(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第4536718号
(P4536718)

(45)発行日　平成22年9月1日(2010.9.1)　(24)登録日　平成22年6月25日(2010.6.25)

(51)Int.Cl.　　F I
G06F 9/455 (2006.01)　　G06F 9/44 ３１０A
G06F 9/30 (2006.01)　　G06F 9/30 ３１０E

請求項の数 20　(全 26 頁)

(21)出願番号	特願2006-506194 (P2006-506194)
(86)(22)出願日	平成16年4月28日 (2004.4.28)
(65)公表番号	特表2006-525572 (P2006-525572A)
(43)公表日	平成18年11月9日 (2006.11.9)
(86)国際出願番号	PCT/GB2004/001824
(87)国際公開番号	WO2004/097631
(87)国際公開日	平成16年11月11日 (2004.11.11)
審査請求日	平成19年4月19日 (2007.4.19)
(31)優先権主張番号	0310180.5
(32)優先日	平成15年5月2日 (2003.5.2)
(33)優先権主張国	英国 (GB)
(31)優先権主張番号	0326437.1
(32)優先日	平成15年11月13日 (2003.11.13)
(33)優先権主張国	英国 (GB)

(73)特許権者　390009531
インターナショナル・ビジネス・マシーンズ・コーポレーション
ＩＮＴＥＲＮＡＴＩＯＮＡＬ　ＢＵＳＩＮＥＳＳ　ＭＡＳＣＨＩＮＥＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国１０５０４　ニューヨーク州　アーモンク　ニュー　オーチャード　ロード
(74)代理人　100108501
弁理士　上野　剛史
(74)代理人　100112690
弁理士　太佐　種一
(74)代理人　100091568
弁理士　市位　嘉宏

最終頁に続く

(54)【発明の名称】プログラム・コード変換用の中間表現を生成するための改善されたアーキテクチャ

(57)【特許請求の範囲】

【請求項１】

プロセッサと該プロセッサに結合されたメモリとを有するとともに、目的コンピューティング環境において使用され、対象コンピューティング環境において適切な対象プログラム・コードをトランスレートして、前記目的コンピューティング環境に適切な目的プログラム・コードを生成するためのトランスレータ装置であって、

前記対象プログラム・コードにおける命令をデコードするように構成されたデコーディング機構と、

前記デコードされた命令の中間表現を生成するように構成された中間表現生成機構であって、該生成機構は、前記中間表現における複数のＩＲノードを、少なくともベース・ノードおよび複合ノードを含む複数の可能なタイプのＩＲノードから選択された対象プログラム・コードの命令によって行われる式、計算、および処理の抽象表現として設けることを含み、前記ベース・ノードのセマンティクスを、より単純なセマンティクスを表わす他のノードに分解することができないように、前記ベース・ノードは、対象プログラム・コードの最も基本的なセマンティクスを表わし、前記複合ノードによって、対象プログラム・コードにおける複合命令のセマンティクスは、ベース・ノード表現の場合よりもコンパクトに表現される、中間表現生成機構と、

前記デコードされた対象プログラム・コードにおける個々の命令に対する中間表現においてどのタイプのＩＲノードを生成するのかを決定するように構成された中間表現タイプ決定機構と、

を備えるトランスレータ装置。
【請求項２】
前記ベース・ノードは、複数の可能な対象アーキテクチャに渡って汎用的である、請求項１に記載のトランスレータ装置。
【請求項３】
前記複合ノードは、直接タイプ命令を表わし、該直接タイプ命令では、定数オペランド値が直接フィールドにおいて直接タイプ命令自体にエンコードされている、請求項１に記載のトランスレータ装置。
【請求項４】
前記複合ノードを、複数のベース・ノードに分解して、前記デコードされたプログラム・コードにおける命令の同じセマンティクスを表わすようにすることができる、請求項１に記載のトランスレータ装置。
【請求項５】
前記プログラム・コードは、対象アーキテクチャによって実行されるように構成され、前記中間表現生成機構は、さらに、対象アーキテクチャ上で対応して構成可能な特性に対してのみ複合ノードを生成するための複合ノード生成機構を含む、請求項１に記載のトランスレータ装置。
【請求項６】
前記複数の可能なタイプのＩＲノードは、さらに、ポリモーフィック・ノードを含む、請求項１に記載のトランスレータ装置。
【請求項７】
前記対象プログラム・コードは、対象アーキテクチャ上での実行のために構成されるとともに、目的アーキテクチャ上での実行のために目的コードに動的にトランスレートされ、前記中間表現生成機構は、さらに、
ポリモーフィック・ノードを含むように中間表現を生成するためのポリモーフィック・ノード生成機構であって、前記ポリモーフィック・ノードは、前記対象コードにおける特定の命令に特有の目的アーキテクチャの関数に対する関数ポインタを含む、ポリモーフィック・ノード生成機構、
を含む、請求項６に記載のトランスレータ装置。
【請求項８】
前記ポリモーフィック・ノード生成機構は、目的アーキテクチャの特性によって、特定の対象命令のセマンティクスが、ベース・ノードとして実現されたときに失われる場合に、ポリモーフィック・ノードを生成する、請求項７に記載のトランスレータ装置。
【請求項９】
各ポリモーフィック・ノードは、前記対象コードにおける特定の命令と目的アーキテクチャの関数との組み合わせに特有である、請求項７に記載のトランスレータ装置。
【請求項１０】
前記中間表現タイプ決定機構は、さらに、ポリモーフィック・ノードとして実現すべきポリモーフィック命令のリスト上の命令に対応する対象コードにおける命令を識別するためのポリモーフィック識別機構を含み、
対象命令が、前記ポリモーフィック命令のリスト上の命令に対応する場合、前記中間表現生成機構は、前記ポリモーフィック命令のリスト上の命令に対応する対象命令に対してのみポリモーフィック・ノードを生成する、請求項７に記載のトランスレータ装置。
【請求項１１】
前記複数の可能なタイプのＩＲノードは、さらに、アーキテクチャ特定ノードを含む、請求項１に記載のトランスレータ装置。
【請求項１２】
前記対象プログラム・コードは、対象アーキテクチャ上での実行のために構成されるとともに、目的アーキテクチャ上での実行のために目的コードに動的にトランスレートされ、

前記中間表現生成機構は、さらに、
対象アーキテクチャと目的アーキテクチャとの特定の組み合わせに特有のアーキテクチャ特定ノードを含むように中間表現を生成するためのアーキテクチャ特定ノード生成機構を含む、請求項１１に記載のトランスレータ装置。
【請求項１３】
前記中間表現生成機構は、
最初に、前記対象プログラム・コードにおけるすべての命令を、複数の対象アーキテクチャ特定ノードとして表わすことであって、各対象アーキテクチャ特定ノードは、前記対象プログラム・コードにおける個々の命令に対応する、表わすこと、
前記対象プログラム・コードにおける命令が、目的アーキテクチャに特化された変換関数を与える命令であるか否かを判断すること、
目的アーキテクチャに特化された変換関数を与えると判断された命令に対しては、対象アーキテクチャ特定ノードを目的アーキテクチャ特定ノードに変換すること、
目的アーキテクチャに特化されたコード生成関数を与えるとは識別されない残りの対象アーキテクチャ特定ノードから、ベース・ノードを生成すること、
を実行するように構成されている、請求項１２に記載のトランスレータ装置。
【請求項１４】
前記目的アーキテクチャ特定ノードから、目的アーキテクチャに対して特化されている対応する目的コードを生成するための特化目的コード生成機構、
をさらに備える、請求項１２に記載のトランスレータ装置。
【請求項１５】
前記ベース・ノードから、目的アーキテクチャに対して特化されていない対応する目的コードを生成するための非特化目的コード生成機構、
をさらに備える、請求項１２に記載のトランスレータ装置。
【請求項１６】
前記生成されたポリモーフィック・ノードによって、目的コードを生成する間に割り当てられるべきレジスタが特定される、請求項７に記載のトランスレータ装置。
【請求項１７】
前記生成されたポリモーフィック・ノードを汎用的なカーネル最適化において利用することは、前記ポリモーフィック・ノードにおける関数ポインタからの情報を推測することによって行なわれ、該情報は、推測されない場合には、前記ポリモーフィック・ノードからは決定不可能な場合もある、請求項７に記載のトランスレータ装置。
【請求項１８】
対象命令が、前記ポリモーフィック命令のリスト上の命令に対応する場合、前記中間表現生成機構は、前記ポリモーフィック命令のリスト上の命令に対応する対象命令に対してポリモーフィック・ノードまたはベース・ノードのいずれかを生成する、請求項１０に記載のトランスレータ装置。
【請求項１９】
対象コードの目的コードへのトランスレーションの間にコンピュータが中間表現を生成する方法であって、
前記コンピュータのデコーディング機構が、対象コードにおける複数の命令をデコードするステップと、
前記コンピュータの中間表現生成機構が、前記デコードされた命令の中間表現をメモリ内に生成するステップであって、該生成するステップは、前記中間表現における複数のＩＲノードを、少なくともベース・ノードおよび複合ノードを含む複数の可能なタイプのＩＲノードから選択された対象コードの命令によって行われる式、計算、および処理の抽象表現として設けることを含み、前記ベース・ノードのセマンティクスを、より単純なセマンティクスを表わす他のノードに分解することができないように、前記ベース・ノードは、対象コードの最も基本的なセマンティクスを表わし、前記複合ノードによって、プログラム・コードにおける複合命令のセマンティクスは、ベース・ノード表現の場合よりもコ

ンパクトに表現される、前記生成するステップと、
前記コンピュータの中間表現タイプ決定機構が、前記複数の可能なタイプのＩＲノードから少なくとも一つのタイプのＩＲノードを決定するステップであって、前記デコードされた対象コードにおける個々の命令に対する中間表現において生成するように決定する、前記決定するステップと
を含む、前記方法。
【請求項２０】
対象コードの目的コードへのトランスレーションの間に中間表現を生成するコンピュータに、請求項１９に記載の方法の各ステップを実行させるためのコンピュータ・プログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、コンピュータおよびコンピュータ・ソフトウェアの分野に関する。より詳細には、たとえばコード・トランスレータ、エミュレータ、アクセラレータにおいて有用なプログラム・コード変換方法および装置に関する。
【背景技術】
【０００２】
組込型および非組込型ＣＰＵの市場では、主流的な命令セット・アーキテクチャ（ＩＳＡ：Instruction Set Architecture）に対する大量のソフトウェアが存在する。これらのソフトウェアを、性能を上げるために「アクセラレート」することもできるし、より優れたコスト／性能の利点を示す無数の高性能なプロセッサに「トランスレート」することもできる。ただし、このようなプロセッサは、関連するソフトウェアに透過的にアクセスできなければならない。主流的なＣＰＵアーキテクチャもまた、そのＩＳＡに歩調を合わせられるため、性能または市場の範囲を発展させることができず、「複合ＣＰＵ（Synthetic CPU ）」共通アーキテクチャの利点を受ける。
【０００３】
第１のタイプのコンピュータ・プロセッサ（「対象（subject ）」プロセッサ）用に書かれたプログラム・コードを、第２のタイプのプロセッサ（「目的（target）」プロセッサ）上で実行することが望まれることは珍しいことではない。この場合、対象プログラムを目的プロセッサ上で実行できるように、エミュレータまたはトランスレータを用いて、プログラム・コードのトランスレーションを行なう。エミュレータを用いて対象プロセッサをエミュレートすることによって、対象プログラムを対象プロセッサ上でネイティブに実行しているかのような仮想環境が実現される。
【発明の開示】
【発明が解決しようとする課題】
【０００４】
従来では、いわゆるベース・ノードを用いた実行時トランスレーションの間に、対象コードをコンピュータ・プログラムの中間表現に変換している。このことは、「プログラム・コード変換」と題された出願公開ＷＯ００／２２５２１に、その出願の図１～５に関連して記載されている。中間表現「ＩＲ（Intermediate Representation ）」は、コンピュータ業界で広く使用されている用語であり、プログラムを表現することができる抽象的なコンピュータ言語の形態を指すものである。しかし、中間表現「ＩＲ」は、何らかの特定のプロセッサに特有でもなければ、何らかの特定のプロセッサ上で直接実行されることが意図されたものでもない。中間表現を用いて前述のアクセラレーション、トランスレーション、および共通アーキテクチャ能力を促進するプログラム・コード変換方法および装置が、たとえば、前述の出願公開ＷＯ００／２２５２１において取り扱われている。
【課題を解決するための手段】
【０００５】
本発明によれば、添付の特許請求の範囲で述べるような装置および方法が提供される。

