Codes und die (i + 1)te Zeile von dem Sourcecode (S3).
[0057] Danach bestimmt der Codekompressionsabschnitt **33**, ob ein Etikett nicht zu dem Code der (i + 1)ten Reihe addiert worden ist, die von dem Codeextraktionsabschnitt **32** extrahiert worden ist, und ob mindestens zwei Unterinstruktionen des Codes der i-ten und der (i + 1)ten Zeile nop-Instruktionen sind (S4). Wenn ein Etikett zu dem Code der (i + 1)ten Zeile addiert worden ist oder wenn es weniger als zwei Unterinstruktionen gibt, die nop-Instruktionen sind, werden die Codes der i-ten und der (i + 1)ten Zeile eingegeben (S4, NEIN), der Schritt S11 wird betreten.
[0058] Wenn ein Etikett nicht zu dem Code der (i + 1)ten Zeile addiert worden ist und mindestens zwei Unterinstruktionen des Codes der i-ten und der (i + 1)ten Zeile nop-Instruktionen sind (**54**, JA), bestimmt der Codekompressionsabschnitt **33**, ob der Code der i-ten Zeile eine Kombination einer linken Unterinstruktion und einer nop-Instruktion ist und der Code der (i + 1)ten Zeile eine Kombination einer linken Unterinstruktion und einer nop-Instruktion ist (S5). Wenn die i-te Zeile nicht eine Kombination einer linken Unterinstruktion und einer nop-Instruktion ist oder wenn die (i + 1)te Zeile nicht eine Kombination einer linken Unterinstruktion und einer nop-Instruktion ist (S5, NEIN), wird Schritt S7 betreten.
[0059] Wenn die i-te Zeile eine Kombination einer linken Unterinstruktion und einer nop-Instruktion ist und die (i + 1)te Zeile eine Kombination einer linken Unterinstruktion und einer nop-Instruktion ist (S5, JA), komprimiert der Codekompressionsabschnitt **33** Instruktionen so, daß die linke Unterinstruktion der i-ten Zeile an den linken Container **2** eingegeben wird und die linke Unterinstruktion der (i + 1)ten Zeile an den rechten Container **3** eingegeben wird (S6). Dann speichert der Codeausgabeabschnitt **34** den komprimierten Code in den Speicherabschnitt **35** des komprimierten Sourcecodes.
[0060] In Schritt S7 bestimmt der Codekompressionsabschnitt **33**, ob die i-te Zeile eine Kombination einer nop-Instruktion und einer rechten Unterinstruktion ist und die (i + 1)te Zeile eine Kombination einer nop-Instruktion und einer rechten Unterinstruktion ist. Wenn die i-te Zeile nicht eine Kombination einer nop-Instruktion und einer rechten Unterinstruktion ist oder wenn die (i + 1)te Zeile nicht eine Kombination einer nop-Instruktion und einer rechten Unterinstruktion ist (S7, NEIN), wird Schritt S9 betreten.
[0061] Wenn die i-te Zeile eine Kombination einer nop-Instruktion und einer rechten Unterinstruktion ist und die (i + 1)te Zeile eine Kombination einer nop-Instruktion und einer rechten Unterinstruktion ist (S7, JA), komprimiert der Codekompressionsabschnitt **33** Instruktionen so, daß die rechte Unterinstruktion der (i + 1)ten Zeile in den linken Container **2** eingegeben wird und die rechte Unterinstruktion der i-ten Zeile in den rechten Container **3** eingegeben wird (S8). Dann speichert der Codeausgabeabschnitt **34** den komprimierten Code in den Speicherabschnitt **35** des komprimierten Sourcecodes.
[0062] In Schritt S9 bestimmt der Codekompressionsabschnitt **33**, ob die i-te Zeile eine Kombination einer nop-Instruktion und einer nop-Instruktion ist und die (i + 1)te Zeile eine Kombination einer linken Unterinstruktion und einer rechten Unterinstruktion ist. Wenn die i-te Zeile nicht eine Kombination von nop-Instruktionen ist oder wenn die (i + 1)te Zeile nicht eine Kombination einer linken und einer rechten Unterinstruktion ist (S9, NEIN), wird Schritt S11 betreten.
[0063] Wenn die i-te Zeile eine Kombination von nop-Instruktionen ist und die (i + 1)te Zeile eine Kombination einer linken und einer rechten Unterinstruktion ist (S9, JA), komprimiert der Codekompressionsabschnitt **38** Instruktionen so, daß die rechte Unterinstruktion der (i + 1)ten Zeile in den linken Container **1** eingegeben wird und die linke Unterinstruktion der (i + 1)ten Zeile in den rechten Container **3** eingegeben wird (S10). Dann speichert der Codeausgabeabschnitt **34** den komprimierten Code in den Speicherabschnitt **35** des komprimierten Sourcecodes.
[0064] In Schritt S11 wird eine Bestimmung gemacht, ob die Kompressionsverarbeitung für alle Zeilen des

Sourcecodes durchgeführt worden ist (S11). Wenn es irgendeine Zeile gibt, die nicht komprimiert worden ist (S11, NEIN), wird (i + 1) für die Variable i gesetzt, und der Vorgang kehrt zu Schritt S3 zurück. Wenn die Kompression für alle Zeilen durchgeführt worden ist (S11, JA), ist die Kompressionsverarbeitung beendet.

[0065] **Fig.** 7 zeigt ein Beispiel der externen Ansicht der Codekompressionseinrichtung gemäß der vorliegenden Erfindung. Die Codekompressionseinrichtung enthält einen Computer **41**, eine Graphikanzeigeeinrichtung **42**, eine Magnetbandeinrichtung **43**, in die ein Magnetband **44** eingeführt ist, eine Tastatur **45**, eine Maus **46**, eine CD-ROM-(Nurlesespeicher einer Kompaktdisc)Einrichtung **47**, in die ein CD-ROM **48** eingeführt ist, und ein Kommunikationsmodem **49**. Ein Codekompressionsprogramm wird durch ein Speichermedium wie ein Magnetband **44** oder ein CD-ROM **48** geliefert. Das Codekompressionsprogramm wird durch den Computer **41** ausgeführt, und ein Bediener komprimiert Sourcecodes durch Betätigen der Tastatur **45** oder der Maus **46**, wobei auf die Graphikanzeigeeinrichtung **42** Bezug genommen wird. Das Codekompressionsprogramm kann zu dem Computer **41** von einem anderen Computer über Kommunikationsleitungen und das Kommunikationsmodem **49** geliefert werden.

[0066] **Fig.** 8 ist ein Blockschaltbild, das ein Beispiel des Aufbaus der Codekompressionseinrichtung gemäß der vorliegenden Erfindung zeigt. Der in **Fig.** 7 gezeigte Computer **41** enthält eine CPU **50**, einen ROM (Nurlesespeicher) **51**, einen RAM (Direktzugriffsspeicher) **52** und eine Festplatte **53**. Die CPU **50** führt Verarbeitung durch Übertragen und Empfangen zu und von der Magnetbandeinrichtung **43**, der Tastatur **45**, der Maus **46**, der CD-ROM-Einrichtung **47**, dem Kommunikationsmodem **49**, dem ROM **51**, dem RAM **52** oder der Festplatte **53** durch. Ein auf dem Magnetband **44** oder dem CD-ROM **48** gespeichertes Codekompressionsprogramm wird einmal auf der Festplatte **53** über die Magnetbandeinrichtung **43** oder die CD-ROM-Einrichtung **47** durch die CPU **50** gespeichert. Die CPU **50** komprimiert einen Sourcecode durch Laden des Codekompressionsprogrammes wenn notwendig in den RAM 52 zum Ausführen.

[0067] Wie oben beschrieben wurde kann gemäß der Codekompressionseinrichtung in dieser Ausführungsform ein Sourcecode in einem Format komprimiert werden, das den Betrieb des in der ersten Ausführungsform beschriebenen VLIW-Prozessors ermöglicht. Somit kann die Kapazität eines Instruktionsspeichers zum Beispiel effektiv benutzt werden.

Fünfte Ausführungsform

[0068] Eine Codekompressionseinrichtung in einer fünften Ausführungsform der vorliegenden Erfindung ist ähnlich zu der unter Bezugnahme auf **Fig.** 5 bis 8 beschriebenen vierten Ausführungsform. Obwohl die Datenabhängigkeit für die Codekompression in der Codekompressionseinrichtung in der vierten Ausführungsform nicht betrachtet worden ist, wird die Datenunabhängigkeit in der Codekompressionseinrichtung in dieser Ausführungsform in Betracht gezogen (in dieser Ausführungsform wird eine Abwicklerfunktion zu der Codekompressionseinrichtung in der vierten Ausführungsform hinzugefügt).

[0069] Es sei als ein Beispiel angenommen, daß das Programm des Codebeispiels 7 unten beschrieben wird.

Codebeispiel 7:

```
Codenummer     Instruktionssequenz für        Instruktionssequenz für
               linke Unterinstruktionen       rechte Unterinstruktionen
     1         store R0 -> mem (200)          :sub R4, R5 -> R6
     2         load mem (100) -> R7           :Rnop
     3         Lnop                           :add R1, R2 -> R3.
```

[0070] Da die in den Codenummern 2 und 3 gezeigten VLIW-Instruktionen keine Datenunabhängigkeit aufweisen, können die Codes wie in dem Codebeispiel 8 unten geändert werden.

