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

(10) **DE 699 24 857 T2** 2006.03.02

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 119 807 B1**
(21) Deutsches Aktenzeichen: **699 24 857.4**
(86) PCT-Aktenzeichen: **PCT/GB99/03168**
(96) Europäisches Aktenzeichen: **99 949 129.3**
(87) PCT-Veröffentlichungs-Nr.: **WO 00/22521**
(86) PCT-Anmeldetag: **11.10.1999**
(87) Veröffentlichungstag der PCT-Anmeldung: **20.04.2000**
(97) Erstveröffentlichung durch das EPA: **01.08.2001**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **20.04.2005**
(47) Veröffentlichungstag im Patentblatt: **02.03.2006**

(51) Int Cl.[8]: ***G06F 9/45*** (2006.01)
G06F 9/455 (2006.01)

(30) Unionspriorität:

9822075	**10.10.1998**	**GB**
115952 P	**14.01.1999**	**US**

(84) Benannte Vertragsstaaten:
AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LI, LU, MC, NL, PT, SE

(73) Patentinhaber:
Transitive Ltd., Hanging Ditch, Manchester, GB

(74) Vertreter:
Betten & Resch, 80333 München

(72) Erfinder:
SOULOGLOU, Jason, Manchester M13 9PL, GB;
RAWSTHORNE, Alasdair, Manchester M13 9PL, GB

(54) Bezeichnung: **PROGRAMM-KODE-UMWANDLUNG**

Beschreibung

[0001] Die vorliegende Erfindung bezieht sich auf ein Verfahren und auf ein System zum Umsetzen von Programmcode aus einem Format in ein anderes. Insbesondere bezieht sich die Erfindung auf ein Verfahren und auf ein System zur Schaffung einer intermediären Repräsentation eines Computerprogramms oder eines Basisblocks eines Programms (ein Basisblock eines Programms ist ein Block von Befehlen, der nur einen Eintrittspunkt bei einem ersten Befehl und nur einen Austrittspunkt bei einem letzten Befehl des Blocks besitzt). Zum Beispiel schafft die vorliegende Erfindung ein Verfahren und ein System zum Übersetzen eines Computerprogramms, das für einen Prozessor geschrieben wurde, so dass das Programm effizient auf einem anderen Prozessor laufen kann; wobei die Übersetzung eine intermediäre Repräsentation verwendet und in einer blockweisen Betriebsart durchgeführt wird.

[0002] Eine intermediäre Repräsentation ist ein in der Computerindustrie umfassend verwendeter Begriff zur Bezugnahme auf Formen der abstrakten Computersprache, in denen ein Programm ausgedrückt werden kann, die aber nicht spezifisch für irgendeinen besonderen Prozessor sind und nicht dafür bestimmt sind, auf irgendeinem besonderen Prozessor ausgeführt zu werden. Eine intermediäre Repräsentation wird z. B. allgemein erzeugt, um eine Optimierung eines Programms zu ermöglichen. Zum Beispiel übersetzt ein Compiler ein Computerprogramm in einer höheren Programmiersprache in eine intermediäre Repräsentation, optimiert das Programm durch Anwenden verschiedener Optimierungstechniken auf die intermediäre Repräsentation und übersetzt daraufhin die optimierte intermediäre Repräsentation in ausführbaren Binärcode. Außerdem wird eine intermediäre Repräsentation verwendet, um zu ermöglichen, Programme in einer Form, die nicht spezifisch für irgendeinen Prozessor ist, über das Internet zu senden. Zum Beispiel hat Sun Microsystems hierfür eine Form einer intermediären Repräsentation entwickelt, die als Bytecode bezeichnet wird. Ein Bytecode kann auf irgendeinem Prozessor interpretiert werden, auf dem das allgemein bekannte Java-(Warenzeichen)-Laufzeitsystem verwendet wird.

[0003] Außerdem wird eine intermediäre Repräsentation häufig von Emulationssystemen verwendet, die eine Binärübersetzung verwenden. Emulationssysteme dieser Art nehmen Software-Code, der für einen gegebenen Prozessortyp kompiliert worden ist, setzen ihn in eine intermediäre Repräsentation um, optimieren die intermediäre Repräsentation und setzen die intermediäre Repräsentation daraufhin in einen Code um, der auf einem anderen Prozessortyp laufen kann. Die Optimierung der Erzeugung einer intermediären Repräsentation ist eine bekannte Prozedur, die zur Minimierung der Menge an Code verwendet wird, der zur Ausführung eines emulierten Programms erforderlich ist. Für die Optimierung einer intermediären Repräsentation gibt es eine Vielzahl bekannter Verfahren.

[0004] Ein Beispiel eines bekannten Emulationssystems, das zur Ausführung einer Binärübersetzung eine intermediäre Repräsentation verwendet, ist das von AT&T betriebene FlashPort-System. Ein Kunde liefert an AT&T ein zu übersetzendes Programm (wobei das Programm in der Weise kompiliert worden ist, dass es auf einem Prozessor eines ersten Typs läuft). Das Programm wird durch AT&T in eine intermediäre Repräsentation übersetzt und die intermediäre Repräsentation daraufhin über die Anwendung automatischer Optimierungsroutinen mit Hilfe von Technikern, die eine Eingabe liefern, wenn die Optimierungsroutinen versagen, optimiert. Daraufhin wird die optimierte intermediäre Übersetzung durch AT&T in Code übersetzt, der auf einem Prozessor des gewünschten Typs laufen kann. Dieser Typ der Binärübersetzung, in der ein gesamtes Programm übersetzt wird, bevor es ausgeführt wird, wird als eine "statische" Binärübersetzung bezeichnet. Die Übersetzungszeiten können irgendwo bis zu mehreren Monaten liegen.

[0005] In einer alternativen Form der Emulation wird ein Programm im Code eines betreffenden Prozessors (d. h. eines ersten Prozessortyps, für den der Code geschrieben ist und der emuliert werden soll) über eine intermediäre Repräsentation in Basisblöcke in Code eines Zielprozessors (d. h. eines zweiten Prozessortyps, auf dem die Emulation ausgeführt wird) übersetzt.

[0006] Afzal, T., u. a.: 'Motorola PowerPC Migration Tools-Emulation and Translation', Digest of Papers der Computer Society Computer Conference Compcon, USA, Los Alamitos, IEEE, Comp.SOC.Press, Bd. CONF. 41, 25.–28. Februar 1996, S. 145–150, ISBN: 0-8186-7414-8, beschreibt Emulations- und Übersetzungsverfahren zur Übertragung vorhandener Anwendungen auf eine Motorola-PowerPC-Architektur. Die in dieser Abhandlung beschriebenen Verfahren bilden den Oberbegriff des beigefügten Anspruchs 1.

[0007] Eine Aufgabe der vorliegenden Erfindung ist die Schaffung eines alternativen Verfahrens zur Erzeugung einer intermediären Repräsentation von Programmcode. Bevorzugte Aufgaben sind die Schaffung eines solchen Verfahrens, das einfach, kostengünstig und leicht zu implementieren ist.