本発明の好ましい特徴は、従属請求項、および以下に述べる説明から明らかとなる。
以下、本発明によるプログラム・コード変換用の改善されたアーキテクチャの種々の実施形態による実現可能な種々の態様および優位性の概要を示す。この概要は、当業者が、後に続く本発明の詳細な説明を迅速に理解するための助けとなる導入部として設けられている。この概要は、添付の特許請求の範囲を限定するものでもなければ、限定することを意図したものでもない。

【０００６】

後述する種々の実施形態は、プログラム・コード変換装置用の改善されたアーキテクチャに関するとともに、対象コンピューティング環境において実行可能な対象コードを、目的コンピューティング環境において実行可能な目的コードに変換するための関連方法に関する。プログラム・コード変換装置によって、対象コードの中間表現（「ＩＲ」）が形成される。次に、中間表現（「ＩＲ」）を、目的コンピューティング環境に対して最適化して、目的コードをより効率的に生成できるようにする。変換に関与する対象および目的コンピューティング環境の特定のアーキテクチャに依存して、一実施形態のプログラム・コード変換装置は、中間表現において、以下のタイプのＩＲノードのうちのどれを生成するかを、決定する。ベース・ノード、複合ノード、ポリモーフィック・ノード、およびアーキテクチャ特定ノードである。初期設定では、プログラム・コード変換アーキテクチャは、中間表現を形成するときにベース・ノードを生成する。これは、別のタイプのノードの方が、実行されている特定の変換に対してより適切であると判断される場合を除いて、行なわれる。

【０００７】

ベース・ノードは、対象コードを実行する任意の対象アーキテクチャのセマンティクスを表わすのに必要最小限のノードの組（すなわち抽象式）を与えることにより、ＲＩＳＣと同様の機能を提供する。複合ノードは、対象コードを実行する対象アーキテクチャのＣＩＳＣと同様のセマンティクスを、ベース・ノードよりもコンパクトな表現で表わす汎用ノードである。すべての複合ノードは、同じセマンティクスを有するベース・ノード表現に分解することができるが、複合ノードは、トランスレータの性能を向上させるために、複合命令のセマンティクスを単一のＩＲノード内に保持する。複合ノードによって、基本的に、対象コード内のＣＩＳＣと同様の命令に対するベース・ノードの組が増える。ベース・ノードおよび複合ノードは両方とも、可能な対象および目的アーキテクチャの広い範囲に渡って汎用的に用いられる。そのため、ベース・ノードおよび複合ノードから構成される対応するＩＲツリー上で、汎用的な最適化を行なうことができる。

【０００８】

プログラム・コード変換装置が、中間表現においてポリモーフィック・ノードを用いるのは、目的コンピューティング環境の特性が原因で、特定の対象命令のセマンティクスが、汎用的なＩＲノードとして実現されたときに失われる場合である。ポリモーフィック・ノードには、ソース・コード内の特定の対象命令に特有の目的コンピューティング環境の関数に対する関数ポインタが含まれる。プログラム・コード変換装置がさらに、アーキテクチャ特定ノードを用いるのは、特定の目的コンピューティング環境に対する特化されたコード生成関数を実行するために、目的に特化された変換コンポーネントを与えるためである。

【０００９】

以下で説明する改善されたＩＲ生成方法によって、プログラム・コード変換装置を、最適なレベルの性能を維持しトランスレーションの速度を最大にする一方で、任意の対象および目的プロセッサ・アーキテクチャの組み合わせに対して構成することができる。

【発明を実施するための最良の形態】

【００１０】

本発明をより良好に理解するために、かつ本発明の実施形態をどのように実行に移すかを示すために、次に、添付図面を一例として参照する。
以下の説明は、いかなる当業者であっても本発明を製造し、かつ用いることができるよ

うに設けられており、また発明を行なう発明者が意図する最良のモードが述べられている。しかし、種々の変更が、当業者には容易に分かるであろう。何故なら、プログラム・コード変換装置に対する改善されたアーキテクチャを提供するために、本明細書では本発明の一般的原理を具体的に規定しているからである。

【００１１】

図１を参照すると、コンピューティング環境の例として、対象コンピューティング環境１および目的コンピューティング環境２を含むものが示されている。対象環境１では、対象コード１０は、対象プロセッサ１２上でネイティブに実行可能である。対象プロセッサ１２には、対象レジスタ１４の組が含まれている。ここで、対象コード１０は、対象コード１０と対象プロセッサ１２との間に中間層（たとえば、コンパイラ）を有するような任意の好適な言語で表わされても良い。これは、当業者には良く知られていることである。

【００１２】

対象コード１０を、目的コンピューティング環境２において実行することが望ましい。目的コンピューティング環境２には、目的レジスタ２４の組を用いる目的プロセッサ２２が設けられている。これら２つのプロセッサ１２，２２は、本質的に非互換性であり、異なる命令セットを用いても良い。したがって、目的コンピューティング環境２には、その非互換性の環境において対象コード１０を実行できるように、プログラム・コード変換アーキテクチャ３０が設けられている。プログラム・コード変換アーキテクチャ３０には、トランスレータ、エミュレータ、アクセラレータ、または一つのプロセッサ・タイプ用に構成されたプログラム・コードを他のタイプのプロセッサ上で実行可能なプログラム・コードに変換するのに好適な任意の他のアーキテクチャが含まれていても良い。以下、本発明を説明する上で、プログラム・コード変換アーキテクチャ３０を「トランスレータ３０」と呼ぶ。２つのプロセッサ１２，２２は、たとえばアクセラレータの場合には、同じアーキテクチャ・タイプになり得るに注意されたい。

【００１３】

トランスレータ３０は、対象コード１０に対してトランスレーション・プロセスを行なって、トランスレート済み目的コード２０を与え、目的プロセッサ２２が実行できるようにする。好適には、トランスレータ３０はバイナリ・トランスレーションを行なう。バイナリ・トランスレーションでは、対象プロセッサ１２に適した実行可能なバイナリ・コードの形態の対象コード１０が、目的プロセッサ２２に適した実行可能なバイナリ・コードにトランスレートされる。トランスレーションは、静的に行なわれても良いし、動的に行なわれても良い。静的なトランスレーションの場合には、プログラム全体をトランスレートすることが、トランスレート済みプログラムを目的プロセッサ上で実行する前に行なわれる。これには、著しい遅延が伴う。したがって、トランスレータ３０は好ましくは、対象コード１０の小さいセクションを動的にトランスレートして、目的プロセッサ２２上で即座に実行できるようにする。これは、はるかに効率的である。何故なら、対象コード１０の大きなセクションは、実際には用いられないか、あるいはめったに用いられないからである。

【００１４】

次に、図２を参照すると、トランスレータ３０の好ましい実施形態が詳細に示されている。トランスレータ３０には、フロントエンド３１、カーネル３２、およびバックエンド３３が含まれている。フロントエンド３１は、対象コードに関連する対象プロセッサ１２に特有に構成される。フロントエンド３１は、対象コード１０の所定のセクションを選択して、汎用的な中間表現のブロック（「ＩＲブロック」）を与える。カーネル３２は、フロントエンド３１が生成する各ＩＲブロックを、最適化技法を用いることによって最適化する。これは、当業者であれば容易に分かることである。バックエンド３３は、最適化されたＩＲブロックをカーネル３２から取得して、目的プロセッサ２２が実行可能な目的コード２０を生成する。

【００１５】

好適には、フロントエンド３１は、対象コード１０を基本ブロックに分割する。各基本

ブロックは、一意的なエントリ・ポイントにおける第１の命令と一意的なエグジット・ポイントにおける最後の命令との間の、連続的な命令（たとえばジャンプ、コール、またはブランチ命令）の組である。カーネル３２は、単一ユニットとして共に処理されるべき２つ以上の基本ブロックを含むグループ・ブロックを選択しても良い。さらにフロントエンド３１は、異なるエントリ条件の下での対象コードの同じ基本ブロックを表わす等ブロック（iso-block ）を形成しても良い。使用時には、対象コード１０の第１の所定のセクションが、たとえば基本ブロックとして識別され、トランスレーション・モードにおいて、目的プロセッサ２２上で実行されるトランスレータ３０によって、トランスレートされる。次いで、目的コード２０の最適化され、かつトランスレート済みの対応するブロックが、目的プロセッサ２２によって実行される。

【００１６】

トランスレータ３０には、複数の抽象レジスタ３４が含まれており、カーネル３２において適切に設けられている。抽象レジスタ３４は、対象コード１０を実行するための対象プロセッサ１２内で使用される物理的な対象レジスタ１４に相当する。抽象レジスタ３４によって、エミュレートされている対象プロセッサ１２の状態が規定される。その規定は、対象プロセッサ・レジスタ上での対象コード命令の予想される効果を表わすことによって行なわれる。

【００１７】

図３に、以上のような実装を採用した構造を示す。図示したように、コンパイルされたネイティブの対象コードが、適切なコンピュータ・メモリ記憶媒体１００内に存在することが示されている。特定のかつ代替的なメモリ記憶機構は、当業者には良く知られている。ソフトウェア・コンポーネントには、トランスレートされるべきネイティブの対象コード、トランスレータ・コード、トランスレート済みコード、およびオペレーティング・システムが含まれる。トランスレータ・コード、すなわちトランスレータを実施するコンパイルされたバージョンのソース・コードは、適切なコンピュータ・メモリ記憶媒体１０２上に、同様に存在している。トランスレータは、メモリに記憶されたオペレーティング・システム１０４、たとえ目的プロセッサ１０６（通常はマイクロプロセッサまたは他の好適なコンピュータ）上で実行されるＵＮＩＸ（登録商標）とともに、実行される。図３に示した構造は典型例に過ぎず、たとえば本発明による方法およびプロセスを、オペレーティング・システムとともに存在するか、あるいはその下に存在するコードにおいて実施しても良いことは理解されよう。トランスレート済みコードは、適切なコンピュータ・メモリ記憶媒体１０８内に存在することが示されている。対象コード、トランスレータ・コード、オペレーティング・システム、トランスレート済みコード、および記憶機構は、多種多様のタイプのいかなるものであっても良い。これは当業者には知られている。

【００１８】

本発明の好ましい実施形態においては、実行時にプログラム・コード変換が動的に行なわれる間に、トランスレート済みプログラムが、目的コンピューティング環境において実行されている。トランスレータ３０は、トランスレート済みプログラムに従って実行される。トランスレート済みプログラムの実行パスは、以下のステップを含む制御ループである。トランスレータ・コードを実行して、対象コードのブロックをトランスレート済みコードにトランスレートするステップと、その後に、そのトランスレート済みコードのブロックを実行するステップである。トランスレート済みコードの各ブロックの最後には、トランスレータ・コードに制御を戻す命令が含まれている。言い換えれば、トランスレートするステップと、その後に対象コードを実行するステップとが組み合わせられて、対象プログラムの一部のみが一度にトランスレートされるようになっている。

【００１９】

トランスレータ３０のトランスレーションの基本単位は、基本ブロックである。すなわち、トランスレータ３０が行なう対象コードのトランスレーションは、一度に１つの基本ブロックである。基本ブロックは正式には、厳密に１つのエントリ・ポイントと厳密に１つのエグジット・ポイントとを有するコードのセクションとして定義される。これによっ

