(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2010-198629
(P2010-198629A)
(43)公開日　平成22年9月9日(2010.9.9)

(51)Int.Cl.	F I	テーマコード（参考）
G06F 9/455 (2006.01)	G06F 9/44 310A	

審査請求 有　請求項の数 23　ＯＬ　（全 33 頁）

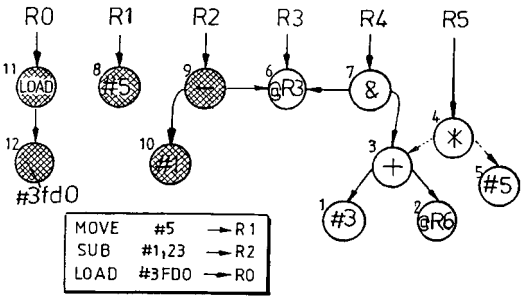
(21)出願番号	特願2010-94372(P2010-94372)
(22)出願日	平成22年4月15日(2010.4.15)
(62)分割の表示	特願2000-576360(P2000-576360)の分割
原出願日	平成11年10月11日(1999.10.11)
(31)優先権主張番号	9822075.9
(32)優先日	平成10年10月10日(1998.10.10)
(33)優先権主張国	英国(GB)
(31)優先権主張番号	60/115,952
(32)優先日	平成11年1月14日(1999.1.14)
(33)優先権主張国	米国(US)

(71)出願人　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方法

(57)【要約】　（修正有）
【課題】あるプロセッサ用に書かれたコンピュータプログラムを異なるプロセッサ上で効率的に実行できるように変換する。
【解決手段】プログラムコードの第１の部分の最初の変換において、プログラムコードの前記第１の部分を優勢な条件セット下で実行するのに必要とされる中間表現だけを生成及び記憶し、プログラムコードの前記第１の部分と同じ部分６が後で入力されたときは常に、サブジェクトコードの第１の部分のための中間表現が後続の条件に対して以前に生成及び記憶されたかどうかを判定し、そのような中間表現が以前に生成されていない場合は、プログラムコードの前記第１の部分を前記後続の条件で実行するのに必要とされる追加の中間表現を生成することにより、前記プログラムコードの中間表現をメモリ内に生成して、エミュレーション実行する。
【選択図】図５



【特許請求の範囲】
【請求項１】
プログラムコードの中間表現を生成する方法であって、
抽象レジスタを表す複数のレジスタオブジェクトを生成するコンピュータ実施ステップであって、単一のレジスタオブジェクトがそれぞれの抽象レジスタを表すコンピュータ実施ステップと、
表現オブジェクトを生成するコンピュータ実施ステップであって、各表現オブジェクトが、プログラム中に生じる異なるサブジェクトコード要素を表し、直接的に、又は他の表現オブジェクトからの参照を介して間接的に関係するレジスタオブジェクトから参照されるコンピュータ実施ステップとを含む方法。
【請求項２】
プログラム可能なマシン上で実行するために書かれたコンピュータプログラムコードの中間表現を生成する方法であって、
（ｉ）プログラムコードによって生成されることになる可変値を保持するためのレジスタオブジェクトを生成すること、ならびに
（ｉｉ）固定値、及び／又は前記固定値と前記プログラムコードに従った前記可変値との関係を表す、複数の表現オブジェクトを生成することを含み、
前記オブジェクトは、すべてのレジスタオブジェクトをネットワークの最下部の基本ルート又はツリートランクのレベルに有する分岐したツリー状のネットワークに構成され、前記レジスタオブジェクトは他のどのレジスタオブジェクトにも供給されない方法。
【請求項３】
前記プログラムコードがサブジェクトプロセッサの命令セットで表される、請求項１又は２に記載の方法。
【請求項４】
前記レジスタオブジェクトが前記サブジェクトプロセッサのレジスタに対応する抽象レジスタを表す、前記請求項のいずれかに記載の方法。
【請求項５】
ただ１つの有効なエントリーポイント命令を有しかつただ１つの有効なエクジットポイント命令を有する、プログラムコードの基本ブロックに対し、前記各ステップが順次行われる、前記請求項のいずれかに記載の方法。
【請求項６】
少なくともいくつかの表現オブジェクトが複数のレジスタオブジェクトに供給される、前記請求項のいずれかに記載の方法。
【請求項７】
表現オブジェクトが複製されない、前記請求項のいずれかに記載の方法。
【請求項８】
サブジェクトコードの所与の要素に対して単一の表現オブジェクトが生成され、各表現オブジェクトが、関係するすべてのレジスタオブジェクトから参照される、前記請求項のいずれかに記載の方法。
【請求項９】
オブジェクトが冗長又は不要になった場合にそれが除去される、前記請求項のいずれかに記載の方法。
【請求項１０】
冗長又は不要なオブジェクトが、レジスタと表現オブジェクトとのネットワークが構築されるときにそのオブジェクトに対してなされる参照の継続カウントを維持することによって識別される、請求項９に記載の方法。
【請求項１１】
各表現オブジェクトに対し、他の表現オブジェクト又は抽象レジスタからその表現オブジェクトへの参照の数のカウントが維持され、特定の表現オブジェクトに関連するカウントが、その表現オブジェクトが作成されるか除去されるたびに調整される、請求項１０に記

載の方法。
【請求項１２】
表現オブジェクトに対する前記カウントが０のとき、その表現オブジェクトと、その表現オブジェクトからのすべての参照とが除去される請求項１１に記載の方法。
【請求項１３】
少なくとも１つの可変サイズのレジスタが複数のレジスタオブジェクトによって表され、可変サイズのレジスタの可能な各サイズにつき１つのレジスタオブジェクトが提供される、前記請求項のいずれかに記載の方法。
【請求項１４】
適切なサイズに対応するレジスタオブジェクトに書き込み、どのレジスタオブジェクトが有効データを含むかの記録を保持することによって、可変サイズのレジスタへの書込み動作が行われる、請求項１３に記載の方法。
【請求項１５】
有効データが複数の対応するレジスタオブジェクトにあってそれを可変サイズのレジスタからの読取りと同じ効果をもたらすように結合しなければならないかどうかを前記記録から判定することにより、可変サイズのレジスタからの読取り動作が行われ、
ｉ）そのような結合が必要でないと判定された場合は、適切なレジスタオブジェクトから直接に読み取り、
ｉｉ）そのような結合が必要であると判定された場合は、適切なレジスタオブジェクトの内容を結合して読取り値を提供する、請求項１４に記載の方法。
【請求項１６】
サブジェクトプロセッサのレジスタが１つ又は複数の可変サイズのレジスタを含む場合、サブジェクトプロセッサのレジスタに対応する抽象レジスタを表す複数のレジスタオブジェクトを生成するステップがさらに、１つの又はそれぞれの可変サイズのプロセッサレジスタを表す関連する抽象レジスタオブジェクトのセットを生成することを含む方法であって、前記セットは、可変サイズの各レジスタの可能な各幅につき１つの抽象レジスタを含み、前記方法は、
可変サイズのレジスタへのあるフィールド幅の書込み動作ごとに、同じ幅の抽象レジスタに書き込むコンピュータ実施ステップと、
どの抽象レジスタが有効データを含むかの記録を保持するコンピュータ実施ステップであって、前記記録は書込み動作ごとに更新されるコンピュータ実施ステップと、
有効データが複数の前記異なるサイズの抽象レジスタのセットにあってそれを可変サイズのレジスタに対して行われる同じ読取り動作と同じ効果をもたらすように結合しなければならないかどうかを、所与のフィールド幅の書込み動作ごとに前記記録から判定するコンピュータ実施ステップと、
ａ）そのように結合が必要とされないと判定された場合は、適切なレジスタから直接に読み取り、あるいは
ｂ）複数のレジスタからのデータをそのように結合しなければならないと判定された場合は、それらのレジスタの内容を結合するコンピュータ実施ステップを含む請求項４及びそれに従属するいずれかの請求項に記載の方法。
【請求項１７】
複数の抽象レジスタの内容を結合しなければならないか否か、また結合しなければならない場合にはどれを結合しなければならないかを決定するステップが、異なるサイズの抽象レジスタの各セットに関して以下の条件、すなわち
ｉ）アクセスに必要とされるデータが１つの有効抽象レジスタ内に完全にある場合は、そのレジスタだけがアクセスされ、
ｉｉ）アクセスに必要とされるデータが複数の有効抽象レジスタ内にある場合は、それらの有効抽象レジスタからデータが結合されてアクセスが行われる、という条件に従って決定される請求項１６に記載の方法。
【請求項１８】

前記中間表現が記憶され、同じコードブロックが後で再入力された場合に後で再利用される、前記請求項のいずれかに記載の方法。
【請求項１９】
前記第１のコンピュータプログラムコードの少なくとも１ブロックが、代替の未使用入口条件又は効果又は機能を有することができ、レジスタオブジェクトを生成するステップが、プログラムコードのその部分をその時の優勢な条件で実行するのに必要な中間表現を最初にだけ生成及び記憶する請求項１８に記載の方法。
【請求項２０】
それにより以前に記憶された中間表現が現在優勢な同じ条件セットに対するものであったかどうかを、変換すべき所与のコードブロックに対して判定し、そうでない場合は、前記コードブロックを実行するのに必要な追加の中間表現を現在優勢な新しい条件セットに対して生成及び記憶する、請求項１９に記載の方法。
【請求項２１】
サブジェクトコードの所与の部分を最初に変換する際に、プログラムコードのその部分を優勢な条件セットで実行するのに必要な中間表現だけを生成及び記憶するコンピュータ実施ステップと、
サブジェクトコードの同じ部分が後で入力されたときは常に、サブジェクトコードのその部分のための中間表現が後続の条件に対して以前に生成及び記憶されたかどうかを判定し、そのような中間表現が以前に生成されていない場合は、サブジェクトコードの前記部分を前記後続の条件で実行するのに必要な追加の中間表現を生成するコンピュータ実施ステップとをさらに含む前記請求項のいずれかに記載の方法。
【請求項２２】
前記中間表現がサブジェクトプログラムの実行中に動的に生成される、前記請求項のいずれかに記載の方法。
【請求項２３】
コンパイル及び／又は変換するために書かれ、第１のプログラム可能なマシン上で実行される第１のコンピュータプログラムコードが、異なる第２のプログラム可能なマシン上で実行するための第２のコンピュータプログラムコードに動的に変換され、
（ａ）前記第１のコンピュータプログラムコードのブロックの中間表現を生成すること、
（ｂ）前記中間表現から前記第２のコンピュータプログラムコードのブロックを生成すること、
（ｃ）第２のコンピュータプログラムコードの前記ブロックを前記第２のプログラム可能なマシン上で実行すること、及び
（ｄ）前記第２のプログラム可能なマシン上で第１のコンピュータプログラムコードをエミュレートする現行の実行に必要な、第１のコンピュータプログラムコードの少なくとも１つのブロックに対し、ステップａ～ｃをリアルタイムで繰り返すことを含む、請求項２２に記載の方法。
【請求項２４】
複数の可能な効果又は機能を有する特定のサブジェクトコード命令の第１の反復時に、その反復で必要とされる特定の機能だけを表す特殊ケース中間表現を生成及び記憶するコンピュータ実施ステップと、
同じサブジェクトコード命令の後続の各反復で、前記後続の反復で必要とされる必要な機能に対して特殊ケース中間表現が生成されているかどうかを判定し、そのような特殊ケース中間表現が以前に生成されていない場合に、その機能特有の追加の特殊ケース中間表現を生成するコンピュータ実施ステップを含む請求項２２又は２３に記載の方法。
【請求項２５】
前記特殊ケース中間表現が生成及び記憶されるとき、必要な機能が関連の記憶済み特殊ケース中間表現によって表される機能と同じかどうかを各サブジェクトコード命令の後続の反復時に判定するための関連テストプロシージャが生成及び記憶され、追加の特殊ケース

中間表現が必要な場合は、その特殊ケース中間表現に関連する追加のテストプロシージャがその追加の特殊ケース中間表現と共に生成及び記憶される請求項２４に記載の方法。
【請求項２６】
特定のサブジェクトコード命令に対する追加の特殊ケース中間表現及び追加の関連テストプロシージャが、少なくとも最初はいずれかの既存の特殊ケース中間表現及び記憶された関連テストプロシージャに対する従属関係で記憶され、したがって、サブジェクトコード命令の第２の反復以降に、必要とされる機能の特殊ケース中間表現が存在すると判定されるか又は必要とされる特殊ケース中間表現が存在しないと判定されるまで前記テストプロシージャをそれが生成及び記憶された順に行うことによって、必要とされる特殊ケース中間表現が以前に生成されたかどうかの判定が行われ、必要とされる中間表現が存在しないと判定された場合は、さらに追加の中間表現及び別の関連テストプロシージャが生成される請求項２５に記載の方法。
【請求項２７】
生成された順にテストプロシージャを順序付けるのではなく、使用頻度のより高い特殊ケース中間表現に関連するテストプロシージャが使用頻度のより低い特殊ケース中間表現に関連したテストプロシージャの前に実行されるようにテストプロシージャの順序を調整することによって中間表現が最適化される請求項２６に記載の方法。
【請求項２８】
プログラム可能なマシン上で実行するために書かれたコンピュータコードの中間表現を生成する方法であって、
（ｉ）プログラムコードによって生成されることになる可変値を保持するための複数のレジスタオブジェクトを生成すること、ならびに
（ｉｉ）固定値、及び／又は前記固定値と前記プログラムコードに従った前記可変値との関係を表す、複数の表現オブジェクトを生成することを含み、
少なくとも１つの可変サイズのレジスタが複数のレジスタオブジェクトによって表され、可変サイズのレジスタの可能な各サイズにつき１つのレジスタオブジェクトが提供される方法。
【請求項２９】
適切なサイズに対応するレジスタオブジェクトに書き込み、どのレジスタオブジェクトが有効データを含むかの記録を保持することによって、可変サイズのレジスタへの書込み動作が行われる、請求項２８に記載の方法。
【請求項３０】
有効データが複数の対応するレジスタオブジェクトにあってそれを可変サイズのレジスタからの読取りと同じ効果をもたらすように結合しなければならないかどうかを前記記録から判定することにより、可変サイズのレジスタからの読取り動作が行われ、
ｉ）そのような結合が必要でないと判定された場合は、適切なレジスタオブジェクトから直接に読み取り、
ｉｉ）そのような結合が必要であると判定された場合は、適切なレジスタオブジェクトの内容を結合して読取り値を提供する、請求項２９に記載の方法。
【請求項３１】
少なくとも１つの可変サイズのレジスタを含むサブジェクトプロセッサの命令セットで表されるプログラムコードの中間表現を生成する方法であって、
可変サイズのレジスタを表す関連する抽象レジスタオブジェクトのセットを生成するコンピュータ実施ステップと、
可変サイズのレジスタへのあるフィールド幅の書込み動作ごとに、同じ幅の抽象レジスタに書き込むコンピュータ実施ステップと、
どの抽象レジスタが有効データを含むかの記録を保持するコンピュータ実施ステップであって、前記記録が書込み動作ごとに更新されるコンピュータ実施ステップと、
有効データが複数の前記異なるサイズの抽象レジスタのセットのうちにあってそれを可変サイズのレジスタに対して行われる同じ読取り動作と同じ効果をもたらすように結合し