[0008] Gemäß einem ersten Aspekt der Erfindung wird ein Verfahren zum Erzeugen einer intermediären Repräsentation von Programmcode geschaffen, wobei das Verfahren die folgenden computerimplementierten Schritte aufweist: Erzeugen einer Mehrzahl von Registerobjekten, welche abstrakte Register repräsentieren; und Erzeugen von Ausdrucksobjekten, von denen jedes ein unterschiedliches Element des Programmcodes repräsentiert, wenn das Element in dem Programmcode auftaucht; dadurch gekennzeichnet, dass ein einzelnes Registerobjekt jeweils ein abstraktes Register repräsentiert; und dadurch dass jedes Ausdrucksobjekt durch ein Registerobjekt referenziert wird, zu dem es entweder direkt in Beziehung steht oder indirekt über Referenzen von anderen Ausdrucksobjekten.

[0009] Vorzugsweise wird der Programmcode durch einen Befehlssatz eines Prozessors ausgedrückt. Vorzugsweise repräsentieren die Registerobjekte abstrakte Register, die den Registern des Prozessors entsprechen.

[0010] Vorzugsweise wird jeder der Schritte sequentiell für Basisblöcke des Programmcodes ausgeführt, die jeweils nur einen effektiven Eintrittspunktbefehl und einen effektiven Austrittspunktbefehl aufweisen.

[0011] Vorzugsweise geben zumindest einige der Ausdrucksobjekte in mehr als eines der Registerobjekte ein.

[0012] Vorzugsweise werden die Ausdrucksobjekte nicht dupliziert.

[0013] Vorzugsweise wird ein einzelnes Ausdrucksobjekt für ein gegebenes Element des Programmcodes erzeugt und jedes Ausdrucksobjekt durch alle der Registerobjekte referenziert, auf die es sich bezieht.

[0014] Vorzugsweise wird eines der Registerobjekte oder eines der Ausdrucksobjekte eliminiert, wenn es redundant oder unnötig wird.

[0015] Vorzugsweise wird ein redundantes oder unnötiges Registerobjekt oder Ausdrucksobjekt durch Führen eines laufenden Zählers von Referenzen identifiziert, die auf dieses Objekt gemacht werden, wenn ein Netzwerk von Registern und Ausdrucksobjekten konstruiert wird. Vorzugsweise wird für jedes Ausdrucksobjekt ein Zähler der Anzahl von Referenzen auf dieses Ausdrucksobjekt von anderen Ausdrucksobjekten oder von Registerobjekten geführt, wobei der Zähler dem bestimmten Ausdrucksobjekt zugeordnet ist, angepasst wird, und zwar jedes Mal dann, wenn eine Referenz auf das Ausdrucksobjekt gemacht oder entfernt wird. Vorzugsweise werden ein Ausdrucksobjekt und alle Referenzen von diesem Ausdrucksobjekt eliminiert, wenn der Zähler für das Ausdrucksobjekt Null wird.

[0016] Vorzugsweise weist das Verfahren das Übersetzen des Programmcodes, der für die Ausführung durch einen Prozessor eines ersten Typs geschrieben ist, so dass der Programmcode von einem Prozessor eines zweiten Typs ausgeführt werden kann, und zwar unter Verwendung der erzeugten intermediären Repräsentation, auf.

[0017] Vorzugsweise wird der Übersetzungsschritt dynamisch durchgeführt, während das Programm läuft.

[0018] Vorzugsweise umfasst das Verfahren den Schritt des Optimierens des Programmcodes durch Optimieren der erzeugten intermediären Repräsentation. Vorzugsweise wird der Optimierungsschritt verwendet, um den Programmcode zu optimieren, der für die Ausführung durch einen Prozessor eines ersten Typs geschrieben wurde, so dass das Programm effizienter durch diesen Prozessor ausgeführt werden kann.

[0019] Vorzugsweise sind die Registerobjekte und die Ausdrucksobjekte in einem verzweigten baumartigen Netzwerk organisiert, das alle Registerobjekte an dem untersten Basisknoten oder der Baumstamm-Ebene des Netzwerks aufweist, wobei kein Registerobjekt in ein anderes Registerobjekt eingibt bzw. einspeist. Vorzugsweise halten die Mehrzahl von Registerobjekten variable Werte, die von dem Programmcode erzeugt werden sollen; und repräsentieren die Mehrzahl von Ausdrucksobjekten fixe Werte und/oder Beziehungen zwischen den fixen Werten und den variablen Werten gemäß dem Programmcode.

[0020] Außerdem bezieht sich die vorliegende Erfindung auf Verfahren, die die Erzeugung der wie hier dargelegten intermediären Repräsentation enthalten.

[0021] Gemäß einem zweiten Aspekt der vorliegenden Erfindung wird ein Verfahren zum dynamischen Übersetzen eines ersten Computerprogrammcodes, der für die Kompilierung und/oder Übersetzung und das Lau-

fen auf einer ersten programmierbaren Maschine geschrieben wurde, in einen zweiten Computerprogrammcode, um auf einer unterschiedlichen zweiten programmierbaren Maschine zu laufen, geschaffen, wobei das Verfahren aufweist: a) Erzeugen einer intermediären Repräsentation eines Blocks des ersten Computerprogrammcodes gemäß dem hier definierten Verfahren; b) Erzeugen eines Blocks des zweiten Computerprogrammcodes aus der intermediären Repräsentation; c) Ausführen des Blocks des zweiten Computerprogrammcodes auf der zweiten programmierbaren Maschine; und d) Wiederholen der Schritte a)–c) in Echtzeit für zumindest die Blöcke des ersten Computerprogrammcodes, die nötig sind, um eine laufende emulierte Ausführung des ersten Computerprogrammcodes auf der zweiten programmierbaren Maschine auszuführen.

[0022] Gemäß einem dritten Aspekt der vorliegenden Erfindung wird ein Verfahren zum Übersetzen eines Computerprogramms, das für die Ausführung durch einen Prozessor eines ersten Typs geschrieben wurde, so dass das Programm durch einen Prozessor eines zweiten Typs ausgeführt werden kann, geschaffen, wobei das Verfahren den folgende Schritt einschließt: Erzeugen einer intermediären Repräsentation in Übereinstimmung mit dem hier definierten Verfahren. Vorzugsweise findet die Übersetzung dynamisch statt und wird ausgeführt, während das Programm läuft.

[0023] Gemäß einem vierten Aspekt der vorliegenden Erfindung wird ein Verfahren zum Optimieren eines Computerprogramms geschaffen, wobei das Verfahren aufweist: Erzeugen einer intermediären Repräsentation in Übereinstimmung mit dem hier definierten Verfahren; und Optimieren der intermediären Repräsentation.