て、そのブロック・コードが単一の制御パスに制限される。この理由により、基本ブロックは、制御フローの基本単位である。

【００２０】

中間表現（ＩＲ）ツリー

トランスレート済みコードを生成するプロセスでは、対象命令シーケンスに基づいて、中間表現（「ＩＲ」）ツリーが生成される。ＩＲツリーには、対象プログラムによって計算された式と対象プログラムによって実行された処理との抽象表現であるノードが含まれる。次に、トランスレート済みコードが、ＩＲツリーに基づいて生成される。本明細書で説明するＩＲノードを集合は、俗称的には「ツリー」と呼ばれる。なお正式には、このような構造は実際には、有向非巡回グラフ（ＤＡＧ：Directed Acyclic Graph）であり、ツリーではないことに注意されたい。ツリーの正式な定義では、各ノードのペアレントは高々１つである必要がある。説明する実施形態では、ＩＲが生成される間に共通部分式の除去が用いられるため、ノードは複数のペアレントを有することが多い。たとえば、フラグに影響を与える命令結果のＩＲは、２つの抽象レジスタ（デスティネーション・対象レジスタおよびフラグ結果パラメータに対応するもの）によって参照されても良い。

【００２１】

たとえば、対象命令（ａｄｄ％ｒ１，％ｒ２，％ｒ３）によって、対象レジスタ％ｒ２および％ｒ３の内容の加算が行なわれて、結果が対象レジスタ％ｒ１内に記憶される。すなわち、この命令は、抽象式「％ｒ１＝％ｒ２＋％３」に対応する。この例には、抽象レジスタ％ｒ１の定義が含まれる。その定義には、命令オペランド％ｒ１および％ｒ２を表わす２つの部分式が含まれる加算式が含まれている。対象プログラムに関連して言えば、これらの部分式は、それより前の他の対象命令に対応していても良いし、あるいはそれらの部分式は、現在の命令の詳細、たとえば直接定数値を表していても良い。

【００２２】

「加算」命令が構文解析されると、加算に対する抽象的な数学的オペレータに対応して、新たな「Ｙ」ＩＲノードが生成される。「Ｙ」ＩＲノードには、オペランド（対象レジスタに保持され、部分式ツリーとして表わされる）を表わす他のＩＲノードに対する参照が記憶される。「Ｙ」ノード自体は、適切な対象レジスタ定義（％ｒ１に対する抽象レジスタ、命令のデスティネーション・レジスタ）によって参照される。当業者であれば理解できるように、一実施形態においては、トランスレータは、オブジェクト指向のプログラミング言語、たとえばＣ＋＋を用いて具体化される。たとえば、ＩＲノードは、Ｃ＋＋オブジェクトとして具体化され、他のノードに対する参照は、これら他のノードに対応するＣ＋＋オブジェクトに対するＣ＋＋参照として具体化される。したがって、ＩＲツリーは、互いに様々に参照し合うＩＲノード・オブジェクトの集合として具体化される。

【００２３】

抽象レジスタ

さらに、説明している実施形態においては、ＩＲ生成は、抽象レジスタ３４の組を用いる。これらの抽象レジスタ３４は、対象アーキテクチャの特定の特性に対応する。たとえば、対象アーキテクチャ１２上の各物理レジスタ１４に対して、一意的な抽象レジスタ３４が存在する。抽象レジスタ３４は、ＩＲ生成の間にＩＲツリーに対するプレースホルダとして機能する。たとえば、対象命令シーケンスにおける所定の点における対象レジスタ％ｒ２の値は、対象レジスタ％ｒ２に対する抽象レジスタ３４に関連する特定のＩＲ式ツリーによって表わされる。一実施形態においては、抽象レジスタ３４は、Ｃ＋＋オブジェクトとして具体化される。Ｃ＋＋オブジェクトは、特定のＩＲツリーに、そのツリーのルート・ノード・オブジェクトに対するＣ＋＋参照を介して関連する。

【００２４】

前述した命令シーケンスの例では、「加算」命令に先行する対象命令を構文解析する間に、トランスレータ３０によって、％ｒ２および％ｒ３の値に対応するＩＲツリーがすでに生成されている。言い換えれば、％ｒ２および％ｒ３の値を計算する部分式は、ＩＲツリーとしてすでに表わされている。「ａｄｄ％ｒ１、％ｒ２、％ｒ３」命令に対するＩＲ

ツリーを生成するときには、新たな「Ｙ」ノードには、％ｒ２および％ｒ３に対するＩＲサブツリーへの参照が含まれる。

【００２５】

抽象レジスタ３４の実施は、トランスレータ３０およびトランスレート済みコードの両方におけるコンポーネント間に分割される。トランスレータに関連して言えば、抽象レジスタは、ＩＲ生成の過程で用いられるプレースホルダである。これは、抽象レジスタ３４が、特定の抽象レジスタ３４が対応する対象レジスタ１４の値を計算するＩＲツリーに関連するように、なされている。したがって、トランスレータにおける抽象レジスタ３４は、ＩＲノード・オブジェクトへの参照（すなわちＩＲツリー）を含むＣ＋＋オブジェクトとして具体化されても良い。トランスレート済みコードに関連して言えば、抽象レジスタ３４は、抽象レジスタ記憶装置内の特定の箇所であり、その箇所まで、かつその箇所から、対象レジスタ１４の値は実際の目的レジスタ２４と同期する。あるいは、抽象レジスタ記憶装置からロードされた値が存在する場合には、トランスレート済みコードにおける抽象レジスタ３４は、トランスレート済みコードがレジスタ記憶装置に戻されて保存される前に実行されている間に、対象レジスタの値を一時的に保持する目的レジスタ２６であると理解することができる。

【００２６】

図４に、説明したプログラム・トランスレーションの例を示す。図４に示すのは、ｘ８６命令の２つの基本ブロックのトランスレーション、およびそのトランスレーションのプロセスにおいて生成される対応するＩＲツリーである。図４の左側に示すのは、トランスレーションを行なう間のエミュレータの実行パスである。トランスレータ３０は、対象コードの第１の基本ブロック１５３を目的コードにトランスレートして（１５１）、次に、その目的コードを実行する（１５５）。目的コードの実行が終了すると、エミュレータに制御が戻される（１５７）。次に、トランスレータ３０は、対象コードの次の基本ブロック１５９を目的コードにトランスレートして（１５７）、その目的コードを実行する（１６１）、等々。

【００２７】

対象コードの第１の基本ブロック１５３を目的コードにトランスレートする過程１５１で、トランスレータ３０は、その基本ブロックに基づいてＩＲツリー１６３を生成する。この場合に、ＩＲツリー１６３は、フラグに影響を与える命令であるソース命令「ａｄｄ％ｅｅｘ，％ｅｄｘ」から生成される。ＩＲツリー１６３を生成する過程で、この命令によって４つの抽象レジスタが定義される。すなわち、デスティネーション・レジスタ％ｅｃｘ１６７、フラグに影響を与える第１の命令パラメータ１６９、フラグに影響を与える第２の命令パラメータ１７１、およびフラグに影響を与える命令の結果１７３である。「加算」命令に対応するＩＲツリーは、単純な「Ｙ」（算術加算）オペレータ１７５であり、そのオペランドは、対象レジスタ％ｅｃｘ１７７，％ｅｄｘ１７９である。

【００２８】

第１の基本ブロックをエミュレートすると、フラグに影響を与える命令のパラメータおよび結果を記憶することによって、フラグが待ち状態に置かれる。フラグに影響を与える命令は、「ａｄｄ％ｅｃｘ，％ｅｄｘ」である。命令のパラメータは、エミュレートされた対象レジスタ％ｅｃｘ１７７，％ｅｄｘ１７９の現在の値である。使用される対象レジスタ１７７，１７９の前にある「@ 」記号は、対象レジスタの値が、グローバル・レジスタ記憶装置の％ｅｃｘ，％ｅｄｘに対応する箇所から、それぞれ取り出されることを示す。この理由は、これらの特定の対象レジスタは、現在の基本ブロックによって事前にロードされてはいなかったからである。これらのパラメータの値は、第１のフラグ・パラメータ抽象レジスタ１６９と第２のフラグ・パラメータ抽象レジスタ１７１とに記憶される。加算操作１７５の結果は、フラグ結果抽象レジスタ１７３に記憶される。

【００２９】

ＩＲツリーが生成された後、対応する目的コードが、そのＩＲに基づいて生成される。汎用的なＩＲから目的コードを生成するプロセスは、当該技術分野において良く理解され

ている。トランスレート済みブロックの最後に目的コードが挿入されている。これは、抽象レジスタ（フラグ結果１７３、およびフラグ・パラメータ１６９，１７１に対するものを含む）を、グローバル・レジスタ記憶装置に保存するためである。目的コードが生成された後、トランスレート済みブロックは実行される（１５５）。

【００３０】

対象コードの第２の基本ブロック１５９をトランスレートする過程１５７で、トランスレータ３０は、その基本ブロックに基づいてＩＲツリー１６５を生成する。ＩＲツリー１６５は、ソース命令「ｐｕｓｈｆ」から生成される。ソース命令「ｐｕｓｈｆ」は、フラグを用いた命令である。「ｐｕｓｈｆ」命令のセマンティクスは、すべての状態フラグの値をスタック上に記憶することである。こうするためには、各フラグを明確に計算する必要がある。そのため４つの状態フラグの値に対応した抽象レジスタが、ＩＲを生成する間に定義される。すなわち、ゼロ・フラグ（「ＺＦ：Zero Flag 」）１８１、符号フラグ（「Ｓ１７：Sign Flag 」）１８３、キャリー・フラグ（「ＣＦ：Carry Flag」）１８５、およびオーバーフロー・フラグ（「ＯＦ：Overflow flag 」）１８７である。ノード１９５は、算術比較オペレータ「無符号未満」である。状態フラグの計算は、フラグに影響を与える事前の命令からの情報に基づいている。この場合、この命令は、第１の基本ブロック１５３からの「ａｄｄ％ｅｃｘ，％ｅｄｘ」命令である。状態フラグの値を計算するＩＲ１６５は、フラグに影響を与える命令の結果１８９およびパラメータ１９１，１９３に基づいている。前述したように、フラグ・パラメータ・ラベルの前にある「@ 」記号は、これらの値を、使用前にグローバル・レジスタ記憶装置からロードするために、目的コードをエミュレータが挿入することを示す。

【００３１】

したがって、第２の基本ブロックによって、フラグの値は強制的に規格化される。フラグの値は、（「ｐｕｓｈｆ」命令をエミュレートする目的コードにより）計算および使用された後、グローバル・レジスタ記憶装置に記憶される。同時に、待ち状態のフラグ抽象レジスタ（パラメータおよび結果）は、フラグの値が明確に記憶されている（すなわち、フラグが規格化されている）という事実を反映するために、未定義の状態におかれる。

【００３２】

図５に、トランスレーションにおいて使用され得る複数の異なるタイプのＩＲノードを生成できるような、本発明の好ましい実施形態により形成されるトランスレータ３０を示す。また、これらの異なるタイプのＩＲノードの実施が、トランスレータ３０のフロントエンド３１、カーネル３２、およびバックエンド３３コンポーネントの間でどのように分散されるのかを示す。用語「実現する」は、対象コードの対象命令１０がデコードされる（すなわち、構文解析される）ときに、フロントエンド３１において行なわれるＩＲ生成を意味する。用語「プラントする」は、バックエンド３３において行なわれる目的コード生成を意味する。

【００３３】

なお、以下ではトランスレーション・プロセスを単一の対象命令に関して述べるが、これらの処理は実際には、前述したように、対象命令の基本ブロック全体に対して一度に行なわれる。言い換えれば、基本ブロック全体が最初にデコードされてＩＲフォレストが生成され、次にカーネル３２によって、ＩＲフォレスト全体に最適化が適用される。最後に、最適化されたＩＲフォレストの１ノードに対する目的コード生成が、バックエンド３３によって、一度に行なわれる。

【００３４】

基本ブロックに対してＩＲフォレストを生成するとき、トランスレータ３０は、所望のトランスレータ性能と、ソース・プロセッサおよび目的プロセッサの組み合わせの特定のアーキテクチャとに依存して、ベース・ノード、複合ノード、ポリモーフィック・ノード、もしくはアーキテクチャ特定ノード（ＡＳＮ：Architecture Specific Node）、またはこれらの任意の組み合わせを生成しても良い。

