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

(10) **DE 698 14 268 T2** 2004.01.22

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 901 071 B1**
(21) Deutsches Aktenzeichen: **698 14 268.3**
(96) Europäisches Aktenzeichen: **98 115 908.0**
(96) Europäischer Anmeldetag: **24.08.1998**
(97) Erstveröffentlichung durch das EPA: **10.03.1999**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **07.05.2003**
(47) Veröffentlichungstag im Patentblatt: **22.01.2004**

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

(30) Unionspriorität:
924518 **05.09.1997** **US**

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

(73) Patentinhaber:
Motorola, Inc., Schaumburg, Ill., US

(74) Vertreter:
SCHUMACHER & WILLSAU, Patentanwaltssozietät, 80335 München

(72) Erfinder:
Moyer, William C., Dripping Springs, Texas 78620, US; Arends, John, Austin, Texas 78748, US; Scott, Jeffrey W., Austin, Texas 78759, US

(54) Bezeichnung: **Verfahren zur Anbindung eines Prozessors an einen Koprozessor**

Beschreibung

Gebiet der Erfindung

[0001] Die vorliegende Erfindung bezieht sich allgemein auf ein Datenverarbeitungssystem mit einem Prozessor und wenigstens einem Koprozessor und insbesondere auf ein Verfahren und eine Vorrichtung zur Anbindung des Prozessors an den Koprozessor.

Hintergrund der Erfindung

[0002] Die Möglichkeit, die Funktionalität eines Prozessors mit grundlegender Architektur durch spezielle und spezialisierte, funktionale Hardware-Elemente zu erweitern, ist ein wichtiger Aspekt für skalierbare und erweiterbare Architekturen.

[0003] Eines der bevorzugten Verfahren zur Erweiterung der Funktionalität eines Prozessors mit grundlegender Architektur ist die Verwendung von Koprozessoren. Diese sind üblicherweise spezielle, Prozessoren für einen einzigen Zweck, die nach Anweisung eines Prozessors arbeiten. Eine der traditionellen Verwendungen von Koprozessoren war als mathematische Koprozessoren zur Selektiven Bereitstellung von Gleitkomma-Fähigkeiten für Architekturen, die solche nicht direkt unterstützten. Einige Beispiele für solche mathematischen Koprozessoren sind der Intel **8087** und **80287**. Einige andere potentielle Verwendungen oder Typen von Koprozessoren schließen ein: Multiplikations-Akkumulatoren, Modulatoren/Demodulatoren (Modems), digitale Signalprozessoren (DSP), Vitturbi-Rechner, kryptographische Prozessoren, Bildverarbeitungsprozessoren und Vektorprozessoren.

[0004] Es gibt zwei verschiedene Ansätze für Koprozessoren. Einerseits war die Gleitkomma-Einheit für die Computerfamilie der Digital Equipment Corporation (DEC) PDP-11 sehr eng mit ihrem Primärprozessor gekoppelt. Ein sich ergebendes Problem ist, dass diese enge Kopplung erforderte, dass der Prozessor viel über den Betrieb des Koprozessors weiß. Dies verkompliziert die Auslegung der Schaltung derart, dass eine Hinzufügung eines neuen Koprozessors in ein integriertes System ein größeres technisches Problem darstellt.

[0005] Die alternative Implementation ist, den Koprozessor locker mit dem Primärprozessor zu koppeln. Dies hatte den Vorteil, den Betrieb des Koprozessors zu abstrahieren und vom Primärprozessor zu isolieren, was die Anstrengungen, die zur Integration eines neuen Koprozessors mit einem bestehenden Prozessor, wesentlich verringert. Dies hat jedoch einen unumgänglichen Preis. Leistungsverlust ist ein Problem dieses Ansatzes. Ein Problem bei der Inkaufnahme dieser Art von Leistungsverlust, der aus dieser lockeren Kopplung resultiert, ist, dass der Amortisationspunkt für den Aufruf eines solchen Koprozessors entsprechend höher liegt. Daher sind viele ansonsten für Koprozessoren attraktive Anwendungen nicht kosteneffizient. Außerdem erfordert solch ein Ansatz oft einen Bus mit all der entsprechenden, zusätzlichen Schaltung und Chipfläche.

[0006] Es ist daher wichtig, eine Koprozessor-Schnittstelle zu haben, die eng genug angebunden ist, dass die Verwendung der Schnittstelle ausreichend schnell ist, so dass ein Aufruf selbst bei ziemlich einfachen Funktionen vorteilhaft ist, während die Abstraktion der Schnittstelle so weitgehend ist, dass die Prozessorarchitektur von so vielen Details wie möglich eines beliebigen Koprozessors wie möglich isoliert ist. Teil dieses letzteren schließt ein, die Schnittstelle programmierfreundlicher zu machen, um den Zuschnitt neuer Koprozessor-Anwendungen in Software statt in Hardware zu erleichtern.

[0007] Ein Artikel mit dem Titel "Coprocessor data transfer instructions", VLSI RISC Architecture and Organization, S. 261–265, S B Furber, XP002061358ISBD: 0–8247–8151–1 offenbart eine Klasse von Anweisungen, die verwendet werden, um ein oder mehrere Datenworte zwischen einem Koprozessor und dem Hauptspeicher zu übertragen. Der beschriebene Koprozessor nimmt die Daten bei sequentiel1en Zyklusraten und ist verantwortlich für die Bestimmung der Anzahl übertragenen Worte.

[0008] Die Europäischen Patentanmeldungen EP-A-0280821 und EP-A-0261685 beschreiben Systeme mit einer Schnittstelle zwischen einem Prozessor und einem Koprozessor, die für die vorliegende Erfindung relevant sind.

Kurze Beschreibung der Zeichnungen

[0009] Die Merkmale und Vorteile der vorliegenden Erfindung werden aus der folgenden, detaillierten Beschreibung in Verbindung mit den beigefügten Figuren klarer verständlich, wobei sich gleiche Bezugszeichen auf gleiche und einander entsprechende Teile beziehen.

[0010] **Fig.** 1 ist ein Blockdiagramm, das eine Ausführungsform eines Datenverarbeitungssystem gemäß der vorliegenden Erfindung illustriert;

[0011] **Fig.** 2 ist ein Blockdiagramm, das einen Teil des Prozessors von **Fig.** 1 illustriert;

[0012] **Fig.** 3 ist ein Blockdiagramm, das eine Ausführungsform eines Teils des Koprozessors von **Fig.** 1 illustriert; **Fig.** 4 ist ein Timing-Diagramm, das eine Register-Snooping-Operation gemäß der vorliegenden Erfindung illustriert;

[0013] **Fig.** 5 ist ein Timing-Diagramm, das die grundlegende Betriebsweise einer Schnittstellenanweisung zur Anweisung eines Handshakes illustriert;

[0014] **Fig.** 6 ist ein Timing-Diagramm, das die Schnittstellenanweisungsoperation illustriert, wenn das H_BUSY*-Signal verwendet wird, um die Ausführung der Schnittstellenanweisung zu steuern;

[0015] **Fig.** 7 ist ein Timing-Diagramm, das das Verwerfen einer Anweisung illustriert;

[0016] **Fig.** 8 ist ein Timing-Diagramm, das ein Beispiel für eine Verzögerung in der Anweisungs-Pipe-

line illustriert;

[0017] **Fig.** 9 ist ein Timing-Diagramm, das ein Beispiel einer End-zu-End- ("back-to-back"-) Ausführung ohne Verzögerungen illustriert;

[0018] **Fig.** 10 ist ein Timing-Diagramm, das eine End-zu-End-Operation mit internen Pipeline-Verzögerungen illustriert;

[0019] **Fig.** 11 ist ein Timing-Diagramm, das eine End-zu-End-Koprozessor-Schnittstellenanweisungen **30** mit H_BUSY*-Verzögerungen illustriert;

[0020] **Fig.** 12 ist ein Timing-Diagramm, das ein Beispiel des H_EXCP*-Signals illustriert, das von einem Koprozessor infolge der Dekodierung und der versuchten Ausführung eines Koprozessor-Schnittstellen-Operationscodes zugeordnet wird;

[0021] **Fig.** 13 ist ein Timing-Diagramm, das ein Beispiel des H_EXCP*-Signals illustriert, das von einem Koprozessor infolge der Dekodierung und der versuchten Ausführung eines Koprozessor-Schnittstellen-Operationscodes gesetzt wird, wenn die Koprozessor-Schnittstellenanweisung verworfen wird;

[0022] **Fig.** 14 ist ein Timing-Diagramm, das ein Beispiel illustriert, in dem H_BUSY* gesetzt wurde, um die Ausführung eines Koprozessor-Schnittstellen-Operationscodes zu verzögern;

[0023] **Fig.** 15 ist ein Timing-Diagramm, das ein Beispiel eines mit dem H_CALL-Grundelement in Beziehung stehenden Registertransfers illustriert;

[0024] **Fig.** 16 ist ein Timing-Diagramm, das ein Beispiel eines mit dem H_RET-Grundelement in Beziehung stehenden Registertransfers illustriert;

[0025] **Fig.** 17 ist ein Timing-Diagramm, das den Ablauf eines H_LD-Transfers an die Koprozessorschnittstelle illustriert;

[0026] **Fig.** 18 ist ein Timing-Diagramm, das das Protokoll illustriert, wenn ein Speicherzugriff zu einer Speicherausnahme führt;

[0027] **Fig.** 19 ist ein Timing-Diagramm, das ein Beispiel eines mit dem H_ST-Grundelement in Beziehung stehenden Transfers illustriert;

[0028] **Fig.** 20 ist ein Timing-Diagramm, das ein Beispiel eines Transfers mit verzögerten Speicherdaten illustriert; **Fig.** 21 ist ein Timing-Diagramm, das die Protokollsignale illustriert, wenn die Speicherung zu einem Zugriffsfehler führt;

[0029] **Fig.** 22 illustriert ein Anweisungsformat für das H_CALL-Grundelement gemäß der vorliegenden Erfindung;

[0030] **Fig.** 23 illustriert ein Anweisungsformat für das H_RET-Grundelement gemäß der vorliegenden Erfindung; **Fig.** 24 illustriert ein Anweisungsformat für das H_EXEC-Grundelement gemäß der vorliegenden Erfindung;

[0031] **Fig.** 25 illustriert ein Anweisungsformat für die H_LD-Anweisung gemäß der vorliegenden Erfindung und **Fig.** 26 illustriert ein Anweisungsformat für die H_ST-Anweisung gemäß der vorliegenden Erfindung.

Beschreibung einer bevorzugten Ausführungsform

[0032] In der folgenden Beschreibung werden eine Vielzahl spezifischer Details aufgestellt, wie etwa spezielle Wort- oder Bytelängen etc., um ein tiefgehendes Verständnis der vorliegenden Erfindung zu ermöglichen. Es ist für den Fachmann jedoch offensichtlich, dass die vorliegende Erfindung ohne solche spezifischen Details angewendet werden kann. In anderen Fällen werden Schaltungen in Form von Blockdiagrammen dargestellt, um die vorliegende Erfindung nicht mit überflüssigen Details zu verschleiern. Meistens werden Details, die Timing-Überlegungen und dergleichen betreffen, insoweit ausgelassen, als solche Details nicht erforderlich sind, um ein vollständiges Verständnis der vorliegenden Erfindung zu erhalten und liegen im Bereich des Fachwissens des entsprechenden Fachmanns.

[0033] Der Ausdruck "Bus" wird verwendet, um eine Mehrzahl von Signalen oder Leitern zu bezeichnen, die verwendet werden können, um einen oder mehrere Typen von Information, wie etwa Daten, Adressen, Steuerbefehle oder Statusangaben zu übermitteln. Die Ausdrücke "setzen" und "negieren" werden verwendet, wenn auf die Änderung eines Signals, Statusbits oder einer ähnlichen Vorrichtung in ihren logisch wahren bzw. logisch falschen Zustand Bezug genommen wird. Hat der logisch wahre Zustand den logischen Pegel Eins, hat der logisch falsche Zustand den logischen Pegel Null. Und wenn der logisch wahre Zustand den logischen Pegel Null hat, hat der logisch falsche Zustand den logischen Pegel Eins.

[0034] **Fig.** 1 ist ein Blockdiagramm, das eine Ausführungsform eines Datenverarbeitungssystems **10** illustriert, welche einen Prozessor **12**, einen Koprozessor **14**, einen Koprozessor **16**, einen Speicher **18**, sonstige Module **20** und eine externe Busschnittstelle **22** enthält, die alle über den Bus **28** bidirektional angeschlossen sind. Alternative Ausführungsformen der vorliegenden Erfindung können auch lediglich einen Koprozessor **14**, zwei Koprozessoren **14** und **16** oder sogar mehrere Koprozessoren (nicht gezeigt) enthalten. Die externe Busschnittstelle **22** ist über die integrierten Schaltungsterminals **35** bidirektional mit dem externen Bus **26** verbunden. Der Speicher **24** ist bidirektional mit dem externen Bus **26** verbunden. Der Prozessor **12** kann optional über integrierte Schaltungsterminals **31** extern mit dem Datenverarbeitungssystem **10** verbunden sein. Der Koprozessor **14** kann optional über integrierte Schaltungsterminals **32** extern mit dem Datenverarbeitungssystem **10** verbunden sein. Der Speicher **18** kann optional über integrierte Schaltungsterminals **32** extern mit dem Datenverarbeitungssystem **10** verbunden sein. Die anderen Module **20** können optional über integrierte Schaltungsterminals **34** extern mit dem Datenverarbeitungssystem **10** verbunden sein. Der Prozessor **12** ist über die Koprozessorschnittstelle **30** bidirektional mit dem Koprozessor **14** und dem Koprozessor **16** verbunden.

[0035] **Fig.** 2 ist ein Blockdiagramm, das einen Teil des Prozessors **12** von **Fig.** 1 illustriert. Bei einem Ausführungsbeispiel enthält der Prozessor **12** eine Steuerschaltung **40**, eine Anweisungs-Dekodierungsschaltung **42**, eine Anweisungs-Pipeline **44**, Register **46**, ein Rechenwerk (ALU: arithmetic logic unit) 48, einen signalspeichernden Multiplexer (MUX) **50**, einen signalspeichernden Multiplexer (MUX) **52** und einen Multiplexer (MUX) **54**. Bei einem Ausführungsbeispiel der vorliegenden Erfindung umfasst die Koprozessorschnittstelle **30** Signale **60** bis **71**. Ein Taktgebersignal **60** wird von der Steuerschaltung **40** erzeugt. Koprozessor-Betriebssignale **61** werden von der Steuerschaltung **40** erzeugt und den Koprozessoren **14** und **16** zur Verfügung gestellt.

[0036] Ein Überwachungsmodus-Signal **62** wird von der Steuerschaltung **40** erzeugt und den Koprozessoren **14** und **16** zur Verfügung gestellt. Ein Dekodierungssignal **63** wird von der Steuerschaltung **40** erzeugt und den Koprozessoren **14** und **16** zur Verfügung gestellt. Ein Koprozessor-beschäftigt ("busy")-Signal 64 wird durch die Steuerschaltung **40** von dem Koprozessor **14** oder dem Koprozessor **16** empfangen. Ein Ausführungssignal **65** wird von der Steuerschaltung **40** erzeugt und den Koprozessoren **14** und **16** zur Verfügung gestellt. Ein Ausnahmesignal **66** wird durch die Steuerschaltung **40** vom Koprozessor **14** oder vom Koprozessor **16** empfangen. Ein "Register Write"-Signal (REGWR*) 67 wird von der Steuerschaltung **40** erzeugt und den Koprozessoren **14** und **16** zur Verfügung gestellt. Registersignale (REG{4 : 0}) 68 werden von der Steuerschaltung **40** erzeugt und den Koprozessoren **14** und **16** zur Verfügung gestellt. Ein Fehlersignal (H_ERR*) 69 wird von der Steuerschaltung **40** erzeugt und den Koprozessoren **14** und **16** zur Verfügung gestellt. Ein Datenfreigabesignal ("data strobe signal") (H_DS*) 70 wird von der Steuerschaltung **40** erzeugt und den Koprozessoren **14** und **16** zur Verfügung gestellt. Ein Datenbestätigungssignal (H_DA*) 71 wird durch die Steuerschaltung **40** vom Koprozessor **14** oder vom Koprozessor **16** empfangen. Hardware-Datenanschlusssignale (HDP{31 : 0}) 72, die auch als Teil der Koprozessorschnittstelle **30** angesehen werden, sind zwischen den Koprozessoren **14** und **16** und der internen Schaltung innerhalb des Prozessors **12** bidirektional.

[0037] Bei einem Ausführungsbeispiel der vorliegenden Erfindung wird eine Mehrzahl von Signalen dem oder von dem Bus **28** bereitgestellt, um die Daten in den Speicher **18** und/oder den Speicher **24** zu laden oder zu speichern. Bei einem Ausführungsbeispiel enthalten diese Signale ein Übertragungsanforderungssignal (TREQ*) 73, das von der Steuerschaltung **40** erzeugt und dem Bus **28** zur Verfügung gestellt wird. Ein Übertragungsfehler-Bestätigungssignal (TEA*) 74 wird über den Bus **28** an die Steuerschaltung **40** geliefert. Ein Übertragungsbestätigungssignal (TA*) 75 wird über den Bus **28** der Steuerschaltung **40** zur Verfügung gestellt. Über die Leiter **76** werden Anweisungen von dem Bus **28** an die Anweisungspipeline **44** geliefert. Daten werden über die Leiter **76** an den MUX **54** geliefert. Ein Treiberdatensignal **79** versetzt den Tristate-Zwischenspeicher **95** in die Lage, über die Leiter **88** und **76** Daten von dem signalspeichernden MUX **52** zu liefern. Ein Adressauswahlsignal **78** versetzt den signalspeichernden MUX **50** in die Lage, über die Leiter **77** Adressen an den Bus **28** zu liefern. Eine weitere Eingabe des MUX **54** wird von dem HDP-Signal (HDP{31 : 0}) 72 bereitgestellt. Eine weitere Eingabe in den MUX **54** wird über die ALU-Ergebnisleiter **86** bereitgestellt. Die Ausgabe des MUX **54**, die Ergebnissignale **83**, werden an die Register **46** und an den Eingang des Tristate-Zwischenspeichers **96** geliefert. Das Treiber-HDP-Signal **82** versetzt den Tristate-Zwischenspeicher **96** in die Lage, die Ergebnissignale **83** auf den HDP-Signalen **72** zu verschicken. Der Ausgang des Tristate-Zwischenspeichers **96** ist auch mit dem Eingang des signalspeichernden MUX **52** verbunden. Alternative Ausführungsformen der vorliegenden Erfindung können in den Registern **46** eine beliebige Anzahl von Registern enthalten. Die Ergebnissignale **83** werden als eine Eingabe an den signalspeichernden MUX **50** geliefert. Die Ergebnissignale **83** werden über den MUX **54** an die Register **46** geliefert. Das Ergebnisauswahlsignal (RESULT_SELECT) **81** wählt aus, welcher Eingabe des MUX **54** auf den Ergebnisleitern **83** verschickt werden soll. Ein Quellenauswahlsignal (SOURCE_SELECT) **80** wird von dem signalspeichernden MUX **52** bereitgestellt, um auszuwählen, welches Signal auf den Leitern **88** zu dem Tristate-Zwischenspeicher **95** geschickt werden soll. Die Steuerschaltung **40** liefert Steuerinformationen und empfängt Statusinformationen von den Registern **46** über die Leiter **91**. Die Steuerschaltung **40** liefert Steuersignale und empfängt Statussignale von dem Rechenwerk **48** über die Leiter **92**. Die Steuerschaltung **40** liefert Steuersignale und empfängt Statussignale von der Anweisungspipeline **44** und der Anweisungsdekodierungsschaltung **42** über die Leiter **93**. Die Anweisungspipeline **44** ist angeschlossen, um der Anweisungsdekodierungsschaltung **42** über die Leiter **89** Anweisungen zu liefern. Die Anweisungsdekodierungsschaltung **42** liefert über die Leiter **90** dekodierte Anweisungsinformationen an die Steuerschaltung **40**. Die Register **46** liefern über die Leiter **84** Quelloperanden an das Rechenwerk **48**. Die Register **46** liefern im Speicher **18** oder im Speicher **24** zu speichernde Daten über die Leiter **84**, den signalspeichernden MUX **52**, den Tristate-Zwischenspeicher **95** und die Leiter **76**. Die Register **46** liefern Adressinformationen an den Speicher **18** oder den Speicher **24** über die Leiter **84**, den signalspeichernden MUX **50** und die Adressleiter **77**. Die Register **46** liefern einen zweiten Quelloperanden an das Rechenwerk **48** über die Leiter **85**.