[0024] Gemäß einem fünften Aspekt der vorliegenden Erfindung wird ein System zum Erzeugen einer intermediären Repräsentation von Programmcode geschaffen, dass umfasst: eine Einrichtung zum Erzeugen einer Mehrzahl von Registerobjekten, welche abstrakte Register repräsentieren; und eine Einrichtung zum Erzeugen von Ausdrucksobjekten, von denen jedes ein unterschiedliches Element des Programmcodes, so wie es im Programmcode auftaucht, repräsentiert; dadurch gekennzeichnet, dass ein einzelnes Registerobjekt jeweils ein abstraktes Register repräsentiert; und jedes Ausdrucksobjekt durch ein Registerobjekt referenziert wird, auf das es sich entweder direkt oder indirekt über Bezugnahmen von anderen Ausdrucksobjekten bezieht.

[0025] Vorzugsweise weist das System einen ersten programmierbaren Prozessor; und ein Emulationssystem, das betreibbar ist, um Programmcode auszuführen, der für einen zweiten Prozessor auf dem ersten programmierbaren Prozessor geschrieben wurde, und zwar durch Erzeugung einer intermediären Repräsentation des Programmcodes, auf. Außerdem oder alternativ weist das System vorzugsweise einen ersten programmierbaren Computer; und ein Emulationssystem, das betreibbar ist, um Programmcode auszuführen, der für einen zweiten Computer auf dem ersten programmierbaren Computer geschrieben wird, und zwar durch Erzeugung einer intermediären Repräsentation des Programmcodes, auf.

[0026] Gemäß einem sechsten Aspekt der vorliegenden Erfindung wird ein Programmspeichermedium geschaffen, das ein Emulationssystem für die Ausführung von für einen ersten Computer auf einem zweiten Computer geschriebenen Programmcode speichert, wobei das Emulationssystem, wenn es durch den zweiten Computer ausgeführt wird, betreibbar ist, um eine intermediäre Repräsentation des Programmcodes gemäß dem hier definierten Verfahren zu erzeugen.

[0027] Gemäß einem siebenten Aspekt der vorliegenden Erfindung wird ein auf einem computerlesbaren Medium gespeichertes Emulationssystem geschaffen, wobei das System betreibbar ist, um durch ein Verfahren gemäß dem hier definierten Verfahren eine intermediäre Repräsentation von Programmcode zu erzeugen.

[0028] Ein Element des Programmcodes ist in der bevorzugten Ausführungsform der vorliegenden Erfindung geeignet eine Operation oder eine Unteroperation eines jeweiligen Codebefehls. Jeder jeweilige Codebefehl kann eine Anzahl solcher Elemente umfassen, so dass eine Anzahl von Ausdrucksobjekten erzeugt werden kann, die einen einzigen Codebefehl repräsentieren.

[0029] Beim Bilden einer intermediären Repräsentation ist es erwünscht, eine Repräsentation des Status eines jeweiligen Prozessors (z. B. seiner Register oder seines Speicherraums) aufzunehmen, der durch die intermediäre Repräsentation repräsentiert wird. In der vorliegenden Erfindung erfolgt dies auf besonders effiziente Weise durch Erzeugen abstrakter Register.

[0030] Wenigstens in bevorzugten Ausführungsformen der vorliegenden Erfindung braucht lediglich ein einziges Registerobjekt erzeugt zu werden, um ein gegebenes abstraktes Register zu repräsentieren (was vorzugsweise für alle abstrakten Register bei der Initialisierung erfolgt), wobei der Zustand jedes abstrakten Registers durch die Ausdrucksobjekte definiert wird, die durch das entsprechende Registerobjekt referenziert

werden. Wo durch ein gegebenes Registerobjekt mehr als ein Ausdrucksobjekt referenziert wird, wird ein "Baum" von Ausdrucksobjekten mit dem Registerobjekt als sein "Knoten" erzeugt. Die durch jedes der Registerobjekte referenzierten Ausdrucksbäume bilden zusammen einen "Ausdruckswald".

[0031] Es ist ein Vorteil der Erfindung, dass irgendein gegebenes Ausdrucksobjekt durch mehr als ein Register referenziert werden kann und folglich ein Ausdruck, der durch mehrere verschiedene Register verwendet wird, nicht für jedes dieser Register getrennt erzeugt und zugewiesen zu werden braucht, sondern einmal erzeugt und für jedes der Register referenziert werden kann. Mit anderen Worten, Ausdrucksbäume können durch Ausdrucksobjekte, die durch mehr als ein Registerobjekt referenziert werden, einander zugeordnet werden. Somit kann ein gegebenes Ausdrucksobjekt für eine Anzahl von Ausdrucksbäumen innerhalb des Ausdruckswalds gemeinsam sein.

[0032] Dadurch, dass die Erfindung vermeidet, mehrere Kopien des gleichen Ausdrucks zu erzeugen, verringert sie die Zeit, die das Erzeugen der intermediären Repräsentation dauert, und verringert sie den von der intermediären Repräsentation belegten Speicherraum.

[0033] Es ist ein weiterer Vorteil der vorliegenden Erfindung, dass Ausdrücke, die redundant werden, sehr effizient identifiziert werden können. Wenn einem Registerobjekt ein neuer Ausdruck zugewiesen wird, wird irgendein Ausdruck, der durch dieses Registerobjekt zuvor referenziert wurde, außer insofern er durch andere Registerobjekte referenziert wird, redundant. Diese mehreren Referenzen werden unter Verwendung der unten beschriebenen Referenzzählung erfasst.

[0034] Irgendein gegebenes Ausdrucksobjekt kann Referenzen von sich auf andere Ausdrucksobjekte und Referenzen auf sich von anderen Ausdrucksobjekten oder von abstrakten Registern haben. Vorzugsweise wird ein Zählerwert der Anzahl der Referenzen geführt, die auf jedes Ausdrucksobjekt führen. Jedes Mal, wenn eine Referenz auf ein Ausdrucksobjekt (entweder von einem Register oder von einem anderen Ausdrucksobjekt) gemacht oder entfernt wird, wird der Zähler für dieses Ausdrucksobjekt angepasst. Ein Zähler von Null gibt für ein gegebenes Ausdrucksobjekt an, dass es keine Referenzen gibt, die auf dieses Ausdrucksobjekt führen, und dass dieses Ausdrucksobjekt somit redundant ist.

[0035] Wenn ein Zähler für ein gegebenes Ausdrucksobjekt Null ist, wird dieses Ausdrucksobjekt vorzugsweise aus der intermediären Repräsentation eliminiert.

[0036] Wenn ein Ausdrucksobjekt eliminiert wird, führt die Löschung aller Referenzen, die von diesem Ausdrucksobjekt führen, zu jedem referenzierten Ausdrucksobjekt, dessen Referenzzähler dekrementiert ist. Wenn dieser dekrementierte Wert Null erreicht hat, kann das referenzierte Objekt seinerseits eliminiert werden, was veranlasst, dass die Referenzzähler des referenzierten Objekts ihrerseits dekrementiert werden.

