(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5448165号
(P5448165)
(45)発行日　平成26年3月19日(2014.3.19)　(24)登録日　平成26年1月10日(2014.1.10)

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

請求項の数 22　(全 35 頁)

(21)出願番号　特願2009-529779 (P2009-529779)
(86)(22)出願日　平成19年9月27日 (2007.9.27)
(65)公表番号　特表2010-506250 (P2010-506250A)
(43)公表日　平成22年2月25日 (2010.2.25)
(86)国際出願番号　PCT/GB2007/050586
(87)国際公開番号　WO2008/068519
(87)国際公開日　平成20年6月12日 (2008.6.12)
審査請求日　平成21年11月17日 (2009.11.17)
(31)優先権主張番号　0619380.9
(32)優先日　平成18年10月2日 (2006.10.2)
(33)優先権主張国　英国 (GB)
(31)優先権主張番号　60/853,924
(32)優先日　平成18年10月24日 (2006.10.24)
(33)優先権主張国　米国 (US)

(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　ＭＡＣＨＩＮＥＳ　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

前置審査　　最終頁に続く

(54)【発明の名称】レジスタ・ウィンドウ・アーキテクチャをサポートするように適合されたコンピュータ・システム、並びに当該コンピュータ・システムを制御する方法及びそのコンピュータ・プログラム

(57)【特許請求の範囲】
【請求項１】
レジスタ・ウィンドウ・アーキテクチャをサポートするように適合されたコンピューティング・システムであって、レジスタ・ウィンドウに基づくサブジェクト・コンピューティング・アーキテクチャのサブジェクト・プロセッサによって実行可能なサブジェクト・コードとともに用いられ、前記レジスタ・ウィンドウが、ウィンドウ表示されたレジスタ・ファイルから、サブジェクト・プロセッサのサブジェクト・レジスタの選択されたサブセットを明確にするように設けられ、前記サブジェクト・コードは、レジスタ・ウィンドウに影響を与えるウィンドウに基づいた命令（以下、前記レジスタ・ウィンドウに基づく命令という）と、前記レジスタ・ウィンドウにおけるサブジェクト・レジスタに対するリファレンスを含むレジスタに基づいた命令（前記レジスタに基づく命令という）とを含み、
前記コンピューティング・システムは、
前記サブジェクト・コードを復号化して、前記レジスタ・ウィンドウに基づく命令から、レジスタ・ウィンドウの動きについての情報を導き出し、且つ前記レジスタに基づく命令から、一つ以上のウィンドウ表示されたサブジェクト・レジスタ・リファレンスを導き出すように構成されているデコーダ・ユニットと、
複数のエントリを記憶するように設けられた、スタック・データ構造を有するメモリと、
前記デコーダ・ユニットによって復号化された前記サブジェクト・コードからターゲッ

ト・コードを生成するように構成されているエンコーダ・ユニットと、
前記ターゲット・コードを実行して、前記メモリにおける前記スタック・データ構造の先頭を指示するスタック・ポインタを設定し、前記レジスタ・ウィンドウの動きについての情報を参照して、前記スタック・ポインタを調整し、前記ウィンドウ表示されたサブジェクト・レジスタ・リファレンスの各々について決定された変位と組み合わされた前記スタック・ポインタを参照して、前記スタック・データ構造内のエントリのうちの少なくとも一つにアクセスするように構成されているターゲット・プロセッサと
を備えている、前記コンピューティング・システム。
【請求項２】
前記デコーダ・ユニットは、少なくとも前記レジスタ・ウィンドウのＳＡＶＥおよびＲＥＳＴＯＲＥタイプの動きを引き起こす前記サブジェクト・コードにおいて、前記ウィンドウに基づく命令のうち、一つ以上を識別し、前記ウィンドウに基づく命令から、少なくともＳＡＶＥおよびＲＥＳＴＯＲＥタイプのレジスタ・ウィンドウの動きに関する情報を導き出すように構成されている、請求項１に記載のコンピューティング・システム。
【請求項３】
前記ターゲット・プロセッサは、前記各ＳＡＶＥおよびＲＥＳＴＯＲＥタイプのレジスタ・ウィンドウの動きに関する情報に応じて、前記スタック・ポインタを調整するように構成されている、請求項２に記載のコンピューティング・システム。
【請求項４】
前記ターゲット・プロセッサは、前記各ＳＡＶＥおよびＲＥＳＴＯＲＥタイプのレジスタ・ウィンドウの動きに関する情報に応じて、所定のオフセットによって、前記スタック・ポインタを調整するように構成されている、請求項３に記載のコンピューティング・システム。
【請求項５】
前記エンコーダ・ユニットは、前記調整されたスタック・ポインタを前記決定された変位に組み合わせることによって、前記ターゲット・プロセッサに、前記スタック・データ構造の選択されたエントリにアクセスさせる、少なくとも一つのターゲット・コード命令を発するように構成されている、請求項１～４のいずれか一項に記載のコンピューティング・システム。
【請求項６】
前記ターゲット・プロセッサは、前記レジスタ・ウィンドウの動きに関する情報にしたがって、所定の数のエントリによって、前記スタック・ポインタを調整するようにフレーム・オフセットを決定し、前記決定されたフレーム・オフセットと前記決定された変位とにしたがって、前記スタック・データ構造から選択されたエントリにアクセスするように、前記スタック・ポインタを調整するように構成されている、請求項１～５のいずれか一項に記載のコンピューティング・システム。
【請求項７】
前記デコーダ・ユニットは、前記レジスタ・ウィンドウの動きに関する情報が導き出される前記サブジェクト・コードにおいて、前記ウィンドウに基づく命令にしたがって、前記サブジェクト・コードを、複数のブロックに分割するように構成されている、請求項１～６のいずれか一項に記載のコンピューティング・システム。
【請求項８】
前記サブジェクト・コードは、被呼び出し部分に対して関数呼び出しを行う、少なくとも一つの呼び出し部分を含み、且つ前記エンコーダ・ユニットは、前記被呼び出し部分を、前記呼び出し部分にインライン化するターゲット・コードの単一のブロックを生成するように構成されている、請求項１～７のいずれか一項に記載のコンピューティング・システム。
【請求項９】
前記エンコーダ・ユニットは、前記ターゲット・プロセッサによって、
ｉ）前記ウィンドウ表示されたサブジェクト・レジスタ・リファレンス及び前記スタッ

ク・ポインタから導き出された前記変位を参照して、前記サブジェクト・コードの前記呼び出し部分において識別された、前記ウィンドウ表示されたサブジェクト・レジスタ・リファレンスにしたがって、前記スタック・データ構造から選択されたエントリにアクセスし、

ｉｉ）前記サブジェクト・コードの前記呼び出し部分から導き出された、前記レジスタ・ウィンドウの動きに関する情報にしたがって、前記スタック・データ構造の所定の数のエントリに等価なフレーム・オフセットを増加し、

ｉｉｉ）ウィンドウ表示されたサブジェクト・レジスタ・リファレンスから導き出された前記変位と、フレーム・オフセットにしたがって調整されたスタック・ポインタを参照して、サブジェクト・コードの前記被呼び出し部分において識別された、ウィンドウ表示されたサブジェクト・レジスタ・リファレンスにしたがって、前記スタック・データ構造から選択されたエントリにアクセスし、

ｉｖ）サブジェクト・コードの前記被呼び出し部分から導き出された、前記レジスタ・ウィンドウの動きに関する情報にしたがって、前記フレーム・オフセットを減少させる

ように、実行される前記ターゲット・コード命令を生成するように構成されている、請求項８に記載のコンピューティング・システム。

【請求項１０】

前記メモリはさらに、前記サブジェクト・コードの実行用に、実行スタックをエミュレートするように構成されているエミュレート実行スタックを備えており、

前記デコーダ・ユニットは、前記サブジェクト・コードを復号化して、少なくともＳＡＶＥタイプおよびＲＥＳＴＯＲＥタイプのレジスタ・ウィンドウの動きに関する情報を識別するように構成されており、

前記エンコーダ・ユニットは、前記ターゲット・プロセッサによって実行された前記ターゲット・コードを生成して、識別されたレジスタ・ウィンドウの動きごとにカウンタをアップデートさせるように構成されており、前記アップデートは、ＳＡＶＥごとにカウンタを増加させることと、ＲＥＳＴＯＲＥごとにカウンタを減分させることを含み、それによって、前記カウンタは、各フレームがスタック・データ構造上に記憶された所定の数のエントリを含む、フレームの数をカウントし、

前記デコーダ・ユニットは、前記サブジェクト・コードにおいて前記レジスタに基づいた命令から導き出された、前記ウィンドウ表示されたサブジェクト・レジスタ・リファレンスにおいて保持される、データの値の流出を要求するような前記サブジェクト・コードにおいて、ＳＰＩＬＬタイプの命令を識別するように構成されており、

前記エンコーダ・ユニットは、前記ターゲット・プロセッサによって実行された前記ターゲット・コードを生成して、前記スタック・データ構造におけるエントリから、メモリ内で前記エミュレートされた実行スタック上に割り当てられた対応するエントリに、データの値をコピーするように構成されており、ここで、前記エントリは、カウンタによって決定されるような多数のフレーム内に存在し、

前記エンコーダ・ユニットは、前記ターゲット・プロセッサによって実行される前記ターゲット・コードを生成して、前記スタック・データ構造内に、エントリの有効なフレームがないことを示す、デフォルト値に前記カウンタをリセットするように構成されており、

前記エンコーダ・ユニットは、前記ターゲット・プロセッサによって実行される前記ターゲット・コードを生成して、前記メモリ内の前記エミュレートされた実行スタックにおける前記エントリ内に記憶された、前記データの値をアドレス指定するように構成されている、請求項１～９のいずれか一項に記載のコンピューティング・システム。

【請求項１１】

前記デコーダ・ユニットはさらに、前記メモリ内の前記エミュレートされた実行スタックにおける前記エントリに保持された、データの値の充てんを要求する、前記サブジェクト・コードにおいて、ＦＩＬＬタイプの命令を識別するように構成されており、

前記エンコーダ・ユニットは、前記ターゲット・プロセッサによって実行された前記タ

ーゲット・コードを生成して、前記デフォルト値について前記カウンタをテストするように構成されており、
ｉ）ここで、前記カウンタをテストすることは、前記スタック・データ構造に一つ以上のエントリの有効なフレームが存在することを示し、識別されたＦＩＬＬ命令にしたがって、前記カウンタをアップデートして、
ｉｉ）ここで、前記カウンタをテストすることは、前記スタック・データ構造にエントリの有効な先行するフレームが存在しないことを示し、前記スタック・データ構造におけるエントリのカレント・フレームとして、前記メモリにおける前記エミュレートされた実行スタックのエントリのフレームから、前記メモリ内の前記スタック・データ構造にデータの値をコピーする、
請求項１０に記載のコンピューティング・システム。
【請求項１２】
前記デコーダ・ユニットは、前記サブジェクト・コードを複数のサブジェクト・コード・ブロックに分割するように構成されており、且つ
前記エンコーダ・ユニットは、前記ターゲット・プロセッサによる変換の合間で前記ターゲット・コードのブロックの実行を行う、対応する複数のターゲット・コード・ブロックとして、前記ターゲット・コードを生成するように構成されている、請求項１に記載のコンピューティング・システム。
【請求項１３】
前記サブジェクト・コードは、実行可能なバイナリコードであり、
前記ターゲット・コードは、実行可能なバイナリコードである、
請求項１に記載のコンピューティング・システム。
【請求項１４】
レジスタ・ウィンドウ・アーキテクチャをサポートするように適合されたコンピューティング・システムを制御する方法であって、
（ａ）レジスタ・ウィンドウに基づくサブジェクト・コンピューティング・アーキテクチャのサブジェクト・プロセッサによって実行可能な、サブジェクト・コードを復号化する工程であって、レジスタ・ウィンドウが、ウィンドウ表示されたレジスタ・ファイルから、サブジェクト・レジスタの選択されたサブセットを明確にするように位置決めされ、
（ａ１）前記サブジェクト・コードにおける命令から、ウィンドウ表示されたサブジェクト・レジスタ・リファレンスを識別する工程であって、前記ウィンドウ表示されたサブジェクト・レジスタ・リファレンスが、前記レジスタ・ウィンドウにおける前記サブジェクト・レジスタのうちの一つに対するリファレンスを含む、前記識別する工程と、
（ａ２）前記レジスタ・ウィンドウの動きを発生させる前記サブジェクト・コードにおける命令から、レジスタ・ウィンドウの動きに関する情報を導き出す工程と
を含む、前記工程（ａ）と、
（ｂ）前記コンピューティング・システムのメモリ内において、複数のエントリを記憶させるように構成されているスタック・データ構造を提供して、前記メモリ内で、前記スタック・データ構造の先頭を指示するスタック・ポインタを設定する工程と、
（ｃ）前記サブジェクト・コードをターゲット・コードに変換して、前記コンピューティング・システムのターゲット・プロセッサ上で前記ターゲット・コードを実行する工程であって、それにより
（ｃ１）前記識別されたレジスタ・ウィンドウの動きについての情報を参照して、前記スタック・ポインタを調整する工程と、
（ｃ２）前記ウィンドウ表示されたサブジェクト・レジスタ・リファレンスの各々について決定された変位と組み合わされた前記スタック・ポインタを参照して、前記スタック・データ構造内のエントリのうちの少なくとも一つにアクセスする工程と
を実行する、前記工程（ｃ）と
を含む、前記方法。
【請求項１５】

前記（ａ２）の工程は、前記レジスタ・ウィンドウの少なくともＳＡＶＥおよびＲＥＳＴＯＲＥタイプの動きを発生させる、前記サブジェクト・コードにおいて、一つ以上の命令を識別して、それによって、少なくともＳＡＶＥおよびＲＥＳＴＯＲＥタイプのレジスタ・ウィンドウの動きに関する情報を導き出すことを含み、

前記（ｃ１）の工程は、前記各ＳＡＶＥタイプのレジスタ・ウィンドウの動きに関する情報に応じて、所定のフレーム・オフセットによって、前記スタック・ポインタを増分し、且つ前記各ＲＥＳＴＯＲＥタイプのレジスタ・ウィンドウの動きに関する情報に応じて、所定のフレーム・オフセットによって、前記スタック・ポインタを減分するか、または前記各ＳＡＶＥタイプのレジスタ・ウィンドウの動きに関する情報に応じて、所定のフレーム・オフセットによって、前記スタック・ポインタを減分し、且つ前記各ＲＥＳＴＯＲＥタイプのレジスタ・ウィンドウの動きに関する情報に応じて、所定のフレーム・オフセットによって、前記スタック・ポインタを増分することを含む、

請求項１４に記載の方法。

【請求項１６】

前記スタック・データ構造における前記エントリの各々が、前記サブジェクト・コード内の前記レジスタに基づく命令における、ウィンドウ表示されたサブジェクト・レジスタ・リファレンスのセットのうち一つに対応する方法であって、

前記ステップ（ｃ１）において、前記フレーム・オフセットは、前記セットの大きさに対応する前記スタック・データ構造上における多数のエントリによって、前記スタック・ポインタを調整し、

前記ステップ（ｃ２）において、前記変位は、セット内の前記ウィンドウ表示されたサブジェクト・レジスタ・リファレンスの相対位置にしたがって、多数のスタック・データ構造上のエントリによって、前記スタック・ポインタを変位させる、

請求項１５に記載の方法。

【請求項１７】

レジスタ・ウィンドウの動きに関する情報が導き出される命令の位置にしたがって、前記サブジェクト・コードを、複数のブロックに分割する工程をさらに含み、このようなブロックごとに、前記サブジェクト・コード・ブロック内のサブジェクト・コード命令について、前記工程（ａ）、（ｂ）、および（ｃ）を行う、請求項１４に記載の方法。

【請求項１８】

前記サブジェクト・コードは、被呼び出し部分に対して関数呼び出しを行う、少なくとも一つの呼び出し部分を含み、

前記工程（ｃ）はさらに、被呼び出し部分を、呼び出し部分にインライン化する、前記ターゲット・コードの単一のブロックを供給することを含む、

請求項１４に記載の方法。

【請求項１９】

第一の値を有する前記スタック・ポインタを提供し、一つ以上のターゲット・コード命令を実行して、ウィンドウ表示されたサブジェクト・レジスタ・リファレンスから導き出された変位と、前記スタック・ポインタの第一の値とを参照して、サブジェクト・コードの前記呼び出し部分において識別された、ウィンドウ表示されたサブジェクト・レジスタ・リファレンスにしたがって、前記スタック・データ構造から選択されたエントリにアクセスする工程と、

サブジェクト・コードの前記呼び出し部分から導き出された前記レジスタ・ウィンドウの動きに関する情報にしたがって、所定の数のエントリによって、フレーム・オフセットを増加させる工程と、