[0038] **Fig.** 3 ist ein Blockdiagramm, das eine Ausführungsform eines Teils des Koprozessors **14** illust-

riert. Bei einer Ausführungsform enthält der Koprozessor **14** eine Steuerschaltung **100**, eine Berechnungsschaltung **102** und eine optionale Speicherschaltung **104**. Die Steuerschaltung **100** ist bidirektional mit dem Prozessor **12** über die Koprozessorschnittstelle **30** verbunden, die die Signale **60** bis **72** umfasst. Bei einem Ausführungsbeispiel der vorliegenden Erfindung enthält die Steuerschaltung **100** eine Dekodierungsschaltung **106**, die die Operationssignale **61** und das Dekodierungssignal **63** vom Prozessor **12** empfängt. Die Steuerschaltung **100** liefert Steuerinformationen und empfängt Statusinformationen von der optionalen Speicherschaltung **104** über die Leiter **108**. Die Steuerschaltung **100** liefert Steuerinformationen und empfängt Statusinformationen von der Berechnungsschaltung **102** über die Leiter **109**. Die Berechnungsschaltung **102** und die optionale Speicherschaltung **104** sind über die Leiter **110** bidirektional verbunden. Ein oder mehrere Signale **110** können dem oder von dem Bus **28** oder den oder von den integrierten Schaltungsterminals **32** zur Verfügung gestellt werden. Die Steuerschaltung **100** kann über die Leiter **112** von dem Bus **28** oder den integrierten Schaltungsterminals **32** Informationen empfangen oder dem Bus **28** oder den integrierten Schaltungsterminals **32** bereitstellen. Die Signale **72** können bidirektional mit der Berechnungsschaltung **102** und der optionalen Speicherschaltung **104** verbunden sein. Außerdem können die Signale **72** bidirektional mit dem Bus **28** oder den integrierten Schaltungsterminals **32** verbunden sein. Bei einer alternativen Ausführungsform der vorliegenden Erfindung muss die optionale Speicherschaltung **104** nicht verwirklicht sein. Bei Ausführungsformen der vorliegenden Erfindung, bei denen die optionale Speicherschaltung **104** verwirklicht ist, kann sie unter Verwendung der Register, jeder Art von Speicher, jeder Art von Speicherschaltung, die Signalspeicher oder programmierbare, logische Arrays enthalten, etc. implementiert sein. Bei alternativen Ausführungsformen der vorliegenden Erfindung kann die Berechnungsschaltung **102** jede Art von logischen oder Rechenfunktionen durchführen.

[0039] Das System unterstützt die Aufgabenbeschleunigung durch einen externen Koprozessor **14** (oder Hardware-Beschleuniger), der für Betriebsweisen optimiert ist, die sich auf spezielle Anwendungen beziehen. Diese externen Koprozessoren **14**, **16** können so einfach wie ein Koprozessor **14** zur Durchführung einer Zählung von Dateneinträgen sein oder eine kompliziertere Funktion erfüllen, wie etwa ein DSP-Beschleunigungskoprozessor **14** oder ein Koprozessor **14**, der in der Lage ist, Multiplikations-/Akkumulations-Operationen mit hoher Geschwindigkeit durchzuführen.

[0040] Daten werden zwischen dem Prozessor **12** und einem Koprozessor **14** mittels einer oder mehrerer von verschiedenen Mechanismen übertragen, je nach Angemessenheit bei der speziellen Implementation. Diese können eingeteilt werden in Transfers zu dem Koprozessor **14** und Transfers von dem Koprozessor **14**.

[0041] Einer der Datentransfermechanismen zum Übertragen von Daten zu einem Koprozessor **14** ist der Register-Snooping-Mechanismus, der kein Anweisungsgrundelement enthält, sondern ein Nebenprodukt des normalen Betriebs des Prozessors **12** ist. Dieser beinhaltet die Spiegelung von Aktualisierungen in die allgemeinen Register (GPR: General Purpose Registers) **46** des Prozessors **12** über die Schnittstelle, so dass der Koprozessor **14** die Aktualisierungen in einem oder mehreren Registern des Prozessors **12** überwachen kann. Dies kann angemessen sein, wenn ein Koprozessor **14** ein GPR **46** als internes Register oder interne Funktion "überlagert". In diesem Fall ist kein expliziter Übergang von Parametern von dem Prozessor **12** an einen Koprozessor **14** erforderlich.

[0042] Anweisungsgrundelemente werden in dem Basisprozessor **12** zum expliziten Transfer von Operanden und Anweisungen zwischen externen Koprozessoren **14**, **16** und dem Prozessor **12** bereitgestellt. Ein Handshake-Mechanismus wird zur Verfügung gestellt, um die Kontrolle über die Rate von Anweisungen und des Datentransfers zu gestatten.

[0043] Man beachte, dass die Funktionen des Koprozessors **14** ausgelegt sind, implementationsspezifische Einheiten darzustellen, weshalb die exakte Funktionalität einer gegebenen Einheit frei ist und über verschiedene Implementationen hinweg verändert werden kann, selbst wenn dieselben Anweisungsumsetzungen vorliegen sollten.

[0044] **Fig.** 4 ist ein Timingdiagramm, das eine Register-Snooping-Operation illustriert. Um den zusätzlichen Aufwand der Parameterübergabe zu einem Koprozessor **14** oder um eine externen Überwachung zu vermeiden, wird ein Register-Snooping-Mechanismus bereitgestellt. Dies gestattet es einem Koprozessor **14**, eine Schattenkopie eines oder mehrerer allgemeiner Register **46** des Prozessors **12** zu implementieren. Diese Fähigkeit wird dadurch verwirklicht, dass der Wert, der in eines der GPRs **46** des Prozessors geschrieben wird sowie ein Hinweis darauf, welches Register **46** für jede GPR-Aktualisierung aktualisiert wird, übertragen wird. Für jede Registeraktualisierung wird ein Freigabesignal REGWR* 67 gesetzt. Der Wert wird über den bidirektionalen 32-Bit Datenpfad HDP [31 : 0] 72 übertragen, und ein 5-Bit Registerzahlenbus liefert einen Pointer auf das aktuelle Prozessorregister **46**, das aktualisiert wird (REG [4 : 0]) 68. Die Registerzahl kann Bezug nehmen auf ein Register **46** in einer normalen Datei oder einer Ausweichdatei. Bei dem bevorzugten Ausführungsbeispiel werden die Register der Ausweichdatei als REG [**4**]==1 bezeichnet und Register der normalen Datei durch REG [**4**]==0. Man beachte jedoch, dass diese Erfindung in keiner Weise von der tatsächlichen Einteilung des Registersatzes abhängt.

[0045] Ein Koprozessor **14** kann den Wert intern zu-

sammen mit einem Hinweis auf die Zahl des Zielregisters **46** latchen bzw. speichern, um später eine ausdrückliche Bewegung zu vermeiden. Diese Funktionalität kann auch von einem Fehlersuch-Koprozessor **14** verwendet werden, um den Zustand der Registerdatei **46** oder einer Untergruppe davon zu verfolgen. **Fig.** 4 zeigt ein Beispiel des Snooping-Vermögens.

[0046] Ein spezieller 12-Bit Anweisungsbus (H_OP[11 : 0]) 61 stellt den Operationscode der Koprozessorschnittstelle **30** bereit, der an den externen Koprozessor **14** ausgegeben wird. Dieser Bus spiegelt die unteren 12 Bit des Operationscodes des Prozessors. Die oberen 4 Bit werden nicht gespiegelt, da sie stets 0b0100 sind. Der Überwachungsmodus-Indikator (H_SUP) **62** wird ebenfalls bereitgestellt, um den aktuellen Zustand des PSR(S)-Bit anzuzeigen, welches anzeigt, ob der Prozessor im Überwachungs- oder Benutzermodus arbeitet. Dies kann nützlich sein, um bestimmte Koprozessorfunktionen auf den Überwachungsmodus zu limitieren. Ein Satz von Handshake-Signalen zwischen dem Prozessor **12** und den externen Koprozessoren **14**, **16** koordiniert die Ausführung der Koprozessor-Schnittstellenanweisungen.

[0047] Die von dem Prozessor **12** erzeugten Steuersignale spiegeln die interne Pipelinestruktur des Prozessors **12** wider. Die Prozessorpipeline **44** besteht aus Stufen für Befehlsabruf, Befehlsdekodierung **42**, Ausführung und Ergebniswiedergabe. Sie enthält ein oder mehrere Anweisungsregister (IR: Instruction Registers). Der Prozessor **12** enthält auch einen Vorabruf-Befehlspuffer, um die Zwischenspeicherung einer Anweisung vor der Dekodierungsstufe **42** zu gestatten. Die Anweisungen gelangen von diesem Puffer zu der Anweisungsdekodierungsstufe **42**, indem sie in das Anweisungsdekodierungsregister IR eingehen.

[0048] Der Anweisungsdekodierer **42** empfängt Eingaben von den IR und erzeugt Ausgaben auf Grundlagen des in dem IR gehaltenen Wertes. Diese Ausgaben des Dekodierers **42** sind nicht immer gültig und können aufgrund von Ausnahmebedingungen oder Änderungen im Anweisungsfluss verworfen werden. Selbst wenn sie gültig sind, können die Anweisungen in den IR gehalten werden, bis sie an die Ausführungsstufe der Anweisungspipeline weiter fortschreiten können. Da dies nicht geschehen kann, bis vorangegangene Anweisungen vollständig abgearbeitet sind (was mehrere Takteinheiten in Anspruch nehmen kann), fährt der Dekodierer fort, den in den IR enthaltenen Wert zu dekodieren, bis das IR aktualisiert wird.

[0049] **Fig.** 5 ist ein Timingdiagramm, das die grundlegende Betriebsweise zur Schnittstellenanweisung für die Anweisung eines Handshakes illustriert. Ein Anweisungsdekodierungs-Freigabesignal (H_DEC*) 63 wird bereitgestellt, um die Dekodierung eines Operationscodes der Koprozessorschnittstelle **30** durch den Prozessor **12** anzuzeigen. Dieses Signal wird gesetzt, wenn in dem IR ein Operationscode der Koprozessorschnittstelle **30** liegt, selbst wenn die Anweisung ohne Ausführung verworfen werden kann. Die H_DEC*-Ausgabe 63 kann für dieselbe Anweisung während mehrerer Takte gesetzt bleiben, bis die Anweisung tatsächlich ausgegeben oder verworfen wird.

[0050] Ein "Beschäftigt"-Signal (H_BUSY*) 64 wird von dem Prozessor **12** überwacht, um zu bestimmen, ob ein externer Koprozessor **14** eine Koprozessor-Schnittstellenanweisung **30** annehmen kann und steuert teilweise, wann die Befehlsausgabe erfolgt. Wenn das H_BUSY*-Signal 64 negiert ist, während H_DEC* 63 gesetzt ist, wird die Befehlsausführung von der Schnittstelle nicht verzögert und das H_EXEC*-Signal 65 kann gesetzt werden, sobald die Befehlsausführung durchgeführt werden kann. Wenn das H_BUSY*-Signal 64 gesetzt ist, wenn der Prozessor **12** einen Koprozessor-Schnittstellen-Operationscode **30** dekodiert (dadurch angezeigt, dass H_DEC* 63 gesetzt ist), wird die Ausführung des Koprozessor-Schnittstellen-Operationscodes **30** zur Verzögerung gezwungen. Sobald das H_BUSY*-Signal 64 negiert ist, kann der Prozessor **12** die Anweisung durch Setzen von H_EXEC* 65 ausgeben. Wenn ein Koprozessor **14** zur Speicherung von Anweisungen fähig ist, kann das H_BUSY*-Signal 64 verwendet werden, um beim Auffüllen des Zwischenspeichers zu helfen.

[0051] **Fig.** 6 ist ein Timingdiagramm, das den Betrieb der Anweisungsschnittstelle illustriert, wenn H_BUSY* 64 verwendet wird, um die Ausführung einer Koprozessor-Schnittstellenanweisung **30** zu steuern. Sobald irgendeine interne Verzögerungsbedingung aufgelöst ist und das H_BUSY*-Signal 64 negiert wird, kann der Prozessor H_EXEC* 65 setzen, um anzuzeigen, dass die Koprozessor-Schnittstellenanweisung **30** die Ausführungsstufe der Pipeline erreicht hat. Ein externer Koprozessor **14** sollte das H_EXEC*-Signal 65 überwachen, um die tatsächliche Befehlsausführung zu kontrollieren, da es für den Prozessor möglich ist, die Anweisung unter gewissen Umständen vor der Ausführung zu verwerfen. Wenn die Ausführung einer früheren Anweisung zu einer Ausnahme führt, wird das H_EXEC*-Signal 65 nicht gesetzt, und die H_DEC*-Ausgabe 63 wird negiert. Ein ähnlicher Prozess kann auftreten, wenn die Anweisung in dem IR als Resultat einer Änderung im Programmfluss verworfen wird.

[0052] **Fig.** 7 ist ein Timingdiagramm, das ein Verwerfen einer Anweisung illustriert. Wenn eine Anweisung verworfen wird, wird das H_DEC*-Signal 63 negiert, bevor ein anderer Koprozessor-Schnittstellen-Operationscode **30** in den H_OP [11 : 0] -Bus 61 gestellt wird.

[0053] **Fig.** 8 ist ein Timingdiagramm, das ein Beispiel einer Verzögerung der Anweisungspipeline illustriert. Es gibt Umstände, bei denen der Prozessor **12** das Setzen von H_EXEC* 65 verzögern kann, obwohl H_DEC* 63 gesetzt und H_BUSY* 64 negiert

ist. Dies kann während des Wartens auf die Vollendung einer vorangegangenen Anweisung auftreten.

[0054] **Fig.** 9 ist ein Timingdiagramm, das ein Beispiel einer End-zu-End-Ausführung ohne Verzögerungen illustriert. Für End-zu-End-Koprozessor-Schnittstellenanweisungen **30** kann das H_DEC*-Signal 63 ohne Negierung gesetzt bleiben, obwohl der H_OP[11 : 0]-Bus 61 aktualisiert wird, wenn neue Anweisungen im IR eingehen. Im Allgemeinen entspricht das Setzen von H_EXEC* 65 der Ausführung der Anweisung, die im vorangegangenen Takt dekodiert wurde.

[0055] **Fig.** 10 ist ein Timingdiagramm, das eine End-zu-End-Operation mit internen Pipelineverzögerungen illustriert. In diesem Fall ist H_BUSY* 64 negiert, der Prozessor setzt jedoch H_EXEC* 65 für die zweite Koprozessor-Schnittstellenanweisung **30** nicht, bis die interne Verzögerungsbedingung verschwindet.

[0056] **Fig.** 11 ist ein Timingdiagramm, das End-zu-End-Koprozessor-Schnittstellenanweisungen **30** mit H_BUSY*-Verzögerungen **64** illustriert. In diesem Beispiel ist der Koprozessor **14** beschäftigt und kann nicht unverzüglich die zweite Anweisung annehmen. H_BUSY* 64 wird gesetzt, um den Prozessor **12** davon abzuhalten, die zweite Anweisung auszugeben. Sobald der Koprozessor **14** frei wird, wird H_BUSY* 64 negiert und die nächste Koprozessor-Schnittstellenanweisung **30** rückt in die Ausführungsstufe vor.

[0057] Ausnahmen, die in Bezug stehen zu der Dekodierung eines Koprozessorschnittstellen-Operationscodes, 30 können von dem externen Koprozessor **14** mit dem H_EXCP*-Signal 66 angezeigt werden. Diese Eingabe des Prozessors **12** wird während des Taktgeberzyklus gesampelt, indem H_DEC* 63 gesetzt und H_BUSY* 64 negiert wird und führt zu einer Hardware-Koprozessorausnahme, falls der Koprozessor-Schnittstellen-Operationscode **30** nicht, wie zuvor beschrieben, verworfen wird. Details dieser Ausnahmebearbeitung sind weiter unten beschrieben.

[0058] **Fig.** 12 ist ein Timingdiagramm, das ein Beispiel für ein H_EXCP*-Signal 66 illustriert, das von dem Koprozessor **14** gesetzt wird als Antwort auf die Dekodierung und versuchte Ausführung eines Koprozessor-Schnittstellen-Operatonscodes **30**. Das H_EXCP-Signal **66** wird von dem Prozessor **12** während des Taktes gesampelt, indem H_DEC* 63 gesetzt und H_BUSY* 64 negiert wird. Das H_EXEC*-Signal 65 wird gesetzt unabhängig davon, ob von der Schnittstelle eine Ausnahme angezeigt wird. Dieses Setzen unterscheidet den Fall, dass eine Ausnahme vorliegt, von dem Fall, dass eine Anweisung verworfen wird.

[0059] Man beachte, dass die Ausführung der im vorangegangenen Taktzyklus dekodierten Anweisung entspricht und dass keine tatsächliche Ausführung stattfinden sollte. Ein Koprozessor **14** muss die verletzende Anweisung annehmen und eine Ausnahme anzeigen, bevor sie von der Ausführungsstufe der Prozessorpipeline erkannt wird. Das H_EXCP*-Signal 66 wird für alle Taktzyklen ignoriert, wenn H_DEC* 63 negiert oder H_BUSY* 64 gesetzt ist.

[0060] **Fig.** 13 ist ein Timingdiagramm, das ein Beispiel eines H_EXCP*-Signals 66 illustriert, das von dem Koprozessor **14** gesetzt wird als Antwort auf die Dekodierung und versuchte Ausführung eines Koprozessor-Schnittstellen-Operationscodes **30**. Im Gegensatz zu dem Timingdiagramm von **Fig.** 14 wird in diesem Beispiel die Koprozessor-Schnittstellen-Anweisung **30** verworfen, so dass das H_EXEC*-Signal **65** nicht gesetzt und das H_DEC* 63 negiert wird.

[0061] **Fig.** 14 ist ein Timingdiagramm, das ein Beispiel illustriert, bei dem H_BUSY* 64 gesetzt ist, um die Ausführung eines Koprozessor-Schnittstellen-Operationscodes **30** zu verzögern, was zu einer Ausnahme führt.

[0062] Die Signale H_BUSY* 64 und H_EXCP* 66 werden von allen Koprozessoren **14**, **16** gemeinsam genutzt, so dass sie in koordinierter Weise betrieben werden müssen. Diese Signale sollten von den Koprozessoren **14**, **16** entsprechend H_OP [11 : 10] 61 bei denjenigen Taktzyklen angesteuert werden (entweder high oder low, was immer angemessen ist), bei denen H_DEC* 63 gesetzt ist. Durch Ansteuerung der Ausgabe nur während des Low-Bereiches des Taktgebers können diese Signale von mehreren Koprozessoren **14**, **16** ohne Konkurrenz gemeinsam genutzt werden. Ein Haltespeicher in dem Prozessor **12** ist für diese Eingabe vorgesehen, um ihn in einem gültigen Zustand für die High-Phase des Taktgebers zu halten, während sie von keiner Einheit angesteuert wird.

[0063] Einige der Koprozessor-Schnittstellen-Anweisungsgrundelemente **30** umfassen auch einen Transfer von Datenposten zwischen dem Prozessor **12** und einem externen Koprozessor **14**. Operanden können über die Koprozessor-Schnittstelle **30** als Funktion der besonderen, ausgeführten Grundelemente übertragen werden. Es sind Vorkehrungen zur Übertragung eines oder mehrerer der Prozessor-GPRs vom oder zum Koprozessor **14** über einen bidirektionalen 32 Bit Datenpfad getroffen. Außerdem sind Vorkehrungen getroffen, einen einzelnen Datenposten in den oder von dem Speicher **18** zu laden oder zu speichern, wobei als Datensenke/Quelle die Koprozessor-Schnittstelle **30** dient. Der Prozessor **12** schickt Parameter über den HDP[31 : 0]-Bus 72 an die externen Koprozessoren **14**, **16**, während des High-Bereiches der CLK **60**. Operanden von der Koprozessor-Schnittstelle **30** werden von dem Prozessor **12** während der Low-Phase des Taktgebers empfangen und gespeichert. Eine Verzögerung ist vorgesehen, wenn der Taktgeber in einen High-Zustand übergeht, bevor eine Ansteuerung erfolgt, um eine kurze Zeitspanne vor einem Bus-Handoff zu ermöglichen. Eine Koprozessor-Schnittstelle **14** muss dieselbe kurze Verzögerung an der abfallenden Taktgeberkante bereitstellen. Das Handshaking von Da-

tenposten wird unterstützt von den Datenfreigabeausgangs(H_DS* 70), den Datenbestätigungseingangs- (H_DA* 71) und den Datenfehlerausgangs-Signalen (H_ERR* 69).