[0037] Somit ermöglicht die intermediäre Repräsentation der Erfindung, effizient redundanten Code aufzufinden und zu eliminieren. Redundanter Code entsteht in binär übersetzten Programmen häufig, wenn der Inhalt eines Registers definiert und nachfolgend neu definiert wird, ohne zunächst verwendet worden zu sein. Die bekannten vorhandenen intermediären Repräsentationen erfordern, dass ein Protokoll geführt wird, das angibt, wann der Inhalt eines gegebenen Registers definiert wird, und das angibt, wann der Inhalt dieses Registers verwendet wird. Diese Protokollführung ist ein ineffizientes Verfahren der Identifizierung von redundantem Code. In der vorliegenden Erfindung wird redundanter Code aus der Sequenz der Zuweisungen zu den Registerobjekten und aus den Verwendungen der Registerobjekte sofort sichtbar.

[0038] Es wird nun lediglich beispielhaft eine spezifische Ausführungsform der vorliegenden Erfindung in Anwendung auf ein dynamisches Emulationssystem anhand der beigefügten Zeichnung beschrieben, in der:

[0039] Fig. 1 bis Fig. 5 schematische Darstellungen der Art und Weise sind, in der ein dynamisches Emulationssystem eine intermediäre Repräsentation eines Programms oder eines Basisblocks eines Programms erzeugt, wobei sie ebenfalls einen Ausdruckswald (Gruppe von Ausdrucksbäumen) zeigen.

[0040] Die im Folgenden beschriebene Ausführungsform der Erfindung ist ein System zur Emulation des Befehlsatzes eines Prozessors auf einem Prozessor eines anderen Typs. In der folgenden Beschreibung bezieht sich der Begriff betreffender Prozessor auf einen Prozessor, der durch ein Emulationssystem zu emulieren ist, während sich der Begriff Zielprozessor auf einen Prozessor bezieht, auf dem das Emulationssystem läuft. Das System ist ein dynamisches Binärübersetzungssystem, das im Wesentlichen dadurch arbeitet, dass es Basisblöcke von Befehlen im Code des betreffenden Prozessors, während sie für die Ausführung benötigt werden,

in Zielprozessorcode übersetzt. Das wie im Folgenden beschriebene Emulationssystem umfasst drei Hauptkomponenten, die als ein Front End, als ein Kern, und als ein Back End bezeichnet werden. Die Befehle des betreffenden Prozessors werden durch das Front End des Emulationssystems decodiert und in eine intermediäre Repräsentation umgesetzt. Der Kern des Emulationssystems analysiert und optimiert die intermediäre Repräsentation der Befehle des betreffenden Prozessors und das Back End setzt die intermediäre Repräsentation in Zielprozessorcode um, der auf dem Zielprozessor läuft.

[0041] Das Front End des Systems ist spezifisch für den betreffenden Prozessor, der emuliert wird. Das Front End konfiguriert das Emulationssystem in Reaktion auf die Form des betreffenden Prozessors, wobei es z. B. die Anzahl und die Namen der Register des betreffenden Prozessors angibt, die von der Emulation benötigt werden, und für das Back End die virtuellen Speicherabbilder angibt, die benötigt werden.

[0042] Die Befehle des betreffenden Prozessors werden in eine intermediäre Repräsentation in Basisblöcke umgesetzt, wobei jeder resultierende Block der intermediären Repräsentation (IR-Block) daraufhin für Emulations-, Caching- und Optimierungszwecke durch den Kern als eine Einheit behandelt wird.

[0043] Der Kern optimiert die durch das Front End erzeugte intermediäre Repräsentation. Der Kern besitzt unabhängig von dem betreffenden Prozessor und von dem Zielprozessor, die mit dem Emulationssystem verbunden sind, eine Standardform. Allerdings werden einige Kernbetriebsmittel, insbesondere Registernummern und -benennungen, und das genaue Wesen der IR-Blöcke durch ein individuelles Front End in der Weise konfiguriert, dass sie den Anforderungen dieser spezifischen Architektur des betreffenden Prozessors genügen.

[0044] Das Back End ist spezifisch für den Zielprozessor und wird durch den Kern aufgerufen, um die intermediäre Repräsentation in Zielprozessorbefehle zu übersetzen. Das Back End ist verantwortlich für das Zuordnen und für das Management der Zielprozessorregister, für das Erzeugen geeigneter Speicherlade- und Speicherspeicherbefehle, um den betreffenden Prozessor richtig zu emulieren, für das Implementieren einer Aufrufsequenz, die ermöglicht, dass der Kern dynamische Routinen aufruft, und um zu ermöglichen, dass diese dynamischen Routinen soweit erforderlich Back-End- und Front-End-Routinen aufrufen.

[0045] Der Betrieb des Emulationssystems wird nun ausführlicher beschrieben. Das System wird initialisiert, um geeignete Zuordnungen zwischen Front End, Kern und Back End zu erzeugen. Am Ende der Initialisierung wird ein Ausführungszyklus begonnen, wobei der Kern das Front End aufruft, um einen ersten Basisblock von Befehlen des betreffenden Prozessors zu decodieren. Das Front End arbeitet befehlsweise, wobei es jeden Befehl des Basisblocks des betreffenden Prozessors der Reihe nach decodiert und Kernroutinen aufruft, um für jede Unteroperation jedes Befehls eine intermediäre Repräsentation zu erzeugen. Wenn das Front End einen Befehl, der möglicherweise eine Änderung der Programmsequenz veranlassen könnte (z. B. einen Sprungbefehl, einen Aufrufbefehl oder einen Verzweigungsbefehl, gleich, ob bedingt oder unbedingt), decodiert, kehrt es zu dem Kern zurück, bevor es weitere Befehle des jeweiligen Prozessors decodiert (wobei es diesen Basisblock an Code abschließt).

[0046] Wenn das Front End einen Basisblock von Befehlen des betreffenden Prozessors in die intermediäre Repräsentation übersetzt hat, optimiert der Kern die intermediäre Repräsentation und ruft daraufhin das Back End auf, um dynamisch eine Sequenz von Befehlen in dem Zielprozessorcode (in den Zielbefehlen) zu erzeugen, die die intermediäre Repräsentation des Basisblocks implementieren. Wenn diese Sequenz von Zielbefehlen erzeugt worden ist, wird sie sofort ausgeführt. Die Sequenz von Zielprozessorbefehlen wird zur nachfolgenden Wiederverwendung (falls sie nicht zuerst überschrieben wird) in einem Cache aufbewahrt.

[0047] Wenn die Zielprozessorbefehle ausgeführt worden sind, wird ein Wert zurückgegeben, der eine Adresse angibt, die als nächste auszuführen ist. Mit anderen Worten, der Zielprozessorcode bewertet irgendeinen Verzweigungs-, Aufruf- oder Sprungbefehl, ob bedingt oder unbedingt, am Ende des Basisblocks und gibt seine Wirkung zurück. Dieser Prozess der Übersetzung und Ausführung von Basisblöcken wird fortgesetzt, bis ein Basisblock festgestellt wird, der bereits übersetzt worden ist.

[0048] Wenn Zielcode, der den nächsten Basisblock repräsentiert, zuvor verwendet und in dem Cache gespeichert worden ist, ruft der Kern einfach diesen Zielcode auf. Wenn das Ende des Basisblocks erreicht ist, liefert der Zielcode wieder die Adresse des nächsten auszuführenden betreffenden Befehls, wobei der Zyklus fortgesetzt wird.