ウィンドウ表示されたサブジェクト・レジスタ・リファレンスから導き出された変位と、フレーム・オフセットにしたがって調整されるようなスタック・ポインタとを参照して、サブジェクト・コードの被呼び出し部分において識別された、ウィンドウ表示されたサブジェクト・レジスタ・リファレンスにしたがって、前記スタック・データ構造から選択されたエントリにアクセスするように、一つ以上のターゲット・コード命令を実行する工

程と、
前記サブジェクト・コードの被呼び出し部分から導き出された、レジスタ・ウィンドウの動きに関する情報にしたがって、フレーム・オフセットを減少させる工程と
を含む、請求項１８に記載の方法。
【請求項２０】
前記コンピューティング・システムの前記メモリにおいてエミュレートされた実行スタックを提供して、サブジェクト・コードの実行に関する実行スタックをエミュレートする工程と、
少なくとも前記ＳＡＶＥおよびＲＥＳＴＯＲＥタイプのレジスタ・ウィンドウの動きに関する情報を識別する、前記サブジェクト・コードを復号化する工程と、
ＳＡＶＥごとにカウンタを増分し、ＲＥＳＴＯＲＥごとにカウンタを減分することを含み、これによって、前記カウンタは、スタック・データ構造上に記憶されたエントリの多数のフレームをカウントする、ＳＡＶＥおよびＲＥＳＴＯＲＥタイプのレジスタ・ウィンドウの動きに関する情報ごとに、カウンタをアップデートする工程と、
前記サブジェクト・コードによって、実行スタックにアドレス指定された、ウィンドウ表示されたサブジェクト・レジスタ内に保持されたデータの値の流出を要求する、前記サブジェクト・コードにおける、ＳＰＩＬＬタイプの命令を識別する工程と、
エントリの多数のフレームからのデータの値を、前記スタック・データ構造から、前記エミュレートされた実行スタック上に割り当てられた対応する空間にコピーする工程であって、エントリのフレームの数はカウンタによって決定される、前記コピーする工程と、
前記スタック・データ構造内にエントリの有効なフレームが存在しないことを示す、デフォルト値に、カウンタをリセットする工程と、
ターゲット・コードを生成して、エミュレートされた実行スタック上に記憶されたデータの値をアドレス指定する工程と
をさらに含む、請求項１５に記載の方法。
【請求項２１】
実行スタックにおいて保持されたデータの値を、サブジェクト・コードによってアドレス指定された、ウィンドウ表示されたサブジェクト・レジスタ内に充てんすることを要求する、前記サブジェクト・コードにおける、ＦＩＬＬタイプの命令を識別する工程と、
デフォルト値について、カウンタをテストする工程と、
をさらに含み、
前記カウンタをテストする工程は、一つ以上のエントリの有効なフレームが、スタック・データ構造内に存在することを示し、識別されたＦＩＬＬ命令にしたがって、カウンタをアップデートして、
前記カウンタをテストする工程は、エントリの先行する有効なフレームが、スタック・データ構造内に存在しないことを示し、エミュレートされた実行スタックからのデータの値のフレームを、エントリのカレント・フレームとして使用するための、スタック・データ構造にコピーする
請求項２０に記載の方法。
【請求項２２】
レジスタ・ウィンドウ・アーキテクチャをサポートするようにコンピューティング・システムを適合させるためのコンピュータ・プログラムであって、コンピューティング・システムに、請求項１４～２１のいずれか一項に記載の方法の各ステップを実行させる、前記コンピュータ・プログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、一般的には、コンピュータおよびコンピュータ・システムの分野に関連する。さらに詳細には、本発明は、レジスタ・ウィンドウ・アーキテクチャをサポートするように適合したコンピュータ・システムおよびレジスタ・ウィンドウ・アーキテクチャをサ

ポートするようにコンピュータ・システムを適合させる方法に関する。
【背景技術】
【０００２】
中央処理装置（ＣＰＵ）あるいはプロセッサは、すべての近代的なコンピュータ・システムの中心部に位置している。プロセッサは、コンピュータ・プログラムの命令を実行し、このようにして、コンピュータに有益な作業を行わせることを可能にする。ＣＰＵは、パソコン、ノート型パソコン、およびＰＤＡなどの専用コンピュータマシンのみでなく、現代生活におけるあらゆる種類のデジタル機器において普及している。近代のマイクロプロセッサは、自動車から、洗濯機、子供の玩具にわたり、あらゆる分野において見られる。
【０００３】
あるタイプのプロセッサによって実行可能であるプログラム・コードは、他のタイプのプロセッサによっては実行不可能である場合が多い点に問題がある。第一に、各タイプのプロセッサは、それ独自の命令セット・アーキテクチャ（ＩＳＡ）を有している。第二に、多くの場合、プロセッサは、他のタイプのプロセッサには存在しない、独自のハードウエアの特徴を有している。したがって、自動的に、一つのタイプのプロセッサに関して書き込まれたプログラム・コードを、他のタイプのプロセッサによって実行可能なコードに変換するか、あるいは、旧式で無効なコードを、同じタイプのプロセッサに関するより新しい、またより速いバージョンに最適化するように、プログラム・コード変換の分野が発展した。すなわち、組み込まれたＣＰＵおよび組み込まれていないＣＰＵの両方において、パフォーマンスが「加速されうる」あるいは、より良いコスト／パフォーマンスによる利益をもたらす他のプロセッサに「変換されうる」、大量のソフトウエアが既に存在する主要なＩＳＡがある。また、それらのＩＳＡに合わせて時間内にロックされて、パフォーマンスあるいは市場参入において発展することが不可能な、主要なＣＰＵアーキテクチャも認識されている。この問題は、スタンドアロンのポケットサイズの装置から、数十個、あるいは数百個の高性能のコンピュータを有する大型ネットワークにいたるまで、電子産業のあらゆるレベルにおいて当てはまる。
【０００４】
このプログラム・コード変換の分野における、背景技術としては、「プログラム・コード変換」と題された国際公開第２０００／２２５２１号パンフレット、「プログラム・コード変換中においてインタープリタの最適化を実行する方法および装置」と題された国際公開第２００４／０９５２６４号パンフレット、発明の名称、「プログラム・コード変換に関して中間表現を生成するための改良されたアーキテクチャ」と題された国際公開第２００４／０９７６３１号パンフレット、「調整可能で正確な例外処理を実行するための方法および装置」と題された国際公開第２００５／００６１０６号パンフレット、および「プログラム・コード変換中における正確な例外処理のための方法および装置」と題された国際公開第２００６／１０３３９５号パンフレットが挙げられる。これらはすべて、参照により本明細書に組み込まれ、本明細書において説明される実施形態の例で用いられうる、プログラム・コード変換能力を促進させる方法および装置を開示している。
【０００５】
最も近代的なプロセッサは、あるタイプの高速アクセス・メモリとして、レジスタを１セット含む。プロセッサは、レジスタを用いて、一時的な値を維持する一方、コンピュータ・プログラムにおいて、一連の命令を実行する。プロセッサのハードウエアは、これらのレジスタを有限数含み、また、用いられる際には、プログラムの実行によって、ただちに、すべての利用可能なレジスタを、一時的なデータ値で満たす。これによって、プロセッサがコードのあるセクションから他のセクションに移動するにつれて、利用可能なレジスタ間で競争が起こる。なぜなら、コードの各セクションは一時的な値を生成して、それらの一時的なデータ値を保存するために、プロセッサにおいてレジスタを活用する必要があるからである。
【０００６】

この困難性に応じて、プロセッサは、レジスタ・ウィンドウ・アーキテクチャによって発展してきた。例えば、１９９０年代初頭のバークリーＲＩＳＣの設計に基づいた、レジスタ・ウィンドウ・アーキテクチャは、ハードウエア・レジスタ（レジスタ・ファイル）の大きいセットを提供するが、コードのカレント・セクションによってアクセスされるこれらのレジスタ（レジスタ・ウィンドウ）については、小さいサブセットのみが可能となる。レジスタ・ウィンドウのカレント位置の外側に存在するレジスタ・ファイル中の他のレジスタは、コンピュータ・プログラムのカレント・セクションによってアクセス可能ではない。例えば、任意のある時間においては、合計で６４個のレジスタのレジスタ・ファイルのうち、８個のレジスタのみが視認できる。プロセッサがコードのあるセクションから他のセクションに移動するとき、例えば、手続き呼び出しが実行されるようなときには、レジスタ・ウィンドウは、レジスタ・ファイル内でその位置を移動させて、コードの新しいセクションに関して、レジスタの異なるサブセットを露出させる。これらの動きは一般的に、ＳＡＶＥタイプの動きを発生させるものとして分類されるか、あるいは、ＲＥＳＴＯＲＥタイプの動きを発生させるものとして分類される。ここで、ＳＡＶＥタイプの動きとは、レジスタ・ウィンドウが、前もって使用されていない位置に移動して、何も記憶されていないハードウエア・レジスタのセットを、実行中のサブジェクト・コードに提供することであり、ＲＥＳＴＯＲＥタイプの動きとは、レジスタ・ウィンドウを、前もって露出させた位置にまで戻し、このようにして、前もって使用したレジスタの１セットを実行コードに対して示すことによって、これらのレジスタ内に維持されるデータ値に対するアクセスを復元することである。しかしながら、最終的には、レジスタ・ファイル内における有限数のハードウエア・レジスタは、プログラムを実行することによって呼び出される、手続き呼び出しレベルの数に応じていっぱいになる。ここで、ＳＰＩＬＬ動作が実行されることによって、レジスタ内のデータの値は、プロセッサの外部にある二階層のメモリなどの安全な位置にまで移動する。その後、ＦＩＬＬ動作は、プロセッサがコードの関連するセクションの実行を継続的に行えるように、これらのデータの値をハードウエア・レジスタに戻す。

【０００７】

このレジスタ・ウィンドウ・アーキテクチャは、特に、プロセッサが頻繁に、コードのあるセクションから他のセクションに移動して、それから、第一のセクションに戻る（すなわち、手続き呼び出しを実行する）場合に、コンピュータ・プログラムの大部分に関する、コストおよび時間のかかるレジスタの流出および充てん動作を避けることによって、コンピューティング・システムが早急に、実行できるようにさせることを目的としている。

【０００８】

このレジスタ・ウィンドウ・アーキテクチャは、大商業規模で、ＳＰＡＲＣ、ＡＭＤ２９０００、およびインテルｉ９６０などのプロセッサに採用されている。このため、これらのアーキテクチャ上でのみ実行されるように、コードの大部分が既に書き込まれている。また、任意の他のタイプのプロセッサによってコードの大部分を実行することはできない。

【０００９】

これらの市販されているプロセッサのうち、ＳＰＡＲＣは特に、普及している。ＳＰＡＲＣレジスタ・ウィンドウ・アーキテクチャについてのさらなる背景情報は、例えば、ＳＰＡＲＣ Ａｒｃｈｉｔｅｃｔｕｒｅ Ｍａｎｕａｌ，Ｖｅｒｓｉｏｎ ８，Ｓｅｃｔｉｏｎ ４．１，“Ｒｅｇｉｓｔｅｒｓ”およびＳＰＡＲＣ Ａｒｃｈｉｔｅｃｔｕｒｅ Ｍａｎｕａｌ，Ｖｅｒｓｉｏｎ ９，Ｓｅｃｔｉｏｎ ５．１，“Ｒｅｇｉｓｔｅｒｓ”（ＳＰＡＲＣ Ｉｎｔｅｒｎａｔｉｏｎａｌ Ｉｎｃ ｏｆ Ｓａｎ Ｊｏｓｅ，Ｃａｌｉｆｏｒｎｉａ）において認められる。この開示は、参照により本明細書に組み込まれる。

【００１０】

実施例として、図１は、従来技術のｖ９　ＳＰＡＲＣアーキテクチャにおけるレジスタ

・ウィンドウの使用を示している。このｖ９　ＳＰＡＲＣアーキテクチャは、複数のコントロール／ステータス・レジスタと、多数の汎用（「ｒ」）レジスタを提供する。汎用レジスタは、８個恒久的なのビジブル・グローバル・レジスタ（およびさらに、８個のグローバルの代替物）と、８個の「イン」レジスタ、８個の「ローカル」レジスタ、および８個の「アウト」レジスタに分割された、移動可能な２４個のレジスタ・ウィンドウを含む。全体的なレジスタ・ファイルへのカレント・ウィンドウは、コントロール／ステータス・レジスタの一つにおいて維持される、カレント・ウィンドウ・ポインタ（ＣＷＰ）によって付与される。このＣＷＰは、「復元」命令が実行されるたびに、増加して、また、「保存」命令に対して、あるいはトラップが発生するときに減少する。この実施例において、一つのウィンドウ位置の８個の「アウト」レジスタが、隣接するウィンドウの位置における８個の「イン」レジスタと重複するように、２４－レジスタ・ウィンドウは、部分的に、隣接するウィンドウの位置と重複する。その一方、ローカルレジスタは、ウィンドウ位置ごとに固有のものである。すなわち、ウィンドウ位置ＣＷＰ＋１の「アウト」レジスタもまた、カレント・ウィンドウＣＷＰの「イン」レジスタとしてアドレス可能である。また、カレント・ウィンドウＣＷＰの「アウト」レジスタは、次のウィンドウＣＷＰ－１の「イン」レジスタと等価である。ｖ９　ＳＰＡＲＣアーキテクチャは、特定のハードウエアの実行に依存して、最小限の３個のウィンドウ位置から、最大限の３２個のウィンドウ位置までをサポートする。このため、永久的に視認可能なグローバル・レジスタとともに、ｖ９　ＳＰＡＲＣアーキテクチャは、６４個から５２８個の汎用ハードウエア・レジスタ（８個のグローバル・レジスタ、８個の代替グローバル、およびウィンドウ位置ごとに１６個のレジスタ）を必要とする。

【００１１】

図２は、ｖ９　ＳＰＡＲＣアーキテクチャの実施例における、ウィンドウ表示されたレジスタ・ファイルの円形の性質を示している。ハードウエア・レジスタの数は有限であり、本実施例において、８個のレジスタ・ウィンドウ位置に対応して、１２８個のウィンドウ表示されたハードウエア・レジスタが存在する。図２は、カレント・ウィンドウ（ＣＷＰ＝０）として、ウィンドウ位置Ｗ０を示している。カレント・ウィンドウ位置Ｗ０を用いる手順が、ＲＥＳＴＯＲＥを実行する場合、その後、ウィンドウ位置Ｗ７は、カレント・ウィンドウ（ＣＷＰ＝＋１）となるであろう。また、位置Ｗ０における手続きがＳＡＶＥを実行する場合、ウィンドウ位置Ｗ１はカレント・ウィンドウとなる。すべてのレジスタ・ウィンドウ位置が消費されたときに、ウィンドウ・オーバーフロウ・トラップが発生する。すなわち、これらのレジスタは既に、以前のプログラム・コードのセクションを実行することから、有効なデータを含んでおり、上書きされるべきではないからである。ｖ９アーキテクチャの実施例において、ウィンドウ・オーバーフロウは、ＣＷＰにリンクされたＣＡＮＳＡＶＥコントロール／ステータス・レジスタを用いて検出される。この時点で、レジスタの内容は、レジスタ・ファイルに保存された有効なデータを上書きすることなく、実行し続けられるようにするために、主要なサブジェクトメモリ８における実行スタックなどの、メモリのより遅いアクセス領域に流出される。プログラムが最終的に、元のレジスタ値が、再度必要とされる時点に対してＲＥＳＴＯＲＥの指示を複数行うと、充てん動作によって、スタックから、レジスタ・ファイルのハードウエア・レジスタにレジスタ値を充てんする。ここで、ウィンドウ・アンダーフロウ・トラップは、ＣＡＮＲＥＳＴＯＲＥコントロール／ステータス・レジスタを参照して、レジスタ・ウィンドウの無効な復元移動を防ぐ。

【００１２】

この実施例において、元のプログラム・コード（ここでは「サブジェクト・コード」と称する）は、レジスタ・ウィンドウ・アーキテクチャを有する、特別なタイプのサブジェクト・ハードウエアに依存する。しかしながら、本発明のプログラム・コード変換の分野において、今度は、サブジェクト・コードはターゲット・コードに変換され、ターゲット・コンピューティング・システムによって実行される。すなわち、レジスタ・ウィンドウ・アーキテクチャに基づいたより旧式のサブジェクト・コンピュータ・システムを、今回

は、より新しいターゲット・コンピュータ・システムに置き換えることが望ましいが、依然として、サブジェクト・コンピュータ・システムのレジスタ・ウィンドウ・アーキテクチャをサポートする、ターゲット・コンピュータ・システムを有していることが望ましい。

【発明の概要】

【発明が解決しようとする課題】

【００１３】

本発明の目的は、レジスタ・ウィンドウ・アーキテクチャをサポートするように適合された、コンピューティング・システムを提供することにある。例示的な実施形態は、コンピュータ・システムを、他システムのアーキテクチャであって、自システムのアーキテクチャとは異なる、レジスタ・ウィンドウ・アーキテクチャをサポートするように適合させることを目的としている。