Codebeispiel 8:

```
Codenummer     Instruktionssequenz für        Instruktionssequenz für
               linke Unterinstruktionen       rechte Unterinstruktionen
     1         store L0 -> mem (200)          :sub R4, R5 -> R6
     2         Lnop                           :Rnop
     3         load mem (100) -> R7           :add R1, R2 -> R3.
```

[0071] Das in dem Codebeispiel 8 gezeigte Programm kann in ein Codebeispiel 9 unten durch die Codekompressionseinrichtung in der vierten Ausführungsform komprimiert werden.

Codebeispiel 9:

```
Codenummer     Instruktionssequenz für        Instruktionssequenz für
               linke Unterinstruktionen       rechte Unterinstruktionen
     1         store R0 -> mem (200)          :sub R4, R5 -> R6
     2         add R1, R2 -> R3               :load mem (100) -> R7.
```

[0072] Wenn das in dem Codebeispiel 9 gezeigte Programm ausgeführt wird, wird es wie in dem Ausführungssequenzbeispiel 4 unten gezeigt ausgeführt.

Sequenzbeispiel 4:

```
Schrittnummer  Instruktionssequenz für        Instruktionssequenz für
               linke Unterinstruktionen       rechte Unterinstruktionen
     1         store R0 -> mem (200)          :sub R4, R5 -> R6
     2         nop [add R1, R2 -> R3]         :nop [load mem (100) -> R7]
     2'        load mem (100) -> R7           :add R1, R2 -> R3.
```

[0073] Durch Ausführen der Abwicklung, während die Datenunabhängigkeit in Betracht gezogen wird, können die Codes effektiver komprimiert werden. Das folgende Codebeispiel 10 ist ein Beispiel von Codes, die nicht komprimiert werden können.

Codebeispiel 10:

```
Codenummer     Instruktionssequenz für        Instruktionssequenz für
               linke Unterinstruktionen       rechte Unterinstruktionen
     1         store R0 -> mem (200)          :R4, R5 -> R6
     2         load mem (100) -> R1           :Rnop
     3         Lnop                           :add R1, R2 -> R3.
```

[0074] Da die linke Unterinstruktion der Codenummer 2 "load mem (100) → R1" bewirkt, daß Daten in das Register R1 geladen werden, und die rechte Unterinstruktion der Codenummer 3 "add R1, R2 → R3" bewirkt, daß auf Daten des Registers R1 zugegriffen wird, kann die Kompression der gleichen VLIW-Instruktion nicht durchgeführt werden.

[0075] Wie oben beschrieben wurde wird gemäß der Codekompressionseinrichtung in dieser Ausführungsform die Abwicklung ausgeführt, während die Datenunabhängigkeit in Betracht gezogen wird, die Codes können effektiver im Vergleich mit der Codekompressionseinrichtung in der vierten Ausführungsform komprimiert werden.

Sechste Ausführungsform

[0076] **Fig.** 9 zeigt schematisch einen VLIW-Prozessor in einer sechsten Ausführungsform der vorliegenden Erfindung. Der VLIW-Prozessor in dieser Ausführungsform weist den Aufbau auf, der durch Verbinden einer Mehrzahl der VLIW-Prozessoren in der in **Fig.** 2 gezeigten ersten Ausführungsform gebildet ist, das heißt, der VLIW-Prozessor dieser Ausführungsform ist ein 2n-Weg-VLIW-Prozessor. **Fig.** 9 zeigt einen VLIW-Prozessor, wenn n gleich 3 ist.

[0077] Wie oben beschrieben wurde, kann der VLIW-Prozessor in dieser Ausführungsform höhere Parallelität und Verarbeitungsgeschwindigkeit als der VLIW-Prozessor in der ersten Ausführungsform aufweisen.

Siebte Ausführungsform

[0078] **Fig.** 10 zeigt schematisch einen VLIW-Prozessor in einer siebten Ausführungsform der vorliegenden Erfindung. Die VLIW-Prozessoren in der ersten bis sechsten Ausführungsform sind die 2n-Weg-VLIW-Prozessoren, die zwei rechte und linke Unterinstruktionen enthalten, während der VLIW-Prozessor in dieser Ausführungsform auf den VLIW-Verfahren von mindestens drei Wegen basiert, die Instruktionen austauschen durch Verschieben derselben statt Austauschen der Instruktionen durch Austauschen derselben.

[0079] Der VLIW-Prozessor in dieser Ausführungsform enthält einen Instruktionsspeicher **61**, einen ersten Container **62**, einen zweiten Container **63**, einen dritten Container **64**, einen vierten Container **65**, einen ersten Instruktionsdecoder **66**, einen zweiten Instruktionsdecoder **67**, einen dritten Instruktionsdecoder **68**, einen vierten Instruktionsdecoder **69**, eine erste Verarbeitungseinheit **70**, eine zweite Verarbeitungseinheit **71**, eine dritte Verarbeitungseinheit **72** und eine vierte Verarbeitungseinheit **73**. Zum Verarbeiten werden Unterinstruktionen

von dem ersten Container **62** zu dem vierten Container **65**, von dem vierten Container **65** zu dem dritten Container **64**, von dem dritten Container **64** zu dem zweiten Container **63** und von dem zweiten Container **63** zu dem ersten Container **62** verschoben.

[0080] **Fig.** 11 ist ein Blockschaltbild, das einen speziellen Aufbau des VLIW-Prozessors in dieser Ausführungsform zeigt. Der VLIW-Prozessor enthält einen Instruktionsspeicher **61**, einen ersten Container **62**, einen zweiten Container **63**, einen dritten Container **64**, einen vierten Container **65**, einen ersten Instruktionsdecoder **66**, einen zweiten Instruktionsdecoder **67**, einen dritten Instruktionsdecoder **68**, einen vierten Instruktionsdecoder **69**, eine erste Verarbeitungseinheit **70**, eine zweite Verarbeitungseinheit **71**, eine dritte Verarbeitungseinheit **72**, eine vierte Verarbeitungseinheit **73**, einen Austauschabschnitt **74**, der die Inhalte des ersten bis vierten Containers **62** bis **65** durch serielles Verschieben der Inhalte austauscht, einen Programmzähler **75**, ein Verzweigungszielregister BTR **76**, Wähler **77** und **78**, einen Inkrementierer **79** und ein Steuersystem **80**, das den gesamten Prozessor steuert. Der Instruktionsspeicher **61** enthält eine Mehrzahl von Speicherzellen **81** und einen Adreßdecoder **82**, der die Inhalte des Programmzählers **75** decodiert zum Auswählen der Speicherzelle **81**. Der Austauschabschnitt **74** enthält ein Register **83**, das die Inhalte des zweiten Containers **63** hält, ein Register **84**, das die Inhalte des dritten Containers **64** hält, ein Register **85**, das die Inhalte des vierten Containers **65** hält, ein Register **86**, das die Inhalte des ersten Containers **62** hält, und Wähler **87** bis **90**.

[0081] Der erste bis vierte Container **62** bis **65**, der Programmzähler **75** und das Verzweigungszielregister **76** halten Eingangssignale an dem Anstieg eines Taktsignales, nicht gezeigt. Die Register **83** bis **86** halten Eingangssignale an einem Abfall des Taktsignales. Wenn der erste bis vierte Container **62** bis **65** Unterinstruktionen von dem Instruktionsspeicher **1** in einem Zyklus halten, können sie daher die Unterinstruktionen verschieben und an die entsprechenden Container in dem nächsten Zyklus eingeben.

[0082] Zur Hauptzyklusverarbeitung steuert das Steuersystem **80** so, daß der Wähler **77** die Ausgabe des Inkrementierers **79** auswählt, der Wähler **78** die Ausgabe des Wählers **77** auswählt, und die Wähler **87** bis **90** die Ausgaben des Instruktionsspeichers **1** auswählen. Für Unterzyklusverarbeitung steuert das Steuersystem **80** so, daß der Wähler **77** die Ausgabe des Programmzählers **75** auswählt, der Wähler **78** die Ausgabe des Wählers **77** auswählt und die Wähler **87** bis **90** die Ausgaben der Register **83** bis **86** auswählen.

[0083] Es wird als ein Beispiel angenommen, daß ein Programmierer das Programm des Codebeispiels 11 unten hergestellt hat.

[0084]

Codebeispiel 11:

Codenummer	Instruktionssequenz für erste Unterinstruktionen	Instruktionssequenz für zweite Unterinstruktionen	Instruktionssequenz für dritte Unterinstruktionen	Instruktionssequenz für vierte Unterinstruktionen
1	erste Unterinstruktion 1	: nop	: nop	: nop
2	erste Unterinstruktion 2	: nop	: nop	: nop
3	erste Unterinstruktion 3	: nop	: nop	: nop
4	erste Unterinstruktion 4	: nop	: nop	: nop
5	erste Unterinstruktion 5	: zweite Unter-Instruktion 6	: nop	: vierte Unterinstruktion 7

[0085] Wenn das Programm des Codebeispiels 11 komprimiert wird, wird es wie in dem Codebeispiel 12 unten beschrieben.