【００３５】

ベース・ノード

ベース・ノードは、任意の対象アーキテクチャのセマンティクス（すなわち、式、計算、および処理）の抽象表現であり、対象アーキテクチャのセマンティクスを表わすのに必要な標準的または基本的なノードの最小の組を与えるものである。したがって、ベース・ノードによって、単純な縮小命令セット・コンピュータ（ＲＩＳＣ：Reduced Instruction Set Computer）と同様の機能、たとえば「加算」処理が得られる。他のタイプのノードとは異なり、各ベース・ノードは縮小することができない。すなわち、これ以上分解して他のＩＲノードにすることはできない。またベース・ノードは、単純であるために、すべてのバックエンド３３上で、トランスレータ３０によって容易に目的命令にトランスレートされる（すなわち、目的アーキテクチャ）。

【００３６】

ベースＩＲノードのみを利用するときには、トランスレーション・プロセスは全体に、図５の上部（すなわち、「ベースＩＲ」ブロック２０４を通過するパス）において行なわれる。フロントエンド３１は、対象プログラム・コード１０からの対象命令を、デコード・ブロック２００においてデコードし、ベース・ノードからなる対応するＩＲツリーを、実現ブロック２０２において実現する（生成する）。次に、ＩＲツリーは、フロントエンド３１から、カーネル３２におけるベースＩＲブロック２０４へ送られて、そこでＩＲフォレスト全体に最適化が適用される。ベースＩＲブロック２０４によって最適化されたＩＲフォレストは、ベース・ノードのみから構成されるため、任意のプロセッサ・アーキテクチャに対して完全に汎用的である。次に、最適化されたＩＲフォレストは、カーネル３２におけるベースＩＲブロック２０４からバックエンド３３へ送られる。そこでは、プラント・ブロック２０６において、各ＩＲノードに対する対応する目的コード命令がプラント（生成）される。次に、目的コード命令は、エンコード・ブロック２０８によってエンコードされて、目的プロセッサが実行できるようにされる。

【００３７】

前述したように、ベース・ノードはすべてのバックエンド３３上において、目的命令に容易にトランスレートされる。またトランスレート済みコードは通常、ベース・ノードのみを利用することによって完全に生成される。ベース・ノードのみを用いることにより、トランスレータ３０の実行は非常に迅速になるが、トランスレート済みコードにおける性能は最適なものとはならない。トランスレート済みコードの性能を高めるために、トランスレータ３０を特化して、目的プロセッサ・アーキテクチャの特性を利用できるようにすることができる。特性の利用は、代替タイプのＩＲノード、たとえば複合ノード、ポリモーフィック・ノード、およびアーキテクチャ特定ノード（ＡＳＮ）を用いることによってなされる。

【００３８】

複合ノード

複合ノードは、対象アーキテクチャのセマンティクスを、ベース・ノードよりもコンパクトな表現で表わす汎用ノードである。複合ノードによって、「複合命令セット・コンピュータ（ＣＩＳＣ：Complex Instruction Set Computer）と同様の機能、たとえば「ａｄｄ＿ｉｍｍ」（レジスタおよび直接定数を加算する）が得られる。具体的には、複合ノードは通常、直接定数フィールドを有する命令を表わす。直接タイプの命令は、定数オペランドの値が、「直接」フィールドにおける命令自体にエンコードされる命令である。定数値が、直接フィールドに組み込まれるほどに十分に小さい場合には、このような命令によって、定数を保持するためにレジスタを１つ用いることが回避される。複合命令の場合には、複合ノードは、同じセマンティクスを特徴付ける同等なベース・ノード表現よりもはるかに少ないノードを用いて、複合命令のセマンティクスを表わすことができる。複合ノードは基本的に、同じセマンティクスを有するベース・ノード表現に分解することができるが、複合ノードは、直接タイプの命令のセマンティクスを単一のＩＲノードに保持するのに有用であるため、トランスレータ３０の性能を向上させることができる。さらに、状況によっては、複合命令のセマンティクスは、複合命令をベース・ノードにより表わすこ

とによって失われる。そのため複合ノードは基本的に、このような「ＣＩＳＣと同様の」命令に対するＩＲノードを含むように、ベース・ノードの組を増やす。

【００３９】

次に、図６を参照して、複合ノードを用いることによって達成される効率の例を、ベース・ノードの場合の効率と比較して説明する。たとえば、ＭＩＰＳ直接加算命令「ａｄｄｉｒ１，＃１０」のセマンティクスは、レジスタｒ１に保持される値に、１０を加える。定数値（１０）をレジスタにロードした後に２つのレジスタを加算するのではなくて、ａｄｄｉ命令では、定数値１０を直接、命令フィールド自体に単純にエンコードするため、第２のレジスタを用いる必要がない。厳密にベース・ノードを用いてこれらのセマンティクスの中間表現の生成を行なってみると、この命令に対するベース・ノード表現では、最初に、ｃｏｎｓｔ（＃１０）ノード６０からレジスタ・ノードｒ（ｘ）６１へ定数値１０をロードし、次に、ａｄｄノード６３を用いて、レジスタ・ノードｒ１　６２とレジスタ・ノードｒ（ｘ）６１との加算を行なう。複合ノード表現は、単一の「直接に加算」ＩＲノード７０からなる。ＩＲノード７０は、ノード７０の部分７２に定数値１０を含み、またレジスタｒ１　７４への参照を含む。ベース・ノードの場合では、バックエンド３３は、「直接に加算」目的命令を認識し、かつ生成するために、図６に示す４つのノード・パターンを認識することができるイディオム認識を行なう必要がある。イディオム認識がないときには、バックエンド３３は、レジスタへの定数値１０のロードをレジスター レジスタ加算の実行の前に行なうための余分な命令を出す。

【００４０】

複合ノードの場合には、バックエンド３３においてイディオム認識を行なう必要が減る。何故なら、複合ノードに含まれるセマンティック情報は、それらのベース・ノード同等物の場合よりも多いからである。具体的には、複合ノードの場合には、バックエンド３３が定数オペランドのイディオム認識を行なう必要がなくなる。比較すると、直接タイプの対象命令がベース・ノードに分解された（かつ、目的アーキテクチャに直接タイプ命令が含まれた）場合には、トランスレータ３０は、高価なバックエンド３３イディオム認識を行なって、複数のノード・クラスタを直接命令候補として識別する必要があるか、あるいは非効率的な目的コードを生成する（すなわち、必要数よりも多い目的レジスタを用いて、必要数よりも多い命令を生成する）。言い換えれば、ベース・ノードのみを利用すると、性能が、トランスレータ３０内で失われるか（イディオム認識を通して）、あるいはトランスレート済みコード内で失われる（イディオム認識を行なわずに、余分に生成されたコードを通して）。より一般的には、複合ノードは、セマンティック情報のよりコンパクトな表現であるために、複合ノードを用いると、トランスレータ３０が形成し、横断し、および削除しなければならないＩＲノードの数が減る。

【００４１】

直接タイプの命令は、多くのアーキテクチャに共通である。したがって、複合ノードは、アーキテクチャの範囲全体において再利用可能であるという点で、汎用的である。しかし、すべてのトランスレータのＩＲノードの組において、すべての複合ノードが存在するわけではない。トランスレータの特定の汎用的な特性が構成可能である。すなわち、特定のソースおよび目的アーキテクチャの組み合わせに対してトランスレータがコンパイルされているときに、そのトランスレータ構成に適用されない特性をコンパイルから除くことができる。たとえば、ＭＩＰＳ　ＭＩＰＳ（ＭＩＰＳ対ＭＩＰＳ）トランスレータでは、いずれかのＭＩＰＳ命令のセマンティクスにマッチングしない複合ノードは、ＩＲノードの組から除かれる。何故なら、このような複合ノードは、利用されることがないからである。

【００４２】

複合ノードの場合、生成される目的コードの性能を、間順走査を行なうことによりさらに向上させることができる。間順走査は、ＩＲツリー内のＩＲノードを目的コードに生成する順番を決定する複数の代替的ＩＲ走査アルゴリズムのうちの１つである。具体的には、間順走査を行なうと、各ＩＲノードが、そのノードが最初に走査されるときに生成され

る。この結果、ＩＲツリー全体に渡って別個の最適化パスが存在しないために起こるバックエンド３３イディオム認識が防がれる。複合ノードの場合、ノード当たりのセマンティック情報はベース・ノードの場合よりも多いため、イディオム認識の作業の一部は、複合ノード自体内において暗黙的である。この結果、トランスレータ３０は、ベース・ノードのみの場合と同様に、目的コード性能におけるペナルティをそれほど被ることなく、間順走査を用いることができる。

【００４３】

トランスレータ３０が複合ノード（すなわち、図５の複合ＩＲブロック２１０を通るパス）を生成する場合、トランスレーション・プロセスは、ベース・ノードに対して前述したトランスレーション・プロセスと同様である。唯一の相違点は、複合ノードのセマンティクスにマッチングする対象命令は、実現ブロック２０２内の複合ノードとして実現され、ベース・ノードとして実現されるわけではないことである（実現ブロック２０２を分離する破線によって示す）。複合ノードの場合でもやはり、広範囲のアーキテクチャに渡って汎用であり、カーネル３２の最適化をやはり、ＩＲフォレスト全体に適用することができる。さらに、ＣＩＳＣタイプの目的アーキテクチャ上での複合ノードに対する目的コード生成は、ベース・ノード同等物の場合よりも効率的であり得る。

【００４４】

ポリモーフィック・ノード

図５に示すトランスレータ３０の好ましい実施形態ではさらに、ポリモーフィック中間表現を利用しても良い。ポリモーフィック中間表現は、バックエンド３３が、特定の性能重要な対象命令に対して目的アーキテクチャの特性を効率的に利用するために特化されたコード生成を与えることができる機構である。ポリモーフィック機構は、バックエンド３３のコード生成関数に対する関数ポインタを含む汎用的なポリモーフィック・ノードとして具体化される。各関数ポインタは、特定の対象命令に特化される。このポリモーフィック機構は、標準的なフロントエンド３１のＩＲ生成機構の代わりに行なわれる。フロントエンド３１のＩＲ生成機構は、さもなければ、対象命令をベース・ノードまたは複合ノードにデコードする。ポリモーフィック機構がない場合には、これらのベース・ノードの生成によって、バックエンド３３において、最適ではない目的コードが生成されるか、あるいは高価なイディオム認識を行なって対象命令のセマンティクスを再構成する必要が生じる。

【００４５】

各ポリモーフィック関数は、特定の対象命令および目的アーキテクチャ関数の組み合わせに特有である。それらの関数についての最小限の情報が、ポリモーフィック・ノードによってカーネル３２に公開される。ポリモーフィック・ノードは、通常のカーネル３２の最適化、たとえば式シェアリングおよび式フォールディングに関与することができる。カーネル３２は、関数ポインタを用いて、２つのポリモーフィック・ノードが同じか否かを判断することができる。ポリモーフィック・ノードには、対象命令のいかなるセマンティック情報も保持しないが、このようなセマンティック情報は、関数ポインタから推測することができる。

【００４６】

ポリモーフィック・ノードは、対象命令に対して使用され、慎重に選択された一連の目的命令によって表現することができるため、最良の目的命令は実行時であることをカーネル３２が判断する必要がない。ポリモーフィック・ノードの実現が、ベース・ノードを用いるフロントエンド３１によってなされていない場合、カーネル３２は、これらのノードをポリモーフィック・ノードとして実現することを選択しても良い。

【００４７】

さらに、ポリモーフィック・ノードは、レジスタ割り当てヒントを含むことができる。目的命令が既知であるため、ＣＩＳＣアーキテクチャ上で必要とされ得る個々のレジスタも既知である。ポリモーフィック・ノードを用いた場合、それらのオペランドおよび結果は、ＩＲ構成時に選択されるレジスタ内に現れることができる。