[0049] Sowohl die intermediäre Repräsentation als auch der Zielprozessorcode sind Basisblöcken der jeweiligen Prozessorbefehle zugeordnet. Die intermediäre Repräsentation ist in der Weise zugeordnet, dass der Op-

timierer effiziente Emulationen von Gruppen häufig ausgeführter IR-Blöcke erzeugen kann, und der Zielcode ist in der Weise zugeordnet, dass die zweite und nachfolgende Ausführungen des gleichen Basisblocks den Zielcode direkt ausführen können, ohne sich den Organisationsaufwand des erneuten Decodierens der Befehle zuzuziehen.

[0050] Das Front End fordert, dass zur Initialisierungszeit eine geforderte Anzahl abstrakter Register in dem Kern definiert werden. Diese abstrakten Register (als Ri bezeichnet) repräsentieren die physikalischen Register, die von den Befehlen des betreffenden Prozessors verwendet würden, falls sie auf einem betreffenden Prozessor laufen würden. Die abstrakten Register definieren den Zustand des betreffenden Prozessors, der emuliert wird, indem sie die erwartete Wirkung der Befehle auf die Register des betreffenden Prozessors repräsentieren.

[0051] Die intermediäre Repräsentation repräsentiert das Programm des betreffenden Prozessors, indem sie den abstrakten Registern Ausdrucksobjekte zuweist. Ausdrucksobjekte sind eine Einrichtung, um die Wirkung z. B. einer einzelnen arithmetischen, logischen oder bedingten Operation in der intermediären Repräsentation zu repräsentieren. Da viele Befehle des betreffenden Prozessors eine Manipulation von Daten ausführen, erzeugen die meisten Befehle Ausdrucksobjekte, um ihre einzelnen Unteroperationen zu repräsentieren. Ausdrucksobjekte werden z. B. verwendet, um Add-Ooperationen, Bedingungseinstelloperationen, die bedingte Bedingungsauswertung in bedingten Verzweigungen sowie Speicherleseoperationen zu repräsentieren. Die abstrakten Register werden auf Ausdrucksobjekte referenziert, die auf andere Ausdrucksobjekte referenziert werden, so dass jeder Basisblock der Befehle des betreffenden Prozessors durch eine Anzahl quer referenzierter Ausdrucksobjekte repräsentiert wird, die als ein Ausdruckswald betrachtet werden können.

[0052] Es werden eine Reihe veranschaulichter Beispiele verwendet, um zu vermitteln, wie das Emulationssystem Ausdrucksobjekte (die als Ausdrücke bezeichnet werden) und abstrakte Register verwendet, um eine intermediäre Repräsentation der Befehle des betreffenden Prozessors aufzubauen. Die Fig. 1 bis Fig. 5 zeigen schrittweise, wie der folgende Pseudo-Assembler-Code im Kern unter Verwendung abstrakter Register repräsentiert wird:

```
1:  MOVE  #3       → R0
2:  MOVE  R6       → R2
3:  ADD   R0,R2    → R1
4:  MUL   R1,#5    → R5
5:  AND   R3,R1    → R4
6:  MOVE  #5       → R1
7:  SUB   #1,R3    → R2
8:  LOAD  #3fd0    → R0
```

[0053] In Fig. 1 ist die Repräsentation des MOVE-Befehls in Zeile **1** gezeigt; es wird ein Langkonstantenausdruck #3 erzeugt und durch Erzeugen einer Bezugnahme, die von R0 auf #3 führt, dem abstrakten Register R0 zugewiesen. Der MOVE-Befehl in Zeile **2** referenziert den Wert des abstrakten Registers R6, wobei ein Registerreferenzausdruck verwendet wird, um diesen zu repräsentieren, und R2 zugewiesen wird. Der Registerreferenzausdruck (RegRef-Ausdruck) in Fig. 1, @R6, repräsentiert den Wert des Registers R6, welcher es auch sein mag. Der RegRef-Ausdruck @R6 wird zu der momentanen Definition des Registers R6. Von diesem Punkt an gibt er als eine Definition den Ausdruck @R6 zurück, es sei denn, dass das Register R6 neu definiert wird.

[0054] Der Operand eines Befehls des betreffenden Prozessors kann entweder eine Konstante oder eine Referenz auf ein Register sein. Die Repräsentation eines konstanten Operanden ist wie in Fig. 1 gezeigt unkompliziert. Anders ist die Situation dagegen, wenn ein Operand auf ein Register Bezug nimmt. Die Repräsentation von Zeile **3** des Pseudo-Assembler-Codes ist in Fig. 2 gezeigt, aus der zu sehen ist, dass die ADD-Operation durch Referenz von R1 auf einen Add-Ausdruck dem abstrakten Register R1 zugewiesen wird. Der ADD-Befehl in Zeile **3** nimmt auf die Register R0 und R2 Bezug, wobei der Ausdruck, der jedes dieser Register definiert, bereits in der intermediären Repräsentation gebaut worden ist. Wenn der Add-Ausdruck erzeugt wird, fragt er die abstrakten Register R0 und R2 ab, um ihre definierenden Ausdrücke zu liefern, wobei der Add-Ausdruck (der dem abstrakten Register R1 zugewiesen ist) eine Referenz auf diese vornimmt. In Fig. 2 ist die interme-

diäre Repräsentation des ADD-Befehls gezeigt. Mit anderen Worten, der Inhalt des abstrakten Registers R1 ist ein Ausdruck, der die in den abstrakten Registern R0 und R2 gehaltenen Ausdrücke referenziert. Jeder Pfeil in den Fig. 1 und Fig. 2 repräsentiert eine Referenz, die entweder wie im Fall von R0 → #3 ein Register auf einen Ausdruck referenzieren kann oder wie im Fall von #3 ← + → @R6 einen Ausdruck auf einen anderen Ausdruck referenzieren kann. Der Ausdruck @R6 besitzt zwei Referenzen, eine vom Register R2 und die andere vom Add-Ausdruck.

[0055] Ein wie in Zeile **4** des obigen Codes enthaltener MUL-Befehl kann als ein typischer Datenflussbefehl betrachtet werden. Entweder durch Erzeugen neuer Unterausdrücke oder durch Referenzieren vorhandener Ausdrücke wird ein Ausdruck der obersten Ebene gebaut, wobei dieser Ausdruck der obersten Ebene als seine Definition einem Register zugewiesen wird. Die intermediäre Repräsentation des MUL-Befehls ist in Fig. 3 gezeigt. Es wird ein Mul-Ausdruck erzeugt, der den in dem abstrakten Register R1 gehaltenen Ausdruck referenziert und einen Langkonstantenausdruck #5 referenziert, und dem abstrakten Register R5 zugewiesen.

[0056] In Fig. 4 ist der Und-Ausdruck aus Zeile **5** des obigen Codes gezeigt. Dieser Ausdruck referenziert unter Verwendung eines RegRef-Ausdrucks auf die gleiche Weise wie oben in Bezug auf Fig. 1 beschrieben ein Register, dessen Definition noch zu bauen ist (d. h. R3).