Codebeispiel 12:

Codenummer	Instruktionssequenz für erste Unterinstruktionen	Instruktionssequenz für zweite Unterinstruktionen	Instruktionssequenz für dritte Unterinstruktionen	Instruktionssequenz für vierte Unterinstruktionen
1	erste Unterinstruktion 1	:erste Unterinstruktion 2	:erste Unterinstruktion 3	:erste Unterinstruktion 4
2	erste Unterinstruktion 5	:zweite Unterinstruktion 6	: nop	:vierte Unterinstruktion 7

[0086] Wenn das in dem Codebeispiel 12 gezeigte Programm durch den VLIW-Prozessor in dieser Ausführungsform ausgeführt wird, wird es als ein Ausführungssequenzbeispiel 5 unten verarbeitet.

Ausführungssequenzbeispiel 5:

Schritt-nummer	Instruktions-sequenz für erste Unter-instruktionen	Instruktions-sequenz für zweite Unter-instruktionen	Instruktions-sequenz für dritte Unter-instruktionen	Instruktions-sequenz für vierte Unter-instruktionen
1	erste Unter-instruktion 1	:nop [erste Unterin-struktion 2]	: nop [erste Unterinstruk-tion 3]	: nop [erste Unterinstruk-tion 4]
1'	erste Unter-instruktion 2	:nop [erste Unterinstruk-tion 3]	: nop [erste Unterinstruk-tion 4]	: nop [erste Unterinstruk-tion 1]
1"	erste Unter-instruktion 3	: nop [erste Unterinstruk-tion 4]	: nop [erste Unterinstruk-tion 1]	: nop [erste Unterinstruk-tion 2]
1'''	erste Unter-instruktion 4	: nop [erste Unterinstruk-tion 1]	: nop [erste Unterinstruk-tion 2]	: nop [erste Unterinstruk-tion 3]
2	erste Unter-instruktion 5	: zweite Unter-instruktion 6	: nop	: vierte Unter-instruktion 7.

[0087] Wie oben beschrieben wurde kann gemäß dem VLIW-Prozessor in dieser Ausführungsform das VLIW-Verfahren von mindestens 3-Wegen angewendet werden, und Codes können effektiver im Vergleich mit dem 2n-Weg-VLIW-Prozessor komprimiert werden.

[0088] Obwohl die vorliegende Erfindung im Detail beschrieben und dargestellt worden ist, ist klar zu verstehen, daß dieses nur als Illustration und Beispiel getan ist, und es kann nicht als Begrenzung genommen werden, der Umfang der vorliegenden Erfindung wird nur durch den Inhalt der beigefügten Ansprüche begrenzt.

Patentansprüche

1. VLIW-Prozessor mit:

einer Mehrzahl von Containern (**2**, **3**, **62**–**65**), die Instruktionen halten;

einer Mehrzahl von Decodern (**4**, **5**, **66**–**69**), wobei einer der Decoder mit jedem der Mehrzahl von Containern (**2**, **3**, **62**–**65**) verbunden ist und die von seinem zugehörigen der Mehrzahl von Containern (**2**, **3**, **62**–**65**) ausgegebenen Instruktionen decodieren; und

einer Mehrzahl von Verarbeitungseinheiten (**6**, **7**, **70**–**73**), wobei eine der Verarbeitungseinheiten mit jedem der Mehrzahl von Decodern (**4**, **5**, **66**–**69**) verbunden ist und die von seinem zugehörigen der Mehrzahl von Decodern (**6**, **7**, **70**–**73**) decodierten Instruktionen ausführt;

gekennzeichnet durch

eine Austauscheinheit (, 8', 74), die den in einem der Mehrzahl von Containern (**2**, **3**, **62**–**65**) gehaltenen Inhalt mit dem in einem anderen der Mehrzahl von Containern (**2**, **3**, **62**–**65**) gehaltenen Inhalt aktualisiert und den in einem anderen der Mehrzahl von Containern (**2**, **3**, **62**–**65**) gehaltenen Inhalt mit dem in dem einen der Mehrzahl von Containern (**2**, **3**, **62**–**65**) gehaltenen Inhalt aktualisiert.

2. VLIW-Prozessor nach Anspruch 1,

bei dem die Mehrzahl von Containern (**2**, **3**, **62**–**65**) einen linken Container (**2**), der eine erste Instruktion in einer VLIW-Instruktion hält, und einen rechten Container (**3**), der eine zweite Instruktion in der VLIW-Instruktion hält, enthält,

die Mehrzahl von Decodern (**4**, **5**, **66**–**69**) einen linken Instruktionsdecoder (**4**), der die in dem linken Container (**2**) gehaltene erste Instruktion decodiert, und einen rechten Decoder (**5**), der die in dem rechten Container (**3**) gehaltene zweite Instruktion decodiert, enthält,

die Mehrzahl von Verarbeitungseinheiten (**6**, **7**, **70**–**73**) eine linke Verarbeitungseinheit (**6**), die durch den linken Instruktionsdecoder (**4**) decodierte Instruktion ausführt, und eine rechte Verarbeitungseinheit (**7**), die die durch den rechten Instruktionsdecoder (**5**) decodierte Instruktion ausführt, enthält,

der linke Instruktionsdecoder (**4**) eine linke Instruktion decodiert und die Instruktion an die linke Verarbeitungseinheit (**6**) ausgibt, wenn die erste Instruktion die linke Instruktion ist, und eine rechte Instruktion als eine nop-Instruktion ausgibt, wenn die erste Instruktion die rechte Instruktion ist, und

der rechte Instruktionsdecoder (**5**) eine rechte Instruktion decodiert und die Instruktion an die rechte Verarbei-

tungseinheit (**7**) ausgibt, wenn die zweite Instruktion die rechte Instruktion ist, und eine linke Instruktion als eine nop-Instruktion ausgibt, wenn die zweite Instruktion die linke Instruktion ist.

3. VLIW-Prozessor nach Anspruch 2, weiter mit:
einem Programmzähler (**9**–**13**), der eine Adresse zum Holen einer VLIW-Instruktion ausgibt,
worin in mindestens einem der Fälle, wenn die erste Instruktion eine rechte Instruktion ist und wenn die zweite Instruktion eine linke Instruktion ist, der linke Instruktionsdecoder (**4**) die erste Instruktion decodiert und der rechte Instruktionsdecoder (**5**) die zweite Instruktion decodiert, in einem Hauptzyklus,
in einem Unterzyklus der linke Instruktionsdecoder (**4**) die zweite Instruktion decodiert, die durch den Austauschabschnitt (**8**) ausgetauscht ist, und der rechte Instruktionsdecoder (**5**) die erste Instruktion decodiert, die durch den Austauschabschnitt (**8**) ausgetauscht ist, und
der Programmzähler (**9**–**13**) einen Programmzählerwert während der Hauptzyklusverarbeitung inkrementiert und den Programmzählerwert unverändert während der Unterzyklusverarbeitung hält.

4. VLIW-Prozessor nach Anspruch 3,
bei dem der Programmzähler (**9**–**13**) ein Verzweigungszielregister (**10**) enthält, das eine Verzweigungszieladresse hält, und
der Programmzähler (**9**–**13**) bevorzugt einen Wert des Verzweigungszählregisters (**10**) ausgibt, wenn eine Verzweigung vor dem Unterzyklus gefunden wird.

5. VLIW-Prozessor nach Anspruch 3,
bei dem der Austauschabschnitt (**8**) enthält
einen ersten Wähler (**19**), der die erste Instruktion in dem Hauptzyklus und die zweite Instruktion in dem Unterzyklus auswählt und die Instruktionen zu dem linken Container (**2**) ausgibt, und
einen zweiten Wähler (**20**), der die zweite Instruktion in dem Hauptzyklus und die erste Instruktion in dem Unterzyklus auswählt und die Instruktionen zu dem rechten Container (**3**) ausgibt.

6. VLIW-Prozessor nach Anspruch 2,
bei dem der Austauschabschnitt (**8**) enthält
ein erstes Register (**17**), das die erste Instruktion hält, die von dem linken Container (**2**) gehalten wird, und die Instruktion zu dem rechten Container (**3**) ausgibt und
ein zweites Register (**18**), das die zweite Instruktion hält, die von dem rechten Container (**3**) gehalten wird, und die Instruktion zu dem linken Container (**2**) ausgibt.

7. VLIW-Prozessor nach Anspruch 1, bei dem der Austauschabschnitt (**74**) sequentiell eine Mehrzahl von Instruktionen, die in der Mehrzahl von Containern (**62**–**65**) gehalten werden, verschiebt und die Instruktionen in die Mehrzahl von Containern (**62**–**65**) eingibt.