【００４８】
トランスレータ３０がポリモーフィック・ノード（すなわち、図５のポリモーフィックＩＲブロック２１２を通るパス）を利用することになるように、バックエンド３３は、対象命令－目的関数ポインタの組み合わせのリストを、フロントエンド３１に与える。与えられたリスト上にある対象命令は、対応するバックエンド３３の関数ポインタを含むポリモーフィック・ノードとして実現される。リスト上にない対象命令は、前述したように、複合またはベースＩＲツリーとして実現される。図５において、バックエンド３３からフロントエンド３１へ向かう矢印２１４によって反映されるパスは、フロントエンド３１でブロック２１５を実現するために与えられる対象命令－目的関数ポインタの組み合わせのリストを示す。フロントエンド３１は、実現ブロック２１５において実現（すなわち、ＩＲノードへの対象命令のマッピング）を行なうが、このプロセスは、パス２１４を通してバックエンド３３から受け取る情報によって変更される。

【００４９】
カーネル３２のポリモーフィックＩＲブロック２１２においては、ポリモーフィック・ノードの場合でもやはり、汎用的な最適化に関与することができる。何故なら、カーネル３２は、ポリモーフィック・ノードのセマンティクスを、各ノード内の関数ポインタから推測することができるからである。バックエンド３３では、目的コード生成関数を指し示す目的関数ポインタが、単純に逆参照されて実行される。この状況は、バックエンド３３が特定のＩＲノードを特定のコード生成関数に対してマッピングするベース・ノードおよび複合ノードの場合とは異なる。ポリモーフィック・ノードを用いた場合、ポリモーフィック関数は、ノード自体において直接エンコードされるため、バックエンド３３が行なう計算は少なくなる。図５において、この相違点は、ポリモーフィック・プラント・ブロック２１６が、ポリモーフィックＩＲブロック２１２およびバックエンド３３の両方と隣接する（すなわち、ポリモーフィックＩＲブロック２１２とポリモーフィック・プラント・ブロック２１６との間に、重要な計算を表す矢印が示されていない）という事実によって表されている。

【００５０】
例１：ポリモーフィックＩＲの例
ＩＲにおいてポリモーフィック・ノードを利用するためにトランスレータ３０を最適化するプロセスを示すことを目的として、以下の例では、ＰＰＣ（PowerPC （登録商標））「ＳＨＬ６４」命令（左シフト、６４ビット）のトランスレーションを説明する。これは、最初にベース・ノードを使用して、次にポリモーフィック・ノードを使用するＰＰＣＰ４（PowerPC to Pentium4 （登録商標））トランスレータにおいて必要とされるものである。

【００５１】
ポリモーフィック・ノードの実施に対してトランスレータを最適化することなく、ＰＰＣ　ＳＨＬ６４命令のトランスレーションにおいてベース・ノードのみを用いる。
ＰＰＣ　ＳＨＬ６４＝＞ベースＩＲの複数ノード＝＞Ｐ４の複数命令
最適化されていないトランスレータのフロントエンド・デコーダ２００は、現在のブロックをデコードして、ＰＰＣ　ＳＨＬ６４命令を生成する。次に、フロントエンド実現ブロック２０２は、カーネル３２に、複数のベース・ノードからなるＩＲを構成するように命令を出す。次に、カーネル３２は、（命令の現在のブロックから生成された）ＩＲフォレストを最適化して、順序付け走査を行なってベースＩＲブロック２０４におけるコード生成の順番を決定する。次に、カーネル３２は、各ＩＲノードに対して順番にコード生成を行ない、バックエンド３３に、適切なＲＩＳＣタイプ命令をプラントするように命令を出す。最後に、バックエンド３３は、プラント・ブロック２０６においてコードをプラントして、エンコード・ブロック２０８において、１つまたは複数の目的アーキテクチャ命令を用いて各ＲＩＳＣタイプ命令をエンコードする。

【００５２】
性能重要な命令に対してフロントエンド３１およびバックエンド３３を特化することに

よって、特定の目的アーキテクチャに対して最適化する場合は、以下の通り。

ＰＰＣ　ＳＨＬ６４＞ポリＩＲの単一ノード＞Ｐ４の単一／少数の命令

最適化されたトランスレータ３０のフロントエンド・デコーダ２００は、現在のブロックをデコードして、ＰＰＣ　ＳＨＬ６４命令を生成する。次に、フロントエンド実現ブロック２０２は、カーネル３２に、単一のポリモーフィックＩＲノードからなるＩＲを構成するように命令を出す。単一のポリモーフィック・ノードが形成されたとき、バックエンド３３には、ＳＨＬ６４のシフト・オペランドが特定のレジスタ（Ｐ４上の％ｅｃｘ）内に存在しなければならないということが知らされる。この要求は、ポリモーフィック・ノードにおいてエンコードされる。次に、カーネル３２は、現在のブロックに対してＩＲフォレストを最適化するとともに、ポリモーフィックＩＲブロック２１２におけるコード生成の順番を固定するために順序付け走査を行なう。次に、カーネル３２は、各ノードに対してコード生成を行ない、バックエンド３３に、適切なＲＩＳＣタイプ命令をプラントするように命令を出す。しかし、コードを生成する間、ポリモーフィック・ノードを処理する仕方は、ベース・ノードの場合とは異なる。各ポリモーフィック・ノードによって、バックエンド３３に存在する特化されたコード・ジェネレータ関数が呼び出される。バックエンド３３の特化されたコード・ジェネレータ関数は、プラント・ブロック２１６においてコードをプラントし、エンコード・ブロック２０８において、１つまたは複数の目的アーキテクチャ命令を用いて各対象アーキテクチャ命令をエンコードする。生成段階におけるレジスタ割り当ての間に、特定のレジスタ情報を用いて、正確なレジスタが割り当てられる。この結果、不適切なレジスタが割り当てられていたならば必要とされたであろうバックエンド３３によって行なわれる計算が減る。このコード生成には、一時的なレジスタに対するレジスタ割り当てが伴っても良い。

【００５３】

例２：困難な命令

以下の例では、本発明のトランスレータ３０が行なうＰＰＣ　ＭＦＦＳ命令（３２ビットＦＰＵ制御レジスタを６４ビットの一般的なＦＰＵレジスタへ移動させる）のトランスレーションおよび最適化を示す。この対象命令は、ベース・ノードにより表わすには非常に複雑である。

【００５４】

最適化されていない場合には、この命令は、置換関数を用いてトランスレートされる。置換関数は、標準的なトランスレーション方式を用いてトランスレートすることが特に困難な対象命令の特別な場合に対する明示的なトランスレーションである。置換関数のトランスレーションは、対象命令のセマンティクスを行なう目的コード関数として具体化される。置換関数のトランスレーションは、標準的なＩＲ命令に基づくトランスレーション方式よりもはるかに高い実行コストがかかる。この命令に対する最適化されていないトランスレーション方式は、以下のようになる。

【００５５】

ＰＰＣ　ＭＦＦＳの命令＝＞ベースＩＲの置換関数＝＞Ｐ４の置換関数

ポリモーフィックＩＲを用いるトランスレータ３０においては、このような特別な場合の命令は、ポリモーフィック・ノードを用いてトランスレートされる。ポリモーフィック・ノードの関数ポインタによって、困難な対象命令のカスタムなトランスレーションをバックエンド３３が提供するためのより効率的な機構が得られる。したがって、同じ命令に対する最適化されたトランスレーション方式は、以下のようになる。

【００５６】

ＰＰＣ　ＭＦＦＳの命令＝＞単一のポリモーフィックのＩＲノード＝＞Ｐ４　ＳＳＥ２の命令

アーキテクチャ特定ノード

本発明のトランスレータ３０の他の好ましい実施形態においては、トランスレータ３０は、図５に示すように、アーキテクチャ特定ノード（ＡＳＮ：Architecture Specific Node）を用いても良い。アーキテクチャ特定ノードは、特定のアーキテクチャ（すなわち、

特定のソース・アーキテクチャおよび目的アーキテクチャの組み合わせ）に特有である。各アーキテクチャ特定ノード（ＡＳＮ）は、特定の命令に対して特定的に仕立てられるため、ＡＳＮは、特定のアーキテクチャに特有のものとなる。ＡＳＮ機構を用いている場合、アーキテクチャ特定の最適化として、ＡＳＮのセマンティクスを包含し、したがってＡＳＮ上で動作可能なものを実施することができる。

【００５７】

ＩＲノードには、最大３つのコンポーネントが含まれていても良い。すなわち、データ・コンポーネント、実施コンポーネント、および変換コンポーネントである。データ・コンポーネントは、ノード自体においては固有ではない何らかのセマンティック情報（たとえば定数直接命令フィールドの値）を保持する。実施コンポーネントは、コード生成を実行し、したがって、特定のアーキテクチャに特定的に関係する。変換コンポーネントは、ノードを、異なるタイプのＩＲノード、ＡＳＮノードまたはベース・ノードに変換する。トランスレータにおける本発明の所定の実施形態においては、生成されたＩＲにおけるベース・ノードおよびＡＳＮのそれぞれに、変換コンポーネントまたは実施コンポーネントのいずれかが含まれるが、両方ではない。

【００５８】

各ベース・ノードは、目的アーキテクチャに特有の実施コンポーネントを有する。ベース・ノードは、変換コンポーネントは有さない。何故なら、ベース・ノードがＩＲノード階層内でエンコードするセマンティック情報の量は可能な限り少ないものであるため、ベース・ノードを他のタイプのＩＲノードに変換しても、何ら利点がないからである。ベース・ノードを他のタイプのＩＲノードにする前述のような変換はいかなるものであっても、イディオム認識を通してセマンティック情報を再収集する必要がある。

【００５９】

ＡＳＮの実施コンポーネントは、ノードのアーキテクチャに、そのＡＳＫに対応するアーキテクチャ特定の命令をコンポーネントが生成するように、特有である。たとえば、ＭＩＰＳロードＡＳＮの実施コンポーネントによって、ＭＩＰＳ「１ｄ」（ロード）命令が生成される。本発明のトランスレータを、対象および目的のアーキテクチャが同じ状態で（すなわちアクセラレータとして）用いる場合、対象ＡＳＮは実施コンポーネントを有する。トランスレータを、対象および目的のアーキテクチャが異なる状態で用いる場合、対象ＡＳＮは変換コンポーネントを有する。

【００６０】

たとえば、図７に示すのは、本発明の実施形態をＭＩＰＳ－ＭＩＰＳアクセラレータで用いた場合の、ＭＩＰＳ命令に対するＡＳＮである。フロントエンド３１は、ＭＩＰＳ「ａｄｄｉ」（直接加算）命令７０１をデコードして、対応するＡＳＮ，ＭＩＰＳ＿ＡＤＤＩ７０３を含むようにＩＲを生成する。対象および目的アーキテクチャは、アクセラレータに対して同じであるため、変換コンポーネント「ＣＶＴ」７０７は定義されていない。実施コンポーネント「ＩＭＰＬ」７０５は、同じＭＩＰＳ「加算」命令７０９を生成するように定義され、コード生成パスにおいてレジスタ割り当ての差を受ける。

【００６１】

図８に示すのは、本発明の実施形態をＭＩＰＳ　Ｘ８６トランスレータで用いた場合の、同じＭＩＰＳ命令に対するＩＲ内のＡＳＮである。フロントエンド３１は、ＭＩＰＳ「ａｄｄｉ」対象命令をデコードして、対応する対象ＡＳＮ，ＭＩＰＳ　ＡＤＤＩ８０１を生成する。ソースおよび目的アーキテクチャは、このトランスレータに対して異なっているため、対象ＡＳＮ８０１の実施コンポーネント８０３は定義されていない。ＭＩＰＳ　ＡＤＤＩの変換コンポーネント８０５は、特化された変換コンポーネントであり、対象ＡＳＮ８０１を目的ＡＳＮ８０７に変換するものである。汎用的な変換コンポーネントの場合には、比較により、対象ＡＳＮ８０１をベース・ノード表現に変換する。ＭＩＰＳ　ＡＤＤＩノード８０１の目的ＡＳＮ表現は、単一のＸ８６　ＡＤＤＩノード８０７である。目的ＡＳＮ８０７の変換コンポーネント８１１は、定義されていない。目的ＡＳＮ８０７の実施コンポーネント８０９は、目的命令８１３（この場合にはＸ８６命令「ＡＤＤ＄Ｅ