なければならないどうかを、所与のフィールド幅の書込み動作ごとに前記記録から判定するコンピュータ実施ステップと、
ａ）そのように結合が必要とされないと判定された場合は、適切なレジスタから直接に読み取り、あるいは
ｂ）複数のレジスタからのデータをそのように結合しなければならないと判定された場合は、それらのレジスタの内容を結合するコンピュータ実施ステップを含む方法。
【請求項３２】
複数の抽象レジスタの内容を結合しなければならないか否か、また結合しなければならない場合にはどれを結合しなければならないかを決定するステップが、異なるサイズの抽象レジスタの各セットに関して以下の条件、すなわち
ｉ）アクセスに必要とされるデータが１つの有効抽象レジスタ内に完全にある場合は、そのレジスタだけがアクセスされ、
ｉｉ）アクセスに必要とされるデータが複数の有効抽象レジスタ内にある場合は、それらの有効抽象レジスタからデータが結合されてアクセスが行われる、という条件に従って決定される、請求項３１に記載の方法。
【請求項３３】
コンピュータプログラムコードの中間表現を生成する方法であって、
サブジェクトコードの所与の部分を最初に変換する際に、プログラムコードのその部分を優勢な条件セットで実行するのに必要な中間表現だけを生成及び記憶するコンピュータ実施ステップと、
サブジェクトコードの同じ部分が後で入力されたときは常に、サブジェクトコードのその部分のための中間表現が後続の条件に対して以前に生成及び記憶されたかどうかを判定し、そのような中間表現が以前に生成されていない場合は、サブジェクトコードの前記部分を前記後続の条件で実行するのに必要な追加の中間表現を生成するコンピュータ実施ステップとを含む方法。
【請求項３４】
前記条件が入口条件であり、
プログラムによって必要とされるときにプログラムコードのＢａｓｉｃ　Ｂｌｏｃｋごとに中間表現の中間表現ブロック（ＩＲ　Ｂｌｏｃｋ）を生成するコンピュータ実施ステップであって、各ＩＲ　Ｂｌｏｃｋが特定の入口条件に対するプログラムコードの各Ｂａｓｉｃ　Ｂｌｏｃｋを表すコンピュータ実施ステップと、
各ＩＲ　Ｂｌｏｃｋに対応するターゲットコードを記憶するコンピュータ実施ステップと、
プログラムが所与の入口条件に対するＢａｓｉｃ　Ｂｌｏｃｋの実行を必要とするときに、
ａ）その所与の入口条件に対するそのＢａｓｉｃ　Ｂｌｏｃｋを表す記憶済みターゲットコードがある場合は、前記記憶済みターゲットコードを使用し、あるいは
ｂ）その所与の入口条件に対するそのＢａｓｉｃ　Ｂｌｏｃｋを表す記憶済みターゲットコードがない場合は、その所与の入口条件に対するそのＢａｓｉｃ　Ｂｌｏｃｋを表す別のＩＲ　Ｂｌｏｃｋを生成するコンピュータ実施ステップを含む請求項３３に記載の方法。
【請求項３５】
プログラムの実行中にプログラムの中間表現が動的に生成され、
複数の可能な効果又は機能を有する特定のサブジェクトコード命令の第１の反復時に、その反復で必要とされる特定の機能だけを表す特殊ケース中間表現を生成及び記憶するコンピュータ実施ステップと、
同じサブジェクトコード命令の後続の各反復で、前記後続の反復で必要とされる必要な機能に対して特殊ケース中間表現が生成されているかどうかを判定し、そのような特殊ケース中間表現が以前に生成されていない場合に、その機能特有の追加の特殊ケース中間表現を生成するコンピュータ実施ステップとを含む請求項３３に記載の方法。

【請求項３６】
前記特殊ケース中間表現が生成及び記憶されるとき、必要な機能が関連する記憶済み特殊ケース中間表現によって表される機能と同じかどうかを各サブジェクトコード命令の後続の反復時に判定するための関連テストプロシージャが生成及び記憶され、追加の特殊ケース中間表現が必要な場合は、その特殊ケース中間表現に関連する追加のテストプロシージャが、その追加の特殊ケース中間表現と共に生成及び記憶される請求項３３に記載の方法。
【請求項３７】
特定のサブジェクトコード命令に対する追加の特殊ケース中間表現及び追加の関連テストプロシージャが、少なくとも最初はいずれかの既存の特殊ケース中間表現及び記憶済み関連テストプロシージャに対する従属関係で記憶され、したがって、サブジェクトコード命令の第２の反復以降に、必要とされる機能の特殊ケース中間表現が存在すると判定されるか又は必要とされる中間表現が存在しないと判定されるまで前記テストプロシージャをそれが生成及び記憶された順に行うことによって、必要とされる特殊ケース中間表現が以前に生成されたかどうかの判定が行われ、必要とされる中間表現が存在しないと判定された場合は、さらに追加の中間表現及び別の関連テストプロシージャが生成される請求項３６に記載の方法。
【請求項３８】
生成された順にテストプロシージャを順序付けるのではなく、使用頻度のより高い特殊ケース中間表現に関連するテストプロシージャが使用頻度のより低い特殊ケース中間表現に関連したテストプロシージャの前に実行されるようにテストプロシージャの順序を調整することによって中間表現が最適化される請求項３７に記載の方法。
【請求項３９】
プログラム可能なマシン上で実行するために書かれたコンピュータコードの中間表現を生成する方法であって、
（ｉ）プログラムコードによって生成されることになる可変値を保持するための複数のレジスタオブジェクトを生成すること、ならびに
（ｉｉ）固定値、及び／又は前記固定値と前記プログラムコードに従った前記可変値との関係を表す、複数の表現オブジェクトを生成することを含み、
前記中間表現は、コンピュータコードのブロックに対して生成及び記憶され、同じコードブロックが後で再入力された場合は後で再利用され、前記第１のコンピュータプログラムコードの少なくとも１ブロックが、代替の未使用入口条件又は効果又は機能を有することができ、前記中間表現は、プログラムコードのそのブロックをその時の優勢な条件で実行するのに必要なとき、最初にだけ生成及び記憶される方法。
【請求項４０】
それにより以前に記憶された中間表現が現在優勢な同じ条件セットに対するものであったかどうかを、変換すべき所与のコードブロックに対して判定し、そうでない場合は、コードのブロックを実行するのに必要な追加の中間表現を現在優勢な新しい条件セットに対して生成及び記憶する、請求項３９に記載の方法。
【請求項４１】
コンピュータプログラムコードのターゲットコード表現を生成する方法であって、
サブジェクトコードの所与の部分を最初に変換する際に、プログラムコードのその部分を優勢な条件セットで実行するのに必要なターゲットコードだけを生成及び記憶するコンピュータ実施ステップと、
サブジェクトコードの同じ部分が後で入力されたときは常に、サブジェクトコードのその部分のためのターゲットコードが後続の条件に対して以前に生成及び記憶されたかどうかを判定し、そのようなターゲットコードが以前に生成されていない場合は、サブジェクトコードの前記部分を前記後続の条件で実行するのに必要な追加のターゲットコードを生成するコンピュータ実施ステップとを含む方法。
【請求項４２】

コンパイル及び／又は変換するために書かれ、第１のプログラム可能なマシン上で実行される第１のコンピュータプログラムコードを、異なる第２のプログラム可能なマシン上で実行するための第２のコンピュータプログラムコードに動的に変換する方法であって、
（ａ）前記第１のコンピュータプログラムコードのブロックの中間表現を生成すること、
（ｂ）前記中間表現から前記第２のコンピュータプログラムコードのブロックを生成すること、
（ｃ）第２のコンピュータプログラムコードの前記ブロックを前記第２のプログラム可能なマシン上で実行すること、及び
（ｄ）前記第２のプログラム可能なマシン上で第１のコンピュータプログラムコードをエミュレートする現行の実行に必要な、第１のコンピュータプログラムコードの少なくとも１つのブロックに対し、ステップａ～ｃをリアルタイムで繰り返すことを含む方法。
【請求項４３】
第１のタイプのプロセッサによって実行するために書かれたコンピュータプログラムを、第２のタイプのプロセッサによって実行できるように変換する方法であって、請求項１から３８までのいずれか一項に記載にしたがって中間表現を生成することを含む方法。
【請求項４４】
前記変換が動的であってプログラムの実行中に行われる、請求項４３に記載の方法。
【請求項４５】
コンピュータプログラムを最適化する方法であって、請求項１から４０までのいずれか一項の記載にしたがって中間表現を生成すること、及び前記中間表現を最適化することを含む方法。
【請求項４６】
第１のタイプのプロセッサによって実行するために書かれたコンピュータプログラムを、そのプロセッサによってより効率的に実行できるように最適化するのに使用される、請求項４５に記載の方法。
【請求項４７】
請求する方法の各ステップを行うための手段を備えた、前記請求項のいずれかの方法を行うためのシステム。
【請求項４８】
添付の図面を参照しながら以下に実質的に述べる、プログラムコードの中間表現を生成する方法。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、プログラムコードをあるフォーマットから別のフォーマットに変換するための方法及びシステムに関する。特に、本発明は、コンピュータプログラム又はプログラムのＢａｓｉｃ　Ｂｌｏｃｋの中間表現を提供するための方法及びシステムに関する（プログラムのＢａｓｉｃ　Ｂｌｏｃｋとは、ただ１つのエントリーポイントを第１の命令に有し、ただ１つのエクジットポイントをそのブロックの最後の命令に有する、命令のブロックである）。例えば本発明は、あるプロセッサ用に書かれたコンピュータプログラムを異なるプロセッサ上で効率的に実行できるように変換するための方法及びシステムを提供する。この変換は、中間表現を利用し、ブロックごとのモードで行われる。
【背景技術】
【０００２】
中間表現とは、プログラムを表現できるが、どの特定プロセッサにも特有ではなく、どのプロセッサ上で直接に実行するようにも意図されていない抽象的なコンピュータ言語の形式を呼ぶのに、コンピュータ業界で広く使用されている用語である。例えば、中間表現は一般に、プログラムの最適化を可能にするために作成される。例えばコンパイラは、高級言語コンピュータプログラムを中間表現に変換し、その中間表現に様々な最適化技術を

適用することによってプログラムを最適化し、次いで、最適化した中間表現を実行可能バイナリコードに変換することになる。中間表現はまた、プログラムを、どのプロセッサにも特有ではない形式でインターネットを横断して送信できるようにもする。例えばＳｕｎ Ｍｉｃｒｏｓｙｓｔｅｍｓは、この目的のためにバイトコードと呼ばれる中間表現の形式を開発した。バイトコードは、周知のＪａｖａ（商標）ランタイムシステムが採用されたプロセッサ上ならどんなプロセッサ上にも実装することができる。

【０００３】

中間表現はまた、バイナリ変換を採用するエミュレーション・システムによっても一般的に使用されている。このタイプのエミュレーション・システムは、所与のプロセッサタイプ用にコンパイルされたソフトウェアコードを取り込み、それを中間表現に変換し、その中間表現を最適化し、次いでその中間表現を別のプロセッサタイプ上で実行できるコードに変換する。中間表現生成の最適化は、エミュレートされたプログラムを実行するのに必要なコードの量を最小限に抑えるのに使用される周知のプロシージャである。中間表現の最適化には、周知の様々な方法が存在する。

【０００４】

バイナリ変換を行うために中間表現を使用する周知のエミュレーション・システムの一例は、ＡＴ＆Ｔによって運営されるＦｌａｓｈＰｏｒｔシステムである。顧客は、変換すべきプログラムをＡＴ＆Ｔに提供する（プログラムは第１のタイプのプロセッサ上で実行されるようにコンパイルされている）。プログラムは、ＡＴ＆Ｔによって中間表現に変換され、中間表現は、技術者の補助により自動最適化ルーチンのアプリケーションを介して最適化される。技術者は、最適化ルーチンが失敗したときに入力を提供する。最適化された中間変換は、次いでＡＴ＆Ｔによって所望のタイプのプロセッサ上で実行できるコードに変換される。このような、実行される前にプログラム全体が変換されるタイプのバイナリ変換は、「静的」バイナリ変換と呼ばれる。変換時間は、数カ月までのどんな時間になる可能性もある。

【０００５】

エミュレーションの代替形式では、サブジェクトプロセッサ（すなわち、コードがそれに向けて書かれている、エミュレートされることになる第１のタイプのプロセッサ）のコードのプログラムが、中間表現を介してＢａｓｉｃ Ｂｌｏｃｋでターゲットプロセッサ（すなわち、エミュレーションがその上で行われる第２のタイプのプロセッサ）のコードに変換される。

【図面の簡単な説明】

【０００６】

【図１】本発明に係る動的エミュレーション・システムが中間表現を生成する方式の概略図（ＭＯＶＥ命令）

【図２】本発明に係る動的エミュレーション・システムが中間表現を生成する方式の概略図（ＡＤＤ命令）

【図３】本発明に係る動的エミュレーション・システムが中間表現を生成する方式の概略図（ＭＵＬ命令）

【図４】本発明に係る動的エミュレーション・システムが中間表現を生成する方式の概略図（ＡＮＤ Ｅｘｐｒｅｓｓｉｏｎ）

【図５】本発明に係る動的エミュレーション・システムが中間表現を生成する方式の概略図（ライン６のＭＯＶＥ命令による再定義）

【図６】本発明に係る動的エミュレーション・システムが中間表現を生成する方式の概略図（サブジェクトプロセッサレジスタの伝搬の例）

【図７】本発明に係る動的エミュレーション・システムが中間表現を生成する方式の概略図（２つのＩｓｏＢｌｏｃｋを適用した例）

【発明を実施するための形態】

【０００７】

本発明の第１の態様の一目的は、プログラムコードの中間表現を生成する方法を提供す

ることであり、この方法は、抽象レジスタを表す複数のレジスタオブジェクトを生成するコンピュータ実施ステップであって、単一のレジスタオブジェクトがそれぞれの抽象レジスタを表すコンピュータ実施ステップと、