8. VLIW-Prozessor nach Anspruch 1,
bei dem die Mehrzahl von Containern (**2**, **3**) einen (2i – 1)ten Container (**2**), der eine (2i – 1)te Instruktion in der VLIW-Instruktion hält, und einen (2i)ten Container (**3**), der eine (2i)te Instruktion in der VLIW-Instruktion hält, enthält, wobei i eine natürliche Zahl von 1 bis n ist,
die Mehrzahl von Decodern (**4**, **5**) einen (2i – 1)ten Decoder (**4**), der die (2i – 1)te Instruktion, die in dem (2i – 1)ten Container (**2**) gehalten ist, decodiert, und einen (2i)ten Decoder (**5**), der die (2i)te Instruktion, die in dem (2i)ten Container (**3**) enthalten ist, decodiert, enthält,
die Mehrzahl von Verarbeitungseinheiten (**6**, **7**) eine (2i – 1)te Verarbeitungseinheit (**6**), die die durch den (2i – 1)ten Decoder (**4**) decodierte Instruktion ausführt, und eine (2i)te Verarbeitungseinheit (**7**), die die durch den (2i)ten Decoder (**5**) ausgeführte Instruktion ausführt, enthält, und
der Austauschabschnitt (**8**, **8'**) die in dem (2i – 1)ten Container (**2**) gehaltene (2i – 1)te Instruktion und die in dem (2i)ten Container (**3**) gehaltene (2i)te Instruktion austauscht.

9. VLIW-Prozessor nach Anspruch 1, mit:
einer ersten Auswahlschaltung (**19**, **21**, **87**), die eine von dem Speicher (**1**) in einem ersten Fall ausgegebene Instruktion auswählt,
die in dem ersten Behälter (**3**, **63**) der Mehrzahl von Behältern (**2**, **3**, **62**–**65**) gehaltene Instruktion in einem zweiten Fall auswählt, und
die ausgewählte Instruktion zu einem zweiten Container (**2**, **62**) der Mehrzahl von Containern (**2**, **3**, **62**–**65**) liefert, der sich von dem ersten Container unterscheidet, als eine zu erhaltende Instruktion.

10. VLIW-Prozessor nach Anspruch 9, weiter mit einem Register (**18**, **83**), das die Instruktion von dem ers-

ten Container empfängt, worin die erste Auswahlschaltung (**19**, **87**) einen ersten Eingang, der mit dem Speicher (**1**) verbunden ist, einen zweiten Eingang, der mit dem zweiten Container verbunden ist, und einen Ausgang, der mit dem ersten Container verbunden ist, aufweist.

11. VLIW-Prozessor nach Anspruch 9, bei dem die erste Auswahlschaltung (**21**) einen ersten Eingang, der mit dem Speicher (**1**) durch eine erste Leitung verbunden ist, einen zweiten Eingang, der mit dem Speicher (**2**) durch eine zweite Leitung verbunden ist, und einen Ausgang, der mit dem ersten Container verbunden ist, aufweist, wobei die erste und die zweite Leitung zum Übertragen der Instruktionen zu dem ersten bzw. zweiten Container dienen.

12. VLIW-Prozessor nach einem der Ansprüche 9 bis 11, weiter mit einer Steuerung (**14**, **80**), die die Instruktion von dem ersten Container (**3**, **63**) empfängt und die erste Auswahlschaltung (**19**, **21**, **87**) steuert, worin die Mehrzahl von Decodern einen ersten und einen zweiten Decoder (**5**, **4**, **67**, **66**) enthält, die mit dem ersten und dem zweiten Container (**3**, **2**, **63**, **62**) verbunden sind, wobei der zweite Decoder mit einer vorbestimmten Art von Instruktion als eine nop-Instruktion tätig ist, durch die die Verarbeitungseinheit, die mit dem ersten Decoder verbunden ist, keine Operation ausführt, worin, wenn der erste und der zweite Container eine erste bzw. eine zweite Instruktion enthalten, die erste Instruktion die vorbestimmte Art von Instruktion ist, durch Steuern der ersten Steuerung der erste und der zweite Decoder die erste bzw. zweite Instruktion decodieren in einem Hauptzyklus und der zweite Decoder die erste Instruktion in einem Unterzyklus zu dem Hauptzyklus decodiert.

13. VLIW-Prozessor nach Anspruch 9, weiter mit einer zweiten Auswahlschaltung (**20**, **22**), die eine Instruktion, die von dem Speicher (**1**) in einem dritten Fall ausgegeben ist, auswählt, die in dem zweiten Container (**2**) in einem vierten Fall auswählt und die ausgewählte Instruktion zu dem ersten Container (**8**) als eine zu erhaltende Instruktion liefert.

14. VLIW-Prozessor nach Anspruch 13,
weiter mit einer Steuerung (**14**), die die Instruktionen von dem ersten und dem zweiten Container (**3**, **2**) empfängt und die erste und die zweite Auswahlschaltung (**19**, **20**, **21**, **22**) steuert,
worin die Mehrzahl von Decodern einen ersten und einen zweiten Decoder (**5**, **4**) enthält, die mit dem ersten und dem zweiten Container (**3**, **2**) verbunden sind, wobei der erste Decoder (**5**) mit einer ersten Art von Instruktion als eine nop-Instruktion tätig ist, durch die die Verarbeitungseinheit, die mit dem ersten Decoder (**5**) verbunden ist, keine Tätigkeiten ausführt, und der zweite Decoder (**49**) mit einer zweiten Art von Instruktion als eine nop-Instruktion tätig ist, durch die die Verarbeitungseinheit, die mit dem zweiten Decoder (**4**) verbunden ist, keine Tätigkeit ausführt,
wenn der erste und der zweite Container eine erste bzw. zweite Instruktion halten und in mindestens einem Fall, wenn die erste Instruktion die erste Art von Instruktion ist und wenn die zweite Instruktion die zweite Art von Instruktion ist, durch Steuern der Steuerung der erste und der zweite Decoder die erste bzw. zweite Instruktion decodieren in einem Hauptzyklus und die zweite bzw. erste Instruktion in einem Unterzyklus zu dem Hauptzyklus decodieren.

15. VLIW-Prozessor nach Anspruch 12 oder 14, weiter mit einem Programmzähler (**9**–**13**), der einen Programmzählwert an den Speicher (**1**) ausgibt zum Holen einer VLIW-Instruktion, wobei der Programmzähler einen Programmzählerwert inkrementiert und die Inkrementierung durch Steuern der Steuerung stoppt.