【課題を解決するための手段】

【００１４】

本発明によれば、コンピューティング・システム、コンピュータ・システムを適合させる方法、および、添付の特許請求の範囲に記載されているような、コンピュータ読取可能な記憶媒体が提供される。本発明の他の特徴は、従属請求項から明白である。これについての説明を以下に述べる。

【００１５】

本発明のある例示的な態様において、少なくとも一つのデコーダ・ユニット、エンコーダ・ユニット、メモリ、およびターゲット・プロセッサを備えるコンピューティング・システムを提供する。前記デコーダ・ユニットは、サブジェクト・コンピューティング・アーキテクチャに応じて、レジスタ・ウィンドウのサブジェクト・プロセッサによって実行可能な、サブジェクト・コードを復号化するように構成されている。ここで、カレント・レジスタ・ウィンドウは、ウィンドウ表示されたレジスタ・ファイルから、サブジェクト・レジスタの選択されたサブセットを明らかにするように設けられている。サブジェクト・コードは、レジスタ・ファイルに関連して、レジスタ・ウィンドウの位置に影響を与える、ウィンドウに基づいた命令と、レジスタ・ウィンドウ内のレジスタのリファレンスを含む、レジスタに基づいた命令を含む。デコーダ・ユニットはさらに、レジスタ・ウィンドウの動作を発生させることを意図した、サブジェクト・コード内におけるウィンドウに基づく命令を識別して、このウィンドウに基づく命令から、レジスタ・ウィンドウの動きについての情報を導き出すように構成されている。さらにまた、デコーダ・ユニットは、サブジェクト・コード内におけるレジスタに基づく命令を識別して、このレジスタに基づく命令から、一つ以上のウィンドウ表示されたサブジェクト・レジスタ・リファレンスを導き出すように構成されている。メモリは、複数のエントリを記憶するように設けられた、スタック・データ構造を含む。エンコーダ・ユニットは、デコーダ・ユニットによって復号化されたサブジェクト・コードからターゲット・コードを生成するように構成されている。ターゲット・プロセッサは、エンコーダ・ユニットによって生成されたターゲット・コードを実行するように構成されている。ターゲット・プロセッサ上でのターゲット・コードの実行は、スタック・データ構造の先頭に関連したスタック・ポインタを設定して、デコーダ・ユニットによって導き出されたレジスタ・ウィンドウの動きについての情報を参照して、スタック・ポインタを調整し、デコーダ・ユニットによって導き出されたウィンドウ表示されたサブジェクト・レジスタ・リファレンスの各々から決定された変位と組み合わされたスタック・ポインタを参照して、スタック・データ構造内のエントリにアクセスする。

【００１６】

本発明の他の例示的な態様において、レジスタ・ウィンドウ・アーキテクチャをサポートするようにコンピューティング・システムを適合させる方法が提供される。本方法は、レジスタ・ウィンドウに基づいたサブジェクト・コンピューティング・アーキテクチャのサブジェクト・プロセッサによって実行可能な、サブジェクト・コードを復号化すること

を含む。ここで、レジスタ・ウィンドウは、ウィンドウ表示されたレジスタ・ファイルから、サブジェクト・レジスタの選択されたサブセットを明らかにするように位置決めされている。また、本方法は、サブジェクト・コード内の命令から、レジスタ・ウィンドウ内の前記サブジェクト・レジスタのうちの一つに対するリファレンスを含む、ウィンドウ表示されたサブジェクト・レジスタ・リファレンスを識別すること、レジスタ・ウィンドウの動きを発生させる、サブジェクト・コード内の命令から、レジスタ・ウィンドウの動きについての情報を導き出すこと、コンピューティング・システムのメモリ内において、複数のエントリを記憶させるように構成されたスタック・データ構造を提供して、前記メモリ内で、スタック・データ構造の先頭を指示するスタック・ポインタを設定すること、サブジェクト・コードをターゲット・コードに変換して、コンピューティング・システムのターゲット・プロセッサ上でターゲット・コードを実行することと、識別されたレジスタ・ウィンドウの動きについての情報を参照して、スタック・ポインタを調整することと、ウィンドウ表示されたサブジェクト・レジスタ・リファレンスから決定された変位と組み合わされたスタック・ポインタを参照して、スタック・データ構造内のエントリのうちの少なくとも一つにアクセスすることを含む。

【００１７】

本発明のさらに他の例示的な態様において、実行されたときに、レジスタ・ウィンドウ・アーキテクチャをサポートするようにコンピューティング・システムを適合させる、コンピュータで実行可能な命令が記録された、コンピュータで読み取り可能な記憶媒体を提供する。ここで、前記コンピュータで読み取り可能な記憶媒体は、レジスタ・ウィンドウに基づいたサブジェクト・コンピューティング・アーキテクチャのサブジェクト・プロセッサによって実行可能なサブジェクト・コードを復号化するように設けられたコード・ユニットを備える。ここで、カレント・レジスタ・ウィンドウは、ウィンドウ表示されたレジスタ・ファイルから、サブジェクト・レジスタの選択されたサブセットを明らかにするように位置決めされている。また本記憶媒体は、サブジェクト・コード内の命令から、ウィンドウ表示されたサブジェクト・レジスタ・リファレンスを識別することを含む。ここで、前記ウィンドウ表示されたサブジェクト・レジスタ・リファレンスは、レジスタ・ウィンドウ内の前記サブジェクト・レジスタのうちの一つに対するリファレンスを含む。また本記憶媒体は、レジスタ・ウィンドウの動きを発生させる、サブジェクト・コード内の命令から、レジスタ・ウィンドウの動きについての情報を導き出すことと、複数のエントリを記憶させるように構成されたコンピューティング・システムのメモリ内にスタック・データ構造を提供して、前記メモリ内で、スタック・データ構造の先頭を示すスタック・ポインタを設定するように構成されたコード・ユニットと、サブジェクト・コードをターゲット・コードに変換して、コンピューティング・システムのプロセッサ上で、このターゲット・コードを実行させて、識別されたレジスタ・ウィンドウの動きについての情報を参照して、スタック・ポインタを調整して、また、ウィンドウ表示されたサブジェクト・レジスタ・リファレンスから決定された変位と組み合わされた調整されたスタック・ポインタを参照して、スタック・データ構造内のエントリのうちの少なくとも一つにアクセスするように構成されたコード・ユニットとを含む。

【００１８】

本発明の例示的な実施形態は、サブジェクト・コンピューティング・システムのサブジェクト・プロセッサに関するサブジェクト・コードを、次に、ターゲット・コンピューティング・システム上のターゲット・プロセッサによって実行されるターゲット・コードに変換する場合に、特に適用可能となる、レジスタ・ウィンドウ・アーキテクチャをサポートするようにコンピューティング・システムを適合させるメカニズムに関する。このメカニズムは、ターゲット・コンピューティング・システムのメモリ内に、スタック・データ構造（「ＳＲスタック」）を提供する。このＳＲスタックは、複数のフレームを有している。ＳＲスタックの各フレームは、サブジェクト・コードによってアドレス指定されるように、レジスタ・ウィンドウのサブジェクト・レジスタのウィンドウ表示されたサブセットに対応する、１セットのエントリを記憶する。ＳＲスタックはその後、ターゲット・コ

ンピューティング・アーキテクチャ上におけるターゲット・コードの実行によってアクセスされる。ＳＲスタックは、このようなフレームを多量に記憶することが可能であり、サブジェクト・アーキテクチャのウィンドウ表示されたレジスタ・ファイルからのモデリング・オートマチック・スピルや充てん動作などの高額な経費を回避する。

【００１９】

例示的な一実施形態において、ワーキング・レジスタを１６個のみ有するコンピューティング・システムは、数十個、あるいは数百個のハードウエア・レジスタに依存して、ウィンドウ表示されたレジスタ・ファイルを表す、レジスタ・ウィンドウ・アーキテクチャをサポートするように適合される。さらに、例示的な本実施形態は、ターゲット・コンピューティング・システムの有効な動作を可能にし、本システムが、レジスタ・ウィンドウ・アーキテクチャをサポートするように適合させられた場合でさえも、特に、処理速度において、ターゲット・コンピューティング・システムの有効な動作を可能にする。

【図面の簡単な説明】

【００２０】

【図１】従来技術のサブジェクト・コンピューティング・アーキテクチャの一例における、レジスタ・ファイルの一部を示す図である。

【図２】従来技術のサブジェクト・コンピューティング・アーキテクチャの一例における、レジスタ・ファイルをさらに示す図である。

【図３】本発明の例示的な実施形態によって採用されるような、装置を示すブロック図である。

【図４】本発明の例示的な実施形態によって採用されるような、プログラム・コードの変換プロセスの概略概観図である。

【図５】本発明の例示的な実施形態によって提供されるような、レジスタ・ウィンドウ・エミュレーション・メカニズムの概略概観図である。

【図６】図５のレジスタ・ウィンドウ・エミュレーション・メカニズムのさらに詳細な概略図である。

【図７】図５および図６のレジスタ・ウィンドウ・エミュレーションのメカニズムの実施形態の例をさらに示す表である。

【図８】８Ａおよび８Ｂは図５および図６のレジスタ・ウィンドウ・エミュレーションのメカニズムの他の実施形態の例をさらに示す表である。

【図９】本発明の例示的な実施形態による、レジスタ・ウィンドウ・アーキテクチャをサポートするようにコンピューティング・システムを適合させる方法を示す、概略的なフローチャートである。

【図１０】本発明の例示的な実施形態による、コンピューティング・システム内のメモリの選択された部分を示す概略図である。

【図１１】本発明の例示的な実施形態による、転送メカニズムを実行する実施例による方法を示す概略的なフローチャート図である。

【図１２Ａ】さらに詳細な転送メカニズムを示す概略図である。

【図１２Ｂ】さらに詳細な転送メカニズムを示す概略図である。

【発明を実施するための形態】

【００２１】

本発明をより良く理解するために、また、本発明の実施形態がどのようにして、効果的に実施されうるかを示すために、一例として、ここに添付する図表を参照することにする。

【００２２】

以下の説明は、当業者が本発明を作成して用いることが可能となるように提供されており、また、本発明を実施する発明者らによって検討された最良の形態を記載するものである。しかしながら、改良されたプログラム・コード変換方法および装置を提供するために、本発明の一般的な原理が、ここで明確に定義されているため、種々の変形例は、当業者にとって容易に自明であり続ける。

【００２３】
図３を参照すると、サブジェクト・プログラム１７は、サブジェクト・プロセッサ３を有するサブジェクト・コンピューティング・プラットフォーム１上で実行されることが意図されている。ここで、サブジェクト・コンピューティング・プラットフォーム１は、任意の形態の電子装置であってもよい。この装置は、これを操作するサブジェクト・プロセッサ３におけるコンピューティング・オペレーションに依存する。しかしながら、ターゲット・コンピューティング・プラットフォーム１０は、プログラム・コード変換を実行するトランスレータ・ユニット１９を介して、サブジェクト・プログラム１７を実行するために用いられる。ここで、トランスレータ・ユニット１９は、サブジェクト・コード１７をターゲット・コード２１に変換して、その結果、ターゲット・コード２１はその後、ターゲット・コンピューティング・プラットフォーム１０によって実行されることが可能となる。

【００２４】
当業者にとって周知となるように、サブジェクト・プロセッサ３は、１セットのサブジェクト・レジスタ５を有している。サブジェクトメモリ８は、とりわけ、サブジェクト・プログラム１７およびサブジェクト・オペレーティング・システム２を保持している。同様に、図３における例示されたターゲット・コンピューティング・プラットフォーム１０は、複数のターゲット・レジスタ１５を有するターゲット・プロセッサ１３、および、ターゲット・オペレーティング・システム２０と、サブジェクト・コード１７と、トランスレータ・コード１９と、変換されたターゲット・コード２１とを含む複数のオペレーションの構成要素を記憶するメモリ１８とを備える。ターゲット・コンピューティング・プラットフォーム１０は、典型的に、マイクロプロセッサに基づくコンピュータあるいは他の適切なコンピュータである。

【００２５】
一実施形態において、トランスレータ・コード１９は、最適化を用いたり用いなかったりして、サブジェクト命令セット・アーキテクチャ（ＩＳＡ）のサブジェクト・コードを、他のＩＳＡの変換されたターゲット・コードに変換するエミュレータである（「ディス・トゥ・ザット（ｔｈｉｓ－ｔｏ－ｔｈａｔ）」トランスレータとして知られる）。他の実施形態において、トランスレータ１９は、プログラム・コードの最適化を行うことによって、サブジェクト・コードを、各々が同一のＩＳＡのターゲット・コードであるターゲット・コードに変換するように機能する（「ディス・トゥ・ザット」のトランスレータ、あるいは「アクセラレータ」として知られる）。

【００２６】
トランスレータ・コード１９は、適切に、トランスレータを実行するソースコードのコンパイルされたバージョンであり、ターゲット・プロセッサ１３上のオペレーティング・システム２０と連動して動作する。図３に示される構造は、一例にすぎないため、例えば、本発明の実施形態による、ソフトウエア、方法およびプロセスは、オペレーティング・システム２０の中あるいは下に存在するコードで実行されてもよいことが理解されよう。サブジェクト・コード１７、トランスレータ・コード１９、オペレーティング・システム２０、メモリ１８の記憶メカニズムは、当業者にとって既知であるように、多種多様のタイプのうちの任意のタイプであってもよい。

【００２７】
図３による装置において、ターゲット・コード２１の動作中に、プログラム・コード変換が、ランタイムで動的に実施され、ターゲット・アーキテクチャ１０上で実行される。すなわち、トランスレータ１９は、変換されたターゲット・コード２１に沿って動作する。トランスレータ１９を介するサブジェクト・プログラム１７の動作は、交互的な方法で実行する二つの異なるタイプのコード、すなわち、トランスレータ・コード１９とターゲット・コード２１とを含む。ここで、ターゲット・コード２１は、ランタイムを通して、変換されつつあるプログラムの記憶されたサブジェクト・コード１７に基づいて、トランスレータ・コード１９によって生成される。

【００２８】
一実施形態において、トランスレータ・ユニット１９は、サブジェクト・プロセッサ３、特にサブジェクト・レジスタ５などの、サブジェクト・アーキテクチャ１の関連部分をエミュレートする。その一方で、ターゲット・プロセッサ１３上のターゲット・コード２１として、サブジェクト・プログラム１７を実際に、実行する。好適な実施形態において、少なくとも一つのグローバル・レジスタ・ストア２７（サブジェクト・レジスタ・バンク２７あるいは理論レジスタ・バンク２７とも称される）が提供される。マルチプロセッサ環境において、任意に、一つ以上の理論レジスタ・バンク２７が、検討中のサブジェクト・プロセッサのアーキテクチャにしたがって提供される。サブジェクトの状態は、トランスレータ１９と、ターゲット・コード２１との構成要素によって描写される。すなわち、トランスレータ１９は、変数および／またはオブジェクトなどの、種々の明示的なプログラミング言語装置に、このサブジェクトの状態を記憶する。変換されたターゲット・コード２１は、比較すると、ターゲット・コード２１のターゲット命令によって操作された、ターゲット・レジスタ１５およびメモリ位置１８にサブジェクト・プロセッサ状態を暗黙のうちに提供する。例えば、グローバル・レジスタ・ストア２７の低レベルの描写は単に、割り当てられたメモリの領域である。しかしながら、トランスレータ１９のソースコードにおいて、グローバル・レジスタ・ストア２７は、高レベルでアクセスされ、操作されることが可能な、データアレイあるいはオブジェクトである。ターゲット・コード２１を実行することによって、サブジェクト・コード１７の期待された作業が行われ、また、サブジェクト・プロセッサ３のエミュレートされたモデルを更新する。その結果、トランスレータ１９は、エミュレートされた実行コンテキスト（サブジェクト状態）を決定することが可能となり、それに応じて、ターゲット・コードとしてのサブジェクト・プログラムの適切なブロックを動的に選択して、変換して、実行するために、実行の流れを正確に制御することが可能となる。

【００２９】
「基本ブロック」という用語は、当業者にとっては周知のものである。基本ブロックは、そのブロックコードを単一の制御パスに限定する、厳密に一つのエントリポイントおよび厳密に一つのエグジットポイントを備えた、コードのセクションである。このため、基本ブロックは、制御フローの有益な基礎的なユニットである。適切に、トランスレータ１９は、サブジェクト・コード１７を、複数の基礎ブロックに分割する。ここで、各基礎ブロックは、単一のエントリポイントにおける第一の命令と、（ジャンプ、呼び出し、あるいは分岐命令などの）単一のエグジットポイントにおける最後の命令間における、一連の命令のセットである。トランスレータ１９は、これらの基礎ブロック（ブロックモード）を一つだけ選択してもよい、あるいは、基礎ブロック（グループ・ブロックモード）のグループを選択してもよい。グループ・ブロックは、適切に、単一のユニットとして、ともに処理されることになる、二つ以上の基礎ブロックを備える。さらに、トランスレータは、サブジェクト・コードの同一の基礎ブロックを示してはいるが、異なるエントリ条件下にある、ｉｓｏブロックを形成してもよい。