[0057] In den bisher gegebenen Beispielen ist angenommen worden, dass ein Register erstmals innerhalb eines besonderen Basisblocks definiert wird. Fig. 5 veranschaulicht, was geschieht, wenn ein Register, das bereits definiert worden ist, wie durch den MOVE-Befehl aus Zeile **6** des obigen Codes neu definiert wird. Während in den Fig. 2 bis Fig. 4 ein Pfeil R1 auf einen Add-Ausdruck referenzierte, wird diese Referenz nun entfernt und ein neuer Referenzpfeil erzeugt, um R1 auf den Langkonstantenausdruck #5 zu referenzieren.

[0058] Ebenso wie der Add-Ausdruck mit R1 verbunden war, war er ebenfalls mit dem Mul-Ausdruck und mit dem Und-Ausdruck verbunden und existiert somit wie in Fig. 5 gezeigt weiter (während der Add-Ausdruck dagegen ohne Referenzen verbleiben würde, nachdem R1 neu definiert wurde, falls der Add-Ausdruck nur eine Referenz, diejenige vom Register R1, gehabt hätte; in diesem Fall wäre der Add-Ausdruck als 'tot' bekannt und redundant). Außerdem veranschaulicht Fig. 5 die Wirkung der SUB-Operation aus Zeile **7** des Pseudo-Assembler-Codes.

[0059] Die letzte Zeile, Zeile **8**, des als intermediäre Repräsentation zu repräsentierenden Pseudo-Assembler-Codes ist ein LOAD-Befehl. Ein Load-Ausdruck, der diesen Befehl repräsentiert, ist in Fig. 5 gezeigt und durch das Register R0 referenziert. Der Load-Ausdruck kann als ein Typ eines unären Operators gedacht werden, der das Ergebnis der Anwendung des LOAD-Befehls auf seinen einzigen Ausdrucksoperanden repräsentiert. In Fig. 5 repräsentiert LOAD → #3fd0 den Wert an einem Speicherplatz 3fd0, welches dieser Wert auch sein mag. Dadurch, dass der Load-Ausdruck, je nachdem, welche Daten im Speicher gespeichert sind, irgendeinen möglichen Wert repräsentieren kann, besitzt der Load-Ausdruck ähnliche Eigenschaften wie der RegRef-Ausdruck.

[0060] Es wird ein Referenzzähler geführt, der die Anzahl der Referenzen angibt, die auf jedes Ausdrucksobjekt führen (der Referenzzähler irgendeines gegebenen Ausdrucksobjekts enthält nicht Referenzen von diesem Ausdrucksobjekt). Der Referenzzähler für dieses Ausdrucksobjekt wird jedes Mal angepasst, wenn eine Referenz auf ein Ausdrucksobjekt (entweder von einem Register oder von einem anderen Ausdrucksobjekt) gemacht oder von diesem Ausdrucksobjekt entfernt wird. Ein Referenzzähler von Null gibt für ein gegebenes Ausdrucksobjekt an, dass es keine Referenzen gibt, die auf dieses Ausdrucksobjekt führen, und dass dieses Ausdrucksobjekt somit redundant ist: Wenn ein Referenzzähler für ein gegebenes Ausdrucksobjekt Null ist, wird dieses Ausdrucksobjekt aus der intermediären Repräsentation eliminiert.

[0061] Wenn ein Ausdrucksobjekt eliminiert worden ist, werden irgendwelche Referenzen, die von diesem Ausdrucksobjekt führen, ebenfalls eliminiert und der Referenzzähler dieser Ausdrucksobjekte, auf die die Referenzen führen, dementsprechend angepasst. Der Prozess des Eliminierens von Ausdrucksobjekten mit einem Referenzzähler Null und des Eliminierens von Referenzen, die von einem solchen Objekt führen, wird den Ausdruckswald entlang verfolgt.

[0062] Durch Eliminieren redundanter Zeilen von Code des betreffenden Prozessors kann eine weitere Optimierung der intermediären Generalisierung erreicht werden.

[0063] Obgleich die obige Beschreibung auf die Emulation gerichtet ist, ist für den Fachmann auf dem Gebiet klar, dass die Erfindung ebenfalls in anderen Anwendungen, z. B. bei der Optimierung von Code während der

Kompilierung, verwendet werden kann.

Patentansprüche

1. Verfahren zum Erzeugen einer intermediären Repräsentation von Programmcode, wobei das Verfahren die folgenden computerimplementierten Schritte aufweist:
Erzeugen einer Mehrzahl von Registerobjekten (R0–R5), welche abstrakte Register repräsentieren; und
Erzeugen von Ausdrucksobjekten (*, #5), von denen jedes ein unterschiedliches Element des Programmcodes repräsentiert, wenn das Element in dem Programmcode auftaucht;
dadurch gekennzeichnet, daß:
ein einzelnes Registerobjekt (R0, R5) ein jeweiliges abstraktes Register repräsentiert; und dadurch daß jedes Ausdrucksobjekt (*, #5) durch ein Registerobjekt (R0, R5) referenziert wird, zu dem es entweder direkt in Beziehung steht oder indirekt über Referenzen von anderen Ausdrucksobjekten (Stern, #5).

2. Verfahren nach Anspruch 1, bei dem der Programmcode durch einen Befehlssatz eines Prozessors ausgedrückt wird.

3. Verfahren nach Anspruch 2, bei dem die Registerobjekte (R0–R5) abstrakte Register repräsentieren, die den Registern des Prozessors entsprechen.

4. Verfahren nach Anspruch 1, 2, oder 3, bei dem jeder der Schritte sequentiell für Basisblöcke des Programmcodes ausgeführt wird, die jeweils nur einen effektiven Eintrittspunktbefehl und einen effektiven Austrittspunktbefehl aufweisen.

5. Verfahren nach einem der vorhergehenden Ansprüche, bei dem zumindest einige der Ausdrucksobjekte (*, #5) in mehr als eines der Registerobjekte (R0, R5) eingeben.

6. Verfahren nach einem der vorhergehenden Ansprüche, bei dem die Ausdrucksobjekte (*, #5) nicht dupliziert werden.

7. Verfahren nach einem der vorhergehenden Ansprüche, bei dem ein einzelnes Ausdrucksobjekt (*, #5) für ein gegebenes Element des Programmcodes erzeugt wird, und bei dem jedes Ausdrucksobjekt (*, #5) durch alle der Registerobjekte (R0–R5) referenziert wird, auf die es sich bezieht.

8. Verfahren nach einem der vorhergehenden Ansprüche, bei dem eines der Registerobjekte (R0, R5) oder eines der Ausdrucksobjekte (*, #5) eliminiert wird, wenn es redundant oder unnötig wird.

9. Verfahren nach Anspruch 8, bei dem ein redundantes oder unnötiges Registerobjekt (R0, R5) oder Ausdrucksobjekt (*, #5) durch Führen eines laufenden Zählers von Referenzen identifiziert wird, die auf dieses Objekt gemacht werden, wenn ein Netzwerk von Registern und Ausdrucksobjekten konstruiert wird.