Es folgen 11 Blatt Zeichnungen

Anhängende Zeichnungen

FIG.1

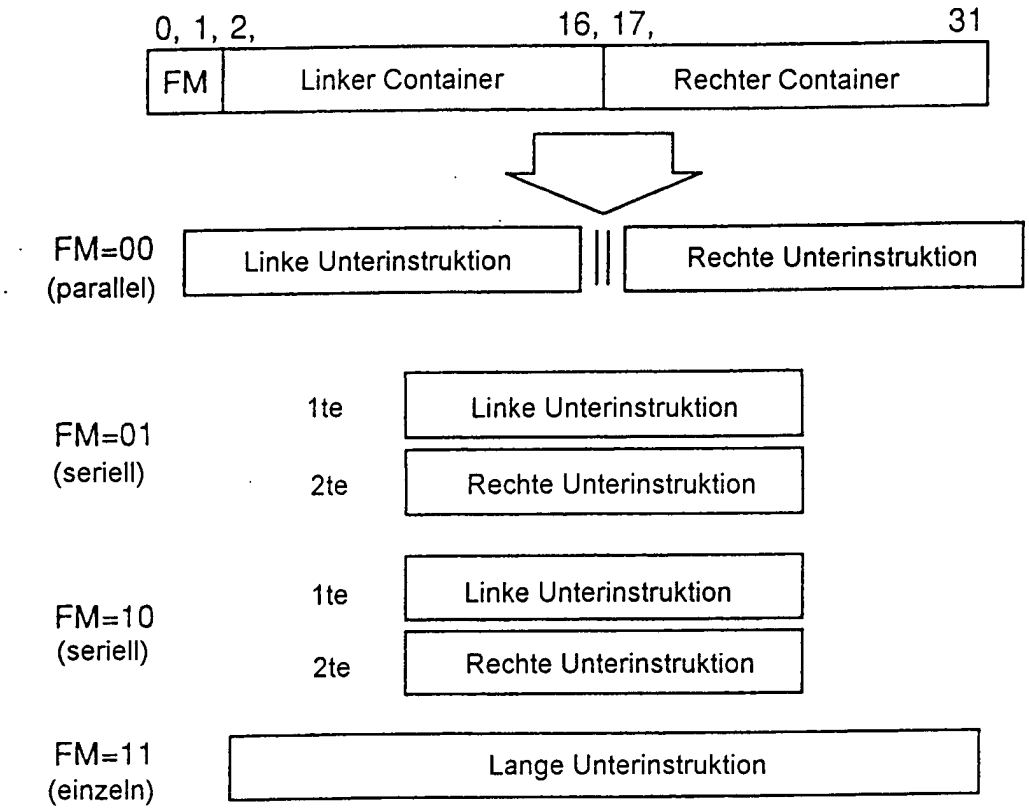


FIG.2

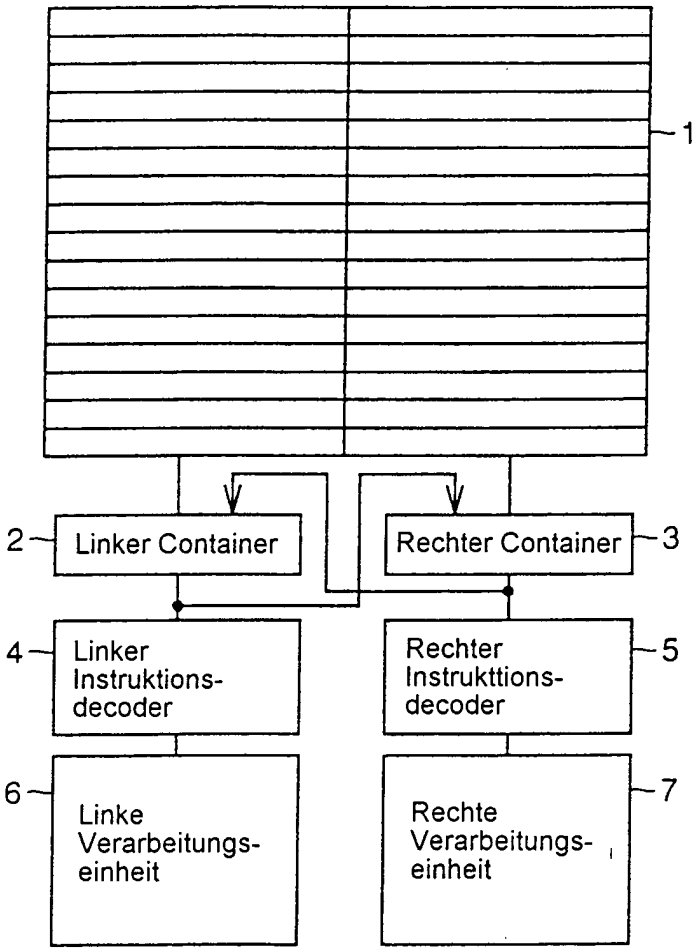

1
2
Linker Container
Rechter Container
3
4
Linker Instruktions-decoder
Rechter Instrukttions-decoder
5
6
Linke Verarbeitungs-einheit
Rechte Verarbeitungs-einheit
7

FIG.3

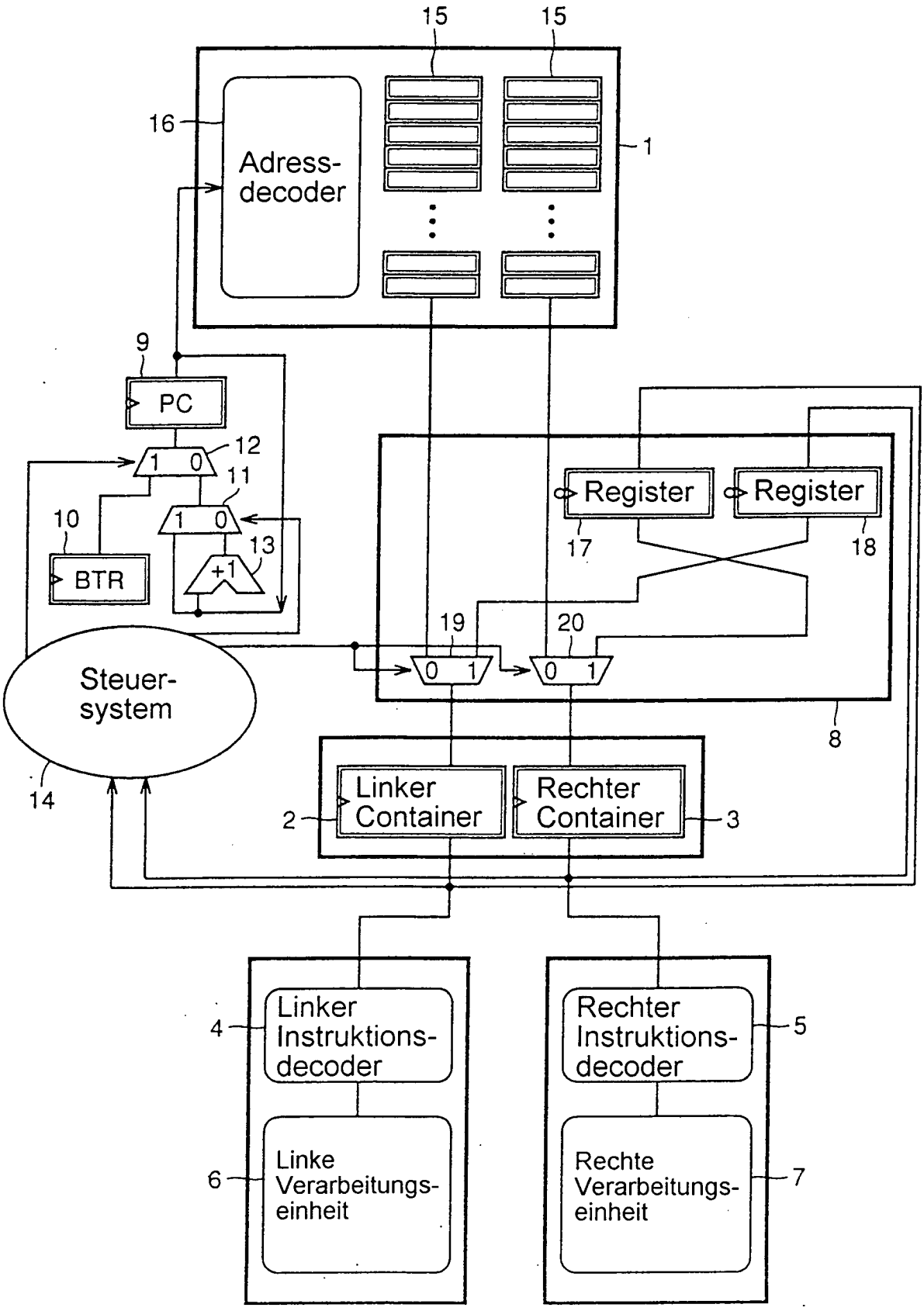

15
15
16
Adress-
decoder
1
9
PC
12
1 0
11
1 0
10
BTR
13
+1
Register
Register
17
18
19
20
0 1
0 1
Steuer-
system
14
8
Linker
Container
Rechter
Container
2
3
4
Linker
Instruktions-
decoder
5
Rechter
Instruktions-
decoder
6
Linke
Verarbeitungs-
einheit
7
Rechte
Verarbeitungs-
einheit

FIG.4

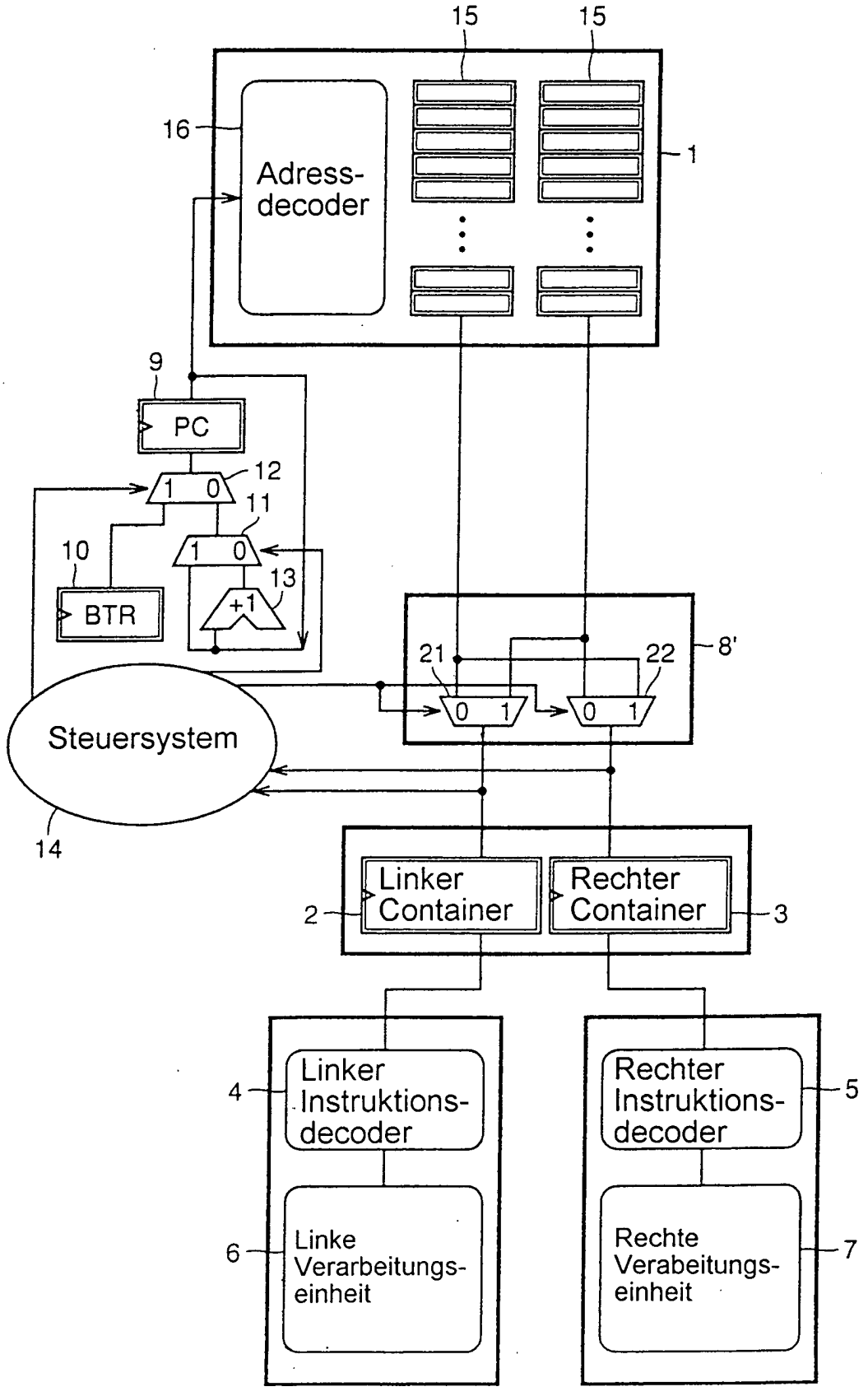

15
15
16
Adress-
decoder
1
9
PC
12
1 0
11
1 0
10
13
+1
BTR
21
22
8'
0 1
0 1
Steuersystem
14
Linker
Container
Rechter
Container
2
3
4
Linker
Instruktions-
decoder
5
Rechter
Instruktions-
decoder
6
Linke
Verarbeitungs-
einheit
7
Rechte
Verabeitungs-
einheit