表現オブジェクトを生成するコンピュータ実施ステップであって、各表現オブジェクトが、プログラム中に生じる異なるサブジェクトコード要素を表し、直接的に、又は他の表現オブジェクトからの参照を介して間接的に関係するレジスタオブジェクトから参照されるコンピュータ実施ステップとを含む。

サブジェクトコードの要素は、サブジェクトコード命令の演算又は副次演算である。各サブジェクトコード命令はこのような要素をいくつか含むこともでき、したがって、いくつかの表現オブジェクトを生成して、単一のサブジェクトコード命令を表すこともできる。

【０００８】

また本発明の第１の態様によれば、プログラム可能なマシン上で実行するために書かれたコンピュータプログラムコードの中間表現を生成する方法が提供され、前記方法は、

（ｉ）プログラムコードによって生成されることになる可変値を保持するための複数のレジスタオブジェクトを生成すること、ならびに

（ｉｉ）固定値、及び／又は前記固定値と前記プログラムコードに従った前記可変値との関係を表す、複数の表現オブジェクトを生成することを含み、

前記オブジェクトは、すべてのレジスタオブジェクトをネットワークの最下部の基本ルート又はツリートランクのレベルに有する分岐したツリーのようなネットワークに構成され、レジスタオブジェクトは他のどのレジスタオブジェクトにも供給されない。

中間表現を形成するときは、中間表現によって表されているサブジェクトプロセッサの状況（例えばそのレジスタ又はメモリ空間の状況）の表現を含めることが必要である。本発明ではこれは、抽象レジスタを作成することにより、特に効率的な方式で行われる。

【０００９】

本発明によれば、所与の抽象レジスタを表すのに単一のレジスタオブジェクトしか生成する必要がなく（これは、初期化時にすべての抽象レジスタに対して行うことが好ましい）、各抽象レジスタの状態は、対応するレジスタオブジェクトから参照される表現オブジェクトによって定義される。所与のレジスタオブジェクトから複数の表現オブジェクトが参照される場合、レジスタオブジェクトをその「ルート」として有する表現オブジェクトの「ツリー」が生成される。各レジスタオブジェクトから参照される表現ツリーは、共に「表現フォレスト」を形成することになる。

【００１０】

本発明の利点は、いずれかの所与の表現オブジェクトを複数のレジスタに関連付けることができることであり、したがって、異なるいくつかのレジスタによって使用される表現をこれらのレジスタそれぞれに別個に作成して割り当てる必要はなく、一度だけ作成して各レジスタに関連付ければよい。言い換えれば、表現ツリーは、複数のレジスタオブジェクトから参照される表現オブジェクトによって相互にリンクさせることができる。したがって、所与の表現オブジェクトを、表現フォレスト内のいくつかの表現ツリーに共通とすることができる。同じ表現のコピーを複数作成することを避けることにより、本発明は、中間表現を作成するのに必要な時間を削減し、中間表現に占有されるメモリ空間を削減する。

【００１１】

本発明の別の利点は、冗長になる表現を非常に効率的に識別できることである。新しい表現がレジスタオブジェクトに割り当てられたとき、以前にそのレジスタオブジェクトから参照されていた表現は、他のレジスタオブジェクトから参照されていない限り冗長になる。これらの多重参照は、以下に述べる参照カウンティングを使用して検出される。

【００１２】

いずれかの所与の表現オブジェクトは、それから他の表現オブジェクトへの参照と、他の表現オブジェクト又は抽象レジスタからそれへの参照とを有することができる。各表現

オブジェクトにつながる参照の数のカウントが維持されることが好ましい。表現オブジェクトへの（レジスタ又は別の表現オブジェクトからの）参照が作成又は除去されるたびに、その表現オブジェクトに対するカウントが調整される。所与の表現オブジェクトに対するカウントが０であるのは、その表現オブジェクトにつながる参照がなく、したがってその表現オブジェクトが冗長であることを示す。所与の表現オブジェクトに対するカウントが０であるとき、その表現オブジェクトは、中間表現から除去されることが好ましい。

【００１３】

ある表現オブジェクトが除去されるとき、その表現オブジェクトからつながるすべての参照が削除されることにより、参照先である各表現オブジェクトはその参照カウントをデクリメントする。このデクリメントされた値が０に達したとき、今度はその参照先オブジェクトを除去することができ、それにより、そのオブジェクトの参照先であるオブジェクトがそれらの参照カウントをデクリメントする。

【００１４】

したがって、本発明の中間表現によれば、効率的に冗長コードを突き止めて除去することができる。バイナリ変換プログラムでは、レジスタの内容が定義された後に一度も使用されることなく再定義されるとき、冗長コードが頻繁に現れる。知られている既存の中間表現は、所与のレジスタの内容がいつ定義されたか、及びそのレジスタの内容がいつ使用されたかを示す記録を保持することを必要とする。この記録保持は、非効率的な冗長コード識別方法である。本発明では、冗長コードは、レジスタオブジェクトへの割当ての順序及びレジスタオブジェクトの使用から即座に明らかとなる。

【００１５】

本発明の第２の態様によれば、プログラム可能なマシン上で実行するために書かれたコンピュータコードの中間表現を生成する方法が提供され、前記方法は、

（ｉ）プログラムコードによって生成されることになる可変値を保持するための複数のレジスタオブジェクトを生成すること、ならびに

（ｉｉ）固定値、及び／又は前記固定値と前記プログラムコードに従った前記可変値との関係を表す、複数の表現オブジェクトを生成することを含み、

少なくとも１つの可変サイズのレジスタが複数のレジスタオブジェクトによって表され、可変サイズのレジスタの可能な各サイズにつき１つのレジスタオブジェクトが提供される。

【００１６】

この本発明の第２の態様によれば、可変サイズのレジスタを少なくとも１つ備えるサブジェクトプロセッサの命令セットで表されたプログラムコードの中間表現を生成する方法が提供され、この方法は、

１つの又はそれぞれの可変サイズのプロセッサレジスタを表す、関連する抽象レジスタオブジェクトのセットを生成するコンピュータ実施ステップであって、このセットが可変サイズの各レジスタの可能な各幅につき１つの抽象レジスタを含むコンピュータ実施ステップと、

可変サイズのレジスタへのあるフィールド幅の書込み動作ごとに、同じ幅の抽象レジスタに書き込むコンピュータ実施ステップと、

どの抽象レジスタが有効データを含むかの記録を保持するコンピュータ実施ステップであって、この記録が書込み動作ごとに更新されるコンピュータ実施ステップと、

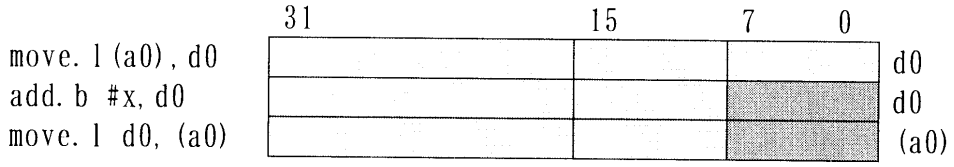
有効データが前記異なるサイズの抽象レジスタのセットのうちの１つ以上あってそれらを可変サイズのレジスタに対して行われる同じ読取り動作と同じ効果をもたらすように結合しなければならないかどうかを、所与のフィールド幅の書込み動作ごとに前記記録から判定するコンピュータ実施ステップと、

ａ）そのように結合が必要とされないと判定された場合は、適切なレジスタから直接に読み取り、あるいは

ｂ）複数のレジスタからのデータをそのように結合しなければならないと判定された場合は、それらのレジスタの内容を結合するコンピュータ実施ステップを含む。

【００１７】
上記において可変サイズのレジスタとは、サブフィールドに値を書き込み、レジスタの幅全体の一部をオーバーレイすることによって、その内容を修正することができるレジスタを意味する。
【００１８】
複数のレジスタからデータを結合しなければならないか否か、また結合しなければならない場合にはどれを結合しなければならないかは、異なるサイズの抽象レジスタの各セットに関して以下の条件に従って決定することができる。
ｉ）アクセスに必要とされるデータが１つの有効抽象レジスタ内に完全にある場合は、そのレジスタだけがアクセスされる。
ｉｉ）アクセスに必要とされるデータが複数の有効抽象レジスタ内にある場合は、それらの有効抽象レジスタからデータが結合されて、アクセスが行われる。
【００１９】
例えば、Ｍｏｔｏｌｏｒａ６８０００シリーズを含めた周知のサブジェクトプロセッサでは、以下のとき、上記のステップ（ｉ）に従ってただ１つのレジスタにアクセスすることが必要となる。すなわち、
ａ）前記抽象レジスタのうち１つだけに有効データがあるときは、そのレジスタがアクセスされる。
ｂ）アクセスの幅に対応するサイズのレジスタ中に有効データがあり、それよりも小さいどのレジスタ中にも有効データがない場合は、そのアクセスの幅にサイズが対応するそのレジスタだけがアクセスされる。
ｃ）有効データを含むレジスタが、アクセスの幅に対応するサイズのレジスタよりも長い場合は、有効データを含むレジスタのうち最小のレジスタがアクセスされる。
【００２０】
周知のサブジェクトプロセッサではまた、アクセスに必要とされるデータが複数の有効抽象レジスタ内にあり、したがって２つ又はそれ以上のレジスタからのデータを結合しなければならない場合、以下のようにして結合を行うことができる。
ａ）読取り動作の幅に対応するか又はそれよりも小さいサイズの２つ又はそれ以上のレジスタ中に有効データがある場合は、これらの各レジスタからのデータが結合される。
ｂ）読取り動作のサイズに対応するサイズのレジスタ中にデータがなく、より大きいレジスタ及びより小さいレジスタ中にデータがある場合は、これらの各レジスタからのデータが結合される。
【００２１】
中間表現が、すべてのレジスタアクセスが同じ幅であるプログラムの領域（１つ又は複数のＢａｓｉｃ　Ｂｌｏｃｋを含む）を表すときは、抽象レジスタの内容を結合する必要はなく、データは単に、単一の動作で単一の抽象レジスタに書き込むか又はそれから読み取ることができる。したがって、ターゲットプロセッサコードが単純化されることになる。２つの抽象レジスタの内容を結合するより複雑なプロシージャは、いずれかの特定のコード領域が、異なるビット幅のレジスタアクセスを含む場合にのみ必要となる。
【００２２】
この本発明の第２の態様は、プロセッサのエミュレーション中、具体的には、エミュレートされるプロセッサが可変サイズのレジスタを利用するときに起こる問題を克服する。対象とするこの問題の性質は、例によって最もよく理解される。　可変サイズのレジスタを使用する命令セットの一例は、Ｍｏｔｏｒｏｌａ６８０００アーキテクチャである。６８０００アーキテクチャでは、「ロング」（．ｌ）と指定された命令は、レジスタ又はメモリ位置の３２ビットすべてに作用する。「ワード」（．ｗ）又は「バイト」（．ｂ）と指定された命令は、レジスタ又はメモリ位置のボトム１６ビット及びボトム８ビットにしかそれぞれ作用しない。例えば、バイト追加によって繰上がりが生成される場合でも、その繰上がりはレジスタの９番目のビットには伝わらない。可変サイズのレジスタ中で発生する状況は、以下の表１に示す６８０００コードの例に示される。

【００２３】
【表１】



【００２４】
この例において、最初の「ｍｏｖｅ．ｌ」命令は、レジスタアドレス「ｄ０」の３２ビットすべてに書き込む。これを、レジスタ「ｄ０」を表すボックスの全部分をカバーする薄い方の陰影によって上に示す。「ａｄｄ．ｂ」命令は、レジスタ「ｄ０」のボトム８ビットだけに書き込み、トップ２４ビットは「ａｄｄ．ｂ」命令の前の状態と全く同じ状態のままである。レジスタ「ｄ０」の、「ａｄｄ．ｂ」命令によって影響された部分を、濃い方の陰影によって示す。ここで、レジスタ「ｄ０」の全内容が別のレジスタ又はメモリにコピーされる場合、コピーされるボトム８ビットは、「ａｄｄ．ｂ」命令によって生成されたものとなり、コピーされるトップ２４ビットは、「ｍｏｖｅ．ｌ」命令によって生成されたものとなる。

【００２５】
エミュレーション・システムは、エミュレートしているサブジェクトプロセッサによって使用される各レジスタを表さなければならない。エミュレーションの一部としてプログラムの中間表現が作成されるとき、その中間表現は、どんなアーキテクチャのターゲットプロセッサ上でも実行されるコードに変換できることが好ましい。したがって、好ましくは中間表現は、コードを実行するのに使用されるターゲットプロセッサのタイプに関するどんな仮定も含むべきではない。この場合、回避すべき特定の仮定は、上記の例で述べたように８ビットのデータがレジスタに書き込まれるときにターゲットプロセッサ上の３２ビットレジスタの上位２４ビットがそれらの既存の形で維持されるであろうという仮定である。この代わりに、８ビットのデータをレジスタの最下位８ビットに書き込み、次いで残りの２４ビットを０で埋めることになるターゲットプロセッサもあり得る。中間表現は、（適切なコードに変換された後は）どちらの形式のターゲットプロセッサ上でも実行できるような方式で構築することが好ましい。

【００２６】
この問題を克服できる方式の１つは、ターゲットプロセッサレジスタの異なるセクションを適切な方式で操作する複雑な式を作成することであり、この例で必要とされる式は次のようになる。
ｄ０＝（（ｄ０＋ｘ）＆０ｘｆｆ）｜（ｄ０＆０ｘｆｆｆｆｆｆ００）
この式は、ターゲットプロセッサレジスタ上で３２ビット追加を行い、ボトム８ビットを抽出し、次いでトップ２４ビットをそれらの元の値に復元する。