【００３０】
好適な実施形態において、中間表現（ＩＲ）のツリーは、元のサブジェクト・コード１７からターゲット・コード２１を生成する過程の一部として、サブジェクト命令シーケンスに基づいて生成される。ＩＲツリーは、計算された式と、サブジェクト・プログラムによって実行された操作の理論的な描写である。後に、ターゲット・コード２１が、ＩＲツリーに基づいて生成（「植え付けられる」）される。ＩＲノードの収集は、実際には、有向非循環グラフ（ＤＡＧ）であるが、口語的には「ツリー」と称されている。

【００３１】
当業者は、一実施形態において、トランスレータ１９が、Ｃ＋＋などのオブジェクト指向プログラミング言語を用いて実施されることを理解している。例えば、ＩＲノードは、Ｃ＋＋オブジェクトとして実施され、他のノードに対する参照は、これらの他のノードに対応する、Ｃ＋＋オブジェクトに対する、Ｃ＋＋リファレンスとして実施される。したが

って、ＩＲツリーは、お互いに対する種々のリファレンスを含む、ＩＲノード・オブジェクトの収集として実施される。

【００３２】

さらに、検討中の実施形態において、ＩＲ生成は、サブジェクト・プログラム１７がその上で動作することが意図された、サブジェクト・アーキテクチャの特定の特徴に対応する、レジスタ定義の１セットを用いる。例えば、サブジェクト・アーキテクチャ上で、物理的なレジスタごとに、固有のレジスタ定義がある（すなわち、図３のサブジェクト・レジスタ５）。このように、トランスレータにおけるレジスタ定義は、ＩＲノード・オブジェクト（すなわち、ＩＲツリー）に対するリファレンスを含むＣ＋＋オブジェクトとして実施されてもよい。レジスタ定義の１セットによって参照されるすべてのＩＲツリーの集合は、ワーキングＩＲフォレスト（「フォレスト」とは、それが、複数の理論レジスタルーツを含み、そのルーツの各々がＩＲツリーを参照するからである）と称される。これらのＩＲツリーおよび他のプロセスは、適切に、トランスレータ１９の一部を形成する。

【００３３】

図３はさらに、ターゲット・アーキテクチャ１０のメモリ１８における、ネイティブ・コード２８を示している。サブジェクト・コード１７のランタイム変換から生じるターゲット・コード２１と、ターゲット・アーキテクチャに関して、書き込まれる、あるいは直接コンパイルされる、ネイティブ・コード２８との間に区別がある。いくつかの実施形態において、ネイティブ・バインディングは、それが、それに関してサブジェクト・コードのネイティブ・バージョンが存在する、サブジェクト・ライブラリなどの、サブジェクト・コード１７のセクションに、制御のサブジェクト・プログラムの流れが入ることを検出したときに、トランスレータ１９によって実施される。サブジェクト・コードを変換することよりむしろ、トランスレータ１９は、同等のネイティブ・コード２８が、ターゲット・プロセッサ１３上で実施されるようにさせる。実施形態において、トランスレータ１９は、参照してここに組み込まれる、公開された国際公開第２００５／００８４７８号パンフレットにおいてさらに詳細に記載されているように、ネイティブ・コードあるいはターゲット・コードの呼び出しスタブなどの、定義されたインターフェースを用いて、生成されたターゲット・コード２１を、ネイティブ・コード２８に対してバインドする。

【００３４】

図４は、ターゲット・コンピューティング・プラットフォーム１０上で動作する場合のトランスレータ１９を、さらに詳細に示している。上記のように、トランスレータ１９の前端は、サブジェクト・プログラム１７の現在必要とされるセクションを復号化するデコーダ・ユニット１９１を含み、複数のサブジェクト・コード・ブロック１７ａ、１７ｂ、および１７ｃ（これらは、通常、各々がサブジェクト・コードの一つの基礎ブロックを含む）を提供し、各サブジェクト・ブロックと、トランスレータ１９の後の動作を支援するであろう、そこに含まれるサブジェクト命令とに関連して、デコーダ情報１７１も提供してもよい。いくつかの実施形態において、トランスレータ１９のコア１９２におけるＩＲユニットは、復号化されたサブジェクト命令から、中間表現（ＩＲ）を生成して、最適化は、中間表現に関連して、適切に実行される。トランスレータ１９の後端の一部としてのエンコーダ１９３は、ターゲット・プロセッサ１３によって実行可能なターゲット・コード２１を生成（植え込み）する。この単純な実施例において、三つのターゲット・コード・ブロック２１ａから２１ｃが生成されて、サブジェクト・プラットフォーム１上でサブジェクト・コード・ブロック１７ａから１７ｃを実行することに相当する、ターゲット・プラットフォーム１０上に作用する。また、エンコーダ１９３は、ターゲット・ブロックが動作して、適切な場合には、制御をトランスレータ１９に戻すような環境を設定するような機能を実行する、ターゲット・コード・ブロック２１ａから２１ｃのうちのいくつか、あるいはすべてに関して、ヘッダ・コードおよび／またはフッタコード２１１を生成してもよい。一実施形態において、トランスレータ１９は、それに関して、ターゲット・コード２１ａから２１ｃが既に生成されている、サブジェクト・コード・ブロック１７ａから１７ｃの記録を維持している。このように、サブジェクト・コードの同一のブロックが

プログラムにおいて、後に、再び遭遇されるときに、前もって生成されたターゲット・コードはフェッチされて、再利用されることが可能である。しかしながら、本発明の他の実施形態は、他の特定なメカニズムを利用して、動的に、サブジェクト・コードを、生成されたターゲット・コードに整合させてもよい。

【００３５】

図４において、サブジェクト・コード１７は、適切に、トランスレータ１９によって変換されてターゲット・システム上で動作するようになる、アプリケーション・プログラムである。一般的な実施例として、このアプリケーション・プログラム１７は、特に、ウェブ・サーバ、デジタル・コンテンツ・サーバ（例えば、ストリーミング・オーディオ、あるいはストリーミング・ビデオ・サーバ）、ワードプロセッサ、スプレッドシート・エディタ、グラフィック・イメージ編集ツール、あるいはデータベース・アプリケーションなどの複雑なプログラムである。しかしながら、他の実施例においては、サブジェクト・コード１７は、コンピューティング・システムが、電子装置の有益な作業と制御動作とを実行することを可能にさせる、任意の種類のプログラムである。オペレーティング・システム２０とトランスレータ１９とに関連したこのようなタスクなどの他のタスクに加えて、ターゲット・コンピューティング・プラットフォーム１０は、多くの場合、このような多くのプログラムを同時に動作させることが要求される。

【００３６】

実施形態において、サブジェクト・コード１７は、サブジェクト・アーキテクチャ１に固有に作成された（例えば、コンパイルされた）バイナリに実行可能な形態を取る。サブジェクト・コード１７に関して、人間が介入する、あるいは再調査する機会はない。その代わり、ターゲット・コンピューティング・プラットフォーム１０は、ターゲット・コンピューティング・プラットフォーム１０上でバイナリに実行されるように、トランスレータ１９を介して、自動的に、サブジェクト・コード１７を、ターゲット・コード２１に変換する。このように、例示的な実施形態において、トランスレータ１９は、サブジェクトＩＳＡのバイナリに実行可能なコードとしてのサブジェクト・コード１７を、ターゲットＩＳＡのバイナリに実行可能なコードとしてのターゲット・コードに変換する、バイナリ・トランスレータである。さらに、トランスレータ１９は、変換の合間でターゲット・コード２１のブロックの実行を行う動的なバイナリ・トランスレータである。

【００３７】

図５は、本発明の実施形態に採用されるような、ターゲット・コンピューティング・プラットフォーム１０において、レジスタ・ウィンドウ・アーキテクチャを有するサブジェクト・プロセッサ３の主要な構成要素をエミュレートするメカニズムを示している。上記のように、サブジェクト・プロセッサのこれらの構成要素をエミュレートすることによって、トランスレータ１９は、サブジェクト・コード１７の予想される挙動を忠実に模倣する、ターゲット・コード２１を生成することが可能となる。以下の実施形態において、ターゲット・プロセッサ１３のアーキテクチャは、レジスタ・ウィンドウを提供しないか、あるいは、サブジェクト・プロセッサ１３と比較して、異なる形状のレジスタ・ウィンドウを用いる。したがって、サブジェクト・コード１７は、ターゲット・プロセッサ１３上では動作することができない。実施例として、トランスレータ１９は、１６個の汎用ハードウエア・レジスタのみを有する、６４ビットのｘ８６タイプのプロセッサ上のターゲット・コードとして動作する６４ビットの汎用ハードウエア・レジスタを、少なくとも６４個、さらに典型的には数百個有する、ｖ９　ＳＰＡＲＣプロセッサ上での実行を意図した、サブジェクト・コードを変換するように構成される。この例によって示されるように、サブジェクト・プロセッサ３およびターゲット・プロセッサ１３は、利用可能なレジスタの数およびタイプに関して、また、各プロセッサにおいてレジスタが構成される方法に関して、基本的には互換性がない場合がある。

【００３８】

サブジェクト・プロセッサ３の一般的な構造と動作は当業者にとっては周知のものとなるはずであるが、ターゲット・システムによってエミュレートされる、サブジェクト・プ

ロセッサのこれらの構成要素を示して、説明するために、ここで、それらについて簡単に検討してみる。ここで説明された実施形態において、サブジェクト・プロセッサ３は、一般的に、図１および図２を参照して、上記に説明された、ｖ９　ＳＰＡＲＣアーキテクチャの例にしたがって、レジスタ・ウィンドウの配置を採用する。この場合、レジスタ・ウィンドウの位置は重複し、回転構造を有している。しかしながら、本発明の実施形態は、重複しない、および／または回転しない構造などの、他の特定の構造を有する、レジスタ・ウィンドウ・アーキテクチャについても実施されてよいことが理解されよう。このレジスタ・ウィンドウ・アーキテクチャについての、いくつかの従来技術の説明は、複数のレジスタ・ウィンドウに言及しており、それらのうちの一つだけが、カレント・ウィンドウである（すなわち、各々が固定位置を有し、そのうちの一つだけがどの時期においても開かれる、多くのウィンドウが存在する）。一貫性を保つために、以下の説明は、異なるレジスタ・ウィンドウの位置に移動する、単一のカレント・ウィンドウについて検討しているが、当業者は、本発明が、複数の固定されたウィンドウに関して定義されたアーキテクチャにも同等に利用可能であることを容易に理解するであろう。

【００３９】

図５は、サブジェクト・プロセッサ３内に設けられた、多数の汎用ハードウエア・レジスタ５を示している。説明を明確にかつ簡単にするために、コントロール／ステータス・レジスタと、サブジェクト・プロセッサ３の他の多くの部分は示されていない。これらのサブジェクト・レジスタ５は、サブジェクト・コード１７のすべてのセクションに対して静的であり、視認可能である、グローバルに視認可能なレジスタ５０１の小さなセット（「グローバル・サブジェクト・レジスタ」）と、複数のレジスタ・ウィンドウ位置５１１のうちの一つに移動する、カレント「レジスタ・ウィンドウ」５１０として、サブジェクト・コード１７の現在実行中の部分にとって、このレジスタ・ファイルのうちの一つのサブセットのみが視認可能であるように、ウィンドウ表示された、レジスタ５０２の大きなファイル（「ウィンドウ表示されたサブジェクト・レジスタ」）を含む。このように、レジスタ・ウィンドウ５１０の現在位置の下に存在する、ウィンドウ表示されたサブジェクト・レジスタ５０２のサブセットのみが、任意の時期にコードを実行することによってアクセス可能となる。

【００４０】

多数のレジスタがサブジェクト・ハードウエア内に設けられてはいるものの、サブジェクト・コード１７は、どの時期においても、３２個の視認可能な汎用レジスタすなわち、８個のグローバル・サブジェクト・レジスタ５０１（ｇ０からｇ７）と、２４個のレジスタ・ウィンドウ５１０を、ウィンドウ表示されたサブジェクト・レジスタ５０２（ｉ０からｉ７、Ｉ０からＩ７、ｏ０からｏ７）にアドレス指定することのみが可能である。したがって、サブジェクト・コードは、この３２個の、視認可能なレジスタ名のセットに関して、書き込まれる。

【００４１】

ターゲット・コンピューティング・プラットフォーム１０の検討に戻ると、サブジェクト・コード１７は、ターゲット・システムのメモリ１８における利用可能な領域にサブジェクト・コード１７をロードすることなどによって提供され、ブロックごとに、サブジェクト・コード１７は、ターゲット・コード２１として変換され、実行される。上記のように、トランスレータ１９が、最初に、サブジェクト・コードのブロックに遭遇したとき、デコーダ・ユニット１９１は、サブジェクト命令を復号化する。この復号化プロセスは、サブジェクト・アーキテクチャの汎用サブジェクト・レジスタに対する、サブジェクト・コード命令内のリファレンスを識別することを含み、特に、ウィンドウ表示されたサブジェクト・レジスタ５０２およびグローバル・サブジェクト・レジスタ５０１に対するリファレンスを含む。第二に、サブジェクト・コード命令が識別されて、カレント・レジスタ・ウィンドウの、新しい位置へのＳＡＶＥおよびＲＥＳＴＯＲＥタイプの動きを引き起こす。ここで、例示されたｖ９　ＳＰＡＲＣハードウエアに関する命令セット・アーキテクチャは、少なくとも「保存」および「復旧」の命令を含み、これらの命令は、サブジェク

ト・プラットフォーム上で、カレント・レジスタ・ウィンドウ５１０を、位置５１１の他の一方に移動させる。レジスタ・リファレンスおよびデコーダ１９１によって取得されたレジスタ・ウィンドウの動きに関する情報は、コア１９２に渡され、エンコーダ１９３内で、ターゲット・コード２１を生成するために用いられる。

【００４２】

初期段階において、トランスレータ１９は、後に、サブジェクト・プロセッサをエミュレートするために用いられる、種々のメモリ構造を提供する。特に、トランスレータ１９は、上記するように、理論レジスタ・バンク２７すなわち、グローバル・サブジェクト・レジスタ５０１とウィンドウ表示されたサブジェクト・レジスタ５０２とを含む、サブジェクト・プロセッサ３のレジスタ５内に保持されている値を記憶するために用いられるデータ構造を提供する。ここで、メモリ領域４２０は、８個の静的な位置４２１のセットを有するターゲット・メモリ１８において、８個のグローバル・サブジェクト・レジスタ５０１に関連する、理論レジスタ・バンク２７の静的な部分を形成するように定義される。グローバル・サブジェクト・レジスタ５０１に対するリファレンスが、デコーダ・ユニット１９１によって復号化されるサブジェクト・コード命令において識別される場合、等価ターゲット・コード命令が、メモリ領域４２０内のこれらの静的な位置４２１に対する適切なリファレンスによって生成される。すなわち、これらのサブジェクト・グローバル・レジスタ５０１のコンテンツを表現しているデータの値は、最も一般的には、静的な位置４２１からのデータを、ターゲット・プロセッサ１３のワーキング・レジスタ１５にロードして、それから、ターゲット・コードの実行中に適切なものとなるように、結果を、これらのメモリ位置４２１に戻して記憶させることにより、ターゲット・コード２１によって用いられる。このように、ターゲット・コード２１は、グローバル・サブジェクト・レジスタ５０１に依存する、サブジェクト・コード１７のこれらの命令の挙動をエミュレートする。

【００４３】

ウィンドウ表示されたサブジェクト・レジスタ５０２は、上記のように、動的なウィンドウ表示される構成において機能し、ここで、分離メカニズムが、ターゲット・プラットフォーム内に提供されて、これらのサブジェクト・レジスタをエミュレートする。図５に示されるように、このレジスタ・ウィンドウ・メカニズムを実施するために、トランスレータ１９は、ターゲット・プラットフォーム１０のメモリ１８内に、スタック・データ構造を提供する。これは、「ＳＲスタック」４００として後述される。当業者にとって周知であるように、スタックは、最も一般的に利用可能なコンピューティング・アーキテクチャにおいて有効に作成され、管理される、ＬＩＦＯ（後入れ先出し）タイプのメモリ構造である。典型的には、スタックは、所定の基準アドレス（開始アドレス）においてメモリ内に存在し、データがスタックに付加される、あるいはスタックから除去されるにつれて、メモリ内に下方に（あるいは上方に）成長する。スタックのトップ（先頭）のカレント位置は、スタック・ポインタに対するリファレンスによって決定され、このスタック・ポインタは、通常は、メモリ位置に関するアカウントにアップデートされる。メモリ位置は、データがスタック上に押し出されるときに消費され、逆に、データがスタックから飛び出すときに、開放される。一般的に、スタック上のデータは、スタック・ポインタに関連するメモリをアドレス指定することによって操作可能となる。以下の例では、ＳＲスタック４００は、ターゲット・メモリ１８内の所定の基準アドレスＳＲ＿ＢＡＳＥから、下方に（すなわち、徐々に減少するメモリアドレスを用いて）成長し、スタック・ポインタＳＲ＿ＳＰが用いられて、ＳＲスタックのカレントヘッドを決定する。