FIG.5

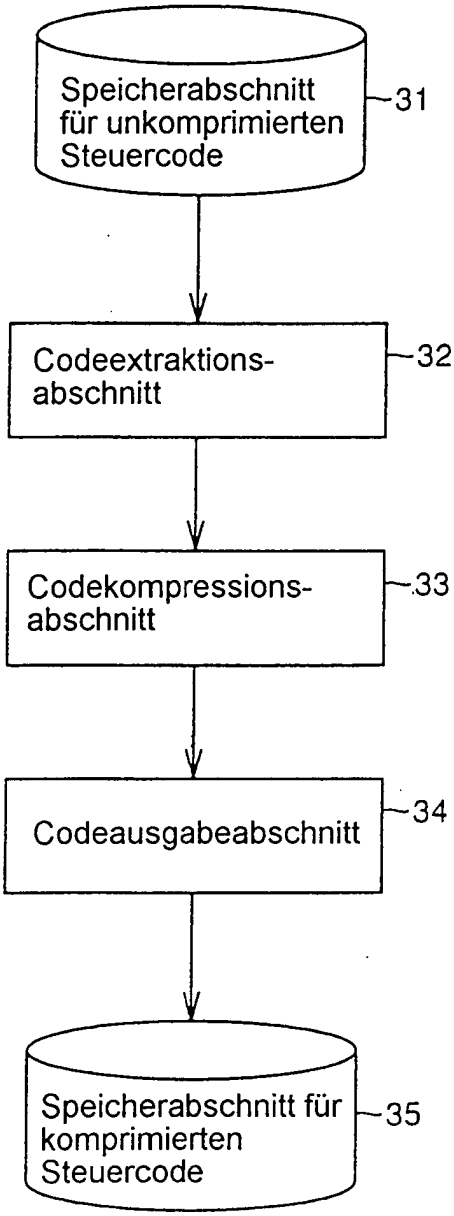

Speicherabschnitt für unkomprimierten Steuercode
31
Codeextraktions-abschnitt
32
Codekompressions-abschnitt
33
Codeausgabeabschnitt
34
Speicherabschnitt für komprimierten Steuercode
35

FIG.6

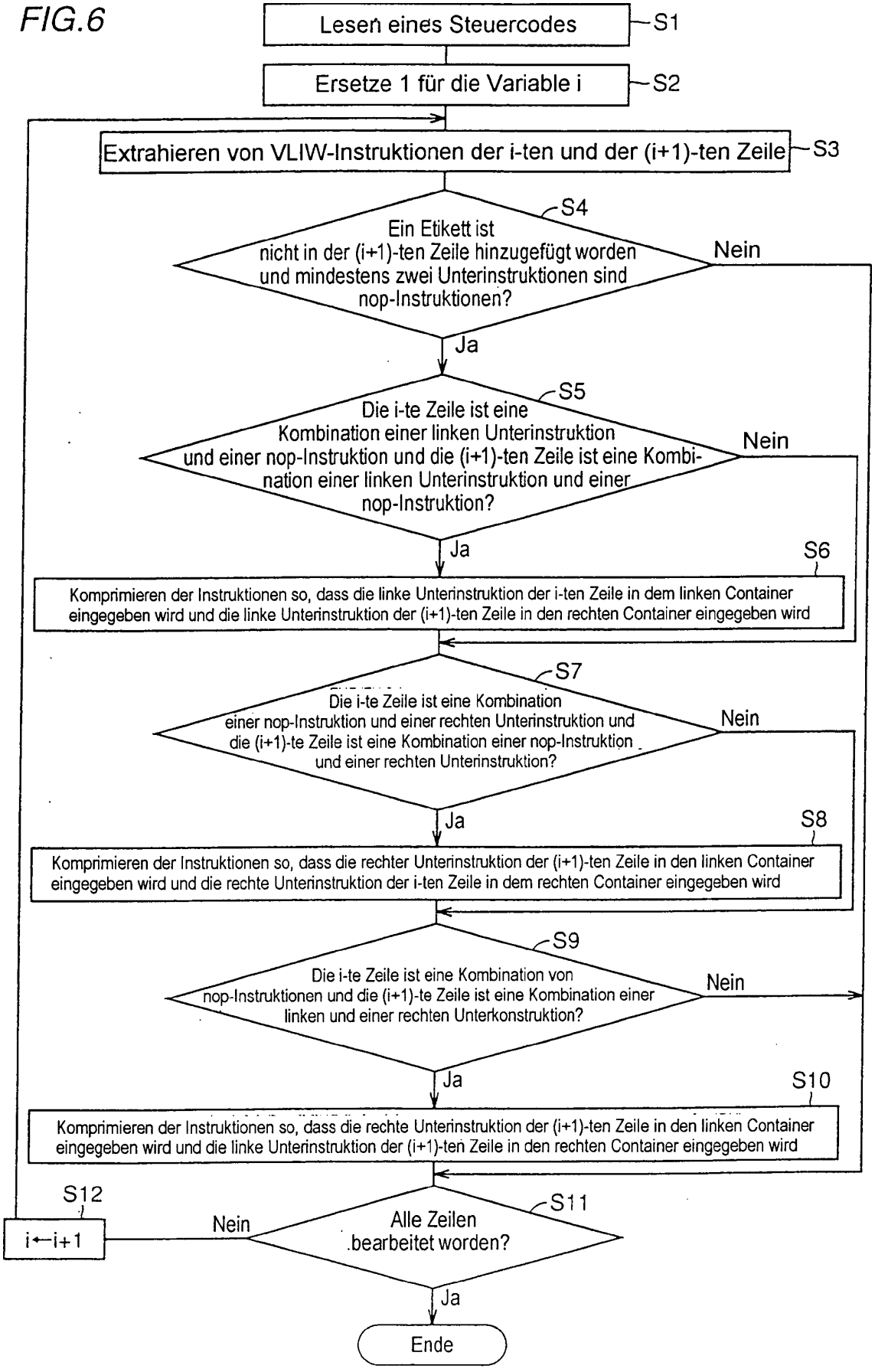

Lesen eines Steuercodes
S1
Ersetze 1 für die Variable i
S2
Extrahieren von VLIW-Instruktionen der i-ten und der (i+1)-ten Zeile
S3
S4
Ein Etikett ist nicht in der (i+1)-ten Zeile hinzugefügt worden und mindestens zwei Unterinstruktionen sind nop-Instruktionen?
Nein
Ja
S5
Die i-te Zeile ist eine Kombination einer linken Unterinstruktion und einer nop-Instruktion und die (i+1)-ten Zeile ist eine Kombination einer linken Unterinstruktion und einer nop-Instruktion?
Nein
Ja
S6
Komprimieren der Instruktionen so, dass die linke Unterinstruktion der i-ten Zeile in dem linken Container eingegeben wird und die linke Unterinstruktion der (i+1)-ten Zeile in den rechten Container eingegeben wird
S7
Die i-te Zeile ist eine Kombination einer nop-Instruktion und einer rechten Unterinstruktion und die (i+1)-te Zeile ist eine Kombination einer nop-Instruktion und einer rechten Unterinstruktion?
Nein
Ja
S8
Komprimieren der Instruktionen so, dass die rechter Unterinstruktion der (i+1)-ten Zeile in den linken Container eingegeben wird und die rechte Unterinstruktion der i-ten Zeile in dem rechten Container eingegeben wird
S9
Die i-te Zeile ist eine Kombination von nop-Instruktionen und die (i+1)-te Zeile ist eine Kombination einer linken und einer rechten Unterkonstruktion?
Nein
Ja
S10
Komprimieren der Instruktionen so, dass die rechte Unterinstruktion der (i+1)-ten Zeile in den linken Container eingegeben wird und die linke Unterinstruktion der (i+1)-ten Zeile in den rechten Container eingegeben wird
S12
i←i+1
Nein
S11
Alle Zeilen bearbeitet worden?
Ja
Ende