[0064] Der Prozessor **12** bietet die Fähigkeit zum Transfer einer Liste von Aufruf- und Rückgabe-Parametern an die Koprozessor-Schnittstelle **30** auf im Wesentlichen die gleiche Weise, wie Software-Subroutinen aufgerufen werden bzw. wie an sie zurückgegeben wird. In den H_CALL- oder H_RET-Grundelementen ist ein Argumentenzähler angezeigt zur Steuerung der Anzahl durchgesetzter Parameter. Die Registerwerte werden, beginnend mit dem Inhalt des Registers R4 des Prozessors **12** zum (vom) externen Koprozessor **14** als Teil der Ausführung der H_CALL (H_RET)-Grundelemente übermittelt. Bis zu 7 Registerparameter können durchgesetzt werden. Diese Konvention ist ähnlich der Software-Subroutinen-Aufrufskonvention.

[0065] Handshaking der Operandentransfers wird von dem Datenfreigabe-Ausgangssignal (H_DS* 70) und dem Datenbestätigungseingangssignal (H_DA* 71) gesteuert. Datenfreigabe wird von dem Prozessor **12** für die Dauer der Transfers gesetzt und die Transfers erfolgen in überlappender Weise, im Wesentlichen auf die gleiche Weise wie die Prozessor-Schnittstellen-Operation. Datenbestätigung (H_DA* 71) wird verwendet, um anzuzeigen, dass ein Datenelement akzeptiert oder von einem Koprozessor **14** angesteuert wurde.

[0066] **Fig.** 15 ist ein Timingdiagramm, das ein Beispiel eines Registertransfers **46** in Verbindung mit dem H_CALL-Grundelement illustriert. Anweisungsgrundelemente werden bereitgestellt, um eine Mehrzahl von Prozessorregistern zu transferieren, und die Transfers können Idealerweise bei jedem Takt erfolgen. Für Transfers zu einem externen Koprozessor **14** fängt der Prozessor automatisch an, den nächsten Operanden (falls notwendig) vor (oder konkurrierend mit) der Bestätigung des aktuellen Elementes anzusteuern. Die externe Logik muss zu einem Zwischenspeicherpegel fähig sein, um sicherzustellen, dass keine Daten verloren gehen. Diese Figur zeigt den Ablauf eines H_CALL-Transfers zu der Koprozessor-Schnittstelle **30**, wobei zwei Register transferiert werden müssen. Der zweite Transfer wird aufgrund einer negierten Datenbestätigung (H_DA*) 71 wiederholt.

[0067] Für Transfers von einem externen Koprozessor **14** zu den Prozessorregistern **46** kann der Prozessor **12** Werte von einem externen Koprozessor **14** während jedes Taktgeberzyklus akzeptieren, nachdem H_DS* 70 gesetzt wurde, und diese Werte werden in die Registerdatei **46** geschrieben, sobald sie empfangen werden, so dass keine Zwischenspeicherung erforderlich ist.

[0068] **Fig.** 16 ist ein Timingdiagramm, das ein Beispiel von Registertransfers **46** in Verbindung mit dem H_RET-Grundelement illustriert. Bei diesem Beispiel werden zwei Registerwerte **46** transferiert. Der Koprozessor **14** kann zu Beginn des Taktes, der auf das Setzen des H_EXEC*-Signals 65 folgt, beginnen, Daten zu schicken, da dies derjenige Takt ist, in dem H_DS* 70 zuerst gesetzt wird. Die H_DS*-Ausgabe **70** ändert sich mit der ansteigenden Flanke von CLK **60**, während die H_DA*-Eingabe 71 während der Low-Phase von CLK **60** gesampelt wird.

[0069] Der Prozessor **12** bietet die Fähigkeit, einen einzelnen Speicheroperanden mit den H_LD- oder H_ST-Anweisungs-Grundelementen zu oder von der Koprozessor-Schnittstelle **30** zu transferieren.

[0070] Das H_LD-Grundelement wird verwendet, um Daten aus dem Speicher **18** zu dem Koprozessor **14** zu transferieren. Das Handshaking des Operandentransfers zum Koprozessor **14** wird von dem Datenfreigabesignal (H_DS*) 70 gesteuert. Die Datenfreigabe wird von dem Prozessor **12** gesetzt, um anzuzeigen, dass ein gültiger Operand in den HDP [31 : 0]-Bus 72 gestellt wurde. Die Datenbestätigungseingabe (H_DA*) 71 wird für diesen Transfer ignoriert.

[0071] **Fig.** 17 ist ein Timingdiagramm, das den Ablauf eines H_LD-Transfers an die Koprozessor-Schnittstelle **30** illustriert. In diesem Fall gibt es einen Zugriff auf den Speicher **18** mit No-Wait-Zustand. Bei Zugriffen auf den Speicher **18** mit n-Wait-Zuständen, würde der Operand und H_DS* 70 n Takte später verschickt. Wird die Option, das Basisregister **46** mit der effektiven Ladeadresse zu aktualisieren, gewählt, wird der Aktualisierungswert auf HDP [31 : 0] 72 beim ersten Takt, nachdem er berechnet wurde, angesteuert (beim Takt der dem Setzen von H_EXEC* 65 folgt).

[0072] **Fig.** 18 ist ein Timingdiagramm, das das Protokoll illustriert, wenn ein Zugriff auf den Speicher **18** zu einer Zugriffsausnahme führt. In einem solchen Fall wird das H_ERR*-Signal 69 zu dem externen Koprozessor **14** zurückgesetzt.

[0073] Das H_ST-Grundelement kann für einen Datentransfer von einem Koprozessor **14** zum Speicher **18** verwendet werden. Wenn die Option, das Basisregister **46** mit der effektiven Speicheradresse zu aktualisieren, ausgewählt ist, wird der Aktualisierungswert auf HDP [31 : 0] 72 beim ersten Takt, nachdem er berechnet wurde, angesteuert (der Takt, der dem Setzen von H_EXEC* 65 folgt).

[0074] **Fig.** 19 ist ein Timingdiagramm, das ein Beispiel eines mit dem H_ST-Grundelement in Beziehung stehenden Transfers illustriert. Das Handshaking, das mit dem H_ST-Grundelement in Beziehung steht, besteht aus zwei Teilen, einem Anfangshandshake vom Koprozessor **14**, der Speicherdaten liefern muss, und ein Vollendungshandshake vom Prozessor **12**, sobald die Speicherung im Speicher **18** vollendet ist.

[0075] Der Anfangshandshake benutzt die H_DA*-Eingabe 71 zum Prozessor **12**, um zu signalisieren, dass der Koprozessor **14** die Speicherdaten dem Prozessor **12** übergeben hat. Das H_DA*-Signal 71 wird in demselben Takt gesetzt, in dem die Daten von dem Koprozessor **14** auf den HDP [31 : 0]-Bus 72

geführt werden. Die Speicherdaten werden zur Speicherung einer halben Wortlänge von der unteren Hälfte des Busses genommen. Die oberen 16 Bit werden nicht in den Speicher **18** geschrieben. Das H_DA*-Signal 71 wird, beginnend mit dem Takt, in dem das H_EXEC*-Signal 65 gesetzt wird, gesampelt. Der Speicherzyklus muss während des Taktes, in dem H_DA* 71 erkannt wird, erfolgen, und die Speicherdaten werden im folgenden Takt dem Speicher **18** zugeführt. Sobald die Speicherung vollendet ist, setzt der Prozessor **12** das H_DS*-Signal **70**.

[0076] **Fig.** 20 ist ein Timingdiagramm, das ein Beispiel eines Transfers mit verzögerten Speicherdaten illustriert.

[0077] **Fig.** 21 ist ein Timingdiagramm, das die Protokollsignale illustriert, wenn die Speicherung zu einem Zugriffsfehler führt. Man beachte hier, dass das H_ERR*-Signal 69 gesetzt ist. Wenn die Hardware-Einheit die Anweisung durch Setzen von H_EXCP* 66 in dem Takt abbricht, in dem H_EXEC* 65 gesetzt wird, sollte das H_DA*-Signal 71 nicht gesetzt werden.

[0078] Die **Fig.** 22 bis 26 illustrieren Anweisungen, die als Teil des Anweisungssatzes der Schnittstelle zu einem Hardware-Beschleuniger (oder Koprozessor) **14** bereitgestellt werden. Der Prozessor **12** interpretiert einige der Felder in den Grundelementen, andere werden allein vom Koprozessor **14** interpretiert.

[0079] **Fig.** 22 illustriert ein Anweisungsformat für das H_CALL-Grundelement. Diese Anweisung wird benutzt, um eine von dem Koprozessor **14** implementierte Funktion aufzurufen ("call"). Das Paradigma ist ähnlich einer Standardsoftware-Aufrufkonvention, jedoch im Handware-Kontext. Das H_CALL-Grundelement wird sowohl vom Prozessor **12** als auch vom Koprozessor **14** interpretiert, um eine Liste von "Aufrufparametern" oder Argumenten vom Prozessor **12** zu transferieren und eine spezielle Funktion im Koprozessor **14** anzustoßen.

[0080] Die W- und CODE-Felder des Anweisungswortes werden nicht vom Prozessor **12** interpretiert. Diese werden verwendet, um eine für den Koprozessor **14** spezifische Funktion zu spezifizieren. Das W-Feld kann einen speziellen Koprozessor **14**, **16** spezifizieren, und das CODE-Feld kann eine besondere Betriebsweise spezifizieren. Das CNT-Feld wird sowohl vom Prozessor **12** als auch vom Koprozessor **14** interpretiert und spezifiziert eine Anzahl von Registerargumenten, die an den Koprozessor **14** gehen.

[0081] Die Argumente werden von allgemeinen Register **46** verschickt, beginnend mit R4 und weiter bis R(4 + CNT – 1). Bis zu sieben Parameter oder Register **46** können in einem einzigen H_CALL-Aufruf verschickt werden.

[0082] Die H_Call-Anweisung kann verwendet werden, um einen modularen Modulaufruf zu implementieren. Von der Verwendung dieses Typs von Schnittstelle ist seit langem bekannt, dass es in Softwaresystemen zu einer höheren Zuverlässigkeit und weniger Fehlern führt. Funktionsparameter werden üblicherweise am Besten durch ihren Wert verschickt. Dies reduziert Seiteneffekte signifikant. In vielen Fällen verschicken moderne Compiler für blockstrukturierte Sprachen, wie etwa C und C++, kurze Sequenzen von Parametern oder Argumenten, um Funktionen oder Subroutinen im Register **46** aufzurufen. Diese Technik kann mit der H_CALL-Anweisung implementiert werden. Ein Compiler kann so konfiguriert werden, dass bis zu sieben Parameter oder Argumente, beginnend bei R4, in aufeinander folgende Register **46** geladen werden und dann die H_CALL-Anweisung erzeugt wird, welche die standardmäßige, vom Compiler erzeugte Subroutinen Verbindungsanweisung ersetzt.

[0083] **Fig.** 23 illustriert ein Anweisungsformat für das H_RET-Grundelement. Diese Anweisung wird für die "Rückkehr von" einer vom Koprozessor **14** implementierten Funktion verwendet. Das Paradigma ist ähnlich der Softwareaufrufskonvention, die vom Prozessor **12** verwendet wird, jedoch in einem Hardware-Kontext. Das H_RET-Grundelement wird sowohl vom Prozessor **12** als auch vom Koprozessor **14** verwendet, um eine Liste von "Rückkehrparametern" oder Werten von einem Koprozessor **14** zum Prozessor **12** zu transferieren.

[0084] Die W- und CODE-Felder des Anweisungswortes werden vom Prozessor **12** nicht interpretiert. Diese werden verwendet, um eine für den Koprozessor **14** spezifische Funktion zu spezifizieren. Das W-Feld kann eine Hardware-Einheit spezifizieren und das CODE-Feld kann eine besondere Betriebsweise oder einen Satz von Registern **46** im Koprozessor **14** für die Rückkehr spezifizieren. Das CNT-Feld wird sowohl vom Prozessor **12** als auch vom Koprozessor **14** interpretiert und spezifiziert die Anzahl von Registerargumenten **46**, die vom Koprozessor **14** zum Prozessor **12** geschickt werden sollen.

[0085] Die Argumente werden von den allgemeinen Registern **46** des Prozessors **12**, beginnend mit R4 und weiter bis R(4 + CNT-1), verschickt. Bis zu sieben Parameter (oder Registerinhalte) können zurückgegeben werden.

[0086] Wie bei der H_CALL-Anweisung kann auch die H_RET-Anweisung zur Implementierung modularer Programmierung verwendet werden. Strukturiertes Programmieren erfordert, dass Funktionsrückgabewerte am Besten über ihren Wert an die aufrufende Routine zurückgeschickt werden. Dies wird von Compilern oft auf effiziente Weise durchgeführt, indem ein oder mehrere Rückgabewerte in Register für eine Subroutinen- oder Funktionsrückkehr gestellt werden. Es sollte jedoch beachtet werden, dass die traditionelle, strukturierte Programmierung erwartet, dass eine Subroutine oder Funktion nach Aufruf der Subroutine oder Funktion unverzüglich zurückkehrt. Im Fall des Koprozessors **14** erfolgt die Ausführung oft asynchron mit derjenigen des aufrufenden Prozessors **12**. Die H_RET-Anweisung kann zur Resynchronisierung von Prozessor **12** und Koprozessor **14** verwendet werden. Der Prozessor **12** kann daher ein

oder mehrere Register **46** laden, den Koprozessor **14** mit einer oder mehreren H_CALL-Anweisungen aktivieren, andere, unabhängige Anweisungen abarbeiten und sich dann mit dem Koprozessor **14** resynchronisieren, während er einen Ergebniswert oder Ergebniswerte, die von der H_RET-Anweisung ausgegeben wurden, vom Koprozessor **14** empfängt.

[0087] **Fig.** 24 illustriert ein Anweisungsformat für das H_EXEC-Grundelement. Diese Anweisung wird verwendet, um eine Funktion zu initiieren oder um in einen von dem Beschleuniger implementierten Operationsmodus einzusteigen. Die H_EXEC-Anweisung kann verwendet werden, um eine Funktion in einem spezifischen Koprozessor **14**, **16**, wie er von dem W-Feld spezifiziert ist, zu spezifizieren. Das CODE-Feld wird vom Prozessor **12** nicht interpretiert, sondern ist vielmehr dem bestimmten Koprozessor **14**, **16** vorbehalten. Die W- und CODE-Felder des Anweisungswortes werden vom Prozessor **12** nicht interpretiert. Diese werden verwendet, um eine für den Koprozessor **14** spezifische Funktion zu spezifizieren. Das W-Feld kann einen spezifischen Prozessor **14**, **16** spezifizieren, und das CODE-Feld kann eine besondere Betriebsweise spezifizieren.

[0088] **Fig.** 25 illustriert ein Anweisungsformat für die H_LD-Anweisung. Diese Anweisung wird verwendet, um einen Wert vom Speicher **18** zu einem Koprozessor **14** zu verschicken, ohne den Speicheroperanden vorübergehend im allgemeinen Register (GPR: General Purpose Register) **46** zu speichern. Der Speicheroperand wird unter Verwendung eines Basispointers und eines Offsets adressiert.

[0089] Die H_LD-Anweisung führt das Laden eines Wertes in den Speicher **18** aus und verschickt den Speicheroperanden an den Koprozessor **14**, ohne ihn in einem Register **46** zu speichern. Die H_LD-Operation hat drei Optionen: w = Wort h = halbes Wort und u = Aktualisierung (update). DISP wird durch Skalierung des IMM2-Feldes mit der Größe der Ladedaten und Auffüllen mit Nullen erreicht. Dieser Wert wird zu dem Wert des Registers RX addiert, und ein Laden der spezifizierten Größe wird von dieser Adresse aus durchgeführt mit dem Ergebnis, dass das Laden an die Hardware-Schnittstelle **28** weitergereicht wird. Beim Laden von halben Wörtern werden die geholten Daten auf 32 Bit mit Nullen aufgefüllt. Wenn die u-Option spezifiziert ist, wird die effektive Ladeadresse, nachdem sie berechnet ist, in das Register RX 46 eingestellt.

[0090] Das W-Feld des Anweisungswortes wird vom Prozessor **12** nicht interpretiert. Dieses Feld kann einen speziellen Koprozessor **14**, **16** spezifizieren. Das SZ-Feld spezifiziert die Größe des Operanden (nur Halbwort oder Wort). Das DISP-Feld spezifiziert einen vorzeichenlosen Offsetwert, der zu dem Inhalt des von dem RBASE-Feld spezifizierten Registers addiert wird, um die effektive Ladeadresse zu bilden. Der Wert des DISP-Feldes wird mit der Größe des zu transferierenden Operanden skaliert. Das Up-Feld spezifiziert, ob das RBASE-Register **46** mit der effektiven Ladeadresse aktualisiert werden soll, nachdem diese berechnet ist. Diese Option erlaubt einen automatisch aktualisierten ("auto-update") Adressierungsmodus.

[0091] **Fig.** 26 illustriert ein Anweisungsformat für die H_ST-Anweisung. Diese Anweisung wird verwendet, um einen Wert von einem Koprozessor **14** an den Speicher **18** weiterzureichen, ohne den Speicheroperanden zwischenzeitlich in einem Register **46** des Prozessors **12** zu speichern. Der Speicheroperand wird unter Verwendung eines Basispointers und eines Offsets adressiert.

[0092] Das UU-Feld des Anweisungswortes wird vom Prozessor **12** nicht interpretiert. Vielmehr kann dieses Feld einen speziellen Koprozessor **14**, **16** spezifizieren. Das SZ-Feld spezifiziert die Größe des Operanden (nur Halbwort oder Wort). Das DISP-Feld spezifiziert einen vorzeichenlosen Offsetwert, der zu dem Inhalt des von dem RBASE-Feld spezifizierten Registers **46** addiert wird, um die effektive Speicheradresse zu bilden. Der Wert des DISP-Feldes wird mit der Größe des zu transferierenden Operanden skaliert. Das Up-Feld spezifiziert, ob das RBASE-Register mit der effektiven Speicheradresse aktualisiert werden soll, nachdem diese berechnet ist. Diese Option erlaubt einen Adressierungsmodus mit "automatischer Aktualisierung".

[0093] Die H_ST-Anweisung führt eine Speicherung eines Operanden vom Koprozessor **14** in den Speicher **18** aus, ohne ihn in einem Register **46** zu speichern. Die H_ST-Operation hat drei Optionen: w = Wort, h = halbes Wort und u = Aktualisierung (update). DISP wird durch Skalierung des IMM2-Feldes mit der Speichergröße und Auffüllen mit Nullen erreicht. Dieser Wert wird zu dem Wert des Registers RX addiert, und eine Speicherung der spezifizierten Größe wird an diese Adresse vorgenommen, wobei die Daten für diese Speicherung von der Hardware-Schnittstelle erhalten werden. Wenn die u-Option spezifiziert ist, wird die effektive Ladeadresse, nachdem sie berechnet ist, in das Register RX eingestellt.

[0094] Die H_LD-Anweisung und die H_ST-Anweisung liefern einen effizienten Mechanismus, um Operanden vom Speicher **18** zu einem Koprozessor **14** und von einem Koprozessor **14** zum Speicher **18** zu bewegen, ohne die zu bewegenden Daten über die Register **46** zu leiten. Die Offset- und Index-Vorkehrungen bieten einen Mechanismus, um effizient durch Arrays zu laufen. Diese Anweisungen sind daher innerhalb von Schleifen besonders nützlich. Es sollte beachtet werden, dass beide Anweisungen für jeden geladenen oder gespeicherten Operanden den Prozessor **12** mit dem Koprozessor **14** synchronisieren. Ist dies nicht erforderlich oder sogar nicht bevorzugt, kann man alternativ Daten zum Koprozessor **14** leiten, indem man ein ausgewähltes Register oder ausgewählte Register **46** wiederholt mit Daten aus dem Speicher **18** belädt und den Koprozessor **14** diese Beladungen detektieren lässt, da der Koprozes-

sor-Schnittstellen-Bus **30** auch für die Registersuchaktionen ("register snooping") verwendet wird.