【００４４】

ＳＲスタック４００が用いられて、サブジェクト・コード１７によって参照されるように、各々がウィンドウ表示された、サブジェクト・レジスタ５０２の一つのコンテンツを表現する、データの値を記憶する。すなわち、サブジェクト・コード１７の復号化によって、サブジェクト・コード１７によってアクセスされる、ウィンドウ表示されたサブジェクト・レジスタ５０２が、実行中に明らかになり、サブジェクト・コードにおいて用いら

れたデータの値が、今度は、ＳＲスタック４００内のエントリ４０１として、記憶される。ターゲット・コード２１が生成されて、ＳＲスタック４００上のこれらのデータの値を、ターゲット・メモリ１８に記憶させ、これらのデータの値を、ターゲット・コード２１の必要に応じて、ターゲット・プロセッサ１３のワーキング・レジスタ１５にロードして、ターゲット・コードの実行中、必要に応じて、その結果をＳＲスタック４００に戻して記憶する。ＳＲスタック４００は、このように、理論レジスタ・バンク２７の一部を形成して、ウィンドウ表示されたサブジェクト・レジスタ５０２をエミュレートして、ターゲット・コード２１が、ウィンドウ表示されたサブジェクト・レジスタ５０２に依存する、サブジェクト・コード命令の実行をエミュレートすることを可能にする。

【００４５】

図５に示されるような、実施形態において、ＳＲスタック４００は、複数のフレーム４１０に分割される。ここで、各フレーム４１０は、１セットのデータエントリ４０１を、含んでいる。この例において、２４個のウィンドウ表示されたサブジェクト・レジスタ５０２のアドレス可能なセットは、サブジェクト・コード１７によって、「ｉ０からｉ７」、「Ｉ０からＩ７」、「ｏ０からｏ７」と（あるいは、ｒ〔８〕からｒ〔３１〕であっても同じである）称される。したがって、サブジェクト・コードにおけるレジスタ定義およびレジスタ・リファレンス（ここでは、まとめて「レジスタ・リファレンス」と称する）は、サブジェクト・コード１７の観点から、このアドレス指定可能なレジスタ名のセットによって表現される。ここで説明されるエミュレーション・メカニズムにおいて、サブジェクト・コード１７によってサブジェクト・レジスタ５０２内に存在するように要求されるこれらのデータの値は、ＳＲスタック４００上の２４個のエントリ４０１の一つのフレーム４１０内に記憶される。

【００４６】

第二に、サブジェクト・プロセッサ３において、サブジェクト・コード１７が、カレント・レジスタ・ウィンドウ５１０を新しい位置５１１に移動させる場合、本エミュレーション・メカニズムにおいて、エントリ４０１の新しいフレーム４１０が、ＳＲスタック４００上に提供される。ウィンドウ表示されたサブジェクト・レジスタ５０２の重複する性質によって、この実施形態は、フレーム４１０ごとに、ＳＲスタック４００上に１６個の新しいエントリ４０１を提供する。一方、８個の旧式なエントリ（前述のフレーム４１０の「ｏ０からｏ８」に対応する）は、カレント・フレームの「ｉ０からｉ８」のレジスタ・リファレンスに関して、データの値を付与する。

【００４７】

図６は、ここで説明される、レジスタ・ウィンドウ・エミュレーション・メカニズムのさらに詳細な概略図であり、ＳＲスタックのアドレス指定に関するメカニズムの例を示す。下記の説明において、代表的な擬似コード命令は、サブジェクト・コード１７ａ、１７ｂ、およびターゲット・コード２１ａ、２１ｂのセクションとして提供され、命令は、ここでは、明確にするために、簡略化されたアセンブラタイプの擬似コードとして示されている。サブジェクト・コードの例は、一般的に、ＳＰＡＲＣ　ｖ９　ＩＳＡ（すなわち、オペレーションソース、デスティネーション）に基づいているが、遅延スロットおよび特定の命令引数などの詳細な部分は、明確にするために省略されている。同様に、ターゲット・コードの例は、一般的に、インテル構文（すなわち、オペレーション・デスティネーション、ソース）を用いた、Ｌｉｎｕｘアセンブリ言語に基づく、アセンブラタイプの擬似コードとして示されている。もちろん、動的なバイナリ・トランスレータに関連する、本発明の実施形態は、サブジェクト・コード１７を、バイナリ・マシン・コードとして受け取り、ターゲット・コードを、バイナリ・マシン・コードとして生成するが、説明を簡単にするために、アセンブラタイプの擬似コードが提供された。図６における代表的な擬似コード命令もまた、図７および図８に関して、さらに詳細に述べられるであろう。

【００４８】

図６に示すように、トランスレータ１９のデコーダ・ユニット１９１は、第一および第二のサブジェクト・コード・ブロック１７ａおよび１７ｂにおいて、命令を復号化する。

好都合なことに、この第一の実施形態は、サブジェクト・コードを、「保存」（ｓａｖｅ）および「復旧」（ｒｅｓｔｏｒｅ）のサブジェクト命令の位置にしたがって、第一および第二のブロックに分割する。実施形態において、サブジェクト・コード１７ａおよび１７ｂの各ブロックは、典型的に、およそ１０個の命令を含むが、およそ１００個までの命令を含むこともあるであろう。

【００４９】

デコーダ１９１は、ウィンドウ表示されたサブジェクト・レジスタ５０２に対する、サブジェクト・コードリファレンス１７５を識別して、レジスタ・ウィンドウの動きに関する情報１７６を導き出す。この代表的な擬似コードにおいて、第一のサブジェクト・ブロック１７ａは、二つの「移動」（ｍｏｖｅ）命令と、ウィンドウ表示されたサブジェクト・レジスタ「Ｉ７」、「Ｉ３」、および「ｏ６」（すなわち、ローカル３、ローカル７、および出力６）に対するリファレンス１７５を含む一つの「追加」（ａｄｄ）命令を含む。レジスタ・ウィンドウの動きに関する情報１７６に関して、「保存」（ｓａｖｅ）命令は、ここで、ＳＡＶＥタイプの動きを起こすこととして識別される。同様に、第二のコード・セクション１７ｂは、「Ｉ３」（ローカル３）に対する他のリファレンス１７５を識別するように復号化され、また、「復旧」（ｒｅｓｔｏｒｅ）命令は、ＲＥＳＴＯＲＥタイプのレジスタ・ウィンドウの動き１７６を誘導するように識別される。この代表的な擬似コードにおける「戻す」（ｒｅｔｕｒｎ）は、この図には示されていない、サブジェクト・コードの他のいくつかの呼び出し側セクションに戻る。

【００５０】

この場合、トランスレータ１９によって生成されるターゲット・コード２１は、ターゲット・コード・ブロック２１ａおよび２１ｂによって図示されている。ここで、トランスレータ１９は、ターゲット・コードの命令を生成して、少なくとも部分的に、識別されたサブジェクト・レジスタ・リファレンスに基づいて、関連するエントリ４０１を、ＳＲスタック４００上にアドレス指定する。

【００５１】

この第一の実施形態において、関連するＳＲスタック・エントリ４０１は、識別された、ウィンドウ表示されたサブジェクト・レジスタ・リファレンス１７５を検討することによって、スタック・ポインタＳＲ＿ＳＰについてアドレス指定される。スタック・ポインタＳＲ＿ＳＰは、好都合に、各サブジェクト・ブロックの端部で認識された、識別されたレジスタ・ウィンドウの動きに関する情報１７６に応じて、各ターゲット・ブロックの端部でアップデートされる。すなわち、ターゲット・コード２１が生成されて、デコーダ１９１によって取得された、ＳＡＶＥおよびＲＥＳＴＯＲＥのレジスタ・ウィンドウの動きに関する情報１７６に応じて、ＳＲスタックの先頭のカレント位置を追跡するように、ＳＲスタック・ポインタＳＲ　ＳＰをアップデートする。

【００５２】

ここで、生成されたターゲット・コード２１は、ＳＲスタック・ポインタＳＲ＿ＳＰを、識別されたＳＡＶＥタイプの動きごとに、１６個の６４ビットのエントリ４０１によって、ＳＲスタック４００に向かって下方に進むようにアップデートさせる、あるいは、識別されたＲＥＳＴＯＲＥタイプの動きごとに、１６個のエントリによって、後退するようにアップデートさせる、ターゲット命令を含む。ここで、スタック・ポインタは、各フレーム４１０内で、「ｉ」（入力）および「ｏ」（出力）を表現するエントリの重複のために、２４個のエントリのフルフレームではなく、１６個のエントリによってアップデートされる。図６の例によって示されるように、ＳＲ＿ＳＰは、第一のフレーム４１０ａに関する、第一のターゲット・コード・ブロック２１ａについて、第一の値「ＳＲ＿ＳＰ１」に対して設定され、また、第二のターゲット・コード・ブロック２１ｂが、第二のフレーム４１０ｂにおけるエントリ４０１を参照するように、第二の値「ＳＲ＿ＳＰ２」にアップデートされる。

【００５３】

また図６に示されるように、第一の実施形態において、要求されたＳＲスタック・エン

トリ４０１が、スタック・ポインタＳＲ＿ＳＰのカレント値によって参照されるように、ＳＲスタック４００の先頭に設けられた、２４個のエントリ４０１のカレント・フレーム４１０において認識される。ここで、各レジスタ・リファレンス１７５は、スタック・ポインタＳＲ＿ＳＰのカレント値に関する、所定の変位を決定する。簡易な実施例として、「ｏ０」に関するエントリが０の変位を有するように取ることによって（すなわち、スタック・ポインタＳＲ＿ＳＰは、スタック上の第一のエントリとして、「ｏ０」を参照する）、その場合には、「ｏ１」および「ｏ２」に関するエントリはそれぞれ＋１および＋２の６４ビットのワードの変位を有し、「ｉ７」に関するエントリが＋２３の６４ビットのワードの変位を有するまで同様である。ここで、ターゲット・コード２１は、スタック・ポインタのカレント値を、要求された変位と組み合わせることによって、スタック上で、要求されたエントリのアドレスを計算する。

【００５４】

好適に、ターゲット・レジスタ１３の一つが、スタック・ポインタＳＲ＿ＳＰを保持するように選択され、コンテキストを、ターゲット・コードに渡すときに、スタック・ポインタのカレント値をロードしている。代表的な６４ビット　ｘ８６のターゲット・アーキテクチャ上で、スタック・ポインタは、ｒｂｐなどの一時的なレジスタの一つに好都合にロードされる。すなわち、一つの選択肢として、スタック・ポインタＳＲ＿ＳＰが、ターゲット・コード２１ａの図示された第一のセクションの呼び出しに先立ち、トランスレータ１９のランループにおいて、ターゲット・レジスタｒｂｐにロードされる。あるいは、好都合に、ｒｂｐは、ヘッダ・コード２１１（図４に示される）の一部としてロードされる。このように、スタック・ポインタＳＲ＿ＳＰはその後、レジスタｒｂｐから、ターゲット・コード２１ａおよび２１ｂのセクションにおいて、用いられることが可能となる。この動作を行う代表的な命令が下記に示される。ここで、「ｆｓ：ＳＲ＿ＳＰ」は、トランスレータ１９によって保持される、サブジェクト状態の一部として、コンテキストの切替中、スタック・ポインタを記憶する、メモリ位置に対するリファレンスである。すなわち、

ｍｏｖ　ｒｂｐ，（ｆｓ：ＳＲ＿ＳＰ）

以下の例は、ＳＲスタック上での変位の計算を示している。ここで、サブジェクト・コードにおいて、サブジェクト・グローバル・サブジェクト・レジスタ「ｇ１」の一つが、ローカルのウィンドウ表示されたサブジェクト・レジスタ「Ｉ５」（ローカルＩ５）にロードされる。ここで、「Ｉ５」エントリは、「＋１３」の変位に設けられる、すなわち、１３個の６４ビットのワード（これは、１３＊８＝１０４、８ビットバイトにおいてアドレス指定を有するいくつかのアーキテクチャにおいて、８ビットバイトとして好都合に表現される）。ターゲット・コードにおいて、「ｇ１」に関する値が、既に、メモリ領域４２０から、利用可能なターゲット・レジスタｒａｘにロードされていると仮定することができる。すなわち、

サブジェクト・コード　ターゲット・コード

ｍｏｖ　ｇ１、Ｉ５　　ｍｏｖ（ｒｂｐ＋１３），ｒａｘ

識別されたＳＡＶＥの動き１７６に応じて、ｒｂｘに保持されたカレント・スタック・ポインタ値ＳＲ＿ＳＰは、ＳＲスタック４００上で、１６個の６４ビットのエントリによって前進させられる。これは、１６個の新たなエントリ４０１を、ＳＲスタックに付加するか、あるいは、カレント・フレーム４１０を、前もって占有された位置にまで移動させるであろう。こうして、これらのメモリ位置において記憶された、データの値を明らかにする。第二に、これは別々に行うことが可能ではあるものの、この時点で、ｆｓ：ＳＲ＿ＳＰにおけるメモリ内に保持された、ＳＲ＿ＳＰのバージョンをアップデートすることも有益である。

【００５５】

スタック・ポインタＳＲ＿ＳＰは、以下のターゲット命令などによって、各ＳＡＶＥタイプの動きに関する情報１７６に応じて、メモリ内で下方に（この例では、１６個の６４ビットのワード分だけ）前進させられる。すなわち、

ａｄｄ　ｒｂｐ，－１６
ｍｏｖ（ｆｓ：ＳＲ＿ＳＰ），ｒｂｐ
反対に、スタック・ポインタＳＲ＿ＳＰは、以下のターゲット命令などによって、各ＲＥＳＴＯＲＥタイプの動きに関する情報１７６に応じて、メモリ内で上方に遅らせられて、その後、特定の変位が、ＳＲスタック４００上で、エントリ４０１の前もって遭遇したフレーム４１０を参照できるようにする。すなわち、
ａｄｄ　ｒｂｐ，＋１６
ｍｏｖ（ｆｓ：ＳＲ＿ＳＰ），ｒｂｐ
図７は、上記の図６における実施形態をさらに詳細に示した表である。すなわち、図７は、上記のように、サブジェクト・コード・ブロック１７ａおよび１７ｂを示している。また、図７は、サブジェクト・コード命令を復号化することによって識別された、識別サブジェクト・レジスタ・リファレンス１７５を示しており、また、識別されたリファレンス１７５ごとに、関連する変位１７７を示している。さらに、図７に示されるように、サブジェクト・コード１７の復号化によって、識別されたレジスタ・ウィンドウの動きに関する情報１７６が導き出される。最終的に、図７は、レジスタ・リファレンス１７５およびレジスタ・ウィンドウの動きに関する情報１７６を参照して、サブジェクト・コード命令から生成される、ターゲット・コード・ブロック２１ａおよび２１ｂを示している。

【００５６】

図７に示されるように、第一のターゲット・コード・ブロック２１ａは、ヘッダ・コード２１１から始まる。この例において、ヘッダ・コード２１１は、スタック・ポインタＳＲ＿ＳＰを、一時的なターゲット・レジスタｒｂｐ内にロードする命令（図示せず）を含む。この点において、ｒｂｐが、値「ＳＲ＿ＳＰ１」を保持して、図６のフレーム４１０ａを参照すると仮定することができる。次に、ターゲット・レジスタｒａｘおよびｒｂｐは、定数「１０」および「３」をロードされており、これらの値も、レジスタ・リファレンス１７５から導き出された変位１７７について参照された、ＳＲスタック４００上のエントリ内に記憶されている。このように、ＳＲスタック４００上の関連エントリは、参照されたレジスタ「Ｉ７」および「Ｉ３」におけるサブジェクト・コードによって期待された、データの値を保持している。これらのターゲット・レジスタｒａｘおよびｒｂｘが、その後、用いられて、代表的な「付加」サブジェクト命令と均等な作業を実行して、その結果は、「ｏ６」のリファレンスに関連したエントリ４０１に記憶される。この実施形態における「保存」サブジェクト命令は、第一のサブジェクト・ブロック１７ａを終了させる。また、この「保存」から導き出されたレジスタ・ウィンドウの動きに関する情報１７６に応じて、ターゲット・コード２１ａが生成されて、ｒｂｐ内に保持されたスタック・ポインタＳＲ＿ＳＰを、ＳＲスタック４００上で１６個のエントリ分、進むように修正する。この結果、ＳＲ＿ＳＰは、値「ＳＲ＿ＳＰ２」を有するようになり、図６におけるフレーム４１０ｂを参照する。この第一のターゲット・ブロック２１ａにおけるフッタコード２１１は、変換されたターゲット・コードとして実行するサブジェクト・コードの次のブロックを選択するために、制御をトランスレータ１９に渡すこと、あるいは、生成済みのターゲット・コードの次のブロックに直接、制御を渡すことなどの、次の行動を決定する。この例において、フッタコード２１１は、実行の制御を、トランスレータ１９に戻すことなく、第二のターゲット・コード・ブロック２１ｂに直接、渡す。

