(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4748829号
(P4748829)
(45)発行日　平成23年8月17日(2011.8.17)　(24)登録日　平成23年5月27日(2011.5.27)

(51)Int.Cl.　F I
G06F 9/45 (2006.01)　G06F 9/44 322L

請求項の数 18　外国語出願　(全 21 頁)

(21)出願番号　特願平11-309657
(22)出願日　平成11年10月29日(1999.10.29)
(65)公開番号　特開2000-181724(P2000-181724A)
(43)公開日　平成12年6月30日(2000.6.30)
審査請求日　平成18年10月24日(2006.10.24)
(31)優先権主張番号　09/183499
(32)優先日　平成10年10月30日(1998.10.30)
(33)優先権主張国　米国(US)

前置審査

(73)特許権者　597004720
オラクル・アメリカ・インコーポレイテッド
アメリカ合衆国、９４０６５　カリフォルニア州、レッドウッド・ショアーズ、オラクル・パークウェイ、５００
(74)代理人　110000028
特許業務法人明成国際特許事務所
(72)発明者　ピーター・ビー．・ケスラー
アメリカ合衆国　カリフォルニア州９４３０６　パロ・アルト，ロス・ロブレス・アベニュー，７６９

審査官　坂庭　剛史

最終頁に続く

(54)【発明の名称】コンパイルする方法をランタイムにおいて選択する方法及び装置

(57)【特許請求の範囲】
【請求項１】
コンピュータによって実行される、コンピュータ・プログラム内の浮動小数点アンダフローを形成する浮動小数点オペレーションに関連するバイトコード命令をコンパイルする方法をランタイムにおいて決定する方法であって、
複数の方法でコンパイルされ得る前記コンピュータ・プログラムに関連するバイトコード命令を取り出す工程と、
前記浮動小数点オペレーションが浮動小数点アンダフローを形成し得るか否かを判定する工程と、
前記浮動小数点アンダフローに関する判定を受けて、前記バイトコード命令を第１の方法でコンパイルする工程と、
前記浮動小数点オペレーションが浮動小数点アンダフローを形成し得るとき、前記バイトコード命令をコンパイルする第２の方法が望ましいことを、ランタイムにおいて決定する工程と、前記第２の方法は、前記浮動小数点アンダフローに関連する予め定められた条件に基づいて決定されることと、
前記バイトコード命令を前記第２の方法でリコンパイルする工程と、
を備える方法。
【請求項２】
請求項１に記載の方法において、
リコンパイルされるべき前記バイトコード命令を備えるモジュールを、キューへ加える

工程
を更に備えることを特徴とする方法。
【請求項３】
請求項１に記載の方法において、
前記バイトコード命令を複数の方法でコンパイルできることを、ランタイムにおいて決定する工程
を更に備えることを特徴とする方法。