10. Verfahren nach Anspruch 9, bei dem für jedes Ausdrucksobjekt (*, #5) ein Zähler der Anzahl von Referenzen auf dieses Ausdrucksobjekt (*, #5) von anderen Ausdrucksobjekten (*, #5) oder von Registerobjekten (R0–R5) geführt wird, wobei der Zähler dem bestimmten Ausdrucksobjekt (*, #5) zugeordnet ist, angepaßt wird, und zwar jedesmal dann, wenn eine Referenz auf das Ausdrucksobjekt (*, #5) gemacht oder entfernt wird.

11. Verfahren nach Anspruch 10, bei dem ein Ausdrucksobjekt (*, #5) und alle Referenzen von diesem Ausdrucksobjekt (*, #5) eliminiert werden, wenn der Zähler für das Ausdrucksobjekt (*, #5) Null wird.

12. Verfahren nach einem der vorhergehenden Ansprüche, welches aufweist:
Übersetzen des Programmcodes, der für die Ausführung durch einen Prozessor eines ersten Typs geschrieben ist, so daß der Programmcode von einem Prozessor eines zweiten Typs ausgeführt werden kann, und zwar unter Verwendung der erzeugten intermediären Representation.

13. Verfahren nach Anspruch 12, wobei der Übersetzungsschritt dynamisch durchgeführt wird, während das Programm läuft.

14. Verfahren nach einem der vorhergehenden Ansprüche, wobei die Registerobjekte (R0–R5) und die Ausdrucksobjekte (*, #5) in einem verzweigten baumartigen Netzwerk organisiert sind, das alle Registerobjekte (R0–R5) an dem untersten Basisknoten oder der Baumstamm-Ebene des Netzwerks aufweist, wobei kein

Registerobjekt in ein anderes Registerobjekt eingibt bzw. einspeist.

15. Verfahren nach einem der vorhergehenden Ansprüche, bei dem
die Mehrzahl von Registerobjekten (R0–R5) variable Werte halten, die von dem Programmcode erzeugt werden sollen; und
die Mehrzahl von Ausdrucksobjekten (*, #5) feste Werte und/oder Beziehungen zwischen den festen Werten und den variablen Werten gemäß dem Programmcode repräsentieren.

16. Verfahren zum Übersetzen eines Computerprogramms, das für die Ausführung durch einen Prozessor des ersten Typs geschrieben wurde, so daß das Programm durch einen Prozessor eines zweiten Typs ausgeführt werden kann, wobei das Verfahren folgende Schritte einschließt:
Erzeugen einer intermediären Repräsentation in Übereinstimmung mit dem Verfahren gemäß einem der Ansprüche 1 bis 11.

17. Verfahren nach Anspruch 16, wobei die Übersetzung dynamisch stattfindet und ausgeführt wird, während das Programm läuft.

18. Verfahren nach Anspruch 16 oder 17, wobei das Verfahren dynamisch ersten Computerprogrammcode, der für die Kompilierung und/oder Übersetzung und das Laufen auf einer ersten programmierbaren Maschine geschrieben wurde, in einen zweiten Computerprogrammcode übersetzt, um auf einer unterschiedlichen zweiten programmierbaren Maschine zu laufen, wobei das Verfahren aufweist:
a) Erzeugen der intermediären Repräsentation für einen Block des ersten Computerprogrammcodes;
b) Erzeugen eines Blocks des zweiten Computerprogrammcodes aus der intermediären Repräsentation;
c) Ausführen des Blocks des zweiten Computerprogrammcodes auf der zweiten programmierbaren Maschine; und
d) Wiederholen der Schritte a)–c) in Echtzeit für zumindest die Blöcke des ersten Computerprogrammcodes, die nötig sind, um eine laufende emulierte Ausführung des ersten Computerprogrammcodes auf der zweiten programmierbaren Maschine auszuführen.

19. Verfahren zum Optimieren eines Computerprogramms, wobei das Verfahren aufweist:
Erzeugen einer intermediären Repräsentation in Übereinstimmung mit dem Verfahren gemäß einem der Ansprüche 1 bis 11; und
Optimieren der intermediären Repräsentation.

20. Verfahren nach Anspruch 19, wobei das Verfahren verwendet wird, um ein Computerprogramm zu optimieren, das für die Ausführung durch einen Prozessor eines ersten Typs geschrieben wurde, so daß das Programm effizienter durch diesen Prozessor ausgeführt werden kann.

21. System zum Erzeugen einer intermediären Repräsentation von Programmcode, welches aufweist:
eine Einrichtung zum Erzeugen einer Mehrzahl von Registerobjekten (R0–R5), welche abstrakte Register repräsentieren; und
eine Einrichtung zum Erzeugen von Ausdrucksobjekten (Stern, #5), von denen jedes ein unterschiedliches Element des Programmcodes, so wie es im Programmcode auftaucht, repräsentiert;
dadurch gekennzeichnet, daß
ein einzelnes Registerobjekt (R0, R5) jeweils ein abstraktes Register repräsentiert; und
jedes Ausdrucksobjekt (*, #5) durch ein Registerobjekt (R0, R5) referenziert wird, auf das es sich entweder direkt oder indirekt über Bezugnahmen von anderen Ausdrucksobjekten (*, #5) bezieht.

22. System nach Anspruch 21, welches aufweist:
einen ersten programmierbaren Prozessor; und
wobei das System von Anspruch 21 zusammen verwendet wird mit
einem Emulationssystem, das betreibbar ist, um Programmcode auszuführen, der für einen zweiten Prozessor auf dem ersten programmierbaren Prozessor geschrieben wurde, und zwar durch Erzeugung einer intermediären Repräsentation des Programmcodes.

23. System nach Anspruch 21, welches aufweist:
einen ersten programmierbaren Computer; und
wobei das System von Anspruch 21 zusammen verwendet wird mit
einem Emulationssystem, das betreibbar ist, um Programmcode auszuführen, der für einen zweiten Computer auf dem ersten programmierbaren Computer geschrieben wird, und zwar durch Erzeugung einer intermediä-

ren Repräsentation des Programmcodes.

24. System nach einem der Ansprüche 21 bis 23, bei dem:
die Registerobjekte (R0–R5) und die Ausdrucksobjekte (*, #5) in einem verzweigten baumartigen Netzwerk lokalisiert sind, wobei alle Registerobjekte (R0–R5) an dem untersten Basisknoten oder Baumstamm-Pegel des Netzwerks liegen, wobei kein Registerobjekt in ein anderes Registerobjekt einspeist.

25. System nach einem der Ansprüche 21 bis 24, bei dem:
die Mehrzahl von Registerobjekten (R0–R5) variable Werte halten, die von dem Programmcode erzeugt werden sollen; und
die Vielzahl von Ausdrucksobjekten (*, #5) fixe Werte und/oder Beziehungen zwischen den fixen Werten und den variablen Werten gemäß dem Programmcode repräsentieren.

26. Programmspeichermedium, welches Befehle speichert, die zur Ausführung all der Schritte gemäß einem der Verfahren nach Anspruch 1 bis 15 dienen.