ＡＸ，＃１０」）を、生成する。

【００６２】

トランスレータ３０がＡＳＮを用いている場合、すべての対象命令は、対象特定のＡＳＮとして実現される。図５において、フロントエンド・デコード・ブロック２００、ＡＳＮ実現ブロック２１８、および対象ＡＳＮブロック２２０が互いに隣接しているという事実は、ＡＳＮがフロントエンド３１によって規定されるという事実、かつ実現することは自明であるという事実を示している。何故なら、対象命令タイプと対象ＡＳＮタイプとの間には、１対１の関係が存在するからである。フロントエンド３１には、対象ＡＳＮのセマンティクスを理解し、対象ＡＳＮ上で動作可能な対象特定の最適化が含まれている。言い換えれば、対象コードは最初に、全体として対象ＡＳＮからなるＩＲフォレストとして実現され、次に、これに対して、対象特定の最適化が適用される。

【００６３】

初期設定により、対象ＡＳＮは、ベース・ノードのＩＲツリーを生成する汎用的な変換コンポーネントを有する。この結果、新しい対象アーキテクチャに対するサポートを、汎用的なＩＲノードを用いて迅速に実施することができる。対象ＡＳＮは、図５のＡＳＮベースＩＲブロック２２２およびプラント・ブロック２０６を通って延びるパスを通して、ベース・ノードとして実現される。このベース・ノードは、すでに詳細に述べた他のベース・ノードの場合と同様の仕方で、目的コードにトランスレートされる。

【００６４】

性能に対して重要である対象命令の場合には、対応する対象ＡＳＮノードによって、特化された変換コンポーネントが得られる。特化された変換コンポーネントによって、目的ＡＳＮノードのＩＲツリーが生成される。特化された変換コンポーネントを実施するか否かを検討する際の要因には、以下のものが含まれる。（１）目的アーキテクチャの特性によって、ベース・ノード・トランスレーションの場合には失われる特に効率的なトランスレーションが得られるか否か、（２）対象命令が、性能に著しい影響を与えるような頻度で行なわれるか否か。これらの特化された変換コンポーネントは、対象及び目的アーキテクチャの組み合わせに特有である。目的ＡＳＮ（定義により、目的と同じアーキテクチャである）には、実施コンポーネントが含まれる。

【００６５】

特化された変換コンポーネントを実施すると、対応する対象ＡＳＮノードによって、目的に特化された変換コンポーネントが与えられ、このコンポーネントにより、対象ＡＳＮが目的ＡＳＮに、目的ＡＳＮブロック２２４を通して変換される。次に、目的ＡＳＮの実施コンポーネントが呼び出されて、目的ＡＳＮプラント・ブロック２２６においてコード生成が行なわれる。各目的ＡＳＮは、１つの特定の目的命令に対応しており、目的ＡＳＮから生成されるコードは単純に、ＡＳＮがエンコードする対応する目的命令となるようになっている。したがって、目的ＡＳＮを用いるコード生成は、計算量が最小である（このことは図５において、目的ＡＳＮプラント・ブロック２２６を、目的ＡＳＮブロック２２４とバックエンド３３内のエンコード・ブロック２０８との両方に隣接するように図示することによって反映されるよう表わされており、これらのコンポーネント間には重要な計算を示す矢印は示されていない）。さらに、ＩＲ走査、変換、およびコード生成プロセスはすべて、カーネル３２によって制御される。

【００６６】

図９に、ＡＳＮ機構を利用する本発明のトランスレータの好ましい実施形態により行なわれるトランスレーション・プロセスを示す。フロントエンド３１では、ステップ９０３において、トランスレータが、対象コード９０１を対象ＡＳＮ９０４にデコードする。ステップ９０５において、トランスレータは、対象ＡＳＮから構成されるＩＲツリー上で、対象特定の最適化を行なう。次にステップ９０７において、対象ＡＳＮの変換コンポーネントを呼び出すことによって、各対象ＡＳＮ９０４が、目的互換性ＩＲノード（目的ＡＳＮ９１１）に変換される。初期設定により汎用的な変換コンポーネントを有する対象ＡＳＮノードは、ベース・ノード９０９に変換される。特化された変換コンポーネントを有す

る対象ＡＳＮノードは、バックエンド９２５によって与えられるときに、目的ＡＳＮ９１１に変換される。したがって、この変換により、ベース・ノード９０９および目的ＡＳＮ９１１の両方を含む混合されたＩＲフォレスト９１３が生成される。カーネル３２では、ステップ９１５において、トランスレータが、混合されたＩＲフォレスト９１３におけるベース・ノード上で、汎用的な最適化を行なう。次に、ステップ９１６において、トランスレータが、混合されたＩＲフォレスト９１３における目的ＡＳＮ上で、目的特定の最適化を行なう。最後に、ステップ９１７において、コード生成によって、混合ツリーにおける各ノードの実施コンポーネントが呼び出され（ベース・ノードおよび目的ＡＳＮノードの両方とも実施コンポーネントを有する）、目的コード９１９が生成される。

【００６７】

コード・アクセラレータを用いる特別な場合には、対象および目的アーキテクチャは、両方とも同じである。この状況では、対象ＡＳＮは、トランスレーションの間中、存続する。フロントエンド３１では、デコーディングによって、対象命令から対象ＡＳＮが生成される。カーネル３２では、対象ＡＳＮは、アーキテクチャ特定の最適化を経る。コード生成によって、対象ＡＳＮの実施コンポーネントが呼び出されて、対応する命令が生成される。したがって、コード・アクセラレータでは、ＡＳＮを用いることによってコード爆発が防止される。その防止は、最小の対象対目的命令変換比率１：１を保証することによってなされる。この比率は、最適化によって増やすことができる。

【００６８】

本発明のトランスレータの種々の実施形態を、特定のトランスレータの応用例（すなわち、特定の対象アーキテクチャおよび目的アーキテクチャの組み合わせ）に対して構成することができる。したがって、本発明のトランスレータは、任意の対象アーキテクチャ上で実行するように構成された対象コードを、任意の目的アーキテクチャ上で実行可能な目的コードに変換するように、構成可能である。各ベース・ノードは、複数のトランスレータの応用例において複数の実施コンポーネントを有する。実施コンポーネントは、サポートされる各目的アーキテクチャに対して１つである。開始されている特定の構成（すなわち、条件付きコンパイル）によって、どのＩＲノードおよびそれらのノードのうちのどのコンポーネントが、特定のトランスレータの応用例に含められるかが決定される。

【００６９】

本発明の好ましい実施形態におけるＡＳＮを用いることによって、複数の優位な利点が得られる。第１に、対象命令の汎用的なＩＲの実施形態を用いることで、スクラッチから構成されるトランスレータ製品を迅速に開発することができる。第２に、性能に対して重要である対象命令（予め分かっているかまたは経験的に決定される）に対して目的特定の変換コンポーネントを実施することで、既存のトランスレータ製品を徐々に増加させることができる。第３に、開発されるトランスレータ製品の数が増加するにつれて、ＡＳＮノード（および実施される機能）のライブラリも時間とともに増加するため、将来のトランスレータ製品の実施または最適化を迅速に行なうことができる。

【００７０】

本発明のこの実施形態では、バックエンド実施により、どの対象命令が（目的に特化された変換コンポーネントを規定することによる）最適化に値するかを、慎重に選ぶ。汎用的な変換コンポーネントによって、ＡＳＮベースのトランスレータを迅速に開発することができる一方で、特化された変換コンポーネントによって、性能重要な命令を、選択的かつ徐々に最適化することができる。

【００７１】

例３：ＡＳＮを用いた困難な命令

前述した例２のＰｏｗｅｒＰＣ　ＳＨＬ６４命令に戻って、ＡＳＮを用いるトランスレータ３０では、以下のステップが行なわれる。フロントエンド・デコーダ２００は、現在のブロックをデコードして、ＰｏｗｅｒＰＣ　ＳＨＬ６４命令を生成する。次に，フロントエンド３１が、その命令に対する単一のＡＳＮ、ＳＨＬ６４　ＰＰＣ　Ｐ４を実現する。次に、カーネル３２が、命令の現在のブロックに対するＩＲを最適化して、コード生成

に備えてＩＲの順序付け走査を行なう。次に、カーネル３２が、各特定のＡＳＮノードのコード・ジェネレータ関数を呼び出すことによって、ＡＳＮノードに対してコード生成を行なう。コード・ジェネレータ関数は、実施コンポーネントの要素である。次に、バックエンド３３が、対象アーキテクチャ（ＰＰＣ）命令を、１つまたは複数の目的アーキテクチャ（Ｐ４）命令にエンコードする。

【００７２】

ＭＩＰＳの例

次に、図１０，１１，１２を参照して、ベースＩＲノード、ＭＩＰＳ－ＭＩＰＳ　ＡＳＮ　ＩＲノード、およびＭＩＰＳ－Ｘ８６　ＡＳＮ　ＩＲノードをそれぞれ用いて、同じＭＩＰＳ命令シーケンスから生成され異なるＩＲツリーを表わす例を示す。ＭＩＰＳ対象命令シーケンスの例（上位の即値をロード、次に即値をビット単位ＯＲ）のセマンティクスは、３２ビット定数値０ｘＩ２３４５６７８を対象レジスタ「ａ１」にロードすることである。

【００７３】

図１０において、バイナリ・デコーダ３００は、対象コードを別個の対象命令にデコード（構文解析）するトランスレータ３０のフロントエンド３１のコンポーネントである。対象命令は、デコードされた後、ベース・ノード３０２として実現され、命令の現在のブロックに対する作業用ＩＲフォレストに追加される。ＩＲマネージャ３０４は、ＩＲを生成する間に作業用ＩＲフォレストを保持するトランスレータ３０の部分である。ＩＲマネージャ３０４は、抽象レジスタおよびそれらに関連するＩＲツリーからなる（ＩＲフォレストのルートは抽象レジスタである）。たとえば、図１０において、抽象レジスタ”ａＶ３０６は、５つのノードからなるＩＲツリー３０８のルートであり、現在のブロックの作業用ＩＲフォレストの一部である。Ｃ＋＋により具体化されるトランスレータ３０の場合には、ＩＲマネージャ３０４を、抽象レジスタ・オブジェクト（またはＩＲノード・オブジェクトに対する参照）の組を含むＣ＋＋オブジェクトとして具体化しても良い。

【００７４】

図１０に示すのは、ベース・ノードのみを用いたＭＩＰＳ対Ｘ８６トランスレータによって生成されるＩＲツリー３０８である。ＭＩＰＳ＿ＬＵＩ命令３１０によって、２つのオペランド・ノード３１６，３１８を有する「ＳＨＬ」（左へシフト）ベース・ノード３１４が実現される。２つのオペランド・ノード３１６，３１８は、この場合には、両方とも定数である。ＭＩＰＳ＿ＬＵＩ命令３１０のセマンティクスは、定数値（０ｘ１２３４）を、一定数のビット（１６）だけ左へシフトさせることである。ＭＩＰＳ＿ＯＲＩ命令３１２によって、２つのオペランド・ノード３１４，３２２、すなわちＳＨＬノード３１４の結果および定数値を有する「ＯＲＩ」（即値をビット単位ＯＲ）ベース・ノード３２０が実現される。ＭＩＰＳ＿ＯＲＩ命令３１２のセマンティクスは、定数値（０ｘ５６７８）を有する既存のレジスタ内容のビット単位ＯＲを行なうことである。

【００７５】

最適化されていないコード・ジェネレータの場合には、ベース・ノードには、即値ロード以外の直接タイプのオペレータは含まれていない。そのため、各定数ノードによって、即値ロード命令が生成される。したがって、最適化されていないベース・ノード・トランスレータの場合には、この対象命令シーケンスに対して５つのＲＩＳＣタイプの処理（ロード、ロード、シフト、ロード、ＯＲ）が必要となる。バックエンド３３イディオム認識によって、この数を５から２へ、定数ノードをそれらのペアレント・ノードと合体させることにより減らして、直接タイプの目的命令（すなわち、即値シフトおよび即値ＯＲ）を生成することができる。この結果、目的命令の数が減って２になるが、これは、コード・ジェネレータでイディオム認識を行なう際のトランスレーション・コストが増加する場合である。