[0095] Der Fachmann wird erkennen, dass Modifikationen und Variationen durchgeführt werden können, ohne sich von dem Umfang der Erfindung zu entfernen. Es ist daher vorgesehen, dass diese Erfindung all solche Variationen und Modifikationen umfasst, die in den Bereich der beigefügten Ansprüche fallen.

Patentansprüche

1. Verfahren zum Betreiben eines Prozessors (**12**), der eingerichtet ist, um mit einem mit ihm über einen Koprozessor-Bus (**30**) verbundenen Koprozessor (**14**, **16**) zusammenzuarbeiten, wobei der Koprozessor-Bus von dem System-Bus getrennt ist und der Prozessor in der Lage ist, bei einer Ausführung wenigstens einer Anweisung (H_CALL), welche ein Zähler-Feld, das eine Anzahl von an den Koprozessor zu transferierenden Argumenten definiert, und ein Code-Feld umfasst, mit dem Koprozessor über den Koprozessor-Bus ohne Verwendung des System-Busses zu kommunizieren, wobei das Verfahren die folgenden Schritte umfasst:

Empfangen der Anweisen,

Liefern des Zähler-Feldes und des Code-Feldes an den Koprozessor, über einen ersten Zyklus auf dem Koprozessor-Bus,

falls das Zähler-Feld einen Wert n hat, der größer als Null ist, liefern eines ersten Operanden an den Koprozessor über einen zweiten Zyklus auf dem Koprozessor-Bus, und

Vollenden der Anweisung.

2. Verfahren nach Anspruch 1, weiter umfassend die Schritte:

Empfangen eines ersten Eingangssignals (**64**) von dem Koprozessor über den Koprozessor-Bus während des ersten Zyklus,

wobei, falls das von dem Koprozessor über den Koprozessor-Bus während des ersten Zyklus empfangene erste Eingangsignal einen ersten Zustand hat, der erste Zyklus wiederholt wird, wobei der Schritt des Wiederholen des ersten Zyklus umfasst:

Liefern des Zähler- und des Code-Feldes an den Koprozessor über den ersten Zyklus auf dem Koprozessor-Bus, und

Empfangen des ersten Eingangssignals von dem Koprozessor über den Koprozessor-Bus während des ersten Zyklus, und

wobei, wenn das von dem Koprozessor über den Koprozessor-Bus während des ersten Zyklus empfangene erste Eingangssignal einen zweiten Zustand hat, ein erstes Ausgangssignal (**65**) an den Koprozessor über den zweiten Zyklus des Koprozessor-Busses geliefert wird.

3. Verfahren nach Anspruch 1, weiter umfassend die Schritte:

wenn ein Zähler-Feld-Wert n größer als 1 ist, dann während jedes von (n–1) Zyklen auf dem Koprozessor-Bus:

Liefern eines nächsten von (n–1) Operanden an den Koprozessor über den Koprozessor-Bus, und

Empfangen eines zweiten Eingangssignals vom Koprozessor über den Koprozessor-Bus.

4. Verfahren nach Anspruch 1, wobei der Prozessor geeignet ist, bei der Ausführung der Anweisung mit einer Mehrzahl von mit ihm über den Koprozessor-Bus verbundenen Koprozessoren zusammenzuarbeiten, wobei das Verfahren weiter den Schritt umfasst:

Liefern eines Identifikator-Feldes an die Mehrzahl von Koprozessoren über den Koprozessor-Bus während des ersten Zyklus, wobei ein Identifikator-Feld einen Wert hat, der einen ausgewählten der Mehrzahl von Koprozessoren eindeutig identifiziert.

5. Verfahren nach Anspruch 1, wobei der Prozessor eine Mehrzahl von Registern zum Speichern ausgewählter Operanden umfasst, und wobei der Schritt des Lieferns des ersten Operanden an den Koprozessor über den Koprozessor-Bus während des zweiten Zyklus weiter gekennzeichnet ist, durch:

wenn das Zähler-Feld einen Wert n hat, der größer als Null ist, Liefern eines in einem vorbestimmten der Mehrzahl von Registern gespeicherten Operanden an den Koprozessor über einen zweiten Zyklus auf dem Koprozessor-Bus.

6. Verfahren zum Betreiben eines Prozessors (**12**), der eingerichtet ist, um mit einem mit ihm über einen Koprozessor-Bus (**30**) verbundenen Koprozessor (**14**, **16**) zusammenzuarbeiten, wobei der Koprozessor-Bus von dem System-Bus getrennt ist und der Prozessor in der Lage ist, bei einer Ausführung von wenigstens einer Anweisung (H_RET), die ein Zähler-Feld, welches eine Anzahl von von dem Koprozessor zu empfangenden Argumenten definiert, und ein Code-Feld umfasst, mit dem Koprozessor über den Koprozessor-Bus ohne Verwendung des System-Busses zu kommunizieren, wobei das Verfahren die Schritte umfasst:

Empfangen der Anweisung,

Liefern des Zähler- und des Code-Feldes an den Koprozessor über einen ersten Zyklus auf dem Koprozessorbus-Bus,

wenn das Zähler-Feld einen Wert n hat, der größer als Null ist, Empfangen eines ersten Operanden von dem Koprozessor über einen zweiten Zyklus auf dem Koprozessor-Bus, und

Vollenden der Anweisung.

7. Verfahren nach Anspruch 6, weiter umfassend den Schritt des Empfangens eines ersten Eingangssignals (**64**) von dem Koprozessor über den Koprozessor-Bus während des ersten Zyklus, wobei:

wenn das von dem Koprozessor über den Koprozes-

sor-Bus während des ersten Zyklus empfangene erste Eingangssignal einen ersten Zustand hat, der erste Zyklus wiederholt wird, wobei der Schritt des Wiederholens des ersten Zyklus umfasst:

Liefern des Zähler- und des Code-Feldes an den Koprozessor über den ersten Zyklus auf dem Koprozessor-Bus, und

Empfangen des ersten Eingangssignals von dem Koprozessor über den Koprozessor-Bus während des ersten Zyklus, und

wenn das von dem Koprozessor über den Koprozessor-Bus während des ersten Zyklus empfangene erste Eingangssignal einen zweiten Zustand hat, Liefern eines ersten Ausgangssignals (**65**) an den Koprozessor über den zweiten Zyklus auf dem Koprozessor-Bus.

8. Verfahren nach Anspruch 6, weiter umfassend die Schritte:

wenn der Zähler-Feld-Wert n größer als eins ist, dann bei jedem von (n-1) Zyklen auf den Koprozessor-Bus:

Empfangen eines zweiten Eingangssignals von dem Koprozessor über den Koprozessor-Bus, wobei, falls das zweite Eingangssignal einen ersten Zustand hat, ein nächster von (n-1) Operanden von dem Koprozessor über den Koprozessor-Bus empfangen wird.

9. Verfahren nach Anspruch 6, wobei der Prozessor eingerichtet ist, um bei der Ausführung der Anweisung mit einer Mehrzahl von mit ihm über den Koprozessor-Bus verbundenen Koprozessoren zusammenzuarbeiten, wobei das Verfahren weiter den Schritt umfasst:

Liefern eines Identifikator-Feldes an die Mehrzahl von Koprozessoren über den Koprozessor-Bus während des ersten Zyklus, wobei das Identifikator-Feld einen Wert hat, der einen ausgewählten der Mehrzahl von Koprozessoren eindeutig identifiziert.

10. Verfahren nach Anspruch 6, wobei der Prozessor eine Mehrzahl von Registern zum Speichern ausgewählter Operanden umfasst, bei der Schritt des Empfangens des ersten Operanden von dem Koprozessor über den Koprozessor-Bus während des zweiten Zyklus weiter gekennzeichnet ist, durch:

wenn das Zähler-Feld einen Wert n hat, der größer als Null ist, Empfangen eines Operanden von dem Koprozessor über einen zweiten Zyklus auf dem Koprozessor-Bus und Speichern des Operanden in einem vorbestimmten der Mehrzahl von Registern.

Es folgen 25 Blatt Zeichnungen

Anhängende Zeichnungen

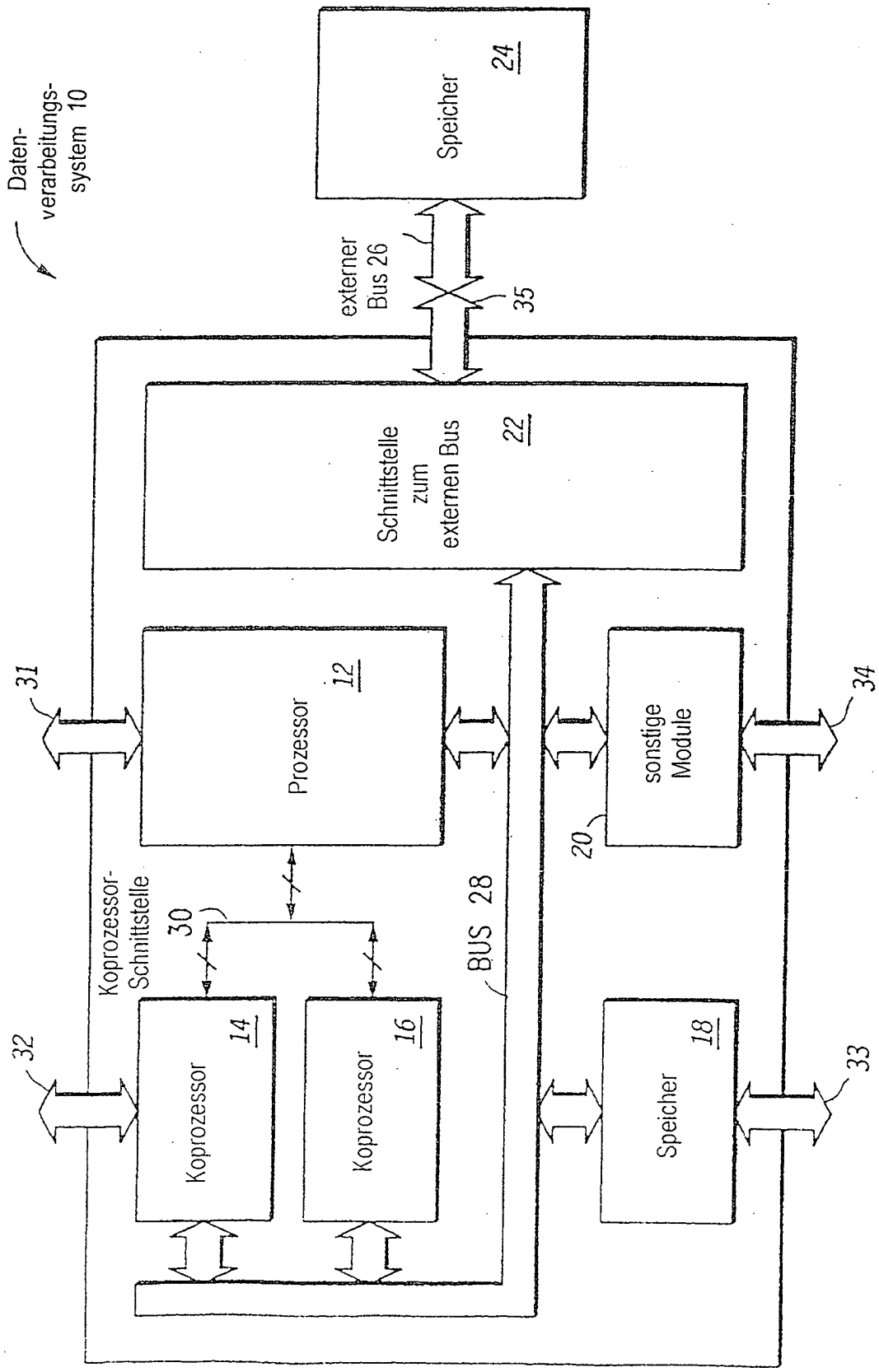
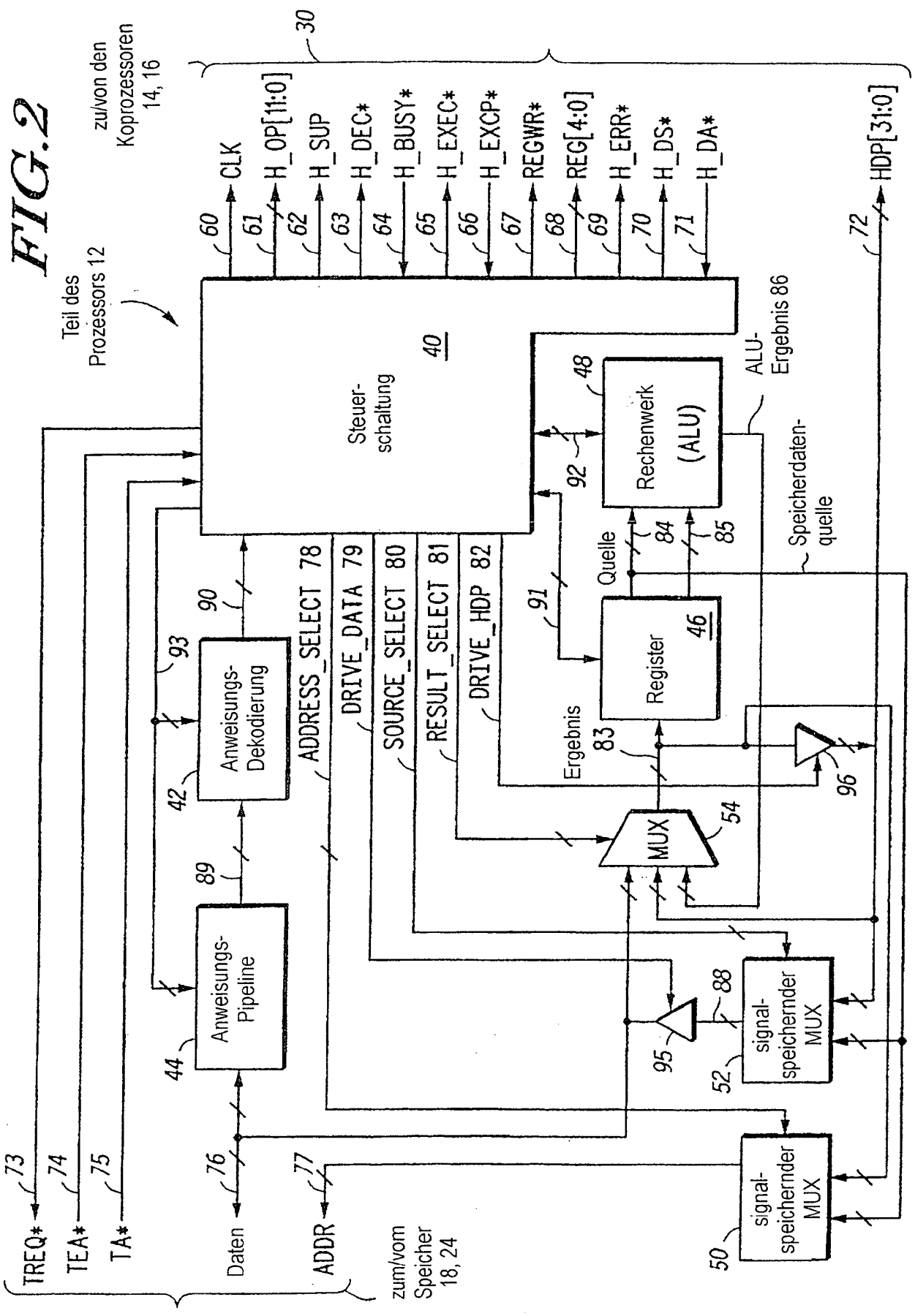


FIG.1

FIG. 2


Teil des Prozessors 12
zu/von den Koprozessoren 14, 16
30
CLK
H_OP[11:0]
H_SUP
H_DEC*
H_BUSY*
H_EXEC*
H_EXCP*
REGWR*
REG[4:0]
H_ERR*
H_DS*
H_DA*
HDP[31:0]
60
61
62
63
64
65
66
67
68
69
70
71
72
Steuer-schaltung
40
Rechenwerk (ALU)
48
92
ALU-Ergebnis 86
Speicherdaten-quelle
Quelle
84
85
91
Register
46
Ergebnis
83
96
MUX
54
Anweisungs-Dekodierung
42
90
93
ADDRESS_SELECT 78
DRIVE_DATA 79
SOURCE_SELECT 80
RESULT_SELECT 81
DRIVE_HDP 82
Anweisungs-Pipeline
44
89
95
88
signal-speichernder MUX
52
signal-speichernder MUX
50
TREQ*
TEA*
TA*
73
74
75
Daten
ADDR
76
77
zum/vom Speicher 18, 24

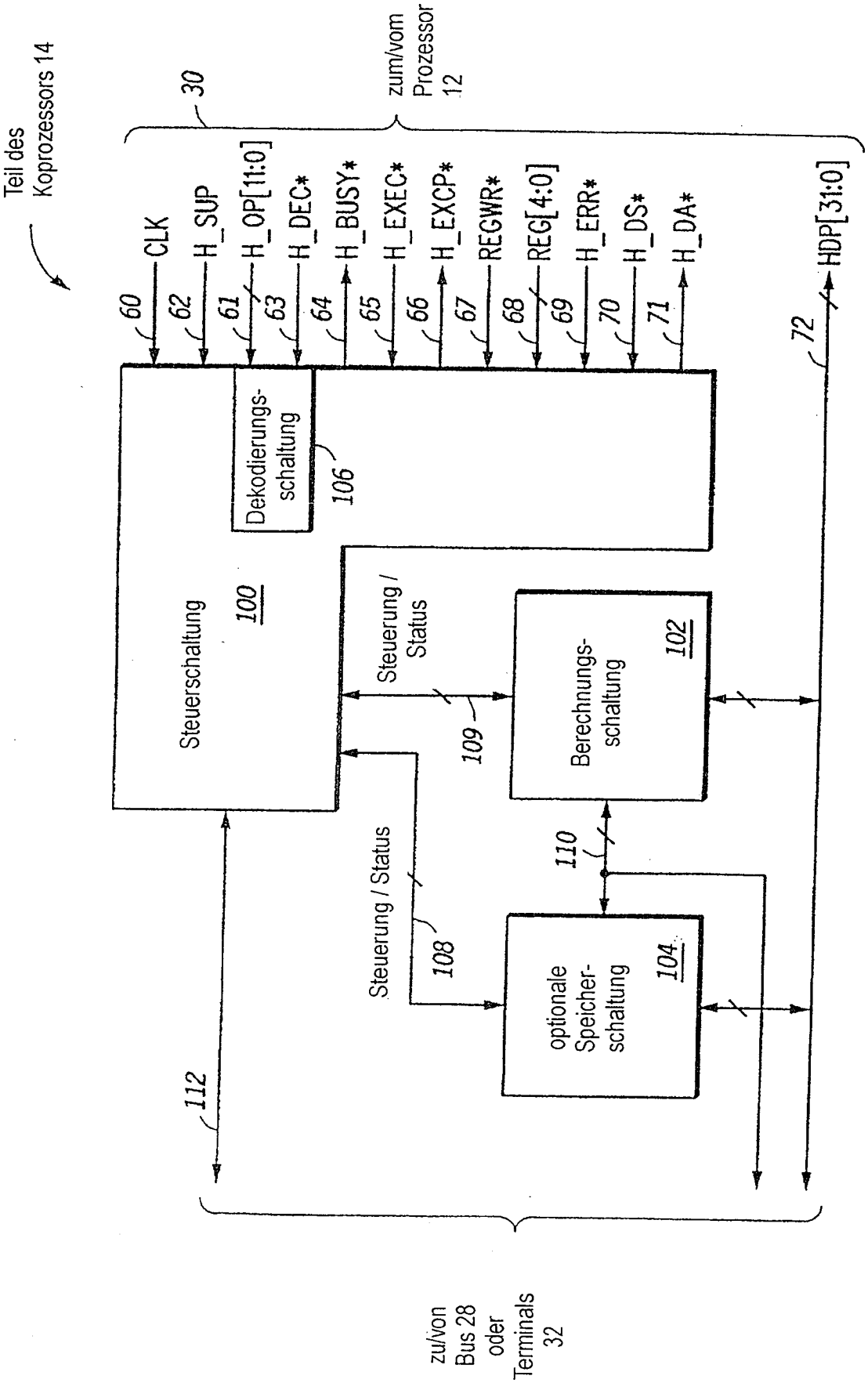

Teil des Koprozessors 14
30
zum/vom Prozessor 12
CLK
H_SUP
H_OP[11:0]
H_DEC*
H_BUSY*
H_EXEC*
H_EXCP*
REGWR*
REG[4:0]
H_ERR*
H_DS*
H_DA*
HDP[31:0]
60
62
61
63
64
65
66
67
68
69
70
71
72
Dekodierungs-schaltung
106
Steuerschaltung
100
Steuerung / Status
109
Berechnungs-schaltung
102
Steuerung / Status
108
110
optionale Speicher-schaltung
104
112
zu/von Bus 28 oder Terminals 32