【００２７】
異なる幅のデータを操作する２つの命令間においてある幅のデータを操作する命令を見つける（上に例示した状況）のは普通でない。プログラム中で共にグループ化される同じ幅のデータを操作する命令のグループを見つけることの方が普通である。例えば、プログラムのある領域はバイトのデータ、例えば文字処理コードに作用し、プログラムの別の領域は３２ビット幅のデータ、例えばポインタ操作コードに作用する場合がある。独立型コード領域のそれぞれが単一の幅しかないデータに作用するこれらの優勢なケースでは、特別なアクションを行う必要はない。例えば、プログラムの領域がバイトだけを動かし操作している場合、これらのバイト値をターゲットプロセッサの３２ビットレジスタに記憶することができ、レジスタのトップ２４ビットは、アクセスされることがないので無視され

る。このプログラムが、次いで１６ビット幅データの操作を開始する場合、１６ビット演算に関わるターゲットプロセッサレジスタには、何らかのワード演算が行われる前に１６ビット項目がロードされる可能性が非常に高く、したがって何の対立も発生しない（すなわちデータのトップ１６ビットが無視される）。しかし、バイト値を使用するより早い段階の演算中は、１６又は３２ビットを使用する演算が発生するまで、レジスタのトップ２４ビット（例えば）を保存する必要があるかどうかが分からない。

【００２８】

レジスタ中に保持されたビットのすべて又はいくつかを廃棄してよいかどうかが分からないので、対立するオペランド幅を使用する演算を表すために複雑な式を構築する前述の技術は、あらゆる命令に適用しなければ正しく機能しない。したがって、周知の中間表現で使用されるこの技術は、たまにしか発生しない問題を解決するために大きなオーバーヘッドを課す。

【００２９】

本発明により、前述のようにサブジェクトプロセッサレジスタの可能な各サイズを表すために別個の抽象レジスタを使用することは、１つのデータ幅しか使用しないプログラム領域の間に追加の処理を必要とせずにデータを中間表現中の抽象レジスタに書き込むか又はそれから動かすことが可能になるので有利である。本発明は、サブジェクトプロセッサレジスタに書き込まれ、それから読み取られる、異なる幅のデータを中間表現が表す必要があるというまれにしか起こらない状況で、計算（すなわち異なる幅のデータの結合）を行うことしか必要としない。

【００３０】

本発明の第３の態様は、変換されるコードの量を削減する。サブジェクトコードの特性は以下のとおりである。

ｉ）コードのＢａｓｉｃ　Ｂｌｏｃｋは、代替かつ未使用の入口条件を有することができる。これは、変換を行う時に検出することができる。

ｉｉ）コードのＢａｓｉｃ　Ｂｌｏｃｋは、代替かつ未使用の可能な効果又は機能を有することができる。一般に、これは、変換されたコードが実行されるときにだけ検出可能となる。

本発明の第３の態様によれば、コンピュータプログラムコードの中間表現を生成する方法が提供され、この方法は、

サブジェクトコードの所与の部分を最初に変換する際に、プログラムコードのその部分を優勢な条件セットで実行するのに必要な中間表現だけを生成及び記憶するコンピュータ実施ステップと、

サブジェクトコードの同じ部分が後で入力されたときは常に、サブジェクトコードのその部分のための中間表現が後続の条件に対して以前に生成及び記憶されたかどうかを判定し、そのような中間表現が以前に生成されていない場合は、サブジェクトコードの前記部分を前記後続の条件で実行するのに必要な追加の中間表現を生成するコンピュータ実施ステップとを含む。

【００３１】

本発明の第３の態様は、サブジェクトコードの単一のＢａｓｉｃ　Ｂｌｏｃｋに対して複数だがより単純な中間表現コードのブロックを可能にすることにより、変換されるコードの量を削減する。ほとんどの場合、より単純な変換ブロックが１つだけ必要とされることになる。

【００３２】

本発明によれば、プログラム可能なマシン上で実行するために書かれたコンピュータコードの中間表現を生成する方法が提供され、前記方法は、

（ｉ）プログラムコードによって生成されることになる可変値を保持するための複数のレジスタオブジェクトを生成すること、ならびに

（ｉｉ）固定値、及び／又は前記固定値と前記プログラムコードに従った前記可変値との関係を表す、複数の表現オブジェクトを生成することを含み、前記中間表現は、コンピ

ュータコードのブロックに対して生成及び記憶され、同じコードブロックが後で再入力された場合には後で再利用され、前記第１のコンピュータプログラムコードの少なくとも１ブロックが、代替の未使用入口条件又は効果又は機能を有することができ、前記中間表現は、プログラムコードのそのブロックをその時の優勢な条件で実行するのに必要なとき、最初にだけ生成及び記憶される。

【００３３】

例えば本発明の好ましい実施形態では、この方法は、プログラムによって必要とされるときに、プログラムコードのＢａｓｉｃ　Ｂｌｏｃｋごとの中間表現の中間表現ブロック（ＩＲ　Ｂｌｏｃｋ）を生成するコンピュータ実施ステップであって、各ＩＲ　Ｂｌｏｃｋが特定の入口条件に対するプログラムコードの各Ｂａｓｉｃ　Ｂｌｏｃｋを表すコンピュータ実施ステップと、

各ＩＲ　Ｂｌｏｃｋに対応するターゲットコードを記憶するコンピュータ実施ステップと、

プログラムが所与の入口条件に対するＢａｓｉｃ　Ｂｌｏｃｋの実行を必要とするときに、

ａ）その所与の入口条件に対するそのＢａｓｉｃ　Ｂｌｏｃｋを表す記憶済みターゲットコードがある場合は、前記記憶済みターゲットコードを使用し、あるいは

ｂ）その所与の入口条件に対するそのＢａｓｉｃ　Ｂｌｏｃｋを表す記憶済みターゲットコードがない場合は、その所与の入口条件に対するそのＢａｓｉｃ　Ｂｌｏｃｋを表す別のＩＲ　Ｂｌｏｃｋを生成するコンピュータ実施ステップを含む。

【００３４】

Ｂａｓｉｃ　Ｂｌｏｃｋは、サブジェクトプロセッサ中の順次命令のグループ、すなわちサブジェクトコードである。Ｂａｓｉｃ　Ｂｌｏｃｋは、ただ１つのエントリーポイントを有し、別のＢａｓｉｃ　Ｂｌｏｃｋの直前に終了するか、あるいはジャンプ、呼出し、又は分岐（条件付きでも無条件でも）の命令時に終了する。ＩＲ　Ｂｌｏｃｋは、中間表現のブロックであり、サブジェクトコードのＢａｓｉｃ　Ｂｌｏｃｋの変換を表す。同じＢａｓｉｃ　Ｂｌｏｃｋだが異なる入口条件に対するＢａｓｉｃ　Ｂｌｏｃｋを表すＩＲ　Ｂｌｏｃｋのセットが生成された場合、そのセット内のＩＲ　Ｂｌｏｃｋは、以下、ＩｓｏＢｌｏｃｋと呼ぶ。

【００３５】

本発明のこの態様は、静的変換に適用することもできるが、動的バイナリ変換を介したエミュレーションに特に適用可能である。本発明によれば、エミュレーション・システムは、サブジェクトプロセッサプログラムをＢａｓｉｃ　Ｂｌｏｃｋごとに変換するように構成することができる。この手法を使用するときは、プログラムのＢａｓｉｃ　Ｂｌｏｃｋの実行に続く、エミュレートされるプロセッサの状態が、プログラムの後続のＢａｓｉｃ　Ｂｌｏｃｋを表すのに使用されるＩＲ　Ｂｌｏｃｋの形式を決定する。

【００３６】

対照的に、変換を利用する周知のエミュレータでは、Ｂａｓｉｃ　Ｂｌｏｃｋの中間表現が生成されるが、これは、プログラムのそのＢａｓｉｃ　Ｂｌｏｃｋの最初の入口条件から独立している。したがって、中間表現は優勢な形式をとることが必要とされ、例えば抽象レジスタの有効性（又はその反対）を決定するテストを含むことになる。これとは対照的に、本発明では、抽象レジスタの有効性（又はその反対）はすでに分かっており、したがってＩＲ　ｂｌｏｃｋは有効性テストを含む必要はない。さらに、抽象レジスタの有効性がわかっているので、ＩＲ　ｂｌｏｃｋは、有効な抽象レジスタを結合するのに必要なコードだけを含むことになり、すべての抽象レジスタを結合できるコードを含む必要はない。これにより、実行するために中間表現に変換する必要のあるコードの量が削減されるので、大きな性能の利点がもたらされる。プログラムのＢａｓｉｃ　Ｂｌｏｃｋが以前に所与の入口条件のセットに対する中間表現に変換されており、かつ、それが異なる入口条件で開始する場合、プログラムのそのＢａｓｉｃ　Ｂｌｏｃｋが中間表現のＩｓｏＢｌｏｃｋに再変換されることになる。

【００３７】
本発明による第３の態様の別の利点は、得られる中間表現のＩＲ　ｂｌｏｃｋ及びＩｓｏＢｌｏｃｋが、すべての入口条件を表すことのできる中間表現よりも複雑でなく、したがってより迅速に最適化でき、また、より迅速に実行されるターゲットプロセッサコードに変換されることである。
【００３８】
本発明の第３の態様はまた、可能な効果又は機能をいくつか有することのできるサブジェクトコード命令を利用し、命令が最初に実行されるとき、これらの効果又は機能のすべてが必要とされるわけではない可能性もあり、これらのいくつかは、実際にはまったく必要とされない可能性もある。本発明のこの態様は、中間表現が動的に生成されるときにだけ使用することができる。
【００３９】
すなわち、本発明によるこの方法は、プログラムの実行中にそのプログラムの中間表現が動的に生成されるときに、
複数の可能な効果又は機能を有する特定のサブジェクトコード命令の第１の反復時に、その反復で必要とされる特定の機能だけを表す特殊ケース中間表現を生成及び記憶するコンピュータ実施ステップと、同じサブジェクトコード命令の後続の各反復で、前記後続の反復で必要とされる機能に対して特殊ケース中間表現が生成されているかどうかを判定し、そのような特殊ケース中間表現が以前に生成されていない場合に、その機能特有の追加の特殊ケース中間表現を生成するコンピュータ実施ステップとを含むことが好ましい。
【００４０】
本発明のこの態様は、エミュレーション・システムに関連付けられた問題、すなわちサブジェクトプロセッサコードの不必要な機能部分の変換という問題を克服する。サブジェクトプロセッサコードから、複雑な命令が中間表現にデコードされるとき、その命令の可能な効果のサブセットだけが、サブジェクトプロセッサプログラム中の所与の位置で使用されるのが普通である。例えばＣＩＳＣ（複雑命令セットコンピュータ）の命令セットにおいては、メモリロード命令は、基準レジスタ内にどのタイプの記述子が含まれているかによって異なった動作をするよう定義されることがある（記述子は情報がどのようにメモリ中に記憶されているかを記述する）。しかしながら大半のプログラムにおいては、そのプログラムの個々のロード命令により１つの記述子タイプだけが使用される。本発明による変換プログラムは、その記述子タイプだけのために定義されたロード命令を含む特殊ケース中間表現を生成する。
【００４１】
好ましくは、特殊ケースの中間表現が生成され記憶されると、関連付けられた試験手順が生成され記憶されてその後各々の対象コード命令が反復される際に、必要とされる機能が、関連付けられ記憶された特殊ケースの中間表現により表される機能と同一であるかを判定し、追加の特殊ケースの中間表現が必要とされる場合は、その特殊ケースの中間表現と関連付けられた追加の試験手順が生成され、その追加の特殊ケース中間表現とともに記憶される。
【００４２】
好ましくは、特定のサブジェクトコード命令及び追加の関連する試験手順についての追加の特殊ケース中間表現は、少なくとも初めは、同一のサブジェクト命令を表すために記憶されている既存の特殊ケース中間表現及び関連する試験手順に対して下位の関係で記憶されるのがよい。その場合、サブジェクトコード命令の２番目の及び後続の反復の際に、前記試験手順をそれが生成され記憶された順番で行うことにより、必要な特殊ケース中間表現がそれ以前に生成されているか否かの判定が行われ、それは必要とされる機能の特殊ケースの中間表現が存在すると判定されるまで、又はそうした必要とされる特殊ケースの中間表現が存在せず、その場合さらなる追加の中間表現及び別の試験手順が生成されるまで行われる。
【００４３】

試験手順をそれが生成される順序で順序付けするのではなく、使用される頻度がより高い特殊ケース中間表現と関連付けられた試験手順が、使用頻度の低い特殊ケース中間表現と関連付けられた試験手順より先に行われるように試験手順の順序付けを調整することにより、中間表現を最適化することが好ましい。

【００４４】

上記方法のいずれにより生成された中間表現を、例えば、第１タイプのプロセッサが実行するために書かれたコンピュータプログラムの変換において使用し、そのプログラムが異なるプロセッサにより実行され、かつコンピュータプログラムの最適化における１つのステップとして実行されるようにする。後者の場合、中間表現は特定のプロセッサが実行するために書かれたコンピュータプログラムを表すように生成することができ、その中間表現は次いで最適化され、次いでその同一のプロセッサにより実行可能なコードに変換し直される。

【００４５】

上記の発明の第３の態様は中間表現の生成に関連するが、その中で説明したステップは、中間表現を生成せずにサブジェクトコードから直接ターゲットコードを生成することにも適用することができる。

【００４６】

したがって、本発明はコンピュータプログラムコードのターゲットコード表現を生成する方法も提供し、この方法は、サブジェクトコードの所与の部分を最初に変換する際、プログラムコードのその部分を、優勢な条件セットで実行するのに必要とされるターゲットコードだけを生成し記憶するステップと、その後サブジェクトコードの同一の部分が入力された時に、そのサブジェクトコードの部分について、ターゲットコードがその後続の条件で以前に生成及び記憶されているかどうかを判定するステップと、そのようなターゲットコードが以前に生成されていない場合は、前記その後続の条件でサブジェクトコードの前記部分を実行するのに必要な追加のターゲットコードを生成する、コンピュータ実施ステップを含む。中間表現の生成に関連して説明した特性及び利点の多くは、ターゲットコードの生成に同様に適用されることが理解されよう。