【００５７】

第二のターゲット・コード・ブロック２１ｂにおけるヘッダ・コード２１１は、適切に、このブロックのためのコンテキストを準備する。大部分の場合、これは、ｆｓ：ＳＲ＿ＳＰにおけるメモリから、ｒｂｐに、ＳＲ＿ＳＰをロードさせることを含む。この例においては、ターゲット・コードの最適化は、既にｒｂｐにある値が、ブロック２１ａから単純に前に移動されるようにさせる。次に、ターゲット・コードは、サブジェクト・ブロック１７ｂからのサブジェクト「追加」（ａｄｄ）命令の作業を実行する。ｒｂｐにおけるＳＲ＿ＳＰのアップデートされた値のために、「Ｉ３」（ローカル３）に対するレジスタ・リファレンスは、第一のサブジェクト・コード・ブロック１７ａにおいてのように、Ｓ

Ｒスタック４００内の同一エントリ４０１に分解されないことに留意されたい。すなわち、サブジェクト・ブロック１７ａにおける、「Ｉ３」に対するリファレンスは、フレーム４１０ａ内のエントリに分解されるのに対して、サブジェクト・ブロック１７ｂにおける、「Ｉ３」に対するリファレンスは、フレーム４１０ｂ内のエントリに分解される。このように、ＳＲスタック４００は、サブジェクト・プロセッサにおけるレジスタ・ウィンドウの期待された挙動をエミュレートする。最終的に、サブジェクト・ブロック１７ｂ内の最終的な「復旧」（ｒｅｓｔｏｒｅ）からのレジスタ・ウィンドウの動きに関する情報１７６に応じて、ターゲット・コード２１ｂが生成され、ｒｂｐ内でＳＲ＿ＳＰの値をアップデートして、１６個のエントリ分、スタック・ポインタを遅らせる。それによって、スタック・ポインタは、再度、図６のフレーム４１０ａを参照することになる。また、サブジェクト「復旧」（ｒｅｓｔｏｒｅ）命令は、この第二のサブジェクト・ブロック１７ｂの終端点を選択するように作用する。大部分の場合、この「復旧」（ｒｅｓｔｏｒｅ）命令は、トランスレータに、サブジェクト・ブロック１７ｂを終了させる。しかしながら、この実施例において、サブジェクト・ブロックは、実際には、「復旧（ｒｅｓｔｏｒｅ）」の後に、「戻す」（ｒｅｔｕｒｎ）においてただちに終了する。この点で、ｒｂｐからｆｓ：ＳＲ＿ＳＰに戻された、アップデートされたスタック・ポインタＳＲ＿ＳＰを保存して、フッタコード２１１が、この場合、サブジェクト・プログラムの実行を継続する「ｆｕｎｃ１」として、第一のサブジェクト・ブロック１７ａを呼び出したサブジェクト・コード（図示せぬ）に制御を戻すような、トランスレータ１９に実行制御を戻すなどの、次の動作を決定することが適切である。

【００５８】

図８は、図６および図７の上記の例に類似した、他の表であり、ＳＲスタック４００をアドレス指定するための、第二の代表的なメカニズムである。

トランスレータ１９のデコーダ・ユニット１９１は、上記のように、ウィンドウ表示されたサブジェクト・レジスタ・リファレンス１７５およびレジスタ・ウィンドウの動き１７６を識別する。さらに、識別されたレジスタ・ウィンドウの動きに関する情報１７６は、スタック・ポインタＳＲ＿ＳＰからのフレーム・オフセットとして、ターゲット・コード２１において表現される、フレーム・オフセット１７８を導き出すために用いられる。それから、レジスタ・リファレンス１７５の各々は、このフレーム・オフセット１７８から変位１７７を提供して、ＳＲスタック４００内の関連するエントリ４０１をアドレス指定する。このため、スタック・ポインタＳＲ＿ＳＰは、エントリ４０１の二つ以上のフレーム４１０が、ブロック中で検討中であるにもかかわらず、ターゲット・コード・ブロック２１ａを通して一定であり続ける。図８に示された第二の実施形態は、特に、トランスレータが、リーフ関数をターゲット・コードの単一のブロック内にインライン化することを可能にさせる。ここで、リーフ関数とは、それ自体は、他の機能を呼び出さない機能である。その機能は、呼び出し側と、呼び出される側の両方のサブジェクト・コード命令を、単一のブロックサブジェクト・コード１７ａとして処理して、そこから、ターゲット・コード２１ａの対応する単一のブロックを作成することによって、インライン化される。図８のこの簡易な例においては、リーフ関数はないが、後記の動作が、同様に、この目的に当てはまる。

【００５９】

さらなる実施形態において、デコーダ１９１が、サブジェクト・コード１７のセクションが、所定の数より多いＳＡＶＥあるいはＲＥＳＴＯＲＥを含んでいることを検出して、それから、その時点で、ブロックの復号化が停止され（中止され）、新しいブロックが、デフォルト値（「ｆ０」）に戻りつつあるフレーム・オフセットによって作成される。すなわち、サブジェクト・コードのセクションは、ＳＡＶＥあるいはＲＥＳＴＯＲＥを所定の数より多く含み、サブジェクト・コードのこのセクションは、付加的なサブジェクト・コード・ブロックを作成することによって、２以上のより小さいセクションに再分割される。この制限は、他の比較的長いブロックを中止させるためには好都合である。

【００６０】

図８Ａに示されるように、識別されたレジスタ・ウィンドウの動き１７６は、仮定された開始位置から、ＳＡＶＥおよびＲＥＳＴＯＲＥ方向の上方、あるいは下方に、フレーム・オフセット１７８を調整する。好適に、フレーム・オフセット１７８は、復号化された各ブロックの開始位置で、「０」に設定され、連続ＳＡＶＥの動きごとに、「－１」、「－２」などまで１カウントだけ、調整される。逆に、フレーム・オフセットは、識別されたＲＥＳＴＯＲＥタイプの動きごとに、１カウントだけ増加する。こうして、フレーム・オフセットは、そのブロック中に遭遇された、識別されたレジスタ・ウィンドウの動きに関する情報１７６に基づいて、デフォルトの開始位置から、カレント累積オフセットを提供する。例えば、「０」のデフォルトから、ＳＡＶＥは、そのフレーム・オフセット１７８を、「－１」にまで調整する。その後、第二のＳＡＶＥは、オフセットを「－２」にまで調整する。しかしながら、ＲＥＳＴＯＲＥは、オフセットを「－１」にまで戻す調整をして、第二のＲＥＳＴＯＲＥは、オフセットを「０」にまで戻す調整をして、さらに、さらなるＲＥＳＴＯＲＥは、オフセットを「＋１」にまで調整するなどをする。図８における表は、識別されたレジスタ・リファレンス１７５と、識別されたレジスタ・ウィンドウの動き１７６とを示している。また、この表は、変位１７７と、そこから導き出されたフレーム・オフセット１７８とを示している。さらに、この表は、フレーム・オフセット１７８と、変位１７７とを組み合わせて、ＳＲスタック４００上の種々の異なるフレーム４１０に存在する個々のエントリ４０１をアドレス指定する、代表的なターゲット・コード２１を示している。すなわち、カレントオフセット１７８および変位１７７は、ターゲット・コードを生成して、植え込むときに、コア１９２およびエンコーダ・ユニット１９３によって用いられる。エンコーダ１９３は、こうして、「ベース・プラス・オフセット」タイプのアドレス指定を可能にすることによって、ＳＲスタック内でのアドレス指定を単純化する、ターゲット・コード２１を植え込むことが可能となる。

【００６１】

図８Ａに示されるように、ターゲット・コード・ブロック２１ａは、前述の例において示されたように、適切に、スタック・ポインタＳＲ＿ＳＰをターゲット・レジスタｒｂｐにロードする、ヘッダ・コード２１１から開始する。この例においては、ｒｂｐにおける値が、図６のフレーム４１０ａを参照することが仮定される。次に、ターゲット・コードは、上記のように、代表的な「移動」（ｍｏｖ）および「付加」（ａｄｄ）サブジェクト命令に相当する作業を実行する。「保存」（ｓａｖｅ）から導き出されたレジスタ・ウィンドウの動きに関する情報１７６に応じて、フレーム・オフセット１７８は、１カウントだけ減少して、ターゲット・コードが生成されて、ＳＲスタック４００上で１６個のエントリだけ前進するように、ｒｂｐに保持されたスタック・ポインタＳＲ＿ＳＰを一時的に修正する。その結果、ＳＲ＿ＳＰは、図６における後続のフレーム４１０ｂを参照するようになる。「保存」（ｓａｖｅ）サブジェクト命令は、ここでサブジェクト・ブロックを終了させないことに留意されたい。すなわち、ヘッダおよびフッタコードのオーバーヘッドは要求されない。図８におけるターゲット・コードは、ｒａｘを用いて、ただちに、第二のサブジェクト「付加」（ａｄｄ）命令の作業を実行する。「復旧」（ｒｅｓｔｏｒｅ）から導き出されたレジスタ・ウィンドウの動きに関する情報１７６に応じて、ターゲット・コードが生成されて、新しいオフセット１７８によって、ｒｂｐ内でＳＲ＿ＳＰの値をアップデートして、１６個のエントリ分だけ、スタック・ポインタを遅らせる。それによって、スタック・ポインタは、再度、図６のフレーム４１０ａを参照することになる。また、最終的なサブジェクト「復旧」（ｒｅｓｔｏｒｅ）命令は、このサブジェクト・ブロック１７ａの終端点を選択するように作用する。フッタ２１１は、こうして、ｒｂｐからｆｓ：ＳＲ＿ＳＰに戻された、アップデートされたスタック・ポインタＳＲ＿ＳＰを保存する。上記の図７に関するように、サブジェクト・ブロック１７ａの端において、「戻す」（ｒｅｔｕｒｎ）は、サブジェクト・ブロック１７ａを「ｆｕｎｃ１」として呼び出したサブジェクト・コード（図示せず）に戻るようなトランスレータに、実行制御を戻してもよい。

【００６２】

図８Ｂは、ＳＲアドレス指定メカニズムにおける、さらなる最適化を示している。ここで、オフセット１７８は、ターゲット・コードを生成するより前に、変位１７７と組み合わせられて、その結果、ｒｂｐにおいてＳＲ＿ＳＰの値を一時的に調整する作業が回避されることになる。すなわち、図８Ａの端部における、三個のターゲット命令、「付加」（ａｄｄ）、「移動」（ｍｏｖ）、および「付加」（ａｄｄ）は、好適に、図８Ｂに示されるように、単一の「付加」（ａｄｄ）命令に組み合わされる。オフセット１７８および変位１７７は、この場合は－１６＋１１＝－５によって組み合わされて、ｒｂｐの元の値から、－５の組み合わされた調整を付与する。ここで、レジスタ・リファレンス１７５と関連する変位１７７を、レジスタ・ウィンドウの動きに関する情報１７６と関連するフレーム・オフセット１７８とともに識別して、サブジェクト・コードの作業の実行と、そのサブジェクト・コードの動作に関連したサブジェクト・プロセッサの重要な構成要素のエミュレートとの両方を行う、有効な最適化されたターゲット・コードが、インライン化されるようにさせる。

【００６３】

図８における、上記のアドレス指定メカニズムは、トランスレータ１９に、ターゲット・コードの単一のブロック内で、サブジェクト・コードの、複数の「保存」（ｓａｖｅ）および／または「復旧」（ｒｅｓｔｏｒｅ）命令の効果をエミュレートする、ターゲット命令を含む、ターゲット・コードの単一のブロックを提供させる。このメカニズムはまた、トランスレータ１９に、サブジェクト関数、特に、リーフ命令を、ターゲット・コードに埋め込むことが理解されよう。すなわち、手続き呼び出しを実行するために用いられるサブジェクト「呼び出し」（ｃａｌｌ）と「戻す」（ｒｅｔｕｎ）命令は、レジスタ・ウィンドウ５１０の同一のＳＡＶＥおよびＲＥＳＴＯＲＥタイプの動きを伴い、ターゲット・コードにおいて、関数呼び出しを要求することなく、達成されることができる。

【００６４】

図９は、ここで説明される、レジスタ・ウィンドウ・エミュレーション・メカニズムの実行方法の概略的なフローチャートである。ステップ９０１において、サブジェクト・コード１７のブロックが選択されて、復号化される。サブジェクト・プロセッサ３のサブジェクト・レジスタ５に対するこのサブジェクト・コード１７のブロックにおけるリファレンス１７５は、ステップ９０２で識別される。とりわけ、リファレンス１７５は、ウィンドウ表示されたサブジェクト・レジスタ５０２のファイルに関連している。また、レジスタ・ウィンドウ５１０の動きを起こすサブジェクト・コード１７内の命令１７６は、ステップ９０３において識別される。ステップ９０４において、ＳＲスタック４００は、上記のように、スタック・ポインタＳＲ＿ＳＰを参照して、ターゲット・コンピューティング・プラットフォーム１０上に提供される。

【００６５】

ステップ９０５において、ターゲット・コード２１は、ＳＲスタック４００とスタック・ポインタＳＲ＿ＳＰに関連して生成される。すなわち、ターゲット・コード２１の各ブロックは、ＳＲスタック４００上のエントリ４０１としてのデータの値を記憶して取り出す、ターゲット・コード命令によって、生成される。ここで、ステップ９０６において、識別されたレジスタ・ウィンドウの動きに関する情報１７６から導き出されたターゲット・コードは、スタックの新しい先頭を参照するようにスタック・ポインタＳＲ＿ＳＰをただちにアップデートすることによって、あるいは、一時的なフレーム・オフセット１７８を調整することによって、スタック・ポインタＳＲ＿ＳＰを調整する。ステップ９０７において、識別されたレジスタ・リファレンス１７５から導き出されたターゲット・コード２１は、調整されたスタック・ポインタＳＲ＿ＳＰからの関連する変位１７６を用いることによって、ＳＲスタックのフレーム４１０内で、所望のエントリ４０１にアクセスする。

【００６６】

有利なことに、ＳＲスタック・メカニズム４００は、それが同時に記憶することが可能なフレーム４０１の数に関して制限はない。図５に戻って、サブジェクト・プロセッサ３

のこの代表的なハードウエアの実施は、８個のレジスタ・ウィンドウ位置５１１のみを提供して、定義によって、ＳＰＡＲＣハードウエアは、最大、３２個のレジスタ・ウィンドウ位置を提供することのみが可能であることに留意されたい。レジスタ・ウィンドウ位置５１１の各々が、いったん、有効なデータを含んでしまうと、任意の付加的なＳＡＶＥは、スピル・オペレーションを課して、ウィンドウ表示されたレジスタ・ファイル５０２を空にして、新しい一時的な作業スペースを開放させる。対照的に、当該エミュレートされたレジスタ・ウィンドウ・メカニズムは、サブジェクト・アーキテクチャの自動的なスピルおよびフィルオペレーションを検出して、実行することに関する、かなりのオーバーヘッドを回避する。すなわち、当該レジスタ・ウィンドウ・メカニズムは、このような有限のハードウエア制限によって束縛されることはなく、ＳＲスタック４００は、ＳＡＶＥごとに、必要な付加的なエントリ４０１を含むことが要求されるとおり、延長する。サブジェクト・コンピューティング・アーキテクチャが、最大有限数ｎ個のレジスタ・ウィンドウ位置を備えている場合、ＳＲスタックは、実行プログレスとして、複数のｍ個のフレーム４１０を有してもよい。ここで、ｍおよびｎは、ともに、正の整数であり、ｍはｎより大きい。ＳＲスタック４００は相対的に大きく、所定の有限な大きさではない。ほとんどのターゲット・コンピューティング・プラットフォームにおいて、メモリ１８は、ＳＲスタック４００が、無限のリソースとしてトランスレータ１９で見られるのには十分な大きさである。最終的に、ＳＲスタック４００は大きさが制限されてはいるが、実際には、論理的に最大の大きさのスタックが、任意の現実的な実施（例えば、１６個の６４ビットのエントリごとに１Ｋｂ）に対して必要な大きさをはるかに上回っている。このため、ＳＲスタックは、無限のリソースとして処理されてもよい（すなわち、数百個のフレーム４１０が、ＳＲスタック４００によって容易に記憶される）。