【００７６】

ＩＲにおいて複合ノードを用いることで、直接タイプＩＲノードを実現することができる。その結果、バックエンド３３でイディオム認識を行なう必要がなくなり、コード・ジ

ェネレータのトランスレーション・コストが減る。本来の対象命令よりも、複合ノードが保持するセマンティクスは多く、実現するＩＲノードは少ない。ノード生成のトランスレーション・コストも、複合ノードを用いると下がる。

【００７７】

図１１に、ＡＳＮを用いたＭＩＰＳ　Ｘ８６（ＭＩＰＳ対Ｘ８６）トランスレータによって生成されるＩＲツリーを示す。対象命令は、バイナリ・デコーダ３００によってデコードされた後、ＭＩＰＳ＿Ｘ８６　ＡＳＮノード３３０として実現される。次に、ＭＩＰＳ＿Ｘ８６　ＡＳＮノード３３０は、現在のブロックに対する作業用ＩＲフォレストに追加される。第１に、ＭＩＰＳ＿Ｘ８６＿ＬＵＩ　ＡＳＮノードは、ＡＳＮの変換コンポーネントによって、Ｘ８６　３２－ビット定数ノード３３２に変換される。第２に、ＭＩＰＳ＿Ｘ８６＿ＯＲＩ　ＡＳＮノードによって、Ｘ８６　ＯＲＩノードが生成され、このＸ８６　ＯＲＩノードは即座に、以前のＸ８６定数ノードによってフォールディングされる（定数フォールディング）。その結果、単一のＸ８６　３２ビット定数ノード３３４となる。このノード３３４は、単一のＸ８６のロード定数命令「ｍｏｖ　％ｅａｘ，＄０ｘ１２３４５６７８」に、エンコードされる。図に示すように、ＡＳＮノードの結果、ノードの数はベース・ノードの例よりも少なくなる。そのため、トランスレーション・コストが下がり、より良好な目的コードが得られる。

【００７８】

図１２に、ＡＳＮを用いたＭＩＰＳ－ＭＩＰＳトランスレータ（すなわち、ＭＩＰＳアクセラレータ）によって生成されるＩＲツリーを示す。対象命令３１０，３１２は、バイナリ・デコーダ３００によってデコードされた後、ＭＩＰＳ＿ＭＩＰＳ　ＡＳＮノード３４０として実現される。次いで、このＭＩＰＳ＿ＭＩＰＳ　ＡＳＮノード３４０は、現在のブロックに対する作業用ＩＲフォレストに追加される。ソースおよび目的アーキテクチャは、ＭＩＰＳ－ＭＩＰＳトランスレータに対して同じであるため、ＭＩＰＳ＿ＭＩＰＳ＿ＬＵＩおよびＭＩＰＳ＿ＭＩＰＳ＿ＯＲＩ　ＡＳＮノード３４０では、変換コンポーネントがヌル（未定義）である。したがって、対象命令と、コードを生成するために使用される最終的なＩＲノードとの間には、直接的な対応関係がある。これによって、何らかの最適化が適用される前であっても、１：１の対象対目的の命令トランスレーション比率が保証される。言い換えれば、ＡＳＮノードによって、同一－同一トランスレータ（アクセラレータ）に対するコード爆発がなくなる。またＡＳＮノードによって、１６ビット定数ノードを共有することができる。これは、ＭＩＰＳプラットフォーム上で、連続するメモリ・アクセスを効率的にトランスレートする上で有用である。

【００７９】

命令の基本ブロックは、一度に１つの対象命令がトランスレートされる。各対象命令によって、ＩＲツリーが形成（実現）される。ＩＲツリーは、所定の命令に対して形成された後、現在のブロックに対する作業用ＩＲフォレストに統合される。作業用ＩＲフォレストのルートは、抽象レジスタである。抽象レジスタは、対象レジスタと対象アーキテクチャの他の特性とに対応する。最後の対象命令がデコードされ、実現され、そのＩＲツリーが作業用ＩＲフォレストに統合されたときに、そのブロックに対するＩＲフォレストは完成する。

【００８０】

図１２では、第１の対象命令３１０は、「ｌｕｉ　ａ１，０ｘＩ２３４」である。この命令３１０のセマンティクスは、定数値０ｘＩ２３４を、対象レジスタ「ａ１」３４２の上位１６ビットにロードすることである。この命令３１０によって、ＭＩＰＳ＿ＭＩＰＳ＿ＬＵＩノード３４４が、０ｘＩ２３４の直接フィールド定数値を用いて実現される。トランスレータによって、このノードは作業用ＩＲフォレストに追加される。この追加は、抽象レジスタ「ａ１」３４２（対象命令のデスティネーション・レジスタ）を、ＭＩＰＳ＿ＭＩＰＳ＿ＬＵＩ　ＩＲノード３４４を指し示すように設定することによって行なわれる。

【００８１】

図１２の同じ例において、第２の対象命令３１２は、「ｏｒｉ　ａ１，ａ１，０ｘ５６７８」である。この命令３１２のセマンティクスは、対象レジスタ「ａ１」３４２の現在の内容を用いて定数値０ｘ５６７８のビット単位ＯＲを行ない、対象レジスタ「ａ１」３４６内の結果を記憶することである。この命令３１２によって、ＭＩＰＳ＿ＭＩＰＳ＿ＯＲＩノード３４８が、０ｘ５６７８の直接フィールド定数値を用いて実現される。トランスレータによって、このノードは作業用ＩＲフォレストに追加される。この追加は、最初にＯＲＩノードを、抽象レジスタ「ａ１」３４２（対象命令のソース・レジスタ）が現在指し示しているＩＲツリーを指し示すように設定し、次に、抽象レジスタ「ａ１」３４６（対象命令のデスティネーション・レジスタ）を、ＯＲＩノード３４８を指し示すように設定することによって行なわれる。言い換えれば、抽象レジスタ３４２をルートとする既存の「ａ１」ツリー（すなわち、ＬＵＩノード）は、ＯＲＩノード３４８のサブツリー３５０になり、次に、ＯＲＩノード３４８は新たなａ１ツリーになる。古い「ａ１」ツリー（ＬＵＩより後だがＯＲＩより前）は、抽象レジスタ３４２をルートとしており、ライン３４５によってリンクされるように示されている。一方で、現在の「ａ１」ツリー（ＯＲＩより後）は、抽象レジスタ３４６をルートとする。

【００８２】

前述したことから分かるように、本発明により形成される改善されたプログラム・コード変換装置は、最適なレベルの性能を維持し、かつトランスレーションの速度とトランスレート済み目的コードの効率とをバランスさせる一方で、任意の対象および目的プロセッサ・アーキテクチャの組み合わせに対して構成可能である。さらに、変換に関与する対象および目的コンピューティング環境の特定のアーキテクチャに依存して、本発明のプログラム・コード変換装置を、汎用的および特定の変換特性のハイブリッド設計を用いて構成することが、ベース・ノード、複合ノード、ポリモーフィック・ノード、およびアーキテクチャ特定ノードの組み合わせを、その中間表現において用いることによって可能である。

【００８３】

本発明の改善されたプログラム・コード変換装置の種々の構造が、前述の実施形態のそれぞれにおいて別個に説明されている。しかし、本明細書で説明した各実施形態の別個の態様を、本明細書で説明した他の実施形態と組み合わせても良いことは、本発明の発明者によって完全に意図されている。たとえば、本発明により形成されるトランスレータには、種々のＩＲタイプのハイブリッド最適化が含まれていても良い。当業者ならば理解するように、説明した好ましい実施形態の種々の適合および変更を、本発明の範囲および技術思想から逸脱することなく構成することができる。したがって、本発明を、添付の特許請求の範囲内で、本明細書で具体的に説明したこと以外で、実行しても良いことが理解される。

【００８４】

いくつかの好ましい実施形態を示し説明してきたが、当業者ならば理解するように、添付の特許請求の範囲において規定される本発明の範囲から逸脱することなく、種々の変形および変更を施しても良い。

【００８５】

本出願に関連し、本明細書と同時にまたは以前に提出され、かつ本明細書を用いて公衆の便覧に公開されたすべての論文および文献に対して注意を払うものであり、このような論文および文献はすべて、その内容が本明細書において参照されることにより取り入れられるものとする。

【００８６】

本明細書（すべての添付の特許請求の範囲、要約書、および図面を含む）で開示されたすべての特徴、および／または同様に開示されたいずれかの方法またはプロセスのすべてのステップは、このような特徴および／またはステップの少なくとも一部が互いに相容れない組み合わせを除いて、任意の組み合わせで組み合わされても良い。

【００８７】

本明細書（すべての添付の特許請求の範囲、要約書、および図面を含む）で開示された各特徴は、特にことわらない限り、同じ、同等、または類似の目的を満たす代替的な特徴と取り替えても良い。したがって、特にことわらない限り、開示された各特徴は、同等または類似の特徴の包括的な組の１つの例に過ぎない。

【００８８】

本発明は、前述した実施形態の詳細に限定されない。本発明は、本明細書（すべての添付の特許請求の範囲、要約書、および図面を含む）で開示された特徴の任意の新しい１つまたは任意の新しい組み合わせに及ぶか、または同様に開示された任意の方法またはプロセスのステップの任意の新しい１つまたは任意の新しい組み合わせに及ぶ。

【図面の簡単な説明】

【００８９】

【図１】対象および目的コンピューティング環境を含むコンピューティング環境の例。

【図２】好ましいプログラム・コード変換装置。

【図３】対象コードの目的コードへのトランスレーションを示す例示的なコンピューティング環境を示す説明図。

【図４】本発明の好ましい実施形態によるプログラム・コード変換装置によって実現される種々の中間表現を示す説明図。

【図５】好ましいプログラム・コード変換装置の詳細な説明図。

【図６】ベース・ノードおよび複合ノードを用いて生成されるＩＲツリーの例。

【図７】アクセラレータにおいて本発明を実施した場合のＡＳＮ生成の例を示す説明図。

【図８】トランスレータにおいて本発明を実施した場合のＡＳＮ生成の例を示す説明図。

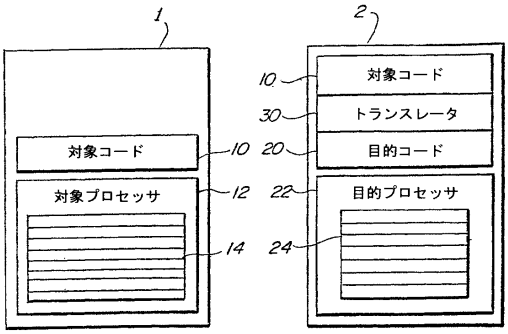
【図９】本発明の好ましい実施形態による、ＡＳＮを利用する際のトランスレーション・プロセスの処理フロー図。

【図１０】トランスレーション・プロセス、およびそのプロセスの間に生成される対応するＩＲの例を示す説明図。

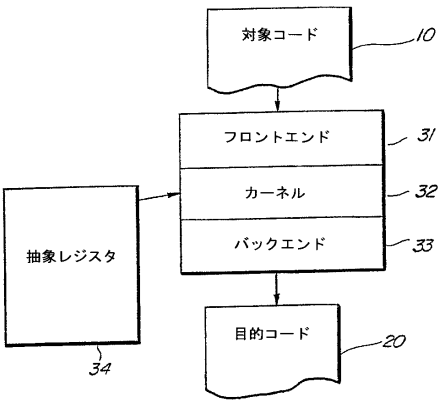
【図１１】トランスレーション・プロセス、およびそのプロセスの間に生成される対応するＩＲの別の例を示す説明図。