【００４７】

本発明の第４の態様によると、第１のプログラム可能なマシン上でコンパイル及び／又は変換するため、及び実行するために書かれた第１のコンピュータプログラムコードを、異なる第２のプログラム可能なマシンで実行するために第２のコンピュータプログラムコードに動的に変換する方法が提供される。前記方法は、

（ａ）前記第１のコンピュータプログラムコードのブロックの中間表現を生成することと、

（ｂ）前記中間表現から前記第２のコンピュータプログラムコードのブロックを生成することと、

（ｃ）第２コンピュータプログラムコードの前記ブロックを前記第２のプログラム可能なマシン上で実行することと、

（ｄ）前記第２のプログラム可能なマシン上の第１のコンピュータプログラムコードのエミュレートされる現在の実行のために必要とされる、少なくとも第１のコンピュータプログラムコードのブロックについてステップａ～ｃをリアルタイムで繰り返すこと、

とを含む。

【００４８】

本発明は、コンピュータコードのリアルタイム変換において中間表現を使用することの利点を実現する。動的エミュレーション・システムに適用される本発明の具体的な実施形態を、図面を例としてのみ参照しながら以下に説明する。

【００４９】

図１から図５は、本発明による動的エミュレーション・システムが、プログラム又はプログラムのＢａｓｉｃ　Ｂｌｏｃｋの中間表現を生成する方式の概略図であり、これらの図はまた本発明の新規性のある特徴である表現フォレスト（表現ツリーのグループ）をも

表している。

【００５０】

図６及び図７は、動的エミュレーション・システムがプログラムのＢａｓｉｃ　Ｂｌｏｃｋの中間表現を生成する方式の概略図であり、それはプログラムのそのＢａｓｉｃ　Ｂｌｏｃｋを開始する際の開始条件に依存する。

【００５１】

下記に説明する本発明の実施形態は、異なるタイプのプロセッサ上で、１つのプロセッサの命令セットをエミュレートするためのシステムである。以下の説明において、サブジェクトプロセッサという用語は、エミュレーション・システムによりエミュレートされるべきプロセッサを言い、ターゲットプロセッサとはエミュレートシステムが実行されるプロセッサを言う。このシステムは、基本的に、サブジェクトプロセッサコード中の命令のＢａｓｉｃ　Ｂｌｏｃｋを、実行する必要に応じてターゲットプロセッサコードに変換することにより動作する動的バイナリ変換システムである。下記で説明するこのエミュレーション・システムは、それぞれＦｒｏｎｔ　Ｅｎｄ、Ｃｏｒｅ、Ｂａｃｋ　Ｅｎｄと呼ばれる３つの主要な構成要素を含む。サブジェクトプロセッサ命令は、エミュレーション・システムのＦｒｏｎｔ　Ｅｎｄによりデコードされ中間表現に変換される。エミュレーション・システムのＣｏｒｅはサブジェクトプロセッサ命令の中間表現を分析及び最適化し、Ｂａｃｋ　Ｅｎｄは中間表現を、ターゲットプロセッサ上で実行するターゲットプロセッサコードに変換する。

【００５２】

システムのＦｒｏｎｔ　Ｅｎｄは、エミュレートされているサブジェクトプロセッサに固有のものである。Ｆｒｏｎｔ　Ｅｎｄは、サブジェクトプロセッサの形態に応じてエミュレーション・システムを構成し、例えばエミュレーションにより必要とされるサブジェクトプロセッサレジスタの番号及び名前を指定し、必要とされる仮想メモリマッピングをＢａｃｋ　Ｅｎｄに指定する。

【００５３】

サブジェクトプロセッサ命令はＢａｓｉｃ　Ｂｌｏｃｋにおいて中間表現に変換され、その効果生じた各中間表現ブロック（ＩＲ　Ｂｌｏｃｋ）は次いでエミュレーション、キャッシング（ｃａｃｈｉｎｇ）、最適化の目的でＣｏｒｅによりユニットとして扱われる。

【００５４】

ＣｏｒｅはＦｒｏｎｔ　Ｅｎｄが生成した中間表現を最適化する。Ｃｏｒｅは、エミュレーション・システムに接続されたサブジェクトプロセッサ及びターゲットプロセッサとは無関係の標準形を有する。ただし、いくつかのＣｏｒｅ資源、具体的にはレジスタ番号、名前付け、ＩＲ　Ｂｌｏｃｋの詳細な性質は個々のＦｒｏｎｔ　Ｅｎｄにより構成されてその固有のサブジェクトプロセッサ構造要件に適応する。

【００５５】

Ｂａｃｋ　Ｅｎｄはターゲットプロセッサに固有のもので、中間表現をターゲットプロセッサ命令に変換するためにＣｏｒｅにより起動される。Ｂａｃｋ　Ｅｎｄは、ターゲットプロセッサレジスタを割当てし管理することと、サブジェクトプロセッサを正確にエミュレートするために適切なメモリロード及び記憶命令を生成することと、Ｃｏｒｅが動的ルーチンを呼び出し、それらの動的ルーチンがＢａｃｋ　Ｅｎｄ及びＦｒｏｎｔ　Ｅｎｄを適切に呼び出せるようにするために呼び出し手順を実施することに責任を負う。

【００５６】

次いでエミュレーション・システムの動作をより詳細に説明する。システムは初期化されるとＦｒｏｎｔ　Ｅｎｄ、Ｃｏｒｅ、Ｂａｃｋ　Ｅｎｄの間に適切なリンクを生成する。初期化の終了時に命令実行ステップが開始され、Ｃｏｒｅはｆｒｏｎｔ　Ｅｎｄを呼び出してサブジェクトプロセッサ命令の第１のＢａｓｉｃ　Ｂｌｏｃｋをデコードする。Ｆｒｏｎｔ　Ｅｎｄは命令ごとに動作し、Ｂａｓｉｃ　Ｂｌｏｃｋの各サブジェクトプロセッサ命令を順にデコードし、Ｃｏｒｅルーチンを呼び出して各命令のサブ操作ごとに中間

表現を生成する。Ｆｒｏｎｔ　Ｅｎｄが、プログラムシーケンスの変更を生じさせる可能性がある命令（例えば条件付又は無条件の、ジャンプ、呼び出し又は分岐命令）をデコードするとき、Ｆｒｏｎｔ　Ｅｎｄはさらなるサブジェクトプロセッサ命令をデコードする前にＣｏｒｅに戻る（それによりコードのＢａｓｉｃ　Ｂｌｏｃｋを終了する）。

【００５７】

Ｆｒｏｎｔ　Ｅｎｄがサブジェクトプロセッサ命令のＢａｓｉｃ　Ｂｌｏｃｋを中間表現に変換すると、Ｃｏｒｅはその中間表現を最適化し、次いでＢａｃｋ　Ｅｎｄを起動して、ターゲットプロセッサコード（ターゲット命令）内に、Ｂａｓｉｃ　Ｂｌｏｃｋの中間表現を実施する命令のシーケンスを動的に生成する。ターゲット命令のそのシーケンスが生成されるとそれは直ちに実行される。ターゲットプロセッサ命令のシーケンスは、後の再使用のためにキャッシュ内で保持される（最初に上書きされる場合を除いて）。

【００５８】

ターゲットプロセッサ命令が実行されると、次に実行されるべきアドレスを示す値が戻される。言い換えれば、ターゲットプロセッサコードは、Ｂａｓｉｃ　Ｂｌｏｃｋの終わりに、条件付き又は無条件に関わらずどの分岐、呼び出し、ジャンプ命令をも評価しその効果を戻す。Ｂａｓｉｃ　Ｂｌｏｃｋのこの変換及び実行のプロセスは、すでに変換されたＢａｓｉｃ　Ｂｌｏｃｋに出会うまで続行する。

【００５９】

次のＢａｓｉｃ　Ｂｌｏｃｋを表すターゲットコードが以前に使用されておりキャッシュ内に記憶されているとき、Ｃｏｒｅは単にそのターゲットコードを呼び出す。Ｂａｓｉｃ　Ｂｌｏｃｋの終わりに達したとき、ターゲットコードは実行されるべき次のサブジェクト命令のアドレスを再度供給し、サイクルは続行する。

【００６０】

中間表現及びターゲットプロセッサコードはどちらもサブジェクトプロセッサ命令のＢａｓｉｃ　Ｂｌｏｃｋとリンクされている。中間表現はオプティマイザが頻繁に実行されるＩＲ　Ｂｌｏｃｋのグループの効果的なエミュレーションを生成できるようにリンクされ、ターゲットコードは、同一のＢａｓｉｃ　Ｂｌｏｃｋの第２及び後続の実行がターゲットコードを直接実行でき、命令を再度デコードするというオーバーヘッドを生じさせないようにリンクされる。

【００６１】

Ｆｒｏｎｔ　Ｅｎｄは、抽象レジスタの必要とされる番号がＣｏｒｅにおいて初期化時に定義されることを要求する。これらの抽象レジスタ（Ｒｉとラベルされる）は、サブジェクトプロセッサ上で実行する場合にサブジェクトプロセッサ命令により使用されることになる物理レジスタを表す。抽象レジスタは、サブジェクトプロセッサレジスタで命令の予測される効果を表すことにより、エミュレートされているサブジェクトプロセッサの状態を定義する。

【００６２】

中間表現は、表現オブジェクトを抽象レジスタに割り当てることにより、サブジェクトプロセッサプログラムを表す。表現オブジェクトは、例えば個々の演算操作、論理演算、条件付演算の効果を中間表現において表す手段である。多数のサブジェクトプロセッサ命令はデータの操作を実行するので、大半の命令はその個々のサブ操作を表すために表現オブジェクトを生成する。表現オブジェクトは、例えば追加操作、条件設定操作、条件付き分岐における条件付き評価、メモリ読出し操作を表すために使用される。抽象レジスタは表現オブジェクトに関係付けられ、表現オブジェクトは他の表現オブジェクトに関係付けられて、サブジェクトプロセッサ命令の各Ｂａｓｉｃ　Ｂｌｏｃｋが、表現フォレストと考えられる相互参照された表現オブジェクトの数によって表されるようにする。

【００６３】

一連の図示した例を使用して、サブジェクトプロセッサ命令の中間表現を作成するために、エミュレーション・システムがどのように表現オブジェクト（Ｅｘｐｒｅｓｓｉｏｎと呼ばれる）及び抽象レジスタを使用するのかを伝える。図１から図５は、抽象レジスタ

を使用して下記の擬似アセンブラコードがＣｏｒｅ内でどのように表されるかを段階的に示している。

【００６４】

【表２】

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

【００６５】

ライン１のＭＯＶＥ命令の表現が図１に示される；Ｌｏｎｇ　Ｃｏｎｓｔａｎｔ　Ｅｘｐｒｅｓｓｉｏｎ＃３が生成され、Ｒ０から＃３に導く参照を生成することにより抽象ＲｅｇｉｓｔｅｒＲ０に割当てられる。ライン２のＭＯＶＥ命令は抽象レジスタＲ６の値を参照し、Ｒｅｇｉｓｔｅｒ　Ｒｅｆｅｒｅｎｃｅ　Ｅｘｐｒｅｓｓｉｏｎはこれを表すために使用され、Ｒ２に割当てられる。図１のＲｅｇｉｓｔｅｒ　Ｒｅｆｅｒｅｎｃｅ（ＲｅｇＲｅｆ）Ｅｘｐｒｅｓｓｉｏｎ@Ｒ６は、それがいずれの値であれＲｅｇｉｓｔｅｒＲ６の値を表す。ＲｅｇＲｅｆＥｘｐｒｅｓｓｉｏｎ@６は、ＲｅｇｉｓｔｅｒＲ６の現在の定義になる。この時点以降、ＲｅｇｉｓｔｅｒＲ６が再定義されない限り、それはＥｘされｐｒｅｓｓｉｏｎ@６をその定義として戻す。

【００６６】

サブジェクトプロセッサ命令のオペランドは、定数又はＲｅｇｉｓｔｅｒへの参照のいずれかである。定数オペランドの表現は図１に示したように単純である。ただし、オペランドがレジスタを参照するときは状況が異なる。擬似アセンブラコードのライン３の表現は図２に示、ＡＤＤ演算は、Ｒ１からＡｄｄＥｘｐｒｅｓｓｉｏｎへの参照により、抽象レジスタＲ１に割当てられていることがそこから分かる。ライン３のＡＤＤ命令はレジスタＲ０及びＲ２を参照し、これらの各レジスタを定義するＥｘｐｒｅｓｓｉｏｎはすでに中間表現に構築されている。ＡｄｄＥｘｐｒｅｓｓｉｏｎが生成されると、それは抽象ＲｅｇｉｓｔｅｒＲ０及びＲ２に問合せを行いそれらを定義するＥｘｐｒｅｓｓｉｏｎを生じ、Ａｄｄ　Ｅｘｐｒｅｓｓｉｏｎ（抽象レジスタＲ１に割当てられている）はこれに対する参照を行う。ＡＤＤ命令の中間表現を図２に示す。言い換えれば、抽象ＲｅｇｉｓｔｅｒＲ１の内容は、抽象ＲｅｇｉｓｔｅｒＲ０及びＲ２に保持されているＥｘｐｒｅｓｓｉｏｎを参照するＥｘｐｒｅｓｓｉｏｎである。図１及び図２中の各矢印は参照を表しており、それはＲ０→＃３の場合にＲｅｇｉｓｔｅｒをＥｘｐｒｅｓｓｉｏｎに関係付けるか、又は＃３←＋→@Ｒ６の場合にＥｘｐｒｅｓｓｉｏｎを別のＥｘｐｒｅｓｓｉｏｎに関係付けることができる。Ｅｘｐｒｅｓｓｉｏｎ@Ｒ６は、ＲｅｇｉｓｔｅｒＲ２からの参照及びＡｄｄ　Ｅｘｐｒｅｓｓｉｏｎからのもう一方の参照の２つの参照を有する。