FIG. 3

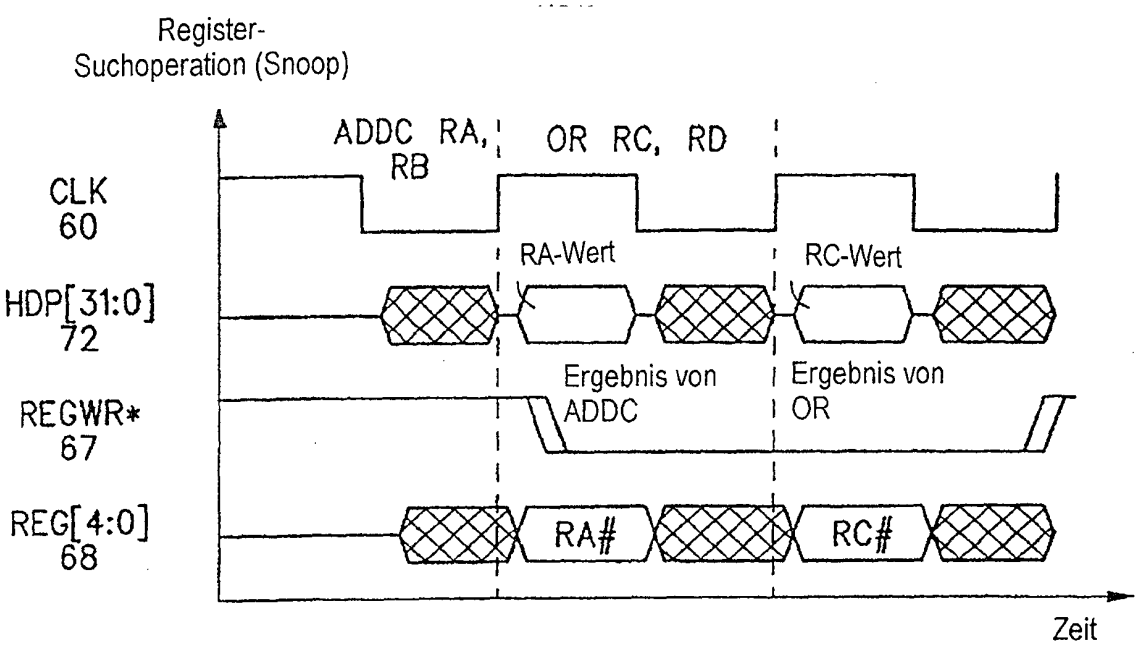

Register-
Suchoperation (Snoop)
ADDC RA, RB
OR RC, RD
CLK
60
RA-Wert
RC-Wert
HDP[31:0]
72
Ergebnis von
ADDC
Ergebnis von
OR
REGWR*
67
REG[4:0]
68
RA#
RC#
Zeit

FIG. 4

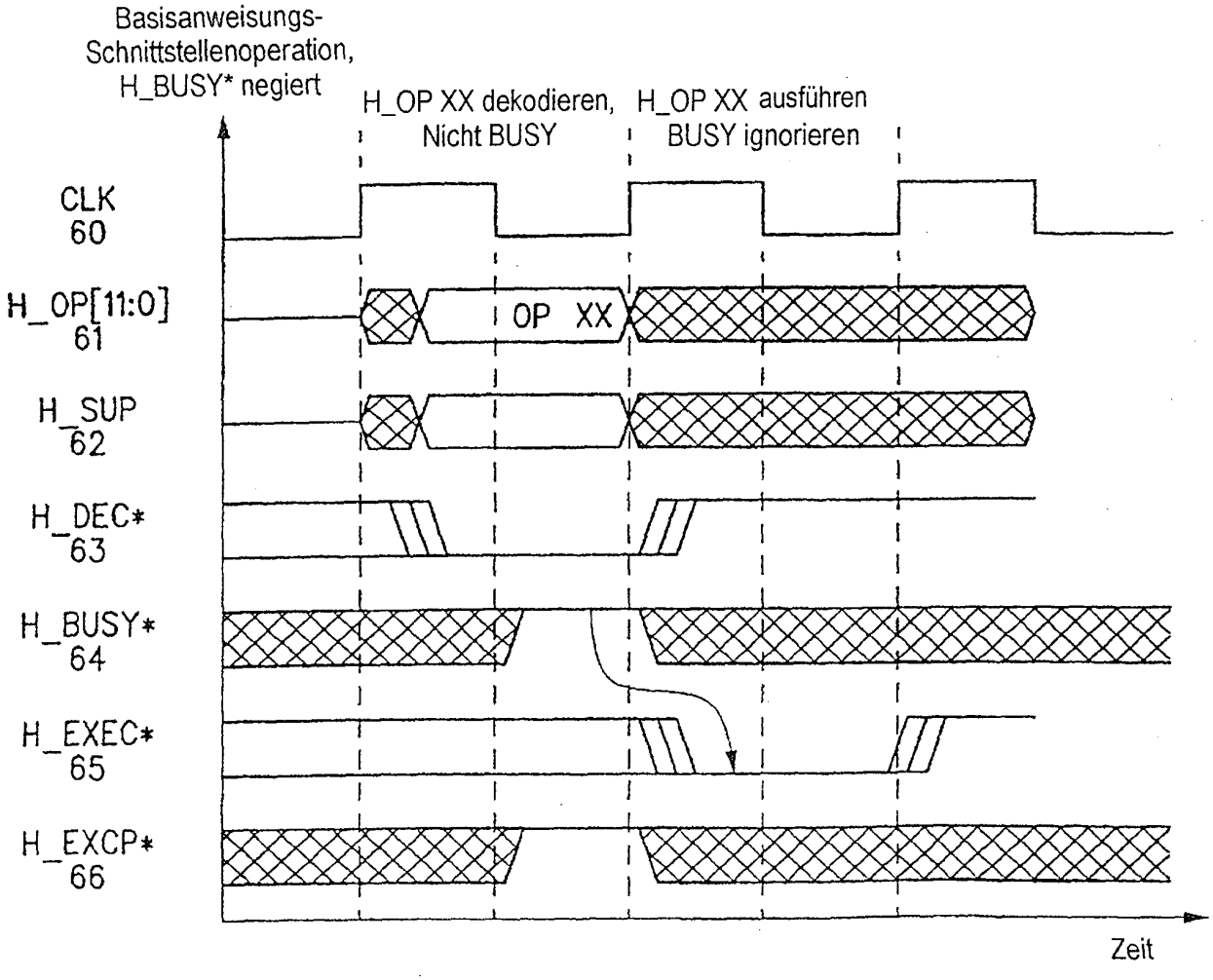

Basisanweisungs-
Schnittstellenoperation,
H_BUSY* negiert
H_OP XX dekodieren,
Nicht BUSY
H_OP XX ausführen
BUSY ignorieren
CLK
60
H_OP[11:0]
61
OP XX
H_SUP
62
H_DEC*
63
H_BUSY*
64
H_EXEC*
65
H_EXCP*
66
Zeit

FIG. 5

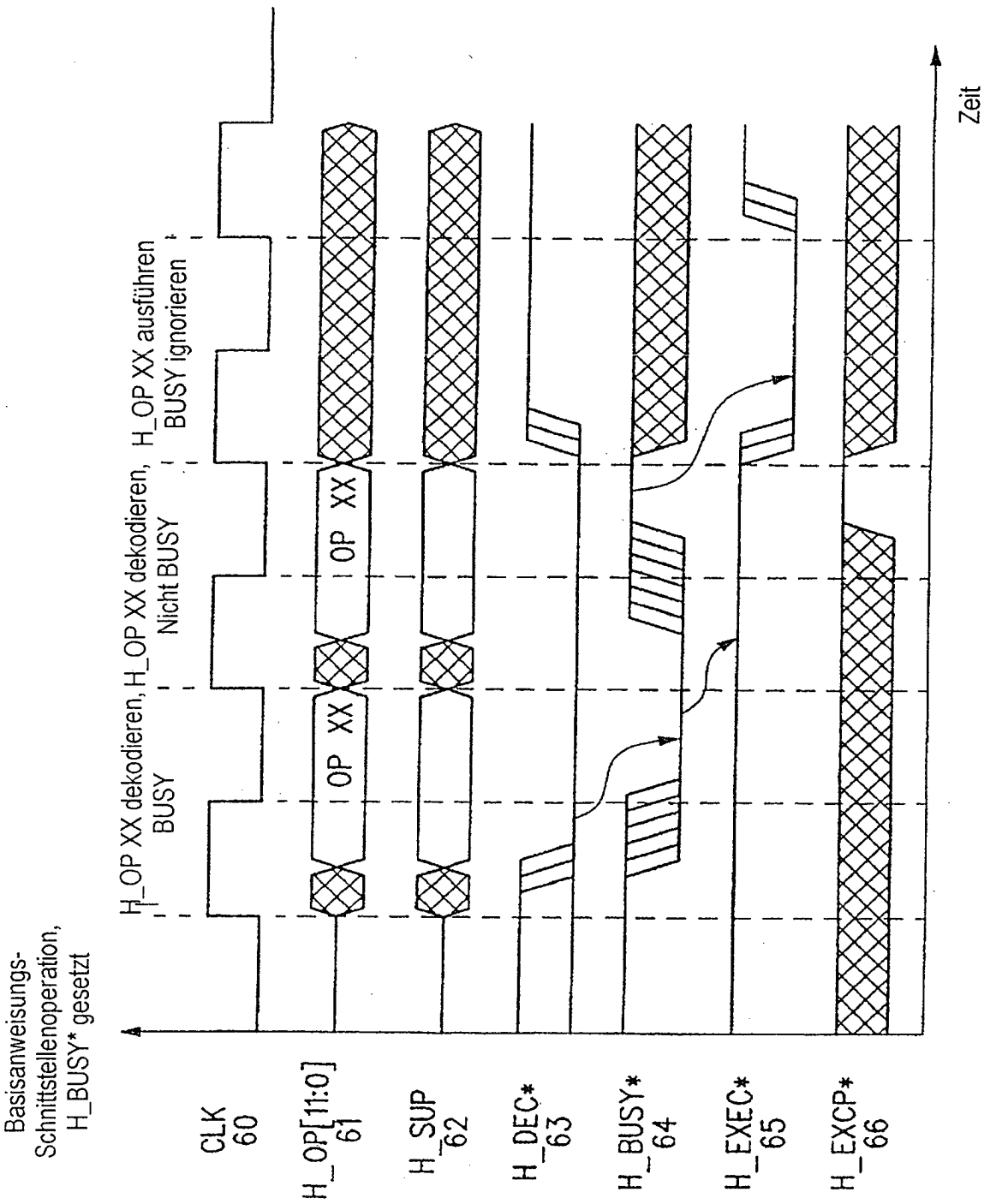

Basisanweisungs-
Schnittstellenoperation,
H_BUSY* gesetzt
H_OP XX dekodieren,
BUSY
H_OP XX dekodieren,
Nicht BUSY
H_OP XX ausführen
BUSY ignorieren
OP XX
OP XX
CLK
60
H_OP[11:0]
61
H_SUP
62
H_DEC*
63
H_BUSY*
64
H_EXEC*
65
H_EXCP*
66
Zeit

FIG.6

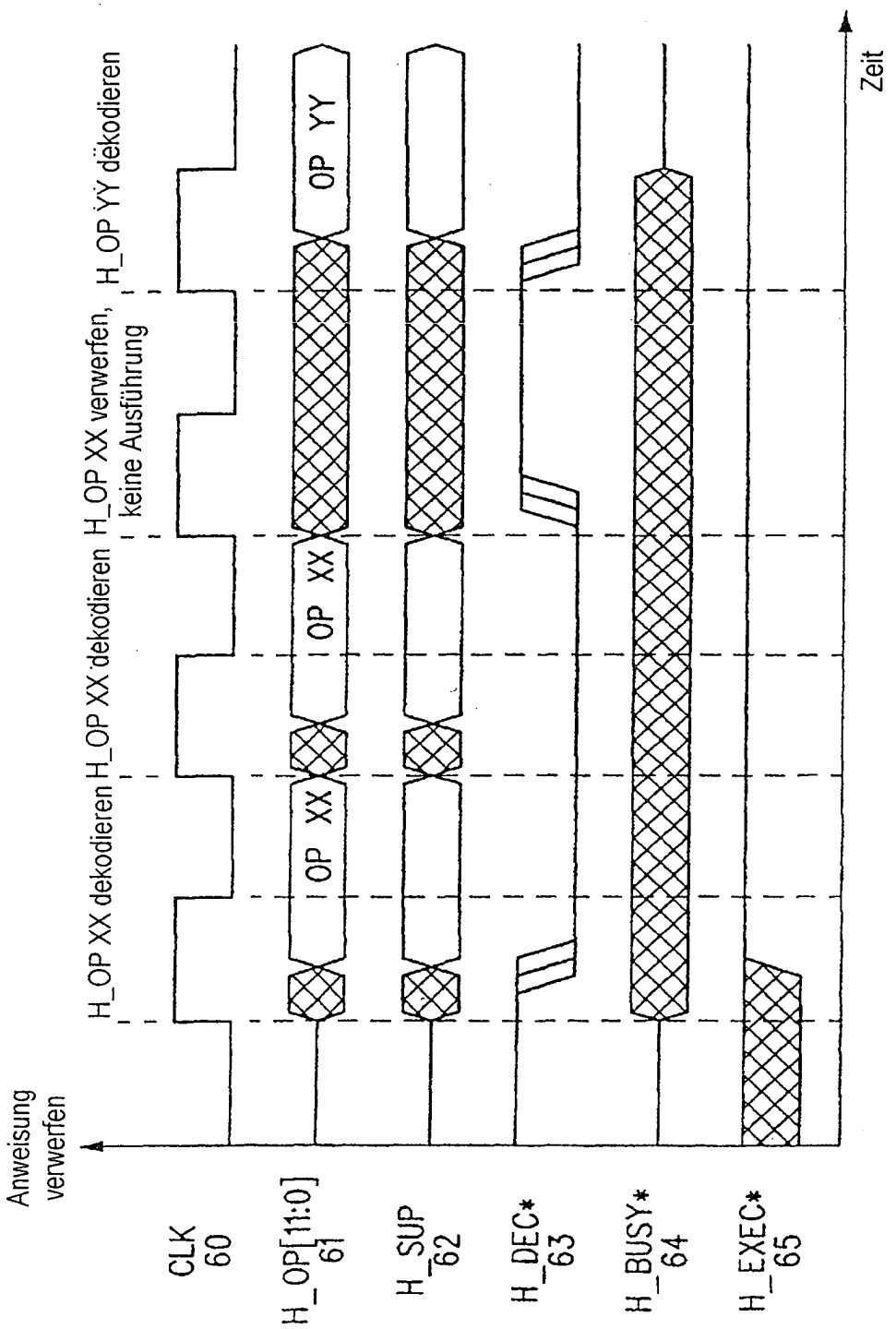

Anweisung verwerfen
H_OP XX dekodieren
H_OP XX dekodieren
H_OP XX verwerfen, keine Ausführung
H_OP YY dekodieren
Zeit
CLK 60
H_OP[11:0] 61
H_SUP 62
H_DEC* 63
H_BUSY* 64
H_EXEC* 65
OP XX
OP XX
OP YY

FIG. 7

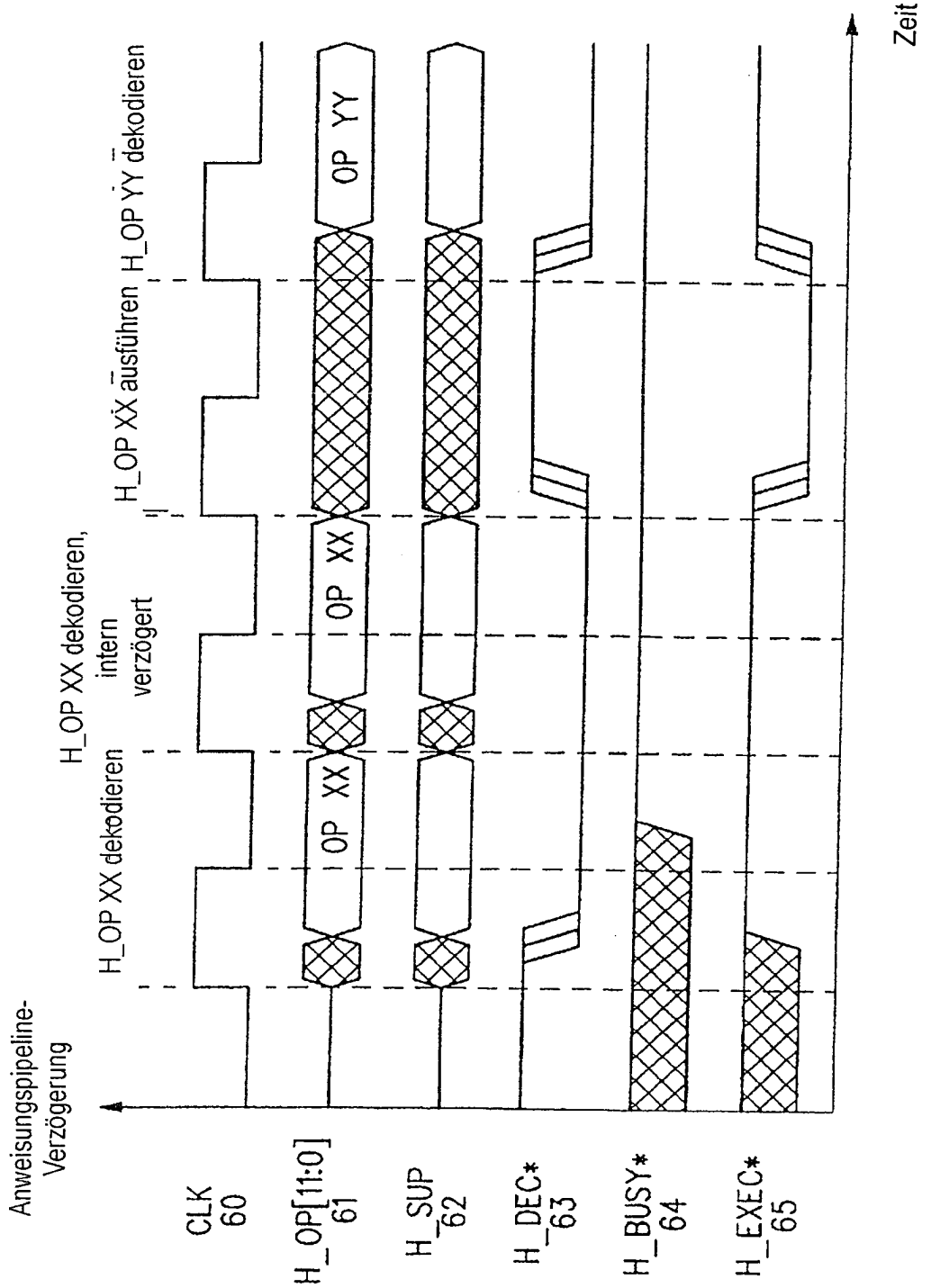

Anweisungspipeline-Verzögerung
H_OP XX dekodieren
H_OP XX dekodieren, intern verzögert
H_OP XX ausführen
H_OP YY dekodieren
Zeit
CLK 60
H_OP[11:0] 61
H_SUP 62
H_DEC* 63
H_BUSY* 64
H_EXEC* 65
OP XX
OP XX
OP YY