【００６７】

図１０は、ターゲット・コンピューティング・アーキテクチャのメモリ内で、複数のスタック・データ構造を示し、本発明のさらなる態様を説明する概略図である。

図１０は、上記のような、ＳＲスタック４００を示す。この場合も、図は垂直に下方に伸びている。また、当業者にとって周知であるように、多数の他のスタックが、ターゲット・コンピューティング・アーキテクチャ１０のメモリ１８内に定義される。図示されたスタックは、特に、サブジェクト・スタック４５０、ターゲットスタック４６０、およびトランスレータスタック４７０を含む。ここで、サブジェクト・スタック４５０は、サブジェクト・コードの実施のために、サブジェクト・アーキテクチャ内に提供されるであろうように、実施スタック（制御スタックあるいは関数スタックとも称される）をエミュレートする。すなわち、サブジェクト・コードからターゲット・コードへのプログラム・コード変換のプロセスの一部は、ターゲット・アーキテクチャ上のサブジェクト・スタック４５０の作成および操作をともなう。ターゲットスタック４６０は、ターゲット・コード２１の実施のために用意されたスタックである。トランスレータスタック４７０は、トランスレータ１９の実行のために用意されたスタックである。これらのスタックの各々と、他のスタックは、ターゲット・アーキテクチャのメモリ１８内に同時に存在して、典型的には、トランスレータ・コード１９とターゲット・コード２１などとともに、ターゲット・オペレーティング・システム２０によって管理される。

【００６８】

サブジェクト・アーキテクチャ１において、レジスタ・スピル・オペレーションは、サブジェクト・システムのメモリ８内に提供されるような、ウィンドウ表示されたハードウエアサブジェクト・レジスタ５０２からサブジェクト・スタックへレジスタ値を転送させるために、サブジェクト・スタック４５０は、特に興味深いものとなっている。逆に、サブジェクト・アーキテクチャにおけるフィルオペレーションは、サブジェクト・スタックからウィンドウ表示されたハードウエアサブジェクト・レジスタ５０２へレジスタ値を転送させる。代表的なＶ９　ＳＰＡＲＣアーキテクチャにおいて、各ＳＡＶＥタイプのレジスタの動きによって、スピル・オペレーションが行われると、各レジスタ・ウィンドウ位置５１１においてレジスタ５０２からレジスタ値が充てんされるであろう実行スタック上

に、空間が蓄えられるようになる。すなわち、図１０に示されるように、サブジェクト・コード１７（ここでは、ターゲット・コード２１に変換されている）は、当然、要求されたデータの値が、所定の状況においては、サブジェクト・スタック４５０のエミュレートされたバージョンの割り当てられた空間内に表れるようになることを仮定してもよい。しかしながら、このようなデータの値は、ＳＲスタック４００からエミュレートされたサブジェクト・スタック４５０へ実際にコピーされていない限り、データの値は期待されるようには表れてこないであろう。

【００６９】

図１０は、ＳＲスタック４００上における、代表的なエントリのフレームの１セットを示している。ここで、フレーム４１０には、Ｆ０からＦ８までの符号が付されている。図５に示されるように、フレームは互いに重複しているが、明確にするために、図１０においては重複しているようには示されていない。さらに、個々のエントリ４０１は、図１０において、やはり明確にするために示されていない。サブジェクト・プラットフォームの挙動をエミュレートするため、新しいフレーム４１０のＦ０からＦ８の各々が、ＳＲスタック４００に付加されるように、空間が、エミュレートされた実行スタック４５０上に蓄えられる。また、サブジェクト・コード１７を実行することによって、サブジェクト・スタック上で、効果をエミュレートすることが要求されるとおり、エミュレートされたサブジェクト・スタック４５０には、一時的な値、関数呼び出し、および戻しパラメータなどの他のデータが散在している。

【００７０】

特に、サブジェクト・レジスタからサブジェクト・スタックへのデータの転送に影響を与える、所定のサブジェクト・コード命令が存在する。ここに記載されたｖ９　ＳＰＡＲＣの例において、「ｆｌｕｓｈｗ」命令は、カレント・ウィンドウ位置を除いて、反復されるスピルトラップを実行することによって、ウィンドウ表示されたハードウエアサブジェクト・レジスタ５０２から、サブジェクト実行スタックへ、すべてのレジスタ５０２をフラッシュ表示する。この「ｆｌｕｓｈｗ」命令は、（カレント・レジスタ・ウィンドウ位置以外の）任意のレジスタ・ウィンドウ位置が、有効なコンテンツを有している場合に、スピルトラップを発生させることによって実行される。サブジェクト・アーキテクチャ上で、有効なコンテンツを有するウィンドウ位置の数は、ＣＡＮＳＡＶＥ制御／状態レジスタを参照して計算される。

【００７１】

また、プログラミング言語Ｃは、特定のサブジェクト・コンピューティング・プラットフォームに、固有にコンパイルされたネイティブ・コードライブラリを含んでもよい。このＣプログラミング言語は、「ｓｅｔｊｍｐ」および「ｌｏｎｇｊｍｐ」などの命令を含む。これらの命令は、現在は、一般的には、時代遅れであり、また、実行が非常に困難であると考えられているものの、現実的には多くのサブジェクト・プログラム、特に、レガシー・プログラムにおいて表現することが可能である。「ｓｅｔｊｍｐ」および「ｌｏｎｇｊｍｐ」命令は典型的に、ローカルではない出口あるいは、ソフトウエアの例外的な処置のための、Ｃプログラミング言語において用いられる。ｓｅｔｊｍｐ関数は、ｓｅｔｊｍｐ関数に対する呼び出しが、サブジェクト・コード内に表れる時点で、実行環境についての情報を保存することによって、戻しポイントを識別する。サブジェクト・プログラムの実行は、実行制御が、ｓｅｔｊｍｐが呼び出されるポイントまで戻って転送されることになるように、通常は、ｓｅｔｊｍｐに対する呼び出しの後に、いくつかの後のポイントで、ｌｏｎｇｊｍｐの呼び出し側が、この戻しポイントに対して出口を発生させるまで、続く。ｓｅｔｊｍｐルーチンは典型的には、サブジェクト・レジスタからサブジェクト・スタックまでレジスタ値をコピーすることを含み、また、ｌｏｎｇｊｍｐ関数は典型的には、これらの値をスタックからサブジェクト・レジスタへ復旧させる。

【００７２】

他の実施例として、プログラミング言語Ｃは、信号処理あるいはユーザのマルチスレッディングにおいて用いられるために、コンテキストを作成し、保存し、復旧する関数の、

プロセッサ・ファミリ・スペシフィック・インプリテーションを、アセンブリにおいて含んでもよい。いくつかのＣライブラリは、このようなルーチンを、「ｇｅｔｃｏｎｔｅｘｔ」、「ｓｅｔｃｏｎｔｅｘｔ」、および「ｍａｋｅｃｏｎｔｅｘｔ」という名の元に供給する。コンテキスト関数は、ネイティブＣライブラリの一部のように提供されるが、ＡＲＭ、ＰｏｗｅｒＰＣ、ＳＰＡＲＣおよびｘ８６などの特定のハードウエアのために、特定の実行によって提供されるものである。

【００７３】

さらなる実施の例として、より高いレベルのプログラミング言語Ｃ＋＋は、具体的には、ソフトウエアの除外を取り扱うような命令を供給する。これらのＣ＋＋の除外を取り扱う命令は、サブジェクト・プログラムの実行中に遭遇する例外的な状況を取り扱うことを第一に、意図しているが、これらの命令はまた、それら自体において、好都合なプログラミング構造を有しており、また、さらに典型的な「ｉｆ」タイプの命令と比較して、プログラムのアルゴリズムを単純化するために頻繁に用いられている。具体的には、Ｃ＋＋の除外の命令は、「ｔｒｙ」、「ｃａｔｃｈ」、および「ｔｈｒｏｗ」命令を含む。サブジェクト・プロセッサによって実行可能な、バイナリ・サブジェクト・コードにおけるこれらの命令を実行することは、このように、サブジェクト・レジスタとサブジェクト・スタックとの間のデータ転送をともなう。

【００７４】

例えば、それによって、サブジェクト・プログラムが実行スタックにまで引き戻るスタックの巻き戻しを含むデータのこのような転送を要求する、他の多くの状況が存在し、このために、実行スタック上に存在する、有効なサブジェクト値を要求する。

【００７５】

図１１は、ウィンドウ表示されたサブジェクト・レジスタ５０２をエミュレートするために用いられるＳＲスタック４００と、サブジェクト・プラットフォームの実行スタックをエミュレートするために用いられるサブジェクト・スタック４５０との間でデータの値を転送するメカニズムを示す、概略フローチャートであり、逆もまた同様である。

【００７６】

ステップ１１０１において、サブジェクト・コード１７は復号化されて、とりわけ、上記のように、ＳＡＶＥおよびＲＥＳＴＯＲＥタイプのレジスタ・ウィンドウの動き１７６を識別する。ステップ１１０２において、ＣＲカウンタは、識別されたレジスタ・ウィンドウの動き１７６ごとに、アップデートされる。適切に、各ＳＡＶＥは、＋１を、ＣＲカウンタに付加して、各ＲＥＳＴＯＲＥは、ＣＲカウンタから－１を減少させる。このように、サブジェクト・プログラムが進行するにつれて、ＣＲカウンタは、ＳＲスタック４００上で、エントリ４０１のフレーム４１０の数をカウントする。また、ステップ１１０３において、サブジェクト命令は、そのサブジェクト・アーキテクチャにおいて、ウィンドウ表示されたサブジェクト・レジスタ５０２において保持されるデータの値が、サブジェクト実行スタックに対して保存されるように要求するものとして識別される。この命令はここでは、ＳＰＩＬＬタイプ命令と称される。ＳＰＡＲＣサブジェクト・アーキテクチャの特定の例として、「ｆｌｕｓｈｗ」サブジェクト命令が識別される。ステップ１１０４において、図１０に示されるように、ターゲット・コード２１が供給されて、エントリ４１０の決められた数のフレーム４１０を、ＳＲスタック４００から、エミュレートされたサブジェクト・スタック４５０上に割り当てられた対応する空間にフラッシュ表示する。この図１０の図示された例において、ＳＲスタック４００は、Ｆ０からＦ８までの符号が付された９個のフレームのためのエントリを含んでいる。ここで、ＣＲカウンタは、先の８個のフレームＦ０からＦ７（すなわち、エントリＦ８のカレント・フレームを含まない）が、ＳＲスタック４００から、サブジェクト・スタック４００（ＣＲ＝８）までコピーされる必要があることを示す。ステップ１１０５において、この反復されるスピル・オペレーションの後に、ＣＲカウンタが０までアップデートされて（リセットされて）、要求されたフレーム４１０のすべてが、ＳＲスタック４００からサブジェクト・スタック４５０にコピーされたことを示す。

【００７７】
上記のように、サブジェクト・アーキテクチャにおいて、「ｆｌｕｓｈｗ」命令は、カレント位置を除く、すべてのレジスタ・ウィンドウ位置５１１を無効にして、このように、サブジェクト実行スタックは、すべての先行するレジスタ・ウィンドウ位置に関して、サブジェクト・レジスタ値の基準のバージョンを保持する。Ｃコンテキスト関数、ｓｅｔｊｍｐ／ｌｏｎｇｊｕｍｐ関数、およびＣ＋＋の除外などのいくつかのサブジェクト・プログラムは、サブジェクト・スタックに記憶されたデータの値を変更するであろう。このため、この図１０に示されたエミュレートされた環境において、ＳＲスタック４００上のエントリ４０１における（潜在的に無効化された）データよりはむしろ、エミュレートされたサブジェクト実行スタック４５０において保持された値を参照することが重要である。したがって、ステップ１１０６において、ターゲット・コードが生成されて、適切な場合に、エミュレートされたサブジェクト・スタック４５０上で、所望のデータの値にアクセスする。

【００７８】
ステップ１１０７は、サブジェクト・アーキテクチャにおいて、レジスタの値を、サブジェクト実行スタックから、ウィンドウ表示されたレジスタ・ファイル５０２にロードされるようにさせるサブジェクト・コード命令を識別することを含む。これはここでは、ＦＩＬＬタイプの命令と称される。代表的なＳＰＡＲＣサブジェクト・アーキテクチャにおいて、これは、適切に、「復旧」（ｒｅｓｔｏｒｅ）命令である。ここに提供された、エミュレートされたメカニズムにおいて、ＣＲカウンタはステップ１１０８において試験される。ＣＲカウンタが非ゼロ（ＣＲ＞０）であって、一つ以上の有効なフレーム４１０が、ＳＲスタック４００において存在することを示す場合、ステップ１１０５において、ＣＲは、ＲＥＳＴＯＲＥ命令に応じて、アップデートされ（この場合、ＣＲ＝ＣＲ－１のように減少する）、サブジェクト・スタック４５０からコピーされるデータはない。ここで、サブジェクト「復旧」（ｒｅｓｔｏｒｅ）は、スタック・ポインタＳＲ＿ＳＰを調整して、上記のように、ＳＲスタック上で異なるフレーム４１０を選択する。しかしながら、ＣＲカウンタがゼロ（ＣＲ＝０）であって、ＳＲスタック上に有効な先行するフレームが存在しないことを示す場合、ステップ１１０９において、フィルオペレーションが行われて、ＲＥＳＴＯＲＥに続く新しいカレント・フレーム４１０として用いられるように、サブジェクト・スタック４５０から、ＳＲスタック４００にデータの値のフレームをコピーする。サブジェクト・スタック４５０からのデータの値のフレームは、サブジェクト・プログラムによって修正されていてもよく、ここで、これらのデータの値の基準のバージョンは、サブジェクト・プログラムにおいて引き続いて用いられるように、ＳＲスタック４００のエントリ４０１において再び、利用可能となる。特に、慣例による、ＳＰＡＲＣアーキテクチャは、サブジェクト・スタックに関するスタック・ポインタを、サブジェクト・レジスタｏ６（エイリアスｓｐによっても既知である）に記憶する。レジスタ・ウィンドウ位置５１１の重複によって、呼び出し側手順に関するスタック・ポインタは、サブジェクト・レジスタｉ６においてもまた利用可能である。この先行するスタック・ポインタは、フレームポインタと称される（また、エイリアスｆｐを用いてアクセスされることが可能である）。記憶されたレジスタの値への修正は、典型的に、フレームポインタ（ｉ６／ｆｐ）および／またはスタック・ポインタ（ｏ６／ｓｐ）に関して、レジスタの値を変更することをともなう。このように、これらのデータの値が、ここで説明されたエミュレーション・メカニズムによって、正確に維持されることが重要である。

【００７９】
図１２Ａは、代表的なスピル・オペレーションをさらに詳細に示している。オペレーション「ｆｌｕｓｈｗ」は、符号を付されたフレーム、Ｆ０およびＦ１に関して先に行われ、このように、ＣＲカウンタをゼロにリセットする。そのポイントに続いて、（少なくとも）１７個のさらなるＳＡＶＥレジスタの動きがあって、ＣＲカウンタ（ＣＲ＝１７）によって留意されるように、ＳＲスタック４００上で、Ｆ２からＦ１９のフレームを形成する。ＳＰＩＬＬ命令（すなわち「ｆｌｕｓｈｗ」）は、サブジェクト・コードにおいて識

別される。このサブジェクト・コードは、レジスタの値Ｆ１９からＦ２の１７個のフレームを、ＳＲスタック４００からサブジェクト・スタック４５０までコピーされるようにして、ＣＲカウンタをゼロに戻す。フレームＦ２からＦ１８のすべてが無効となり、現在、アクティブなフレームＦ１９だけが、ＳＲスタック上で有効のままである。

【００８０】

図１２Ｂは、図１２Ａに示される状態以降の代表的なフィルオペレーションをさらに詳細に示している。ここで、現在アクティブなフレームＦ１９（ここでＣＲ＝０）からのＲＥＳＴＯＲＥは、レジスタの値のフレームを、サブジェクト・スタック４５０からロードさせて、新しいカレント・フレームを、ここではＦ１８で、形成して、ＳＲスタック上のそのフレームにおいて先に、（無効な）値を置き換える。新しいカレント・フレームＦ１８にロードされたレジスタの値を使用して、実行は継続する。ＳＲスタックＳＲ＿ＳＲＳのスタック・ポインタはただちに、あるいはサブジェクト・コード・ブロックの端部においてアップデートされて、このレジスタ・ウィンドウの動きを説明する。