【００６７】

上記コードのライン４に含まれるＭＵＬ命令は、典型的なデータフロー命令と見てよい。トップレベルのＥｘｐｒｅｓｓｉｏｎは新しいサブＥｘｐｒｅｓｓｉｏｎを生成するか又は現存のＥｘｐｒｅｓｓｉｏｎを参照することにより構築され、このトップレベルのＥｘｐｒｅｓｓｉｏｎはその定義としてＲｅｇｉｓｔｅｒに割当てられる。ＭＵＬ命令の中間表現を図３に示す。抽象ＲｅｇｉｓｔｅｒＲ１に保持されるＥｘｐｒｅｓｓｉｏｎを参照し、Ｌｏｎｇ　Ｃｏｎｓｔａｎｔ　Ｅｘｐｒｅｓｓｉｏｎ＃５を参照するＭｕｌ　Ｅｘ

ｐｒｅｓｓｉｏｎは、生成され抽象ＲｅｇｉｓｔｅｒＲ５に割当てられる。

【００６８】

上記コードのＡｎｄ　Ｅｘｐｒｅｓｓｉｏｎを図４に示す。このＥｘｐｒｅｓｓｉｏｎは、図１に関連して上記で説明したのと同様の方法でＲｅｇＲｅｆＥｘｐｒｅｓｓｉｏｎを使用して、定義がまだ構築されていないＲｅｇｉｓｔｅｒ（すなわちＲ３）を参照する。

【００６９】

ここまでに示した例において、Ｒｅｇｉｓｔｅｒは特定のＢａｓｉｃ　Ｂｌｏｃｋ内で最初に定義されることが想定される。図５は、すでに定義されているＲｅｇｉｓｔｅｒが、上記コードのライン６のＭＯＶＥ命令により再定義されるとき何が起こるかを示している。一方図２から図４では、矢印がＲ１をＡｄｄ　Ｅｘｐｒｅｓｓｉｏｎに関係付けているが、この参照はここで消去され、Ｒ１をＬｏｎｇ　Ｃｏｎｓｔａｎｔ　Ｅｘｐｒｅｓｓｉｏｎ＃５に関係付けるために新しい参照矢印を作成する。

【００７０】

Ｒ１に結合されているのと同様に、Ａｄｄ　ＥｘｐｒｅｓｓｉｏｎはＭｕｌ　Ｅｘｐｒｅｓｓｉｏｎ及びＡｎｄ　Ｅｘｐｒｅｓｓｉｏｎにも結合されており、したがって図５に示すように存在し続ける（ただしＡｄｄ　Ｅｘｐｒｅｓｓｉｏｎが、ＲｅｇｉｓｔｅｒＲ１からの参照１つだけを有する場合、Ｒ１が再定義された後ＡｄｄＥｘｐｒｅｓｓｉｏｎには参照が残されない；この場合このＡｄｄ　Ｅｘｐｒｅｓｓｉｏｎは「死んだ」ものとして知られることになり、冗長になる）。さらに、図５は、擬似アセンブラコードのライン７のＳＵＢ演算の効果を図示する。

【００７１】

中間表現として表されるべき擬似アセンブラコードの最後のラインであるライン８はＬＯＡＤ命令である。この命令を表すＬｏａｄ　Ｅｘｐｒｅｓｓｉｏｎは図５にＲｅｇｉｓｔｅｒＲ０に関係付けられて示されている。Ｌｏａｄ　Ｅｘｐｒｅｓｓｉｏｎは、ＬＯＡＤ演算をその単一のＥｘｐｒｅｓｓｉｏｎオペランドに適用した効果を表す、単項演算子のタイプとして考えることができる。図５で、ＬＯＡＤ→＃３ｆｄ０は、それがどのような値であれメモリ位置３ｆｄ０における値を表す。メモリ内に記憶されているデータに応じて１つのＬｏａｄ　Ｅｘｐｒｅｓｓｉｏｎがどの可能な値でも表すという点において、Ｌｏａｄ　ＥｘｐｒｅｓｓｉｏｎはＲｅｇＲｅｆＥｘｐｒｅｓｓｉｏｎと類似の特性を有する。

【００７２】

各表現オブジェクトにつながる参照の数を示す参照カウントは保持される（任意の所与の表現オブジェクトの参照カウントは、その表現オブジェクトからの参照は含まない）。表現オブジェクトに参照が行われる（レジスタ又は別の表現オブジェクトから）たびに、又は参照がその表現から除去されるたびに、その表現オブジェクトに対する参照カウントは調節される。所与の表現オブジェクトに対するゼロの参照カウントは、その表現オブジェクトにつながる参照がなく、したがってその表現オブジェクトが冗長であることを示す。所与の表現オブジェクトに対する参照カウントがゼロの時、その表現オブジェクトは中間表現から除去される。

【００７３】

表現オブジェクトが除去されると、その表現オブジェクトからつながるいずれの参照、及びその参照がつながる先の表現オブジェクトの参照カウントはそれに従って調整される。ゼロ参照カウントの表現オブジェクトを除去するプロセス、及びそのようなオブジェクトからつながる参照を除去するプロセスは、表現フォレストに沿ってたどられる。中間般化のさらなる最適化は、下記で説明するようにサブジェクトプロセッサコードの冗長なラインを除去することにより達成することができる。

【００７４】

複雑な命令がサブジェクトプロセッサコードから中間表現にデコードされるとき、その命令の可能な効果のサブセットだけがサブジェクトプログラムの所与の場所で使用される

のが通常である。例えばＣＩＳＣ命令セットにおいて、メモリロード命令は、基準レジスタに含まれている記述子のタイプに応じて異なった動作をするように定義することができる（記述子は情報がどのようにメモリ内に記憶されているかを記述する）。ただし、大半のプログラムではプログラム中の個々のロード命令により１つの記述子タイプしか使用されない。

【００７５】

本発明のエミュレーション・システムでは、サブジェクトプロセッサプログラムが実行されるとＦｒｏｎｔ　Ｅｎｄは実行時間値を問い合せ、必要に応じて特殊ケースの中間表現を生成する。上記の例では、プログラムが使用しない記述子タイプに関連する、メモリロード命令のその部分を省略する、特殊ケースの中間表現が生成されることになる。

【００７６】

特殊ケースはテストにより保護されており、このテストは追加機能が必要とされていることを実行時に検出した場合に、追加コードを生成するためにＦｒｏｎｔ　Ｅｎｄへの再エントリを生じさせる。最適化中に、当初の推定が誤りであることが発見された場合（例えば特定の記述子タイプがプログラムを通じて使用されるという推定）、オプティマイザはそのテストの方向を反転し、使用頻度の高い機能が、最初に選択された使用頻度の低い機能より迅速に選択されるようにする。

【００７７】

本発明のエミュレーション・システムは、下記で説明するように可変サイズのレジスタを使用するサブジェクトプロセッサをエミュレートすることができる。可変サイズのレジスタを使用する、命令セット構造の例は、Ｍｏｔｏｒｏｌａ６８０００シリーズのプロセッサの構造である。６８０００構造では、「ロング」（．１）と指定された命令はレジスタ又はメモリ位置の３２ビットすべての上で動作する。「ワード」（．ｗ）又は「バイト」（．ｂ）と指定された命令はそれぞれ、３２ビットのレジスタ又はメモリ位置のボトム１６ビット及びボトム８ビットでのみ動作する。例えばバイト追加が繰り上がりを生成する場合でも、その繰り上がりはレジスタの９番目のビットには伝搬されない。

【００７８】

異なる幅のデータで動作する（この例では６８０００プロセッサにおいて）異なる命令間の対立を回避するために、本発明によるシステムは、サブジェクトプロセッサレジスタごとに、３つの抽象レジスタのセットを生成し、このセットの各レジスタは所与の幅のデータ専用になっている（すなわち、バイトデータ、ワードデータ、ロングワードデータごとに１つのレジスタ）。６８０００プロセッサの各レジスタは３２ビットのデータを常に記憶し、一方で命令はこの３２ビットのデータの８ビット又は１６ビットのサブセット上で動作することができる。そのＦｒｏｎｔ　Ｅｎｄが６８０００に結合されるように構成されたシステムＣｏｒｅでは、サブジェクトプロセッサ「ｄ０」についてのバイト値は、例えば「Ｄ０＿Ｂ」とラベルされた抽象レジスタに記憶され、一方でワード値は「Ｄ０＿Ｗ」とラベルされた別の抽象レジスタに記憶され、ロング値は「Ｄ０＿Ｌ」とラベルされた第３の抽象レジスタに記憶される。データレジスタとは対照的に、６８０００のアドレスレジスタは、ワード及びロングの２つの有効なアドレスサイズしか有さない。したがってこの例では、各６８０００アドレスレジスタを表すのにＣｏｒｅが必要とするのは、２つの抽象レジスタ「Ａ０＿Ｌ」及び「Ａ０＿Ｗ」だけである。

【００７９】

サブジェクトプロセッサ命令の特定のＢａｓｉｃ　Ｂｌｏｃｋ内で、命令サイズに関して対立が起きない場合（すなわちそのＢａｓｉｃ　Ｂｌｏｃｋ内の命令のすべてが同一のビット幅である場合）、適切な抽象レジスタに含まれているデータには自由にアクセスすることができる。ただし対立が起こった場合には（すなわち所与のサブジェクトプロセッサレジスタから、異なるビット幅の命令が記憶され／読み出される場合）、２つ又はそれ以上の抽象レジスタの内容を適切な方法で組み合わせることにより、正しいデータを引き出すことができる。この手法の利点は、抽象レジスタ上のすべての演算が３２ビットのデータ項目上で実行されるのでＣｏｒｅが単純化されることである。

【００８０】
サブジェクトプロセッサレジスタと抽象レジスタとの相違は、可変サイズのレジスタの効果を考慮する際に重要である。６８０００構造の「ｄ０」などのサブジェクトプロセッサレジスタは、サブジェクトプロセッサ中の高速記憶域のユニットであり、このユニットはアセンブラのオペランドではそのラベル（この場合は「ｄ０」）により参照される。これとは対照的に、抽象レジスタはＣｏｒｅの中間表現の整数部を形成するオブジェクトであり、サブジェクトプロセッサレジスタのセットを表すために使用される。抽象レジスタは、サブジェクトプロセッサレジスタ中の意味論に加えて余分の意味論を含んでおり、サブジェクトプロセッサとの相互作用に対して正しい意味論が維持されるならば、任意の数の抽象レジスタを使用して単一のサブジェクトプロセッサレジスタを表す。上記で説明したように、本発明では、Ｆｒｏｎｔ　Ｅｎｄは各６８０００データレジスタを表すために３つの抽象レジスタを必要とし（すなわち、バイト、ワード、ロングワードの各データの幅ごとに１つの）、各６８０００アドレスレジスタを表すために２つの抽象レジスタを必要とする。これとは対照的に、例えばＭＩＰＳＦｒｏｎｔ　Ｅｎｄの実施は、単一の抽象レジスタに単一のサブジェクトプロセッサレジスタをマップする。

【００８１】
下記の表は６８０００について、異なるサイズの命令がサブジェクトプロセッサレジスタを読み出し、またこれに書き込む際に、２つ又はそれ以上の抽象レジスタの内容がどのように扱われるかを要約するものである。データが組み合わせられる方式は、サブジェクトプロセッサレジスタの現在の状態によって決まる。

【００８２】
【表３】

現在の状態 State(d0)		
D0_L	D0_W	D0_B
✓	×	×
✓	×	✓
✓	✓	×
✓	✓	✓

表３ａ

書込み後の新状態（d0）								
ロングワード			ワード			バイト		
D0_L	D0_W	D0_B	D0_L	D0_W	D0_B	D0_L	D0_W	D0_B
✓	×	×	✓	✓	×	✓	×	✓
✓	×	×	✓	✓	×	✓	×	✓
✓	×	×	✓	✓	×	✓	✓	✓
✓	×	×	✓	✓	×	✓	✓	✓

表３ｂ

【００８３】
【表４】

現在の状態 State(d0)		
D0_L	D0_W	D0_B
✓	×	×
✓	×	✓
✓	✓	×
✓	✓	✓

表４ａ

読出し前の組合せ		
L	W	B
D0_L	D0_L	D0_L
D0_L/D0_B	D0_L/D0_B	D0_B
D0_L/D0_W	D0_W	D0_W
D0_L/D0_W/D0_B	D0_W/D0_B	D0_B

表４ｂ

【００８４】
表３及び表４は、抽象レジスタＤ０＿Ｌ、Ｄ０＿Ｗ、Ｄ０＿Ｂに関してのサブジェクト

プロセッサレジスタ「ｄ０」の状態を表す（すなわちサブジェクトプロセッサレジスタ「ｄ０」を表す抽象レジスタ）。表３ａの「現在の状態」は、各抽象レジスタＤ０＿Ｌ、Ｄ０＿Ｗ、Ｄ０＿Ｂが有効なデータを含むか含まないかを示すことにより、レジスタｄ０の所与の状態を表す。表３ａの第１行は、レジスタｄ０の所与の状態、すなわちこのレジスタが３２ビットのデータを含んでいることを表し、抽象レジスタＤ０＿Ｌ（３２ビットデータに対応する）だけが有効なデータを含んでいることを示す。例えばサブジェクトプロセッサレジスタ「ｄ０」の３２ビットすべてが有効であると初めに想定される場合、「ｄ０」の現在の状態は、表３ａの第１行により表されるようになる（×の記号は、その印のついたレジスタが有効なデータを含まないことを示す）。

【００８５】

表３ｂの「書込み後の新状態」は、本発明に従って実行された書込み命令の効果を表す。表３ａの第１行で示されるように、ｄ０が３２ビットのデータを含み、長い命令により書込みをされる場合、書込み操作の効果は、表３ｂの「Ｌｏｎｇ　Ｗｏｒｄ」の部分の第１行で示されるものになる。「レ」の記号で示されるように抽象レジスタＤ０＿Ｌは有効なままであり（すなわち有効なデータを含む）、一方で抽象レジスタＤ０＿Ｗ及びＤ０＿Ｂにはデータが書き込まれないので「×」の記号で示されるように無効のままである。したがって「ｄ０」の状態は変更されていない。

【００８６】

表３ａの第１行に示される状態で、「ｄ０」にデータのバイトによって書込みがされる場合、「ｄ０」の新しい現在の状態は表３ｂの「バイト」部分により表される。この場合レジスタはロングデータ及びバイトデータの両方について有効である（すなわち、抽象レジスタＤ０＿Ｌ及びＤ０＿Ｂがどちらも有効データを含む）。