FIG.8

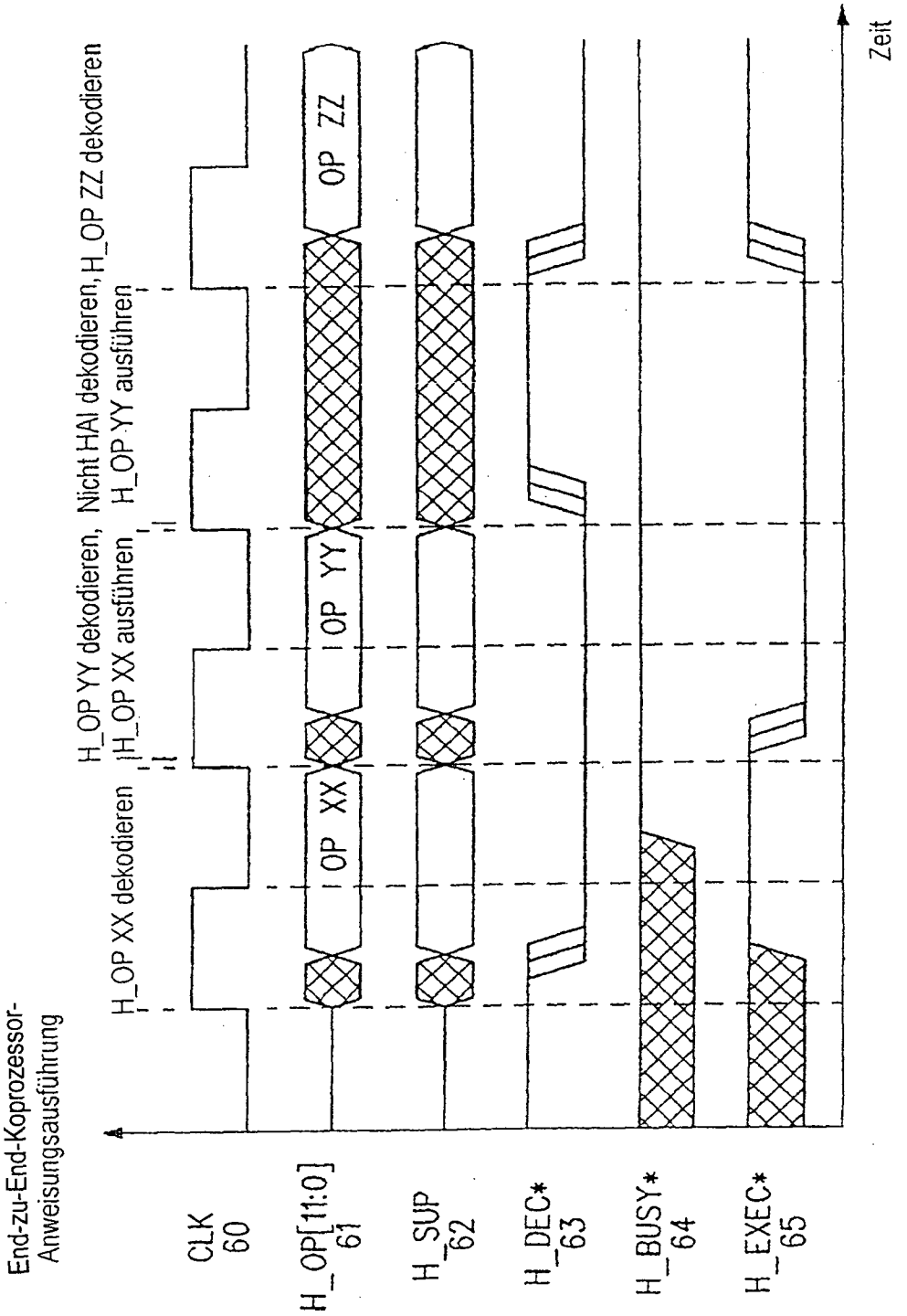

End-zu-End-Koprozessor-Anweisungsausführung
H_OP XX dekodieren
H_OP YY dekodieren,
H_OP XX ausführen
Nicht HAI dekodieren,
H_OP YY ausführen
H_OP ZZ dekodieren
CLK 60
H_OP[11:0] 61
OP XX
OP YY
OP ZZ
H_SUP 62
H_DEC* 63
H_BUSY* 64
H_EXEC* 65
Zeit

FIG.9

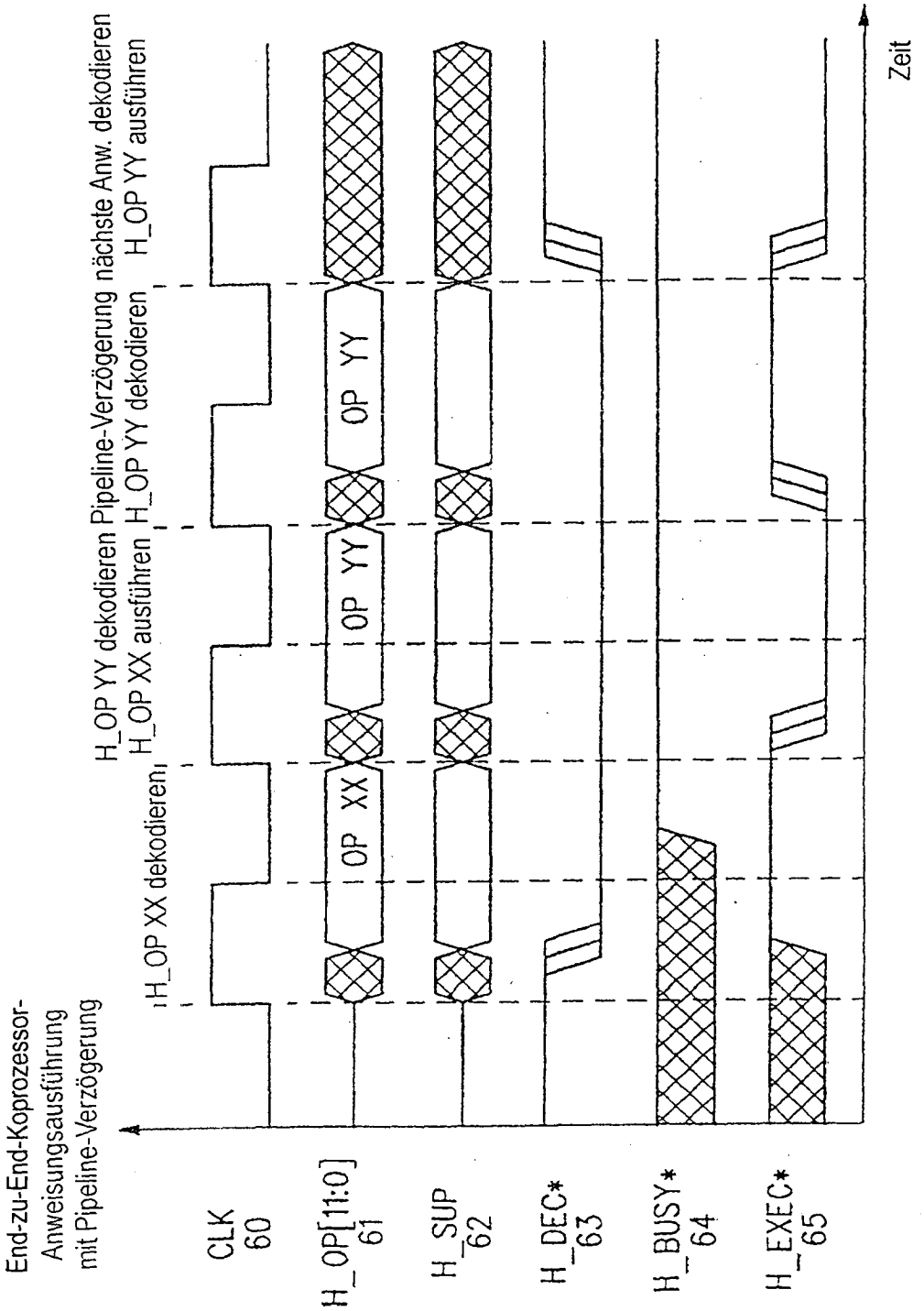

End-zu-End-Koprozessor-
Anweisungsausführung
mit Pipeline-Verzögerung
H_OP XX dekodieren
H_OP XX ausführen
H_OP YY dekodieren
Pipeline-Verzögerung
H_OP YY ausführen
nächste Anw. dekodieren
CLK
60
H_OP[11:0]
61
H_SUP
62
H_DEC*
63
H_BUSY*
64
H_EXEC*
65
OP XX
OP YY
OP YY
Zeit

FIG.10

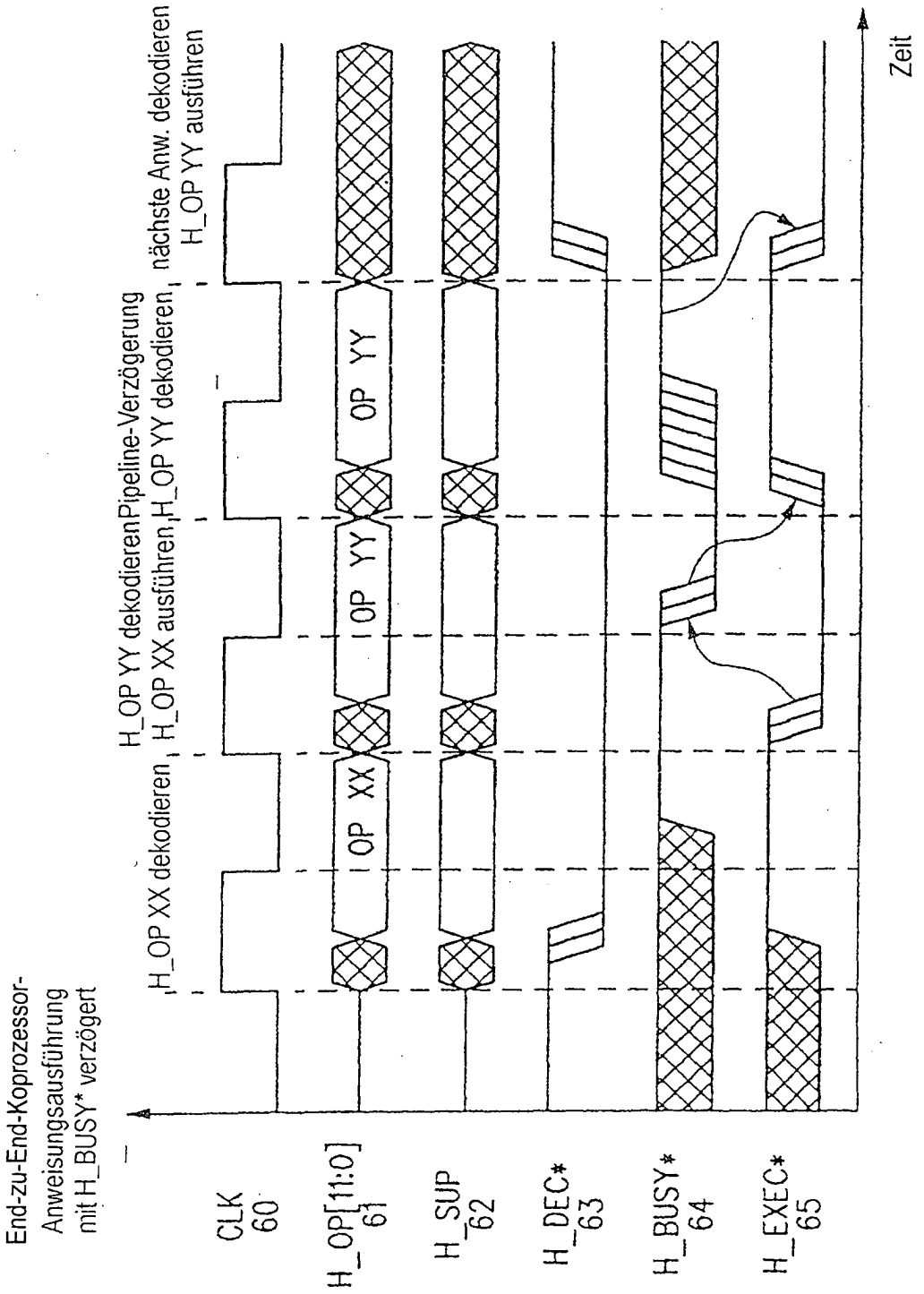

End-zu-End-Koprozessor-
Anweisungsausführung
mit H_BUSY* verzögert
H_OP XX dekodieren
H_OP YY dekodieren
Pipeline-Verzögerung
H_OP XX ausführen, H_OP YY dekodieren
nächste Anw. dekodieren
H_OP YY ausführen
CLK
60
H_OP[11:0]
61
OP XX
OP YY
OP YY
H_SUP
62
H_DEC*
63
H_BUSY*
64
H_EXEC*
65
Zeit

FIG. 11

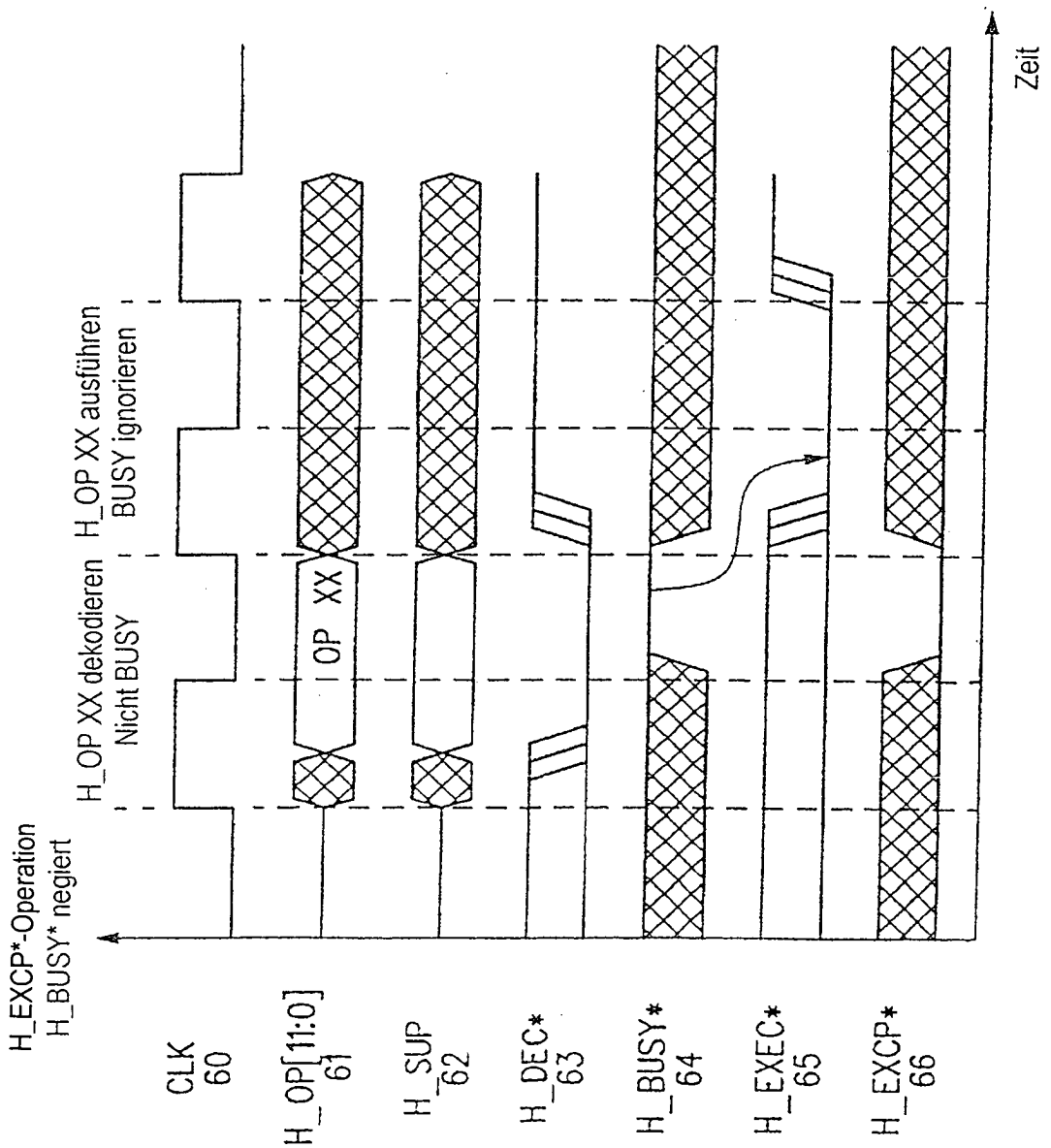

H_EXCP*-Operation
H_BUSY* negiert
H_OP XX dekodieren
Nicht BUSY
H_OP XX ausführen
BUSY ignorieren
OP XX
CLK
60
H_OP[11:0]
61
H_SUP
62
H_DEC*
63
H_BUSY*
64
H_EXEC*
65
H_EXCP*
66
Zeit

FIG.12

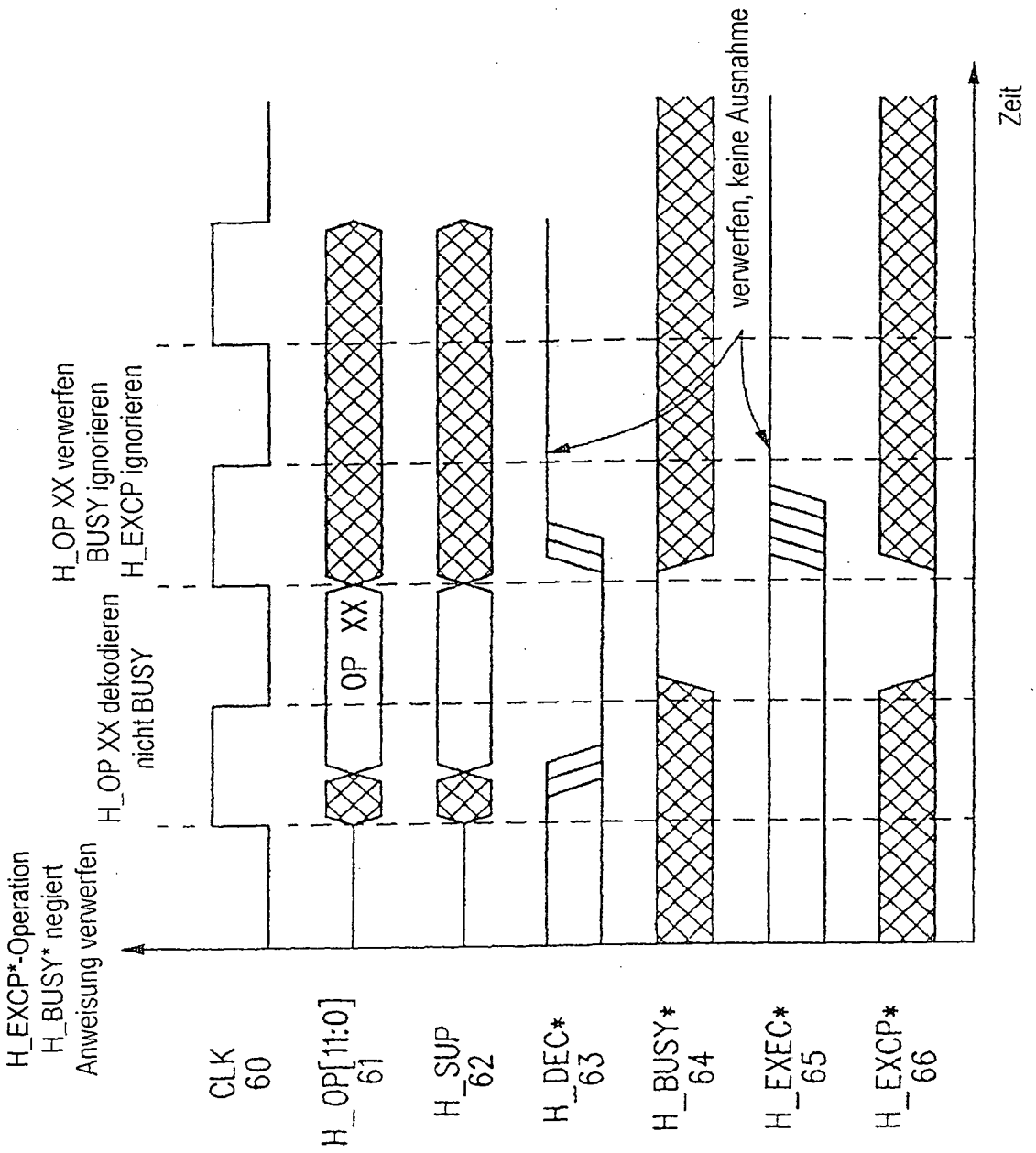

H_EXCP*-Operation
H_BUSY* negiert
Anweisung verwerfen
H_OP XX dekodieren
nicht BUSY
H_OP XX verwerfen
BUSY ignorieren
H_EXCP ignorieren
OP XX
verwerfen, keine Ausnahme
CLK
60
H_OP[11:0]
61
H_SUP
62
H_DEC*
63
H_BUSY*
64
H_EXEC*
65
H_EXCP*
66
Zeit