FIG.7

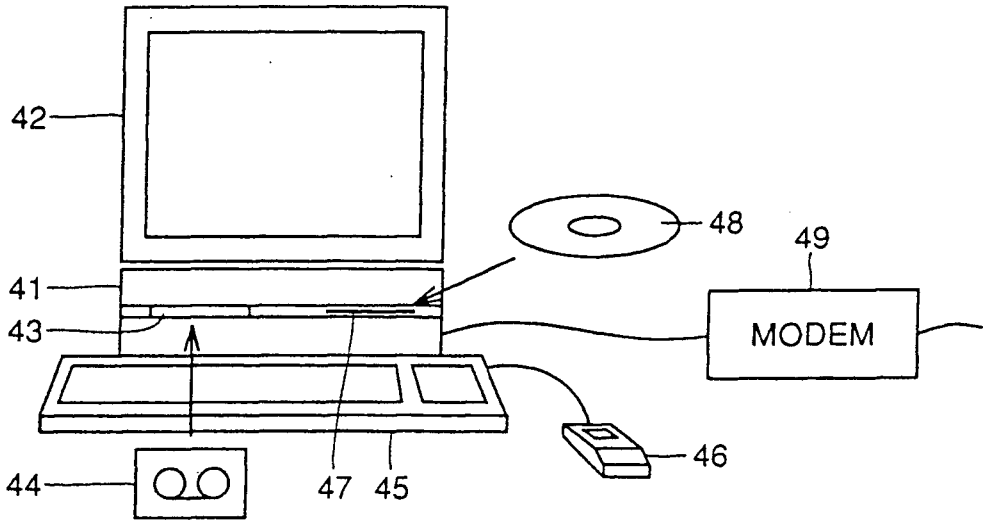


FIG.8

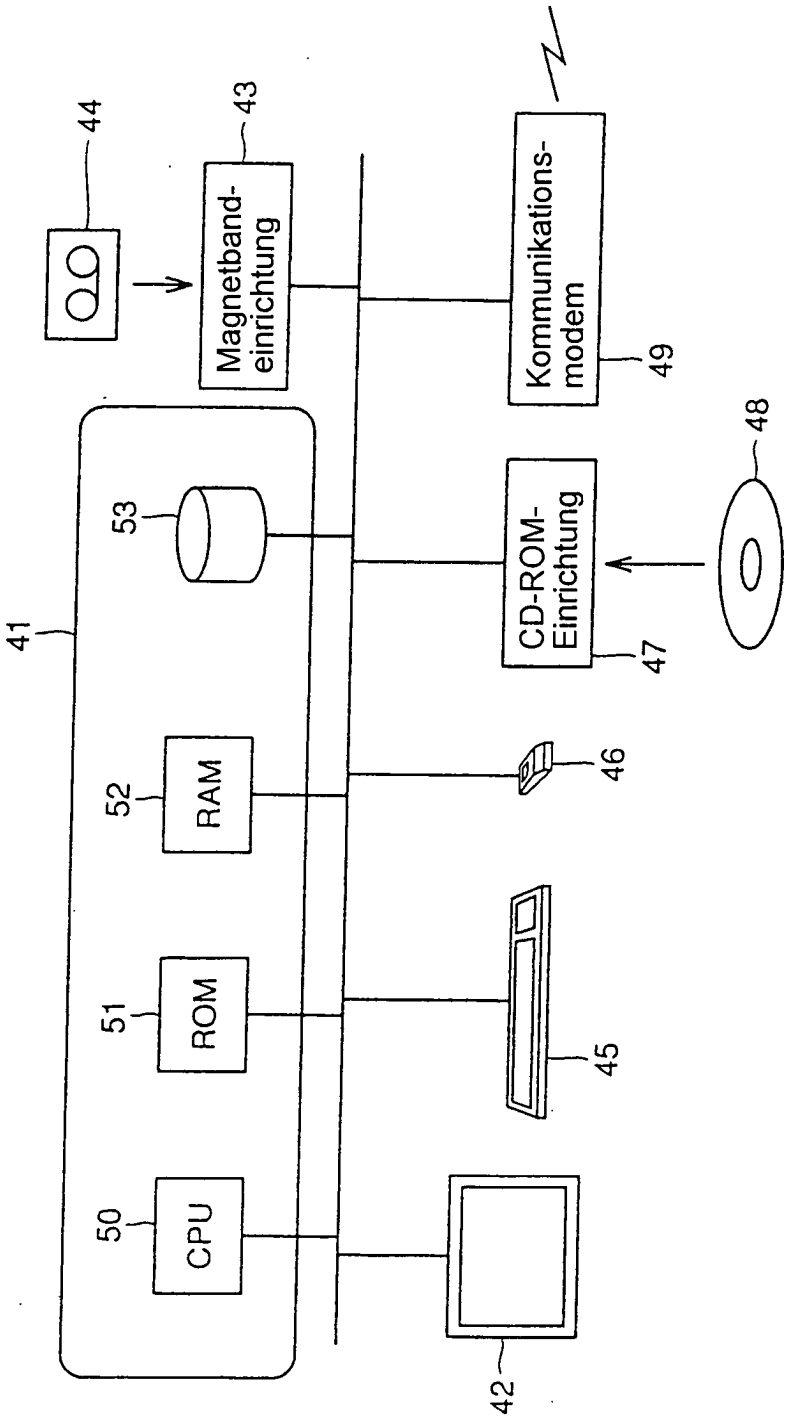

44
43
Magnetband-
einrichtung
Kommunikations-
modem
49
48
53
CD-ROM-
Einrichtung
47
41
52
RAM
46
51
ROM
45
50
CPU
42

FIG.9

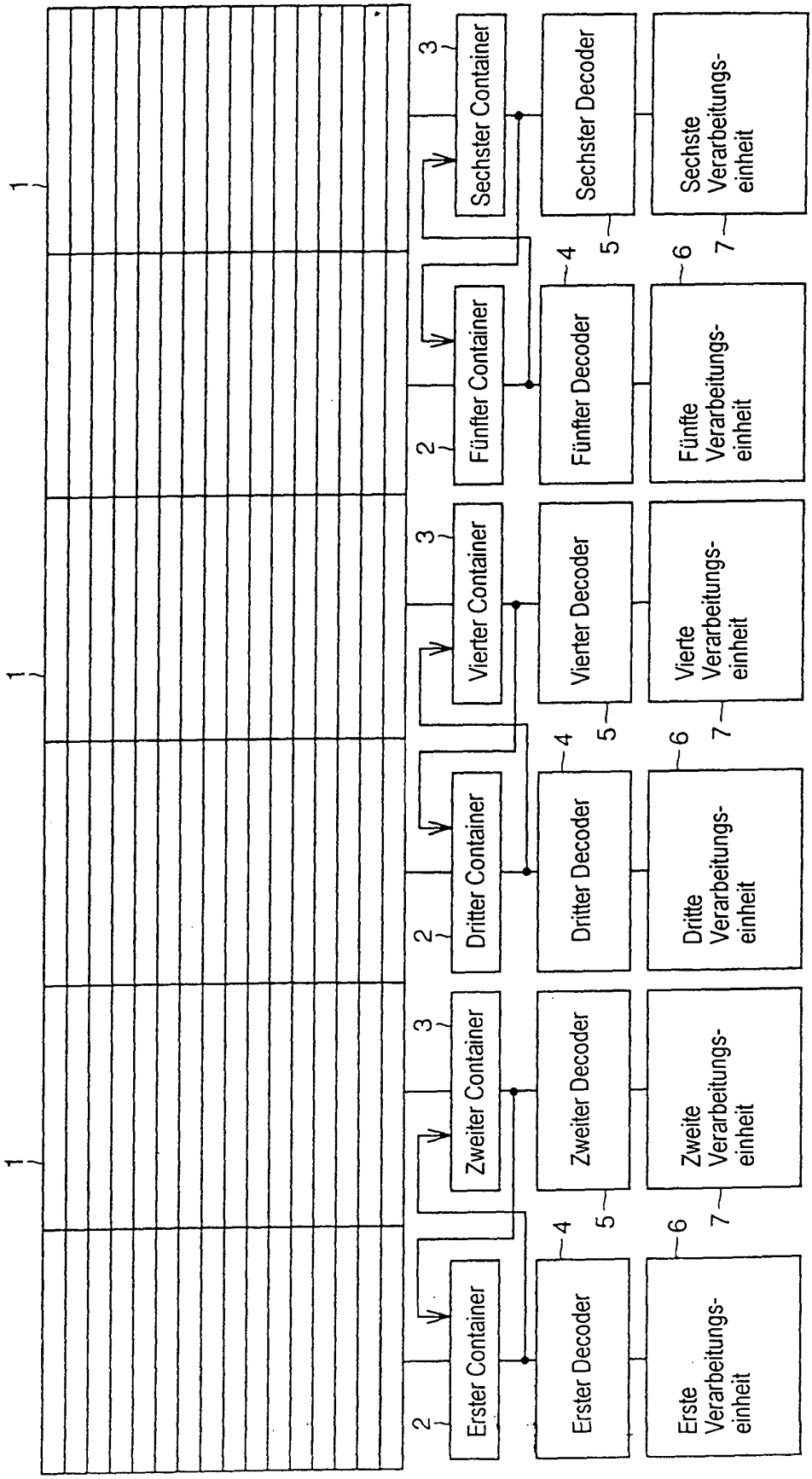

1
2
3
4
5
6
7
Erster Container
Zweiter Container
Dritter Container
Vierter Container
Fünfter Container
Sechster Container
Erster Decoder
Zweiter Decoder
Dritter Decoder
Vierter Decoder
Fünfter Decoder
Sechster Decoder
Erste Verarbeitungs-einheit
Zweite Verarbeitungs-einheit
Dritte Verarbeitungs-einheit
Viere Verarbeitungs-einheit
Fünfte Verarbeitungs-einheit
Sechste Verarbeitungs-einheit