【００８７】

表４ａ「現在の状態」及び表４ｂ「読出し前の組合せ」は、データがサブジェクトプロセッサレジスタ「ｄ０」から読み出される際に、抽象レジスタＤ０＿Ｌ、Ｄ０＿Ｗ、Ｄ０＿Ｂの内容がどのように組み合わされるかを表している。例えばレジスタｄ０の現在の状態が表４ａの第２行に示されるものである場合、抽象レジスタＤ０＿Ｌ及びＤ０＿Ｂは有効データを含んでいる。レジスタｄ０が長い命令により読み出される場合（すなわち３２ビットすべてが「ｄ０」から読み出される場合）、表４ｂの行２は列Ｌで、抽象レジスタＤ０＿Ｌ及びＤ０＿Ｂの内容を適切な方法で組み合わせることにより、「ｄ０」の正しい値を引き出さなければならないことを示している。この場合は、レジスタＤ０＿Ｂのボトムの８ビットをレジスタＤ０＿Ｌのトップ２４ビットと組み合わせなければならない。一方、サブジェクトプロセッサレジスタ「ｄ０」をバイト命令により読み出すべき場合、Ｄ０＿Ｂの内容は抽象レジスタＤ０＿Ｌ又はＤ０＿Ｗを参照せずに直接読み出すことができた。

【００８８】

上記で説明したように、データの幅ごとに別の抽象レジスタを使用することにより、データの単一幅を使用するサブジェクトプロセッサコードのセクションがエミュレートされている時に、データに容易にアクセスすることができるようになる。これは非常に優勢な状況であり、例えばプログラムの１つのセクションがデータのバイト上、例えば文字処理コードで動作し、プログラムの別のセクションが例えばポインタ操作コードの３２ビットのデータ上で動作する場合に生じる。異なる幅のデータがサブジェクトプロセッサレジスタに書き込まれ、またそこから読み出されるというまれな場合に、本発明は行われるべき計算（すなわち異なる幅のデータの組合せ）しか必要としない。

【００８９】

サブジェクトプロセッサレジスタの異なるセクションを適切な方式で操作する、複雑な表現を生成すると知られている技術は、サブジェクトプロセッサレジスタの読出し及び／又はそこへの書込みごとに、計算を行うことを必要とする。これとは対照的に本発明はまれな場合に計算を必要とし、それによりサブジェクトプロセッサレジスタのより効率的な表現を提供する。

【００９０】
本発明は、各サブジェクトプロセッサレジスタの明白な現在の状態（すなわち、構成要素抽象レジスタの有効性又は類似のもの（ｏｔｈｅｒｗｉｓｅ））が常に知られていることを必要とし、それによりそれらの抽象レジスタが表すサブジェクトプロセッサレジスタに対して読出し命令が行われたときに、抽象レジスタの正しい組合せが行われるようにする。

【００９１】
Ｂａｓｉｃ　Ｂｌｏｃｋ２に入力する際のサブジェクトプロセッサレジスタの初期状態が変換時に知られていない場合、レジスタの状態をテストするためのターゲットプロセッサコードを生成する必要がある。この理由により、本発明によるエミュレーション・システムは、各サブジェクトプロセッサレジスタの状態が変換時に常に知られていることを確実にする。本発明によるシステムにおいてこれは、１つのＩｎｔｅｒｍｅｄｉａｔｅ　Ｒｅｐｒｅｓｅｎｔａｔｉｏｎ（ＩＲ）Ｂｌｏｃｋからのレジスタ状態を次に伝搬することにより行われる。例えばＩＲ　Ｂｌｏｃｋ１は「ｄ０」の状態をその後継のＩＲ　Ｂｌｏｃｋ２に伝搬し、ＩＲ　Ｂｌｏｃｋ２は同様の方法でレジスタ状態をＩＲ　Ｂｌｏｃｋ３に伝搬する。サブジェクトプロセッサレジスタのこの伝搬の例は、図６に示される。

【００９２】
図６で、ＩＲ　Ｂｌｏｃｋ２は、ＩＲ　Ｂｌｏｃｋ３又はＩＲ　Ｂｌｏｃｋ２の始めという、可能性のある後継を２つ有している。ＩＲ　Ｂｌｏｃｋ２とＩＲ　Ｂｌｏｃｋ３の間のルートは、「ａ」の印の矢印で表される。エンドバックからＩＲ　Ｂｌｏｃｋ２の始めへのルートは、「ｂ」の印の点線で示されている（このルートは存在するものの、変換されたプログラムの現在の実行においてはまだトラバースされていないので点線を使用している）。変換されたプログラムの実行中に、ＩＲ　Ｂｌｏｃｋ２がルート「ｂ」を通ってそれ自体にブランチバックした場合、それが伝搬する状態は、ＩＲ　Ｂｌｏｃｋ１により初めにＩＲ　Ｂｌｏｃｋ２に伝えられた抽象レジスタの状態とは整合しないものになる。中間表現は抽象レジスタの状態に固有であることから、ＩＲ　Ｂｌｏｃｋ２を再実行することはできない。ＩＲ　Ｂｌｏｃｋの境界を越えて本発明を正しく操作するために、各ＩＲ　Ｂｌｏｃｋはサブジェクトプロセッサレジスタの現在の状態の明白な表現（抽象レジスタにより表現される）を有していなければならない。したがって、ルート「ｂ」の存在は、ＩＲ　Ｂｌｏｃｋ及びＩＲ　Ｂｌｏｃｋ２の境界を超える本発明の動作とは整合しない。

【００９３】
この問題を克服するために、本発明は、異なる入口条件の複数のＩＲ　Ｂｌｏｃｋを使用して、サブジェクトプロセッサコードのＢａｓｉｃ　Ｂｌｏｃｋを表すことができる。異なる入口条件の単一のＢａｓｉｃ　Ｂｌｏｃｋを表すために使用されるＩＲ　Ｂｌｏｃｋは、ＩｓｏＢｌｏｃｋと呼ばれる。各ＩｓｏＢｌｏｃｋはサブジェクトプロセッサコードの、同一ではあるが入口条件が異なるＢａｓｉｃ　Ｂｌｏｃｋを表す。図７には、図６に図示した問題を克服するために使用される２つのＩｓｏＢｌｏｃｋを示す。Ｉｓｏ　Ｂｌｏｃｋ２ａは、Ｂａｓｉｃ　Ｂｌｏｃｋ２の正確な表現であるが、これはＩＲ　Ｂｌｏｃｋ２の始めのサブジェクトプロセッサレジスタ「ｄ０」の状態がレ××の場合に限る（これは図６のＩＲ　Ｂｌｏｃｋ２に対応する）。図７の後継ルート「ｂ」が最初にトラバースされるとき、Ｂａｓｉｃ　Ｂｌｏｃｋ２（この場合１つのＩＲ　Ｂｌｏｃｋしかない）を表すすべての現存のＩｓｏＢｌｏｃｋは、伝搬されるべき抽象レジスタ状態（すなわちレレ×）との整合性をテストされる。整合性のあるＩｓｏＢｌｏｃｋが見つかった場合（すなわちレジスタ状態レレ×で始まるもの）、後継ルート「ｂ」は常にそのＩｓｏＢｌｏｃｋと結合されることになる。図７に示した例では、ルート「ｂ」と整合性のあるＩｓｏ　Ｂｌｏｃｋは存在せず、したがって新しいＩｓｏＢｌｏｃｋ２ｂを生成しなければならない。ＩｓｏＢｌｏｃｋ２ｂは、Ｂａｓｉｃ　Ｂｌｏｃｋ２を構成しているサブジェクトプロセッサ命令を２回目にデコードし、Ｂａｓｉｃ　Ｂｌｏｃｋ２の開始におけるサブジェクトプロセッサレジスタ「ｄ０」の状態がレレ×であるとの当初の推定を使用するこ

とにより生成される。

【００９４】

ＩｓｏＢｌｏｃｋ２ｂから発している後継ルート「ｃ」が最初にトラバースされる際、ＩＲ　Ｂｌｏｃｋ３に整合性テストが行われる。ルート「ｃ」はＩＲ　Ｂｌｏｃｋ３と整合性があるので、新しいＩｓｏＢｌｏｃｋを生成する必要はなく、後継ルート「ａ」及び後継ルート「ｃ」はともにＩＲ　Ｂｌｏｃｋ３に結合される。

【００９５】

上記で述べた整合性テストに関する低レベルの詳細は、それがサブジェクトプロセッサ構造で提供される重複するレジスタの性質そのものに依存することから、異なるＦｒｏｎｔ　Ｅｎｄモジュール間で異なったものになる。これらの詳細の必要な修正点は、当分野の技術者には明らかであろう。

【００９６】

抽象レジスタの状態の所与のセットに対して中間表現のＩｓｏＢｌｏｃｋを入力時に生成することの原理は、幅広いセットの初期状態の固有の値についてのサブジェクトプロセッサコードのＢａｓｉｃ　Ｂｌｏｃｋを表す中間表現に広げることができる。知られている中間表現は、可能性のあるすべての初期開始条件についての命令のブロックを表し、したがってかなりの量の柔軟性を含むことを必要とされる。この方式で形成された中間表現は必然的に複雑になり、一般に実行中には使用されない要素を含むことになる。

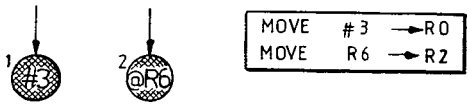
【００９７】

本発明による中間表現は、入口条件の固有の値についてのコードのＢａｓｉｃ　Ｂｌｏｃｋを表し、したがって知られている中間表現よりもコンパクトであるので有利である。本発明のさらなる利点は、生成されるすべての中間表現が少なくとも１回は使用され、不必要な追加表現を生成することに時間が浪費されないことである。

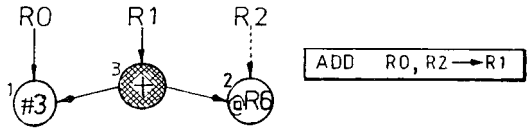
【００９８】

上記の説明はエミュレーションに向けられるものであるが、当分野の技術者は、本発明は例えばコンパイル中のコードの最適化など他のアプリケーションにも使用できることを理解されよう。

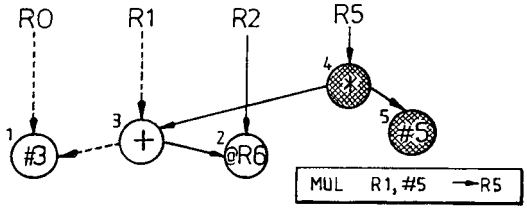
【図１】



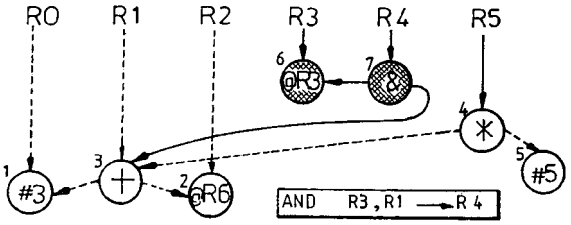
【図２】



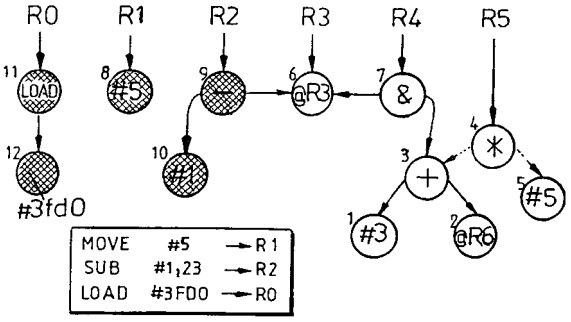
【図３】



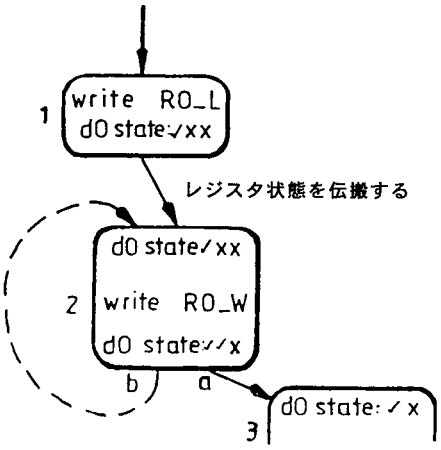
【図４】



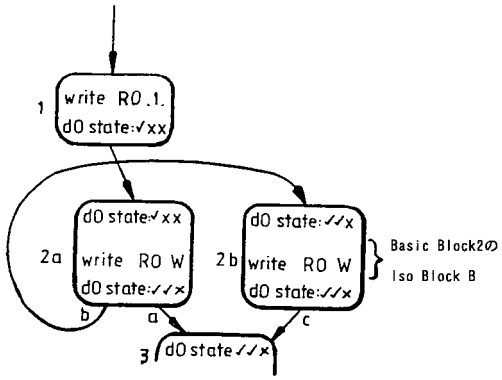
【図５】



【図６】



【図７】



【手続補正書】
【提出日】平成22年4月16日(2010.4.16)
【手続補正１】
【補正対象書類名】特許請求の範囲
【補正対象項目名】全文
【補正方法】変更
【補正の内容】
【特許請求の範囲】
【請求項１】
コンピュータシステムであって、
第１のプロセッサと、
第２のプロセッサのために書かれたプログラムコードを前記第１のプロセッサ上で前記プログラムコードの中間表現をメモリ内に生成することを介して実行可能なエミュレーション・システムであって、
前記中間表現を生成することは、
プログラムコードの第１の部分の最初の変換において、プログラムコードの前記第１の部分を優勢な条件セット下で実行するのに必要とされる中間表現だけを生成及び記憶すること、
プログラムコードの前記第１の部分と同じ部分が後で入力されたときは常に、サブジェクトコードの第１の部分のための中間表現が後続の条件に対して以前に生成及び記憶されたかどうかを判定し、そのような中間表現が以前に生成されていない場合は、プログラムコードの前記第１の部分を前記後続の条件で実行するのに必要とされる追加の中間表現を生成すること
を含む、前記コンピュータシステム。
【請求項２】
コンピュータシステムであって、
第１のコンピュータと、
第２のコンピュータのために書かれたプログラムコードを前記第１のコンピュータ上で前記プログラムコードの中間表現をメモリ内に生成することを介して実行可能なエミュレーション・システムであって、
前記中間表現を生成することは、
プログラムコードの第１の部分の最初の変換において、プログラムコードの前記第１の部分を優勢な条件セット下で実行するのに必要とされる中間表現だけを生成及び記憶すること、
プログラムコードの前記第１の部分と同じ部分が後で入力されたときは常に、サブジェクトコードの第１の部分のための中間表現が後続の条件に対して以前に生成及び記憶されたかどうかを判定し、そのような中間表現が以前に生成されていない場合は、プログラムコードの前記第１の部分を前記後続の条件で実行するのに必要とされる追加の中間表現を生成すること
を含む、前記コンピュータシステム。
【請求項３】
前記条件が入口条件である、請求項１又は２に記載のコンピュータシステム。
【請求項４】
抽象レジスタを表す複数のレジスタオブジェクトを生成することであって、個々のレジスタオブジェクトは個々の抽象レジスタを表す、前記生成すること、
複数の表現オブジェクトを生成することであって、前記プログラムコードの要素が前記プログラムコード中に生じる場合に各表現オブジェクトは前記プログラムコードの種々の演算子又はオペランドを表し、各表現オブジェクトは、直接的に又は前記表現オブジェクトのうちの他の表現オブジェクトからの参照を介して間接的に関係する１つのレジスタオブジェクトによって参照される、請求項１～３のいずれか一項に記載のコンピュータシス