FIG.13

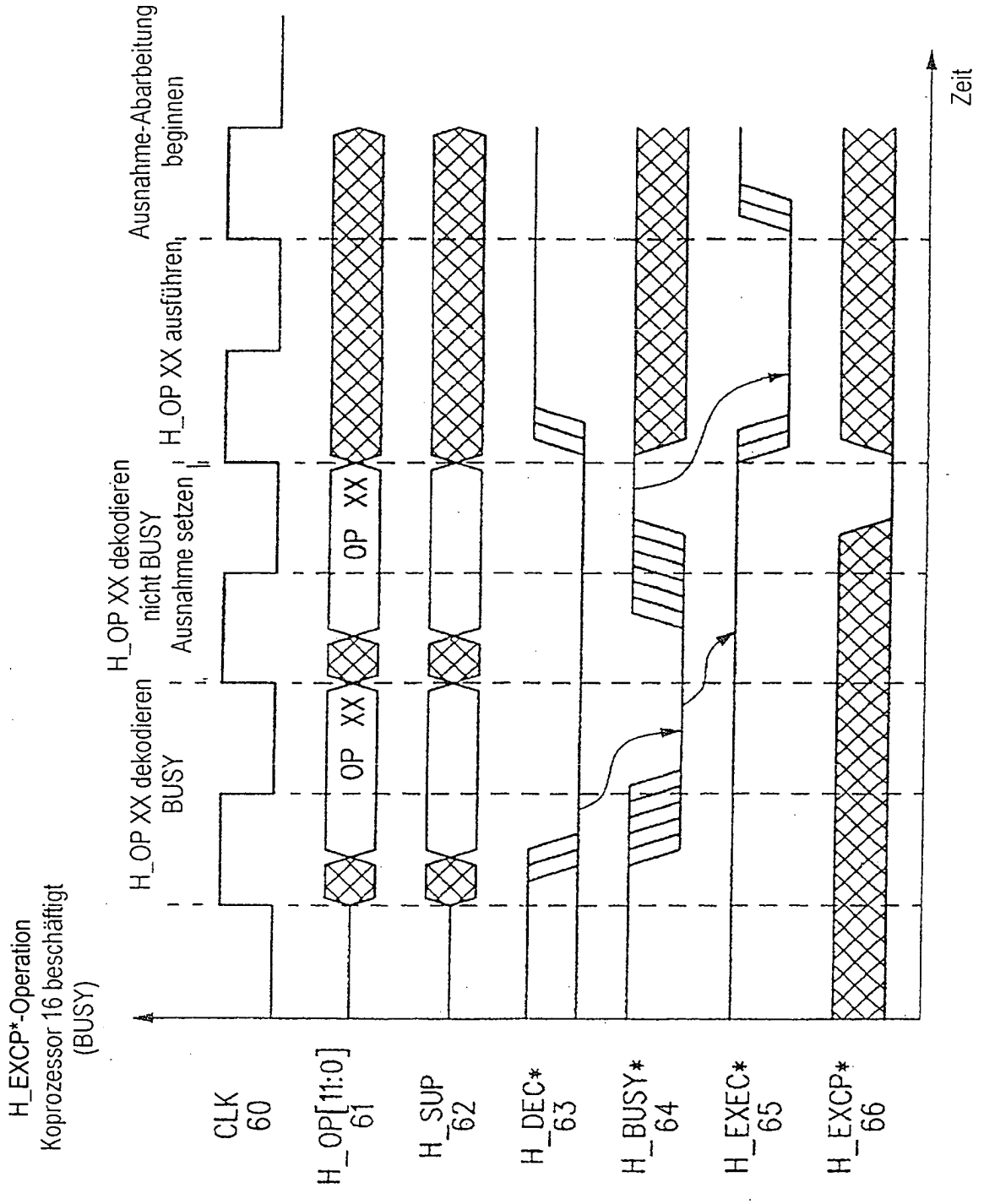

H_EXCP*-Operation
Koprozessor 16 beschäftigt
(BUSY)
H_OP XX dekodieren
BUSY
H_OP XX dekodieren
nicht BUSY
Ausnahme setzen
H_OP XX ausführen
Ausnahme-Abarbeitung
beginnen
Zeit
CLK
60
H_OP[11:0]
61
OP XX
OP XX
H_SUP
62
H_DEC*
63
H_BUSY*
64
H_EXEC*
65
H_EXCP*
66

FIG.14

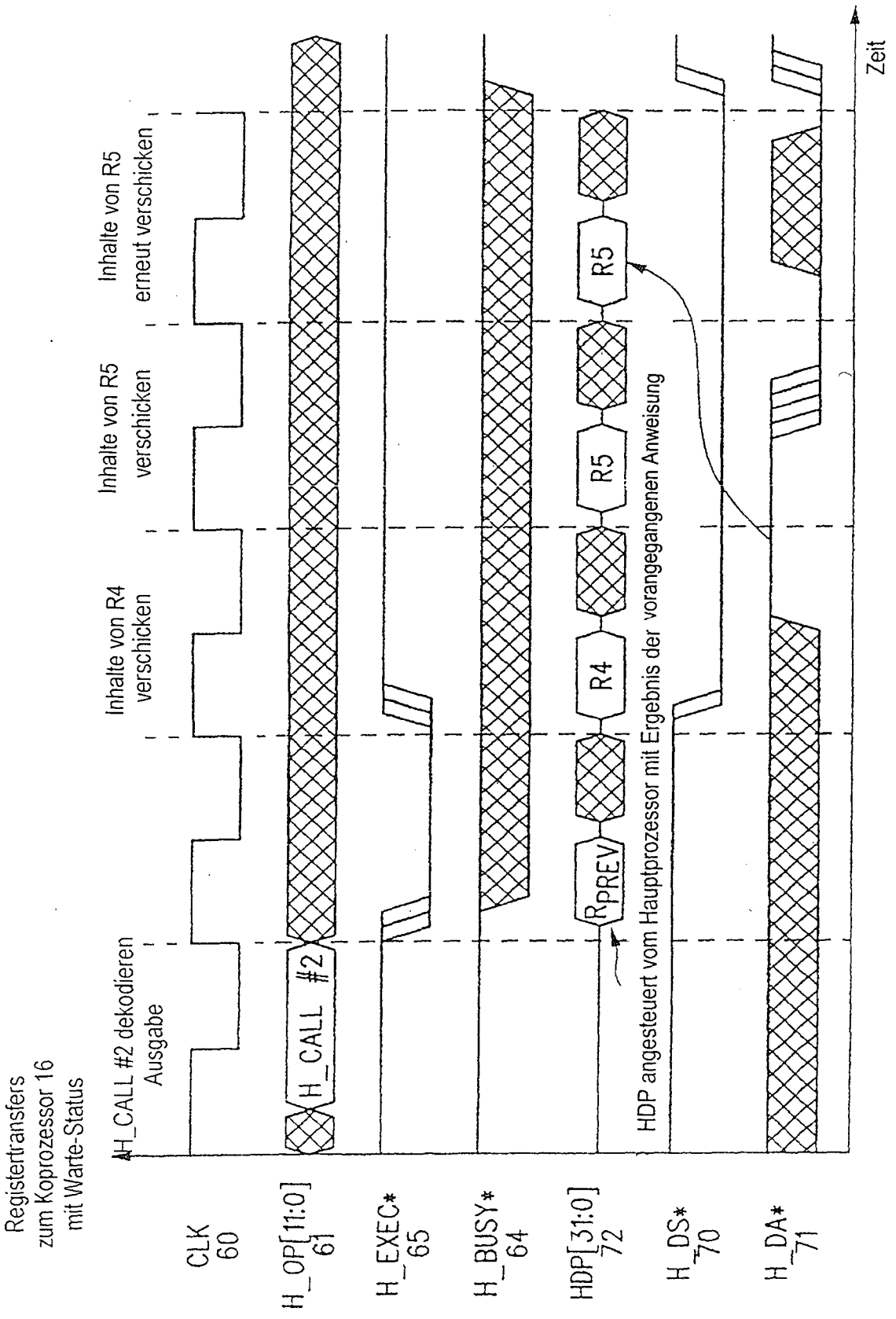

Registertransfers
zum Koprozessor 16
mit Warte-Status
H_CALL #2 dekodieren
Ausgabe
Inhalte von R4
verschicken
Inhalte von R5
verschicken
Inhalte von R5
erneut verschicken
CLK
60
H_OP[11:0]
61
H_CALL #2
H_EXEC*
65
H_BUSY*
64
HDP[31:0]
72
R_PREV
R4
R5
R5
R5
HDP angesteuert vom Hauptprozessor mit Ergebnis der vorangegangenen Anweisung
H_DS*
70
H_DA*
71
Zeit

FIG.15

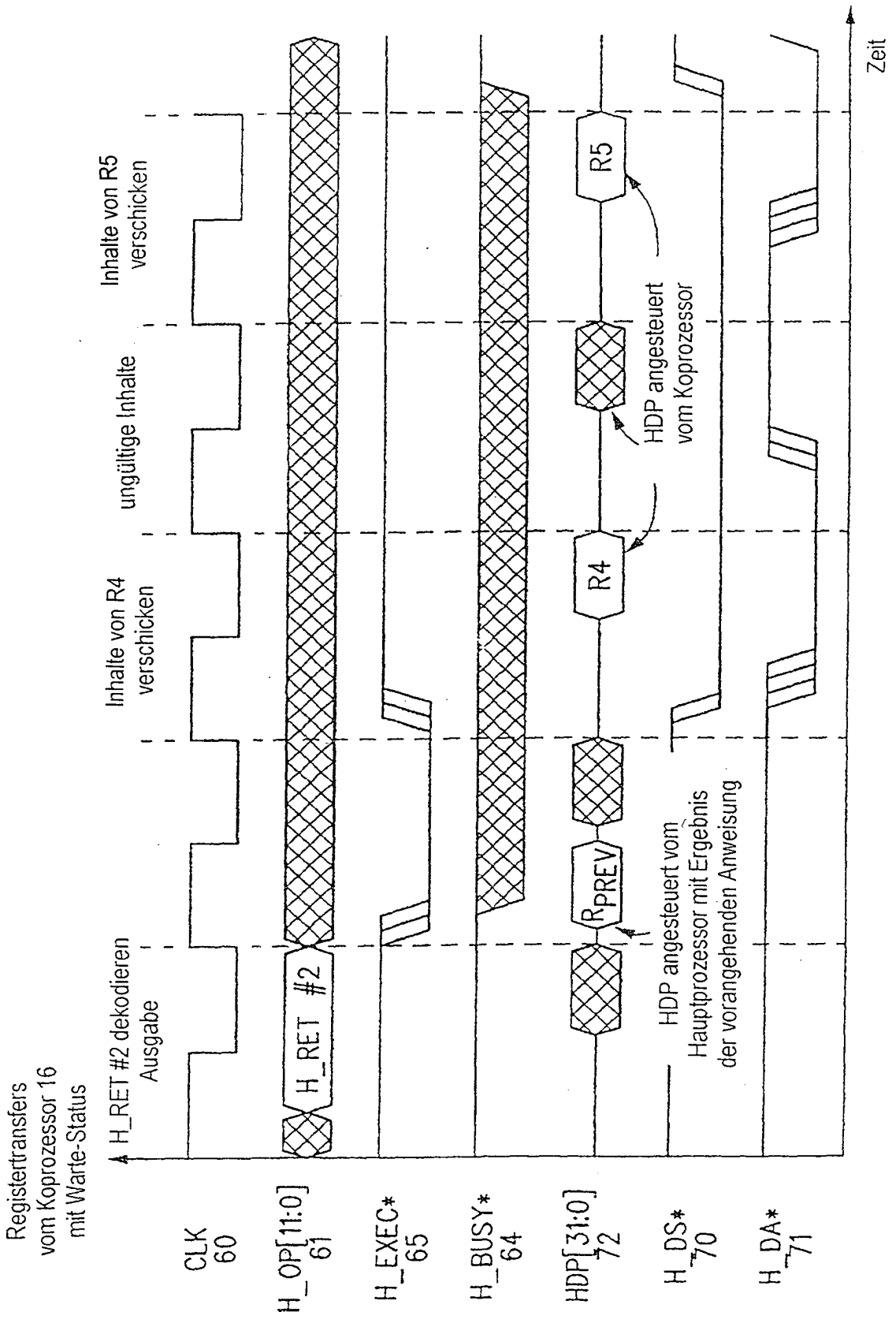

Registertransfers
vom Koprozessor 16
mit Warte-Status
H_RET #2 dekodieren
Ausgabe
Inhalte von R4
verschicken
ungültige Inhalte
Inhalte von R5
verschicken
CLK
60
H_OP[11:0]
61
H_RET #2
H_EXEC*
65
H_BUSY*
64
HDP[31:0]
72
R4
R5
RPREV
HDP angesteuert vom
Hauptprozessor mit Ergebnis
der vorangehenden Anweisung
HDP angesteuert
vom Koprozessor
H_DS*
70
H_DA*
71
Zeit

FIG.16

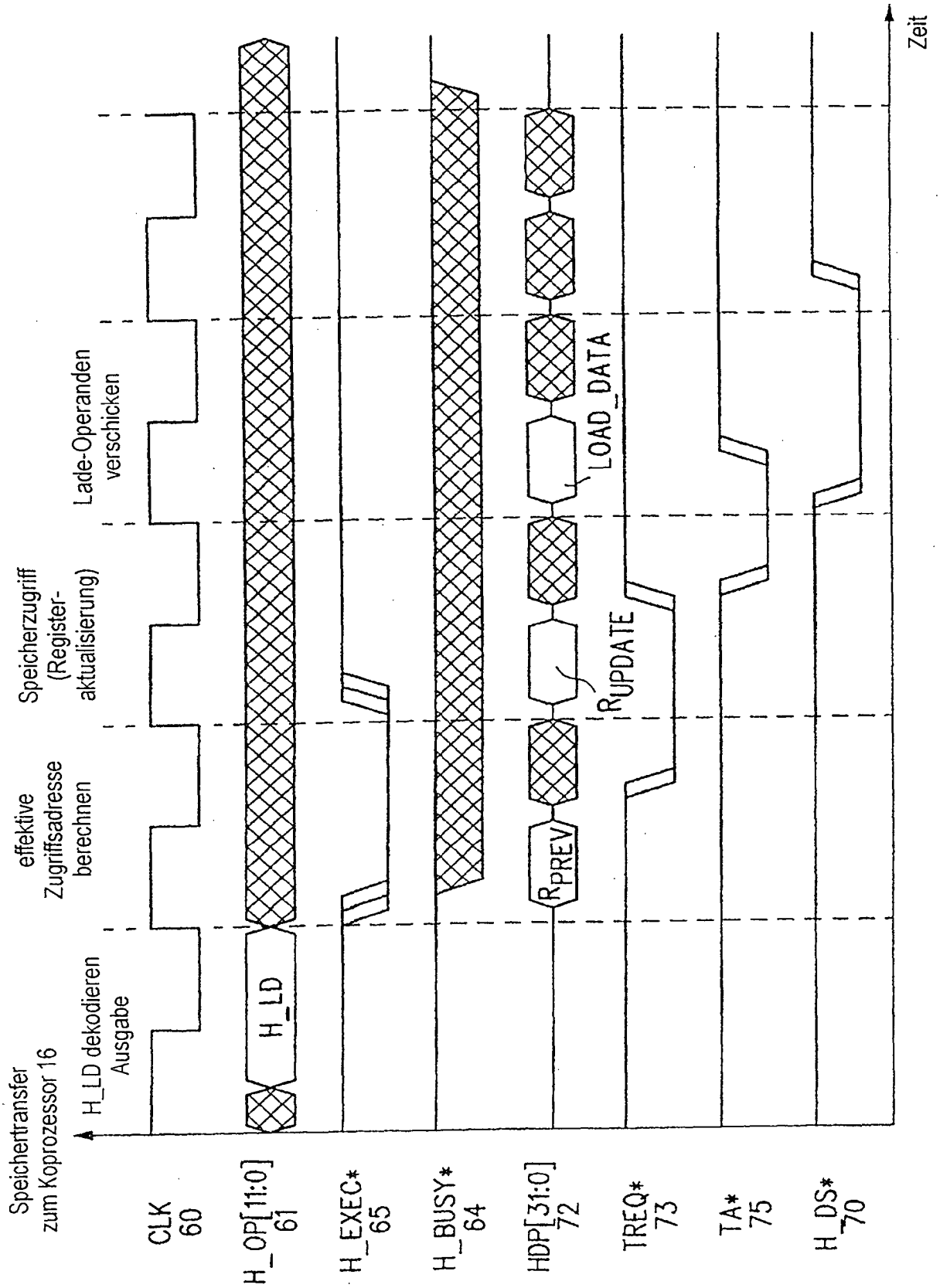

Speichertransfer zum Koprozessor 16
H_LD dekodieren Ausgabe
effektive Zugriffsadresse berechnen
Speicherzugriff (Register-aktualisierung)
Lade-Operanden verschicken
Zeit
CLK 60
H_OP[11:0] 61
H_LD
H_EXEC* 65
H_BUSY* 64
HDP[31:0] 72
R_PREV
R_UPDATE
LOAD_DATA
TREQ* 73
TA* 75
H_DS* 70

FIG 17

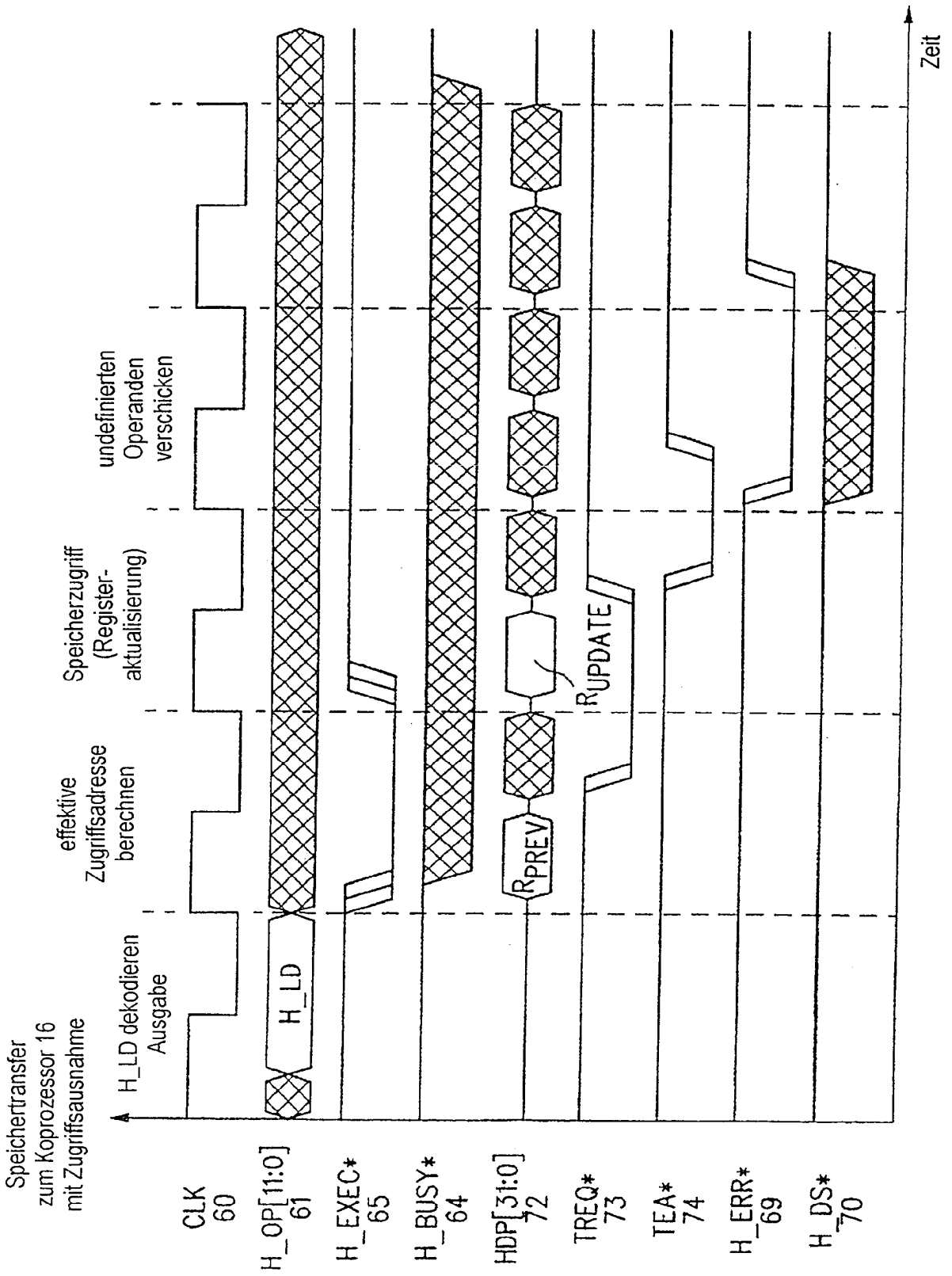

Speichertransfer
zum Koprozessor 16
mit Zugriffsausnahme
H_LD dekodieren
Ausgabe
effektive
Zugriffsadresse
berechnen
Speicherzugriff
(Register-
aktualisierung)
undefinierten
Operanden
verschicken
Zeit
CLK
60
H_OP[11:0]
61
H_LD
H_EXEC*
65
H_BUSY*
64
HDP[31:0]
72
R_PREV
R_UPDATE
TREQ*
73
TEA*
74
H_ERR*
69
H_DS*
70