Es folgen 3 Blatt Zeichnungen

Anhängende Zeichnungen

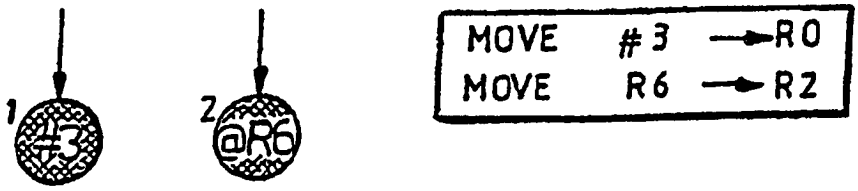

FIG. 1

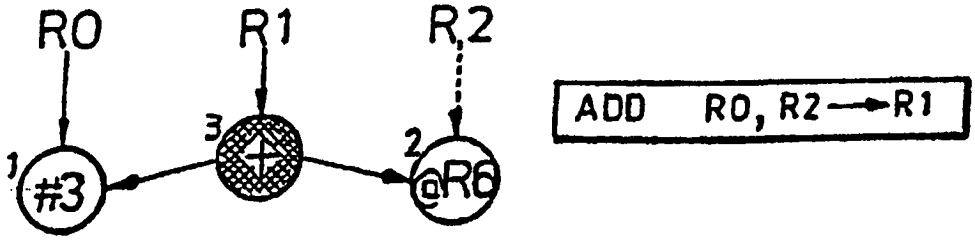

FIG. 2

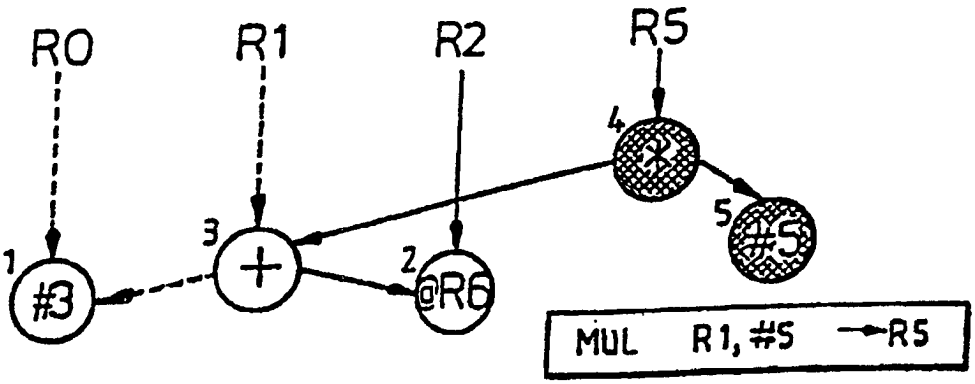
R0
R1
R2
R5
4
*
5
#5
3
+
1
#3
2
@R6
MUL R1, #5 → R5

FIG. 3

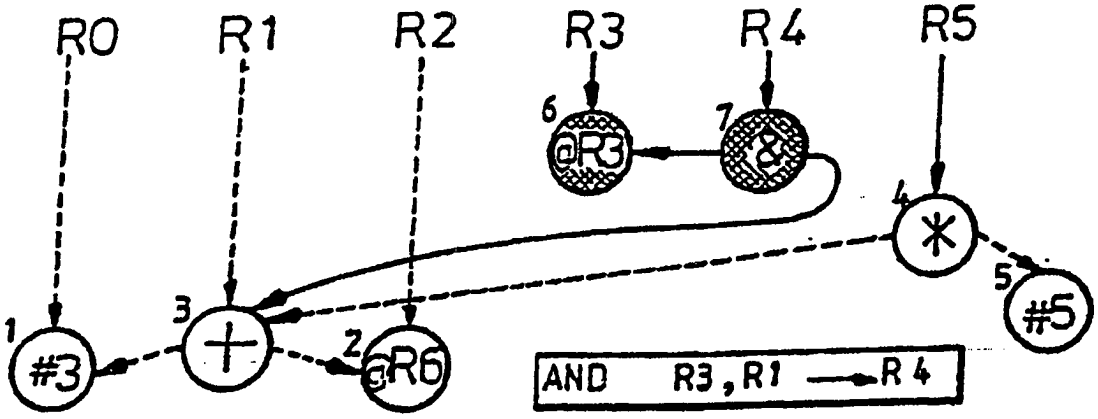
R0
R1
R2
R3
R4
R5
6
@R3
7
&
4
*
5
#5
3
+
1
#3
2
@R6
AND R3, R1 → R4

FIG. 4

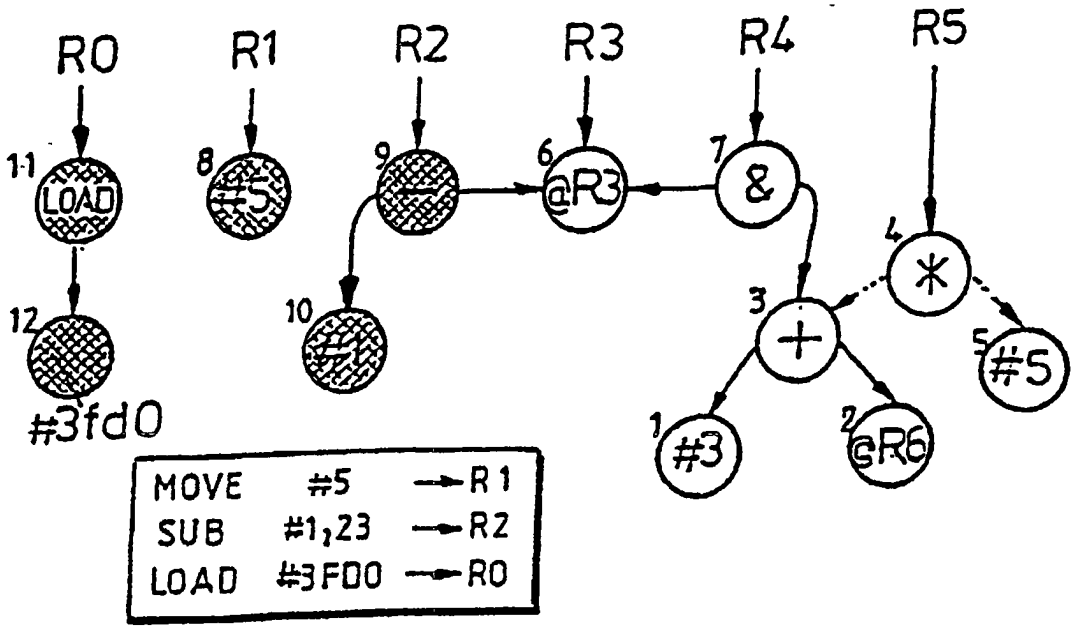
R0
R1
R2
R3
R4
R5
11
LOAD
8
#5
9
6
@R3
7
&
4
*
12
#3fd0
10
#1
3
+
5
#5
1
#3
2
@R6
MOVE #5 → R1
SUB #1,23 → R2
LOAD #3FD0 → R0

FIG. 5