テム。
【請求項５】
コンピュータシステムであって、
第１のプロセッサと、
第２のプロセッサのために書かれたプログラムコードを前記第１のプロセッサ上で前記プログラムコードの中間表現をメモリ内に生成することを介して実行可能なエミュレーション・システムであって、
前記中間表現を生成することは、
前記プログラムによって必要とされる場合にプログラムコードのＢａｓｉｃ　Ｂｌｏｃｋごとに中間表現の中間表現ブロック（ＩＲ　Ｂｌｏｃｋ）を生成することであって、各ＩＲ　Ｂｌｏｃｋが特定の入口条件についてのプログラムコードの各Ｂａｓｉｃ　Ｂｌｏｃｋを表す、前記生成すること、
各ＩＲ　Ｂｌｏｃｋに対応するターゲットコードを記憶すること、
プログラムが所与の入口条件についてのＢａｓｉｃ　Ｂｌｏｃｋの実行を必要とする場合に、
ａ）その所与の入口条件についての前記Ｂａｓｉｃ　Ｂｌｏｃｋを表す記憶済みターゲットコードがある場合は、前記記憶済みターゲットコードを使用し、又は
ｂ）その所与の入口条件についての前記Ｂａｓｉｃ　Ｂｌｏｃｋを表す記憶済みターゲットコードがない場合は、前記所与の入口条件についての前記Ｂａｓｉｃ　Ｂｌｏｃｋを表す別のＩＲ　Ｂｌｏｃｋを生成すること
を含む、前記コンピュータシステム。
【請求項６】
コンピュータシステムであって、
第１のコンピュータと、
第２のコンピュータのために書かれたプログラムコードを前記第１のコンピュータ上で前記プログラムコードの中間表現を生成することを介して実行可能なエミュレーション・システムであって、
前記中間表現を生成することは、
前記プログラムによって必要とされる場合にプログラムコードのＢａｓｉｃ　Ｂｌｏｃｋごとに中間表現の中間表現ブロック（ＩＲ　Ｂｌｏｃｋ）を生成することであって、各ＩＲ　Ｂｌｏｃｋが特定の入口条件についてのプログラムコードの各Ｂａｓｉｃ　Ｂｌｏｃｋを表す、前記生成すること、
各ＩＲ　Ｂｌｏｃｋに対応するターゲットコードを記憶すること、
プログラムが所与の入口条件についてのＢａｓｉｃ　Ｂｌｏｃｋの実行を必要とする場合に、
ａ）その所与の入口条件についての前記Ｂａｓｉｃ　Ｂｌｏｃｋを表す記憶済みターゲットコードがある場合は、前記記憶済みターゲットコードを使用し、又は
ｂ）その所与の入口条件についての前記Ｂａｓｉｃ　Ｂｌｏｃｋを表す記憶済みターゲットコードがない場合は、前記所与の入口条件についての前記Ｂａｓｉｃ　Ｂｌｏｃｋを表す別のＩＲ　Ｂｌｏｃｋを生成すること
を含む、前記コンピュータシステム。
【請求項７】
抽象レジスタを表す複数のレジスタオブジェクトを生成することであって、個々のレジスタオブジェクトは個々の抽象レジスタを表す、前記生成すること、
複数の表現オブジェクトを生成することであって、前記プログラムコードの要素が前記プログラムコード中に生じる場合に各表現オブジェクトは前記プログラムコードの種々の演算子又はオペランドを表し、各表現オブジェクトは、直接的に又は前記表現オブジェクトのうちの他の表現オブジェクトからの参照を介して間接的に関係する１つのレジスタオブジェクトによって参照される、請求項５又は６に記載のコンピュータシステム。
【請求項８】

コンピュータシステムであって、
第１のプロセッサと、
第２のプロセッサのために書かれたプログラムコードを前記第１のプロセッサ上で前記プログラムコードの中間表現をメモリ内に生成することを介して実行可能なエミュレーション・システムであって、
前記中間表現を生成することは、
複数のあり得る効果又は機能を有する特定のサブジェクトコード命令の第１の反復時に、当該反復で必要とされる特定の機能だけを表す特殊ケース中間表現を生成及び記憶すること、
同じサブジェクトコード命令の後続の各反復で、前記後続の反復で必要とされる必要な機能に対して特殊ケース中間表現が生成されているかどうかを判定し、そのような特殊ケース中間表現が以前に生成されていない場合に、その機能特有の追加の特殊ケース中間表現を生成すること
を含む、前記コンピュータシステム。
【請求項９】
コンピュータシステムであって、
第１のコンピュータと、
第２のコンピュータのために書かれたプログラムコードを前記第１のコンピュータ上で前記プログラムコードの中間表現をメモリ内に生成することを介して実行可能なエミュレーション・システムであって、
前記中間表現を生成することは、
複数のあり得る効果又は機能を有する特定のサブジェクトコード命令の第１の反復時に、当該反復で必要とされる特定の機能だけを表す特殊ケース中間表現を生成及び記憶すること、
同じサブジェクトコード命令の後続の各反復で、前記後続の反復で必要とされる必要な機能に対して特殊ケース中間表現が生成されているかどうかを判定し、そのような特殊ケース中間表現が以前に生成されていない場合に、その機能特有の追加の特殊ケース中間表現を生成すること
を含む、前記コンピュータシステム。
【請求項１０】
抽象レジスタを表す複数のレジスタオブジェクトを生成することであって、個々のレジスタオブジェクトは個々の抽象レジスタを表す、前記生成すること、
複数の表現オブジェクトを生成することであって、前記プログラムコードの要素が前記プログラムコード中に生じる場合に各表現オブジェクトは前記プログラムコードの種々の演算子又はオペランドを表し、各表現オブジェクトは、直接的に又は前記表現オブジェクトのうちの他の表現オブジェクトからの参照を介して間接的に関係する１つのレジスタオブジェクトによって参照される、請求項８又は９に記載のコンピュータシステム。
【請求項１１】
前記第１のプロセッサが前記中間用言から得られるターゲットコードを実行する場合に、前記中間表現が動的に生成される、請求項８～１０のいずれか一項に記載のコンピュータシステム。
【請求項１２】
前記特殊ケース中間表現が生成され、前記要求された機能が前記関連付けられ記憶された特殊ケースの中間表現によって表されるものと同じであるかどうかを前記個々のサブジェクト命令の後続の反復時に判定するための関連付けられたテストプロシージャが生成及び記憶され、追加の特殊ケース中間表現が必要な場合は、その特殊ケース中間表現に関連する追加のテストプロシージャが、その追加の特殊ケース中間表現と共に生成及び記憶される、請求項８～１１のいずれか一項に記載のコンピュータシステム。
【請求項１３】
特定のサブジェクトコード命令についての追加の特殊ケース中間表現及び追加の関連テ

ストプロシージャが、少なくとも最初はいずれかの既存の特殊ケース中間表現及び記憶済み関連テストプロシージャについての従属関係で記憶され、したがって、サブジェクトコード命令の第２の反復以降に、必要とされる機能の特殊ケース中間表現が存在すると判定されるか又は必要とされる中間表現が存在しないと判定されるまで前記テストプロシージャをそれが生成及び記憶された順に行うことによって、必要とされる特殊ケース中間表現が以前に生成されたかどうかの判定が行われ、必要とされる中間表現が存在しないと判定された場合は、さらに追加の中間表現及び別の関連テストプロシージャが生成される、請求項８～１２のいずれか一項に記載のコンピュータシステム。

【請求項１４】

使用頻度のより高い特殊ケース中間表現に関連付けられたテストプロシージャが使用頻度のより低い特殊ケース中間表現に関連付けられたテストプロシージャの前に実行されるようにテストプロシージャの順序を調整することによって中間表現が最適化される、請求項８～１３のいずれか一項に記載のコンピュータシステム。

【請求項１５】

前記プログラムコードが前記生成された中間表現を使用して前記第１のプロセッサによって実行されるように前記第２のプロセッサによる実行のために書かれた前記プログラムコードを変換する、請求項８に記載のコンピュータシステム。

【請求項１６】

前記プログラムコードが前記生成された中間表現を使用して前記第１のコンピュータによって実行されるように前記第２のコンピュータによる実行のために書かれた前記プログラムコードを変換する、請求項９に記載のコンピュータシステム。

【請求項１７】

前記プログラムコードが実行されている場合に、前記変換が動的に実行される、請求項６～１２のいずれか一項に記載のコンピュータシステム。

【請求項１８】

前記プロセスが前記中間表現を最適化することによって前記プログラムコードを最適化することを含む、請求項６～１３のいずれか一項に記載のコンピュータシステム。

【請求項１９】

前記エミュレーション・システムが、前記プログラムコードが前記第１のプロセッサによってより効率的に実行されるように前記第１のプロセッサによって実行さするために書かれた前記プログラムコードを最適化するために使用される、請求項８に記載のコンピュータシステム。

【請求項２０】

前記エミュレーション・システムが、前記プログラムコードが前記第１のコンピュータによってより効率的に実行されるように前記第１のコンピュータによって実行さするために書かれた前記プログラムコードを最適化するために使用される、請求項９に記載のコンピュータシステム。

【請求項２１】

コンピュータシステムであって、

第１のプロセッサと、

第２のプロセッサのために書かれたプログラムコードを前記第１のプロセッサ上で前記プログラムコードの中間表現をメモリ内に生成することを介して実行可能なエミュレーション・システムであって、

前記中間表現を生成することは、

抽象レジスタを表す複数のレジスタオブジェクトを生成することであって、各レジスタオブジェクトは個々の抽象レジスタを表す、前記生成することと、

表現オブジェクトを生成することであって、前記プログラムコードの要素が前記プログラムコード中に生じる場合に各表現オブジェクトは前記プログラムコードの個々の演算子又はオペランドを表し、各表現オブジェクトは、直接的に又は前記表現オブジェクトのうちの他の表現オブジェクトからの参照を介して間接的に関係する１以上の前記レジスタオ

ブジェクトによって参照される、前記生成すること、
前記プログラムコードによって生成される可変値を保持するための複数のレジスタオブジェクトを生成すること、
固定値、及び／又は前記固定値と前記プログラムコードに従った前記可変値との関係を表す複数の表現オブジェクトを生成すること
を含み、
前記中間表現は、コンピュータコードのブロックに対して生成及び記憶され、同じコードブロックが後で再入力された場合は後で再利用され、前記第１のコンピュータプログラムコードの少なくとも１ブロックが、代替の未使用入口条件又は効果又は機能を有することができ、前記中間表現は、プログラムコードのそのブロックをその時の優勢な条件で実行するのに必要なとき、最初にだけ生成及び記憶される、前記コンピュータシステム。
【請求項２２】
コンピュータシステムであって、
第１のコンピュータと、
第２のコンピュータのために書かれたプログラムコードを前記第１のコンピュータ上で前記プログラムコードの中間表現をメモリ内に生成することを介して実行可能なエミュレーション・システムであって、
前記中間表現を生成することは、
抽象レジスタを表す複数のレジスタオブジェクトを生成することであって、各レジスタオブジェクトは個々の抽象レジスタを表す、前記生成することと、
表現オブジェクトを生成することであって、前記プログラムコードの要素が前記プログラムコード中に生じる場合に各表現オブジェクトは前記プログラムコードの個々の演算子又はオペランドを表し、各表現オブジェクトは、直接的に又は前記表現オブジェクトのうちの他の表現オブジェクトからの参照を介して間接的に関係する１以上の前記レジスタオブジェクトによって参照される、前記生成すること、
前記プログラムコードによって生成される可変値を保持するための複数のレジスタオブジェクトを生成すること、
固定値、及び／又は前記固定値と前記プログラムコードに従った前記可変値との関係を表す複数の表現オブジェクトを生成すること
を含み、
前記中間表現は、コンピュータコードのブロックに対して生成及び記憶され、同じコードブロックが後で再入力された場合は後で再利用され、前記第１のコンピュータプログラムコードの少なくとも１ブロックが、代替の未使用入口条件又は効果又は機能を有することができ、前記中間表現は、プログラムコードのそのブロックをその時の優勢な条件で実行するのに必要なとき、最初にだけ生成及び記憶される、前記コンピュータシステム。
【請求項２３】
プログラムコードの所与のブロックについて、それにより以前に記憶された中間表現が現在優勢な同じ条件セットに対するものであったかどうかを判定し、そうでない場合は、コードのブロックを実行するのに必要な追加の中間表現を現在優勢な新しい条件セットに対して生成及び記憶する、請求項２１又は２２に記載のコンピュータシステム。

フロントページの続き

(74)復代理人　100085545
弁理士　松井　光夫
(74)復代理人　100118599
弁理士　村上　博司
(72)発明者　ソウログロウ　ジェイソン
イギリス国、エム１３・９ピーエル、マンチェスター、オックスフォード・ロード、ヴィクトリア　ユニバーシティ　オブ　マンチェスター内
(72)発明者　ラウズソーン　アラスデア
イギリス国、エム１３・９ピーエル、マンチェスター、オックスフォード・ロード、ヴィクトリア　ユニバーシティ　オブ　マンチェスター内