【００８１】

この転送メカニズムの他の特定の実施形態もまた、検討される。特に、「ｆｌｕｓｈｗ」が、カレント・ウィンドウ以外のすべてのレジスタ・ウィンドウを無効化するために、スピル・オペレーション（「ｆｌｕｓｈｗ」）に続いて、ＳＲスタック上の任意の適切なポイントは、その後、新しいカレント・フレームとして使用されてもよい。例えば、ＳＲスタックは除去されることが可能であり、各「ｆｌｕｓｈｗ」の後で完全に再生されたスタック空間、あるいはＳＲスタック上のデフォルトの位置は、スタックに関して割り当てられた空間の半分などの、新しいカレント・フレームとして設定されてもよい。

【００８２】

本発明の少なくともいくつかの実施形態は、専用のハードウエアを用いて単独に構成されてもよく、ここで用いられる「モジュール」あるいは「ユニット」などの用語は、所定のタスクを実行する、フィールド・プログラマブル・ゲート・アレイ（ＦＰＧＡ）あるいは特定用途向け集積回路（ＡＳＩＣ）などの、ハードウエア装置を含むがそれらに限定されない。また、本発明の構成要素は、アドレス指定可能な記憶媒体上に存在するように構成されてもよく、また、一つ以上のプロセッサ上で実行されるように構成されてもよい。このように、本発明の機能的な要素は、いくつかの実施形態において、例として、コンポーネント、例えば、ソフトウエア・コンポーネント、オブジェクト指向ソフトウエア・コンポーネント、クラスコンポーネント、およびタスクの構成要素、プロセス、関数、属性、サブルーチン、プログラム・コードのセグメント、ドライバ、ファームウエア、マイクロコード、回路、データ、データベース、データ構造、テーブル、アレイ、および可変要素を含んでもよい。さらに、好適な実施形態は、ここに述べられている、コンポーネント、モジュール、およびユニットに関して記載されてはいるが、このような機能的な要素が組み合わされて、より少ない要素となってもよいし、あるいは付加的な要素に分割されてもよい。

【００８３】

いくつかの実施形態が示され、記載されているものの、添付の特許請求の範囲において定義されるような、本発明の範囲から逸脱していない限り、種々の変形および修正を行うことが可能であると当業者に理解されるであろう。

【００８４】

本願に関連して本明細書と同時にあるいはそれ以前に提出され、本明細書とともに公衆が閲覧できるように公開されたあらゆる文献や書類に注意が向けられており、これらの文献や書類すべての内容が、参照してここに組み込まれる。

【００８５】

本明細書（添付の特許請求の範囲、要約書、および図面を含む）に開示されたすべての特徴、および／または、同様に開示された方法あるい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いくつかが相互排他的である場合の組み合わせをのぞいて、任意の組み合わせにおいて組み合わされてもよい。

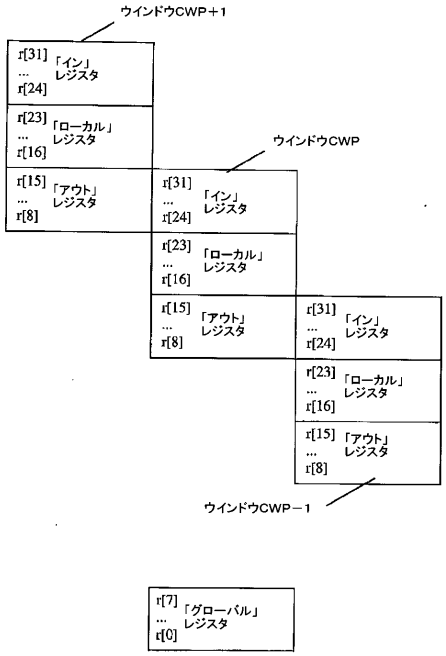
【００８６】

本明細書（添付の特許請求の範囲、要約書、および図面を含む）に開示された各特徴は、別に明確に述べられていない限り、同一の、同等の、あるいは同様の目的を果たす代替的な特徴によって置き換えられてもよい。このように、別に明確に述べられていない限り、開示された各特徴は、同等の、あるいは同様の特徴の一般的なシリーズからなる一例にすぎない。

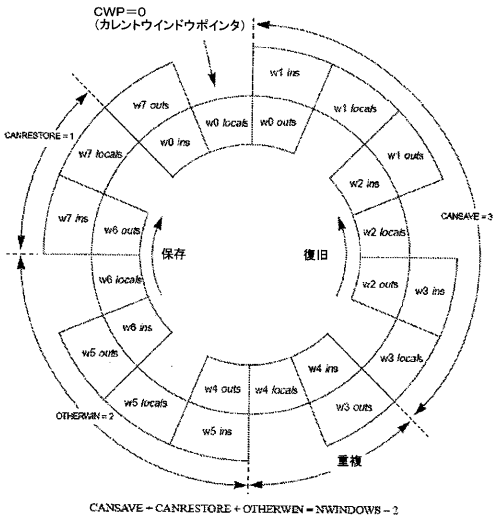
【００８７】

本発明は、上記の実施形態の詳細には限定されない。本発明は、本明細書（添付の特許請求の範囲、要約書、および図面を含む）に開示された特徴における、いかなる新しい特徴、あるいは新しい特徴の組み合わせ、あるいは、同様に開示されたいかなる方法あるいはプロセスのステップにおける、いかなる新しいステップ、あるいは新しいステップの組み合わせにまで発展する。

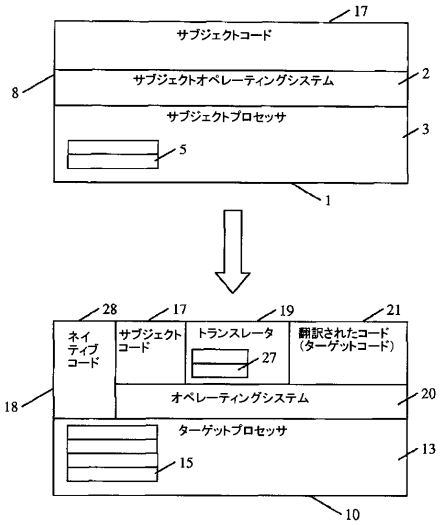
【図１】



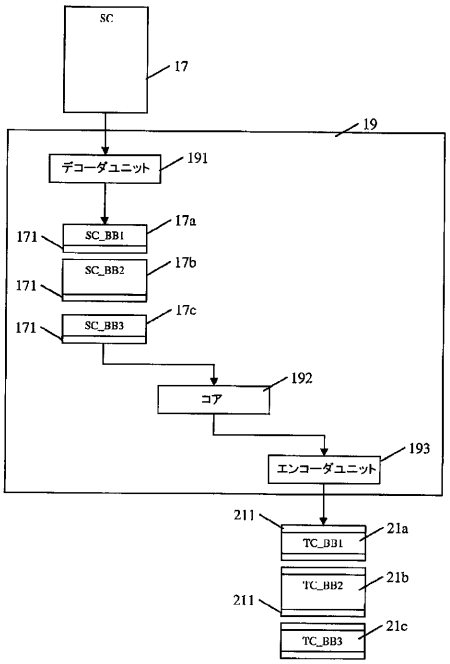
【図２】



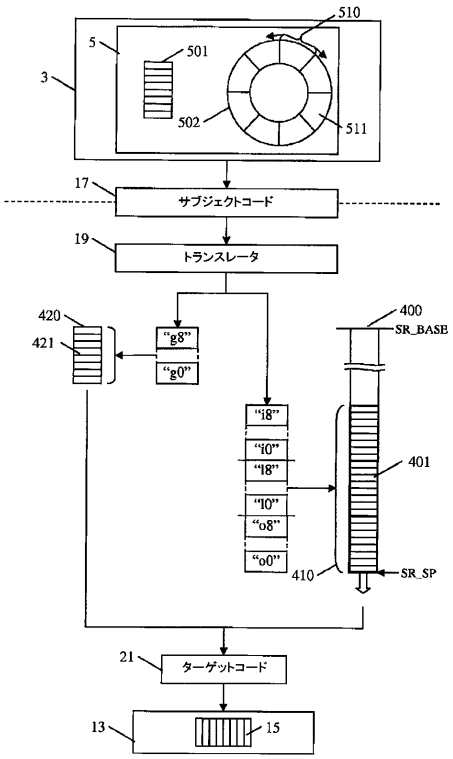
【図３】


17
サブジェクトコード
8
サブジェクトオペレーティングシステム
2
サブジェクトプロセッサ
3
5
1
28
17
19
21
ネイティブコード
サブジェクトコード
トランスレータ
翻訳されたコード（ターゲットコード）
27
18
オペレーティングシステム
20
ターゲットプロセッサ
13
15
10

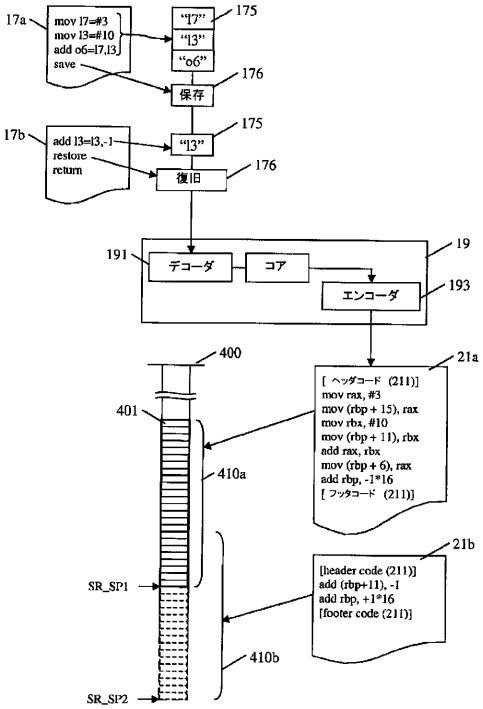
【図４】


SC
17
19
デコーダユニット
191
171
SC_BB1
17a
SC_BB2
17b
SC_BB3
17c
192
コア
193
エンコーダユニット
211
TC_BB1
21a
TC_BB2
21b
TC_BB3
21c

【図５】


510
5
501
3
502
511
17
サブジェクトコード
19
トランスレータ
420
421
"g8"
"g0"
400
SR_BASE
"i8"
"i0"
"l8"
"l0"
"o8"
"o0"
401
410
SR_SP
21
ターゲットコード
13
15

【図６】


17a
mov l7=#3
mov l3=#10
add o6=l7,l3
save
"l7"
"l3"
"o6"
175
保存
176
17b
add l3=l3,-1
restore
return
"l3"
175
復旧
176
191
デコーダ
コア
エンコーダ
19
193
400
401
410a
SR_SP1
410b
SR_SP2
21a
[ヘッダコード (211)]
mov rax, #3
mov (rbp + 15), rax
mov rbx, #10
mov (rbp + 11), rbx
add rax, rbx
mov (rbp + 6), rax
add rbp, -1*16
[フッタコード (211)]
21b
[header code (211)]
add (rbp+11), -1
add rbp, +1*16
[footer code (211)]

【図７】

サブジェクトコード	レジスタリファレンス（１７５）	エントリ調整（６４ビットのワードにおいて）	識別されたレジスタウインドウの動き（１７６）	SR_SPのフレーム の動き	ターゲットコード
サブジェクトブロック１７ａ：					［ヘッダコード（２１１）］
func1:mov17=#3	17	+15	（なし）	0	mov rax, #3
					mov(rbp+15), rax
mov13=#10	13	+11	（なし）	0	mov rax, #10
					mov(rbp+11), rax
add o6=17,13	17	+15	（なし）	0	add rax, rbx
	13	+11			mov(rbp+6), rax
	o6	+6			
save			SAVE(-1)	-1	add rbp, -1*16
					［フッタコード（２１１）］
サブジェクトブロック１７ｂ：					ターゲットブロック２１ｂ：
add l3=l3,-1	13	+13	（なし）	0	［ヘッダコード（２１１）］
restore			RESTORE(+1)	+1	add rbp, +1*16
return			（なし）		［フッタコード（２１１）］

【図８】

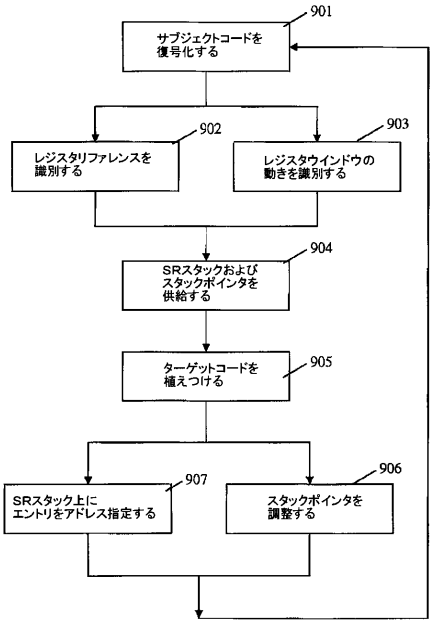
8A

サブジェクトコード	レジスタリファレンス（１７５）	変位（１７７）（６４ビットのワードにおいて）	識別された動き（１７６）	フレームオフセット（１７８）（１６×６４ビットのワードのフレームにおいて）	ターゲットコード
サブジェクトブロック１７ａ：					［ヘッダコード］
Func1:mov17=#3	17	+15	（なし）	0	mov rax, #3
					mov(rbp+15), rax
mov13=#10	13	+11	（なし）	0	mov rax, #10
					mov(rbp+11), rax
add o6=17,o6	17	+15	（なし）	0	add rax, rbx
	13	+11			mov(rbp+6), rax
	o6	+6			
save			SAVE(-1)	-1*16	add rbp, -1*16
add l3=l3,-1	13	+11	（なし）	-1*16	add (rbp+11),-1
restore			RESTORE(+1)	0	add rbp, +1*16
return					［フッタコード（２１１）］

8B

save			SAVE(-1)	-1*16	add (rbp, -5),1
add l3=l3,-1	13	+11	（なし）	-1*16	
restore			RESTORE(+1)	0	

【図９】



【図１０】

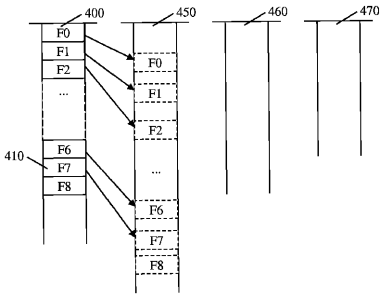
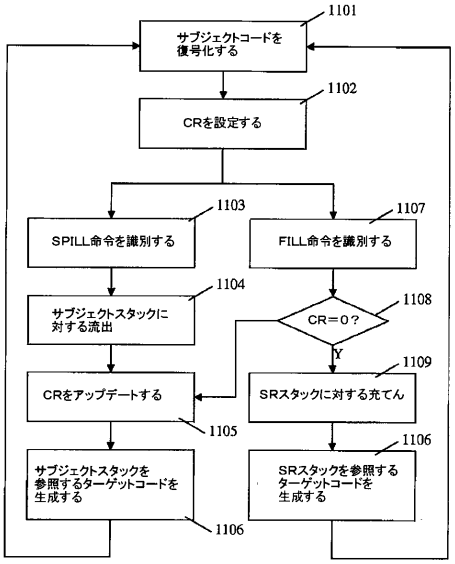


Fig. 10

【図１１】



【図１２Ａ】

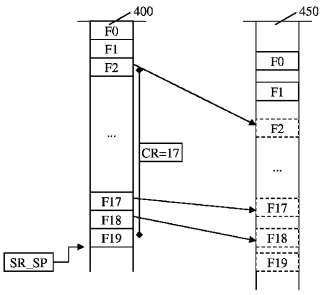


Fig. 12A

【図１２Ｂ】

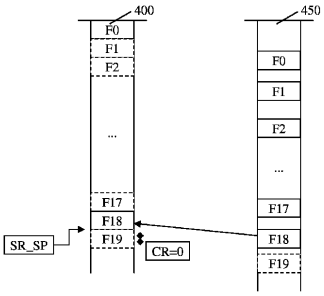


Fig. 12B

フロントページの続き

(72)発明者　ブラウン、アレクサンダー　バラクラフ
イギリス国　Ｓ１０　２ＰＬ　ヨークシャー　シェフィールド　ブルームヒル　ニューボールド　レーン　１２５

審査官　三坂　敏夫

(56)参考文献　国際公開第２００５／００８４８７（WO，Ａ１）
国際公開第２００５／００８４７８（WO，Ａ１）
特表２００２－５４３４９０（ＪＰ，Ａ）
国際公開第２００４／０１２０７９（WO，Ａ１）
特開平０７－０９３２６４（ＪＰ，Ａ）
特表２００７－５３１０７４（ＪＰ，Ａ）
特表２００７－５２９０６３（ＪＰ，Ａ）
Trek Palmer et al., "Experiences Constructing a Lightweight SPARC Interpreter for a Dynamic Binary Translator", [ONLINE], 米国, University of New Mexico, ２００３年　3月１４日, Pages:1-9, 平成２４年９月１９日検索、インターネットＵＲＬ：http://www.cs.unm.edu/~darko/papers/tr-cs-2003-12-sind-exprs.pdf

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