【図１２】トランスレーション・プロセス、およびそのプロセスの間に生成される対応するＩＲのさらに別の例を示す説明図。

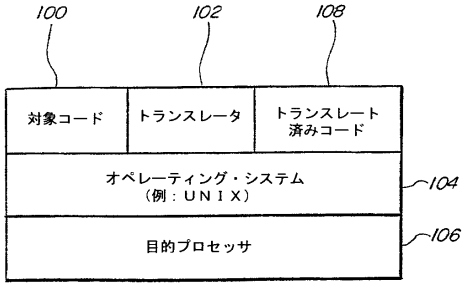
【図１】


1
2
対象コード
10
対象プロセッサ
12
14
対象コード
10
トランスレータ
30
目的コード
20
目的プロセッサ
22
24

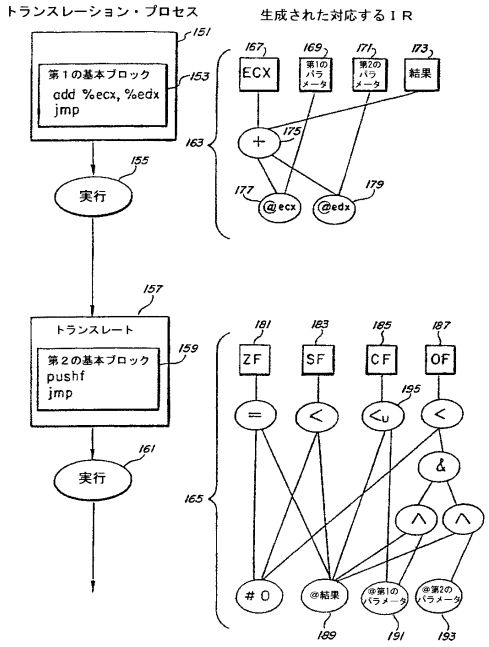
【図２】


対象コード
10
フロントエンド
31
カーネル
32
バックエンド
33
抽象レジスタ
34
目的コード
20

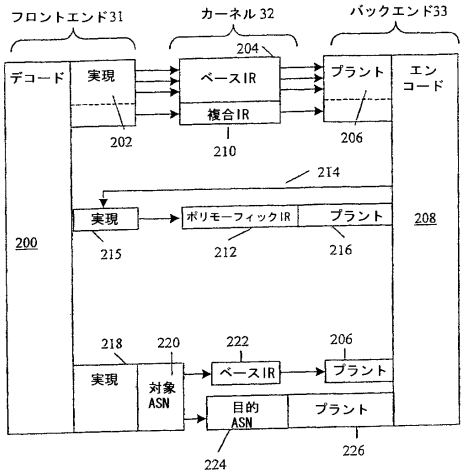
【図３】


100
102
108
対象コード
トランスレータ
トランスレート済みコード
オペレーティング・システム
（例：ＵＮＩＸ）
104
目的プロセッサ
106

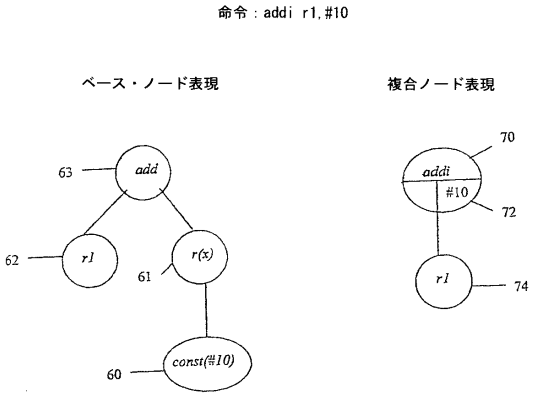
【図４】


トランスレーション・プロセス
生成された対応するＩＲ
151
第１の基本ブロック
add %ecx, %edx
jmp
153
163
155
実行
157
トランスレート
159
第２の基本ブロック
pushf
jmp
161
実行
165
167
ECX
169
第1のパラメータ
171
第2のパラメータ
173
結果
175
177
@ecx
179
@edx
181
ZF
183
SF
185
CF
187
OF
195
=
<
<u
<
&
^
^
#0
@結果
@第1のパラメータ
@第2のパラメータ
189
191
193

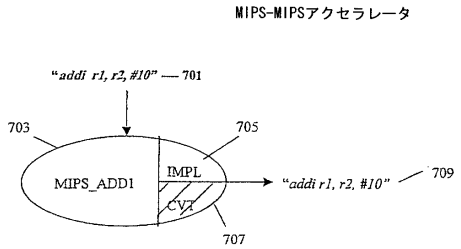
【図５】


フロントエンド31
カーネル 32
バックエンド33
デコード
実現
202
ベースIR
複合IR
204
210
プラント
206
エンコード
214
実現
215
ポリモーフィックIR
212
プラント
216
208
200
218
実現
220
対象ASN
222
ベースIR
206
プラント
目的ASN
224
プラント
226

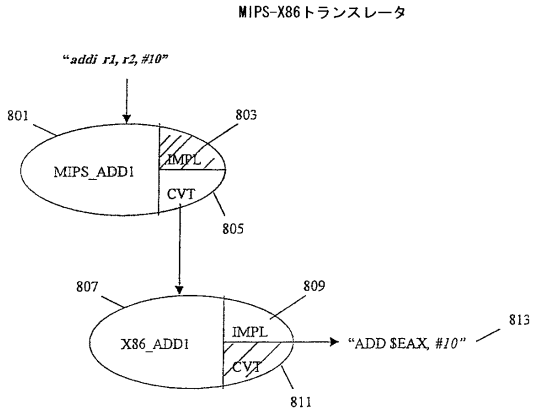
【図６】


命令：addi r1,#10
ベース・ノード表現
複合ノード表現
63
add
62
r1
61
r(x)
60
const(#10)
70
addi
#10
72
r1
74

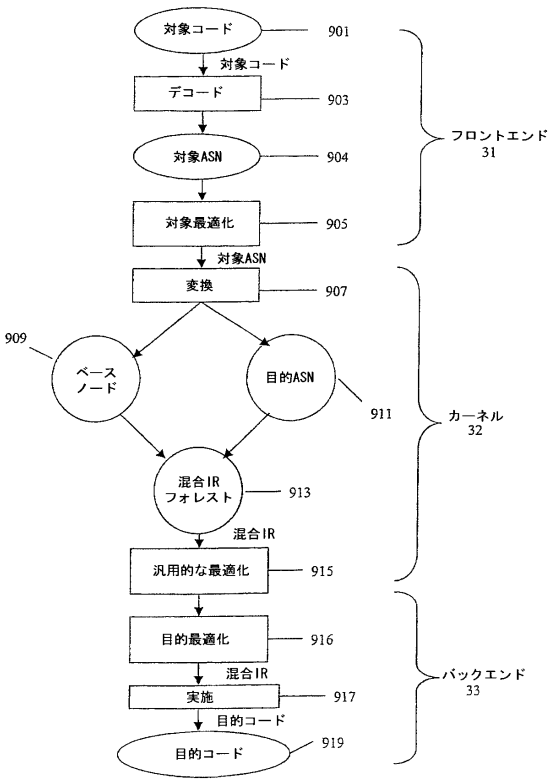
【図７】


MIPS-MIPSアクセラレータ
"addi r1, r2, #10" 701
703
705
MIPS_ADD1
IMPL
CVT
"addi r1, r2, #10"
709
707

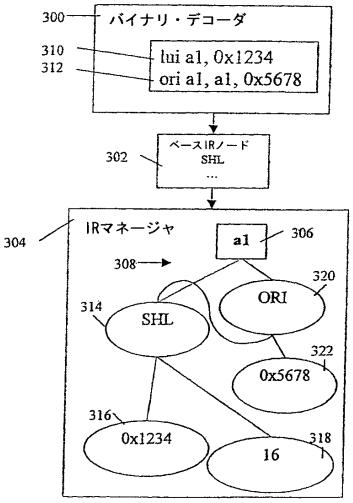
【図８】


MIPS-X86トランスレータ
"addi r1, r2, #10"
801
803
MIPS_ADD1
IMPL
CVT
805
807
809
X86_ADD1
IMPL
CVT
"ADD $EAX, #10"
813
811

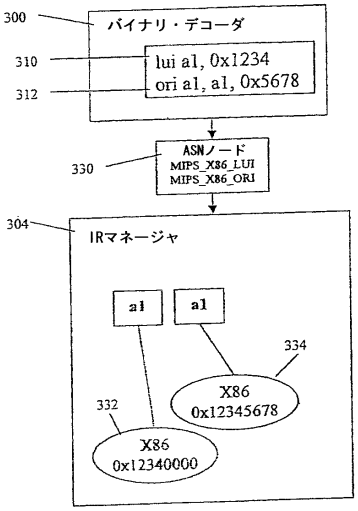
【図９】


対象コード
901
対象コード
デコード
903
対象ASN
904
対象最適化
905
フロントエンド
31
対象ASN
変換
907
909
ベースノード
目的ASN
911
混合IR
フォレスト
913
混合IR
汎用的な最適化
915
カーネル
32
目的最適化
916
混合IR
実施
917
目的コード
目的コード
919
バックエンド
33

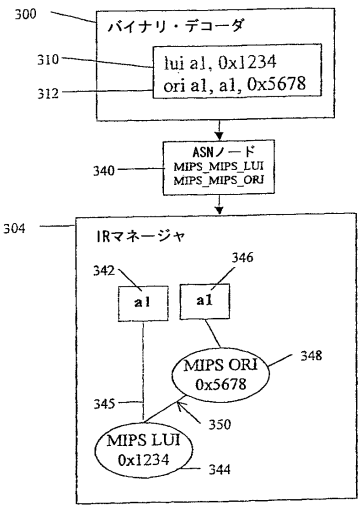
【図１０】


300
バイナリ・デコーダ
310
312
lui a1, 0x1234
ori a1, a1, 0x5678
302
ベースIRノード
SHL
...
304
IRマネージャ
a1
306
308
320
ORI
314
SHL
322
0x5678
316
0x1234
318
16

【図１１】


300
バイナリ・デコーダ
310
312
lui a1, 0x1234
ori a1, a1, 0x5678
330
ASNノード
MIPS_X86_LUI
MIPS_X86_ORI
304
IRマネージャ
a1
a1
334
X86
0x12345678
332
X86
0x12340000

【図１２】


300
バイナリ・デコーダ
310
312
lui a1, 0x1234
ori a1, a1, 0x5678
340
ASNノード
MIPS_MIPS_LUI
MIPS_MIPS_ORI
304
IRマネージャ
342
a1
346
a1
348
MIPS ORI
0x5678
345
350
MIPS LUI
0x1234
344

フロントページの続き

(72)発明者　オーウェン、ダニエル
イギリス国　Ｍ３　２ＥＧ　マンチェスター　ブラックフライアーズ　ストリート　メイブルック　ハウス　トランジティブ　リミテッド　内
(72)発明者　アンドリューズ、ジョナサン　ジェイ
イギリス国　Ｍ３　２ＥＧ　マンチェスター　ブラックフライアーズ　ストリート　メイブルック　ハウス　トランジティブ　リミテッド　内
(72)発明者　ホーソン、マイルズ　フィリップ
イギリス国　Ｍ３　２ＥＧ　マンチェスター　ブラックフライアーズ　ストリート　メイブルック　ハウス　トランジティブ　リミテッド　内
(72)発明者　ハイケン、デイビッド
イギリス国　Ｍ３　２ＥＧ　マンチェスター　ブラックフライアーズ　ストリート　メイブルック　ハウス　トランジティブ　リミテッド　内

審査官　坂庭　剛史

(56)参考文献　特表２００２－５４３４９０（ＪＰ，Ａ）
特表２００２－５２７８１５（ＪＰ，Ａ）
特開２００２－３１２１８０（ＪＰ，Ａ）
特開２０００－３４７８７３（ＪＰ，Ａ）
特開平１１－１９４９４８（ＪＰ，Ａ）
特開平０７－１０５０１５（ＪＰ，Ａ）
特開平０６－２５０８４６（ＪＰ，Ａ）
特開平０４－０１４１４４（ＪＰ，Ａ）
実開平０４－０３６６４６（ＪＰ，Ｕ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F　9/455
G06F　9/30