FIG.10

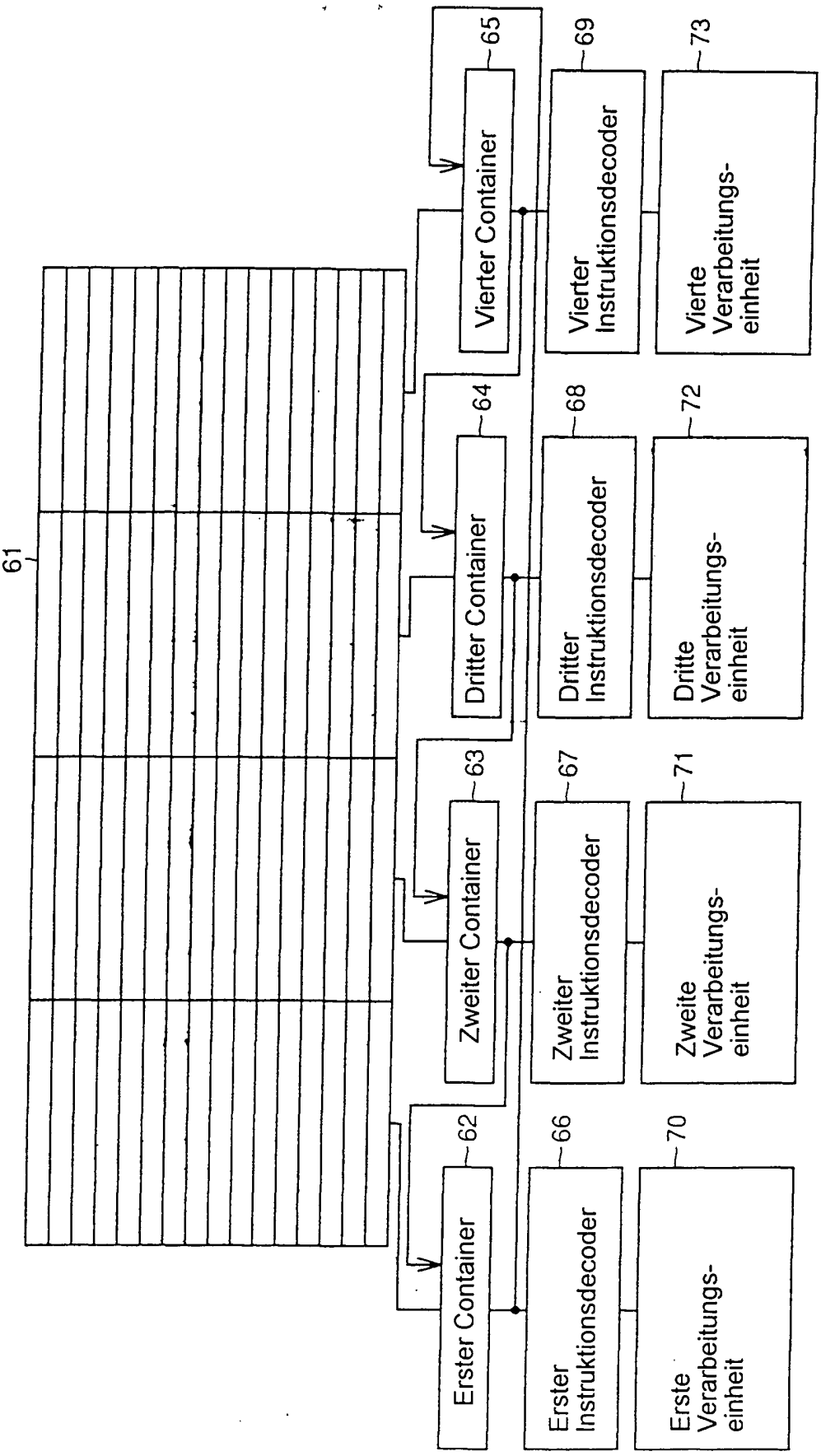
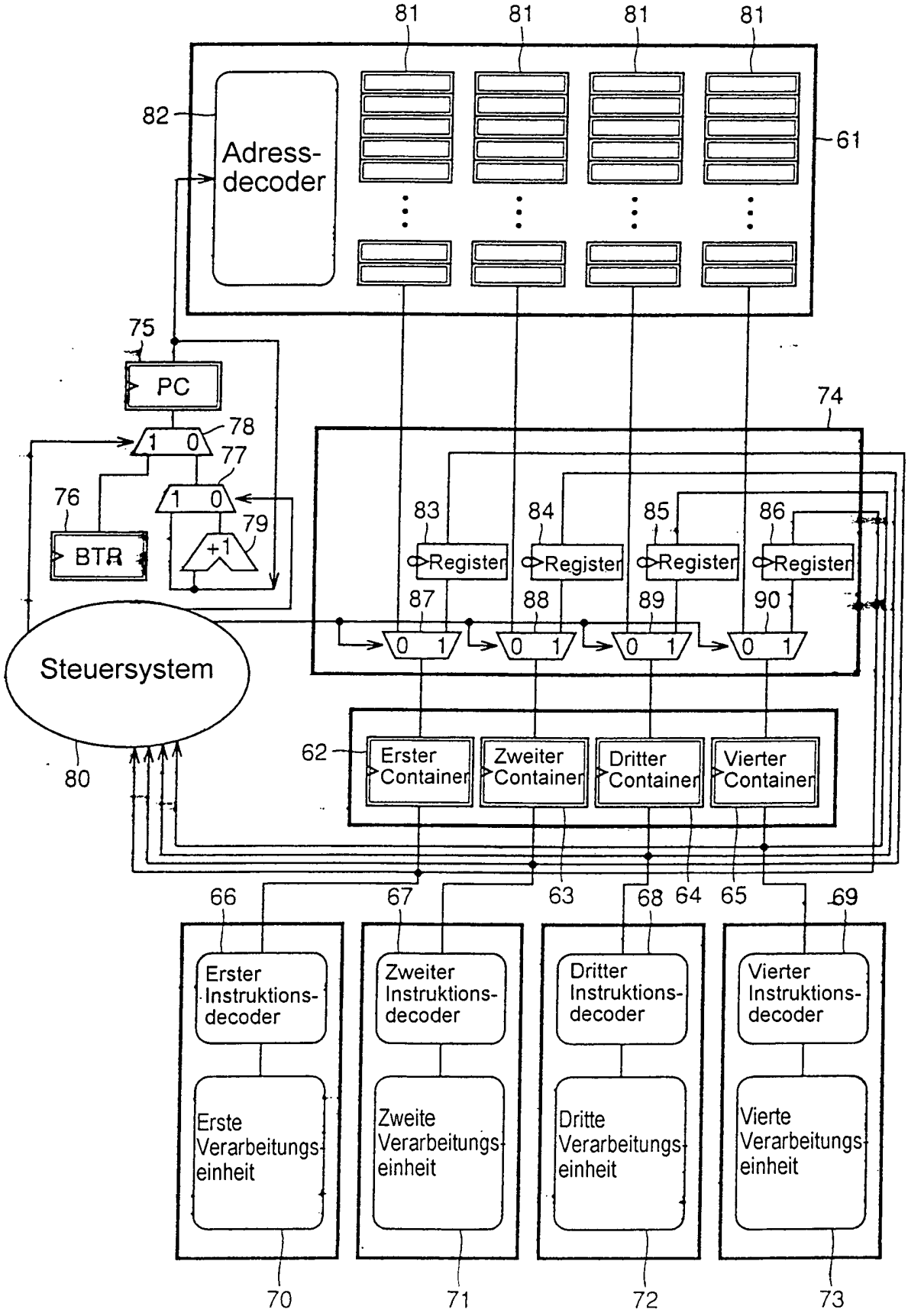

61
Erster Container
62
Zweiter Container
63
Dritter Container
64
Vierter Container
65
Erster Instruktionsdecoder
66
Zweiter Instruktionsdecoder
67
Dritter Instruktionsdecoder
68
Vierter Instruktionsdecoder
69
Erste Verarbeitungs-einheit
70
Zweite Verarbeitungs-einheit
71
Dritte Verarbeitungs-einheit
72
Vierte Verarbeitungs-einheit
73

FIG.11


81
81
81
81
82
Adress-
decoder
61
75
PC
74
78
1 0
77
76
1 0
79
BTR
+1
83
84
85
86
Register
Register
Register
Register
87
88
89
90
0 1
0 1
0 1
0 1
Steuersystem
62
Erster
Container
Zweiter
Container
Dritter
Container
Vierter
Container
80
66
67
63
68
64
65
69
Erster
Instruktions-
decoder
Zweiter
Instruktions-
decoder
Dritter
Instruktions-
decoder
Vierter
Instruktions-
decoder
Erste
Verarbeitungs-
einheit
Zweite
Verarbeitungs-
einheit
Dritte
Verarbeitungs-
einheit
Vierte
Verarbeitungs-
einheit
70
71
72
73