FIG.18

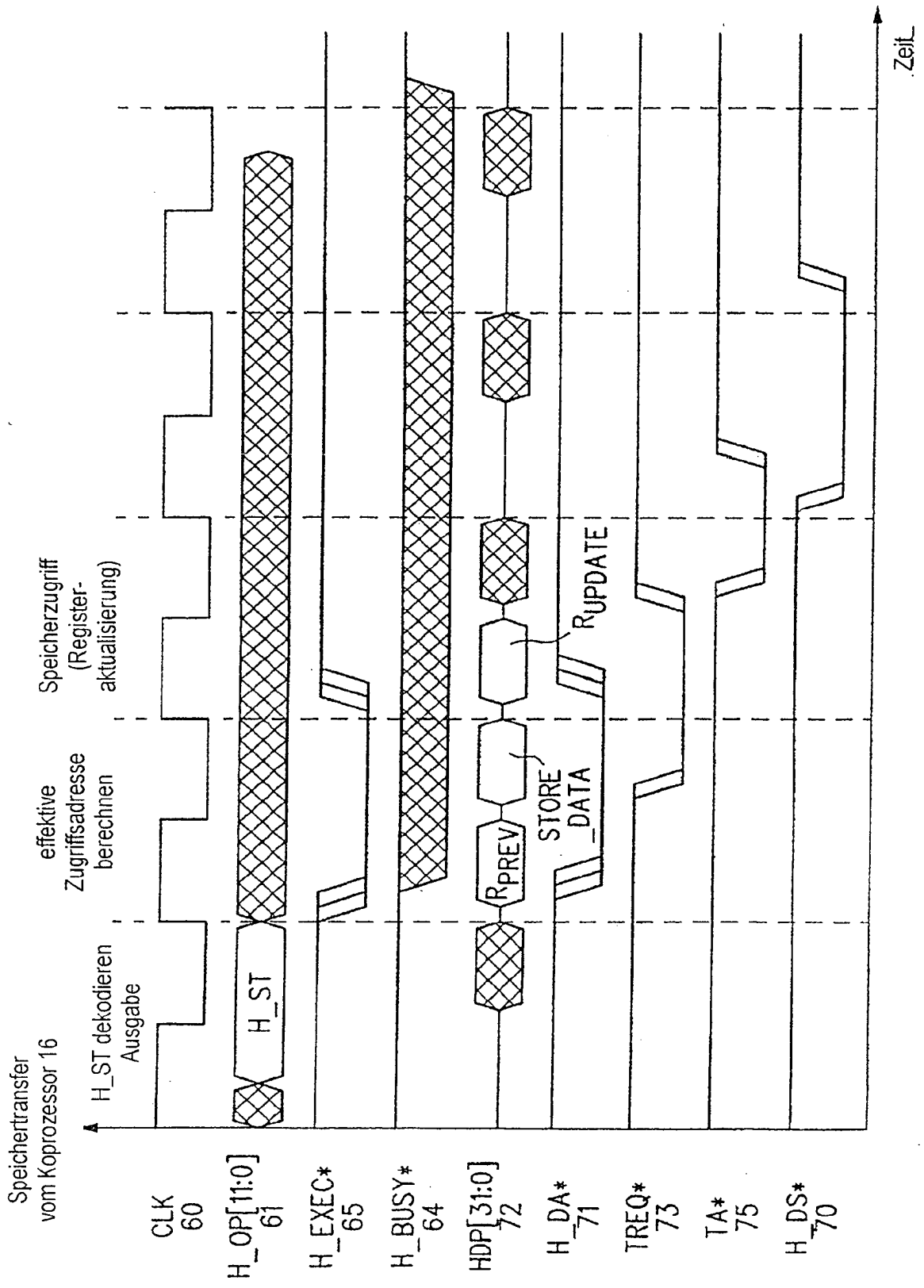

Speichertransfer vom Koprozessor 16
H_ST dekodieren Ausgabe
effektive Zugriffsadresse berechnen
Speicherzugriff (Register-aktualisierung)
Zeit
CLK 60
H_OP[11:0] 61
H_ST
H_EXEC* 65
H_BUSY* 64
HDP[31:0] 72
R_PREV
STORE_DATA
R_UPDATE
H_DA* 71
TREQ* 73
TA* 75
H_DS* 70

FIG.19

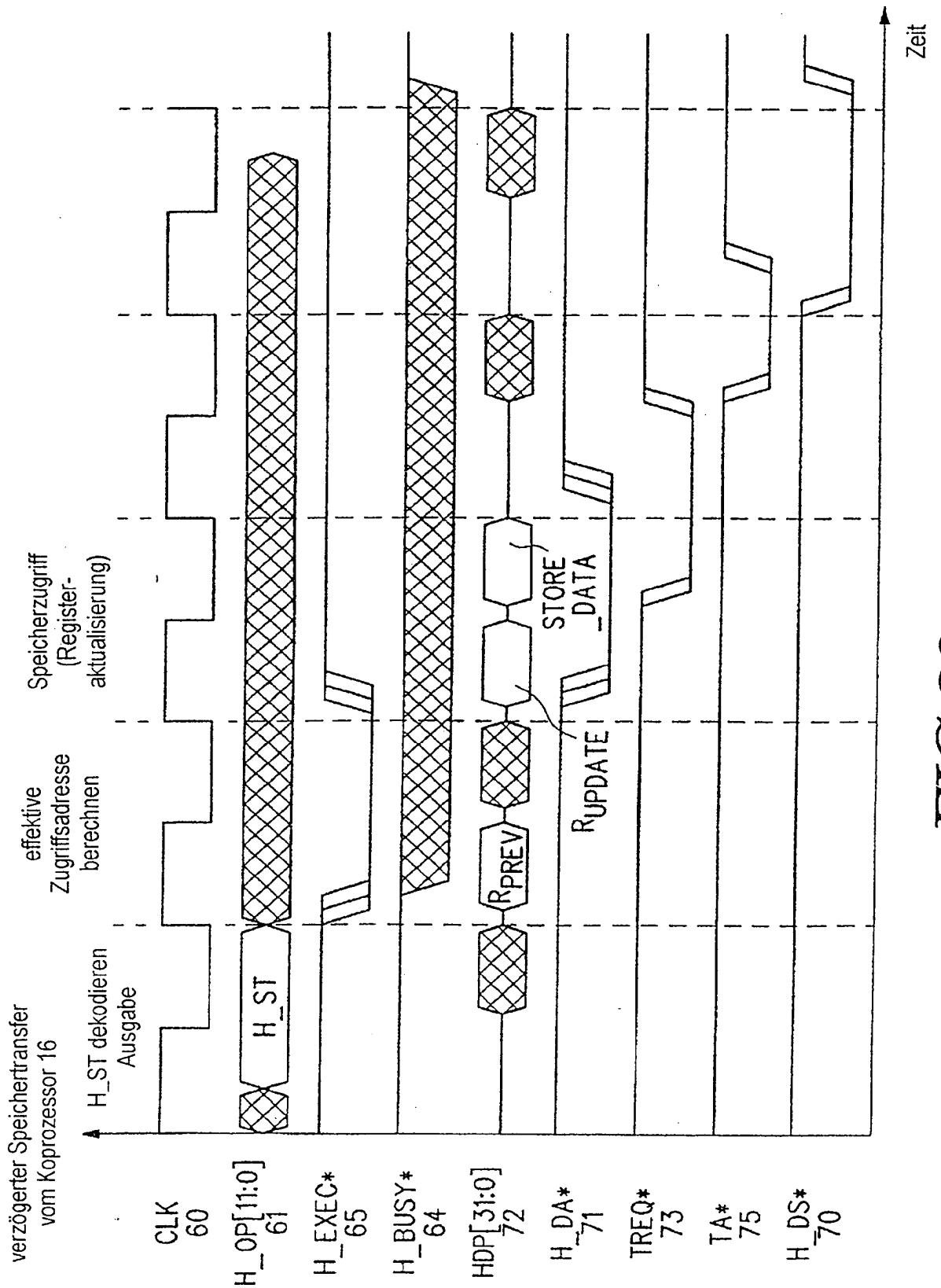
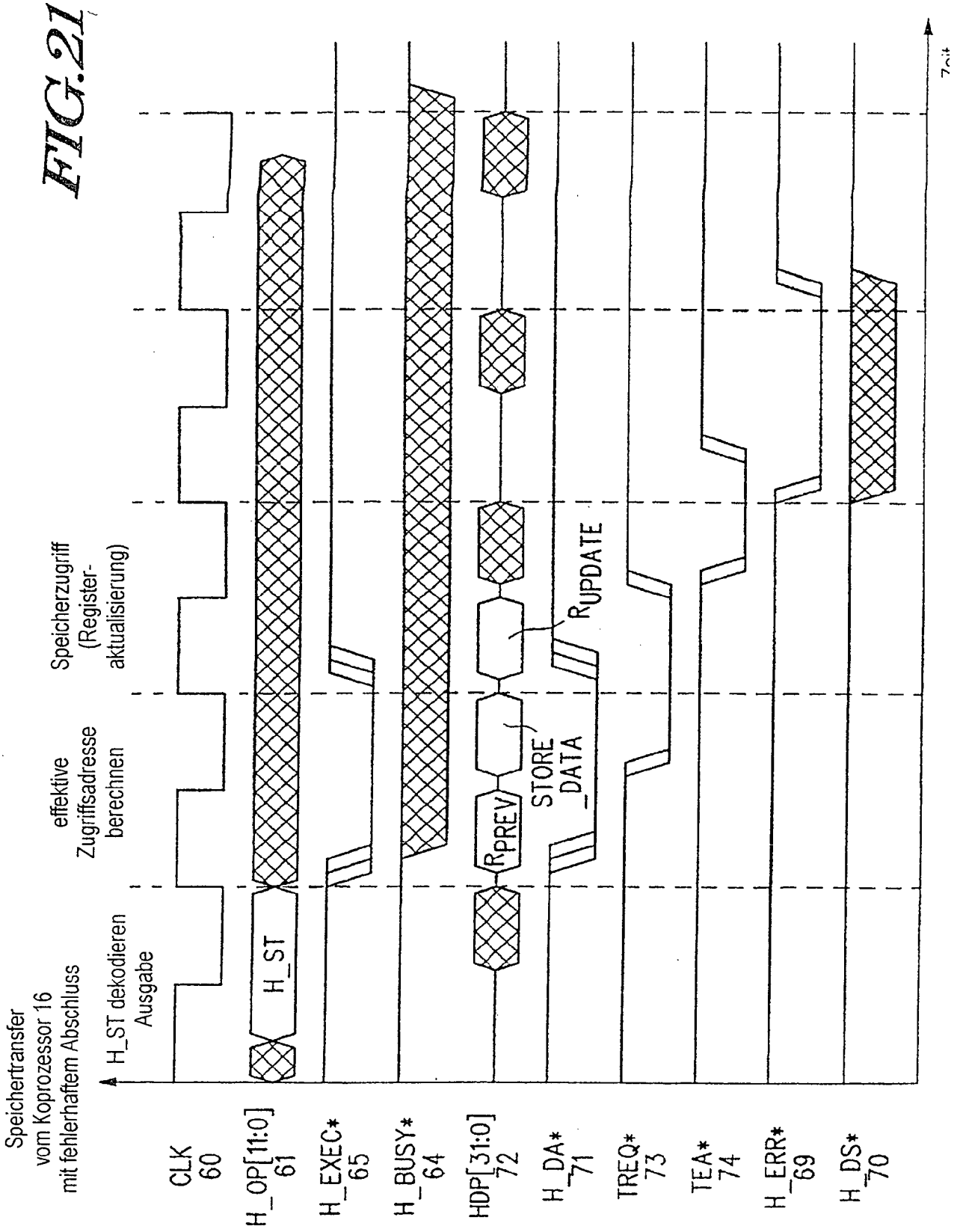

verzögerter Speichertransfer vom Koprozessor 16
H_ST dekodieren Ausgabe
effektive Zugriffsadresse berechnen
Speicherzugriff (Register-aktualisierung)
Zeit
CLK 60
H_OP[11:0] 61
H_ST
H_EXEC* 65
H_BUSY* 64
HDP[31:0] 72
R_PREV
R_UPDATE
STORE _DATA
H_DA* 71
TREQ* 73
TA* 75
H_DS* 70

FIG. 20

FIG.21


Speichertransfer
vom Koprozessor 16
mit fehlerhaftem Abschluss
H_ST dekodieren
Ausgabe
effektive
Zugriffsadresse
berechnen
Speicherzugriff
(Register-
aktualisierung)
CLK
60
H_OP[11:0]
61
H_ST
H_EXEC*
65
H_BUSY*
64
HDP[31:0]
72
RPREV
STORE
_DATA
RUPDATE
H_DA*
71
TREQ*
73
TEA*
74
H_ERR*
69
H_DS*
70
Zeit

H_CALL Hardware-Beschleuniger- (Koprozessor-) Aufruf-Grundelement

OPERATION: Parameter an Hardware-Beschleuniger übergeben

ASSEMBLER SYNTAX: H_CALL #UU, R4–RLAST, #CODE

Beschreibung: H_CALL übergibt einen Satz von Register-basierten Parametern und einen Kode an Hardware-Block (Koprozessor) #UU

Bedingungs-Kode: Unbeeinflusst

Anweisungs-Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	UU		0	1	1	CNT			CODE			

Anweisungs-Felder:

UU-Feld-spezifiziert Hardware-Block (Koprozessor)

00 – BLOCK 0
01 – BLOCK 1
10 – BLOCK 2
11 – BLOCK 3

CNT-Feld-spezifiziert Anzahl zu übergebender Register, beginnend mit R4

000 - reserviert, nicht verwenden
001 - übergibt R4

:
:

111 - übergibt R4-R10

FIG.22

H_RET	Hardware-Beschleuniger- (Koprozessor-) Rückgabe-Grundelement
OPERATION:	Parameter von Hardware-Beschleuniger übergeben
ASSEMBLER SYNTAX:	H_RET #UU, R4-RLAST, #CODE
Beschreibung:	H_RET übergibt einen Kode an Koprozessor #UU und empfängt einen Satz von in die CPU-Register zu ladenden Rückgabe-Parametern

Bedingungs-Kode: Unbeeinflusst

Anweisungs-Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	UU		0	1	0	CNT			CODE			

Anweisungs-Felder:

UU-Feld - spezifiziert Hardware-Block (Koprozessor)

00 – BLOCK 0
01 – BLOCK 1
10 – BLOCK 2
11 – BLOCK 3

CNT-Feld - spezifiziert Anzahl zu übergebender Register, beginnend mit R4

000 - reserviert, nicht verwenden
001 - übergibt R4
010 - übergibt R4 – R5
:
111 - übergibt R4-R10

FIG.23

H_EXEC	Hardware-Beschleuniger- (Koprozessor-) Ausführungs-Grundelement
OPERATION:	Ausführungscode an Hardware-Beschleuniger übergeben
ASSEMBLER SYNTAX:	H_EXEC #UU, #CODE
Beschreibung:	H_EXEC wird verwendet zur Steuerung einer Funktion im Koprozessor #UU. Das CODE-Feld wird von der CPU nicht interpretiert
Bedingungs-Kode: Unbeeinflusst	

Anweisungs-Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	UU		0	0	CODE							

Anweisungs-Felder:

UU-Feld - spezifiziert Hardware-Block (Koprozessor)

00 – BLOCK 0
01 – BLOCK 1
10 – BLOCK 2
11 – BLOCK 3

CODE-Feld - spezifiziert einen Operationskode für einen Hardware-Block

FIG.24

H_LD	Hardware-Beschleuniger- (Koprozessor-) Lade-Grundelement
OPERATION:	Operand aus Speicher laden und an Hardware-Beschleuniger übergeben
ASSEMBLER SYNTAX:	H_LD.[HW][U] #UU, (RX, DISP) H_LD.[U] #UU, (RX, DISP)

Beschreibung: H_LD führt ein Laden eines Wertes im Speicher durch und übergibt den Speicheroperanden an den Koprozessor, ohne ihn in einem GPR zu speichern. Die H_LD-Operation umfasst drei Optionen, W-Wort, H-halbes Wort und U-Aktualisierung (Update). DISP ehält man durch Skalieren des IMM2-Feldes mit der Größe der Ladedaten und Auffüllen mit Null. Dieser Wert wird zum Wert des Registers RX addiert und ein Laden der spezifizierten Größe von dieser Adresse wird durchgeführt, wobei das Ergebnis des Ladens an die Hardware-Schnittstelle übergeben wird. Beim Laden von halben Wörtern, werden die geholten Daten mit Nullen auf 32 Bit aufgefüllt. Falls die U-Option gewählt ist, wird die effektive Adresse des Ladevorgangs in das Register RX gestellt, nachdem sie berechnet wurde.

Bedingungs-Kode: Unbeeinflusst

Anweisungs-Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	UU		1	SZ	0	UP	IMM2		RX			

Anweisungs-Felder:

UU-Feld - spezifiziert Hardware-Block (Koprozessor)

00 – BLOCK 0
01 – BLOCK 1
10 – BLOCK 2
11 – BLOCK 3

SIZE - spezifiziert Lade-Größe
0 – Wort
1 - halbes Wort

UP - spezifiziert, ob Basisregister aktualisiert werden soll
0 - Keine Aktualisierung
1 - Aktualisieren des Basisregisters mit effektiver Adresse

IMM2-Feld - spezifiziert einen 2-Bit skalierten, unmittelbaren Wert

Register X - spezifiziert die dem skalierten, unmittelbaren Feld hinzuzufügende Basisadresse

FIG.25

H_ST	Hardware-Beschleuniger- (Koprozessor-) Speicherungs-Grundelement
OPERATION:	Operand von Hardware-Beschleuniger in Speicher speichern
ASSEMBLER SYNTAX:	H_ST.[HW][U] #UU, (RX, DISP)

Beschreibung: H_ST führt eine Speicherung eines Operanden vom Koprozessor in den Speicher durch, ohne ihn in einem GPR zu speichern. Die H_ST- Operation umfasst W- Wort , H-halbes Wort und U-Aktualisierung (Update). DISP erhält man durch Skalieren des IMM2-Feldes mit der Größe der Speicherdaten und Auffüllen mit Null. Dieser Wert wird zum Wert des Registers RX addiert und eine Speicherung der spezifizierten Größe an diese Adresse wird durchgeführt, wobei man die Daten für die Speicherung von der Hardware-Schnittstelle erhält. Falls die U-Option gewählt ist, wird die effektive Adresse des Ladevorgangs in das Register RX gestellt, nachdem sie berechnet wurde

Bedingungs-Kode: Unbeeinflusst

Anweisungs-Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	UU		1	SZ	1	UP	IMM2		RX			

Anweisungs-Felder:

UU-Feld - spezifiziert Hardware-Block (Koprozessor)

00 – BLOCK 0
01 – BLOCK 1
10 – BLOCK 2
11 – BLOCK 3

SIZE - spezifiziert Speicher-Größe
0 - Wort
1 - halbes Wort

UP - spezifiziert, ob Basisregister aktualisiert werden soll
0 - Keine Aktualisierung
1 - Aktualisieren des Basisregisters mit effektiver Adresse

IMM2-Feld - spezifiziert einen 2-Bit skalierten, unmittelbaren Wert

Register X - spezifiziert die dem skalierten, unmittelbaren Feld hinzuzufügende Basisadresse

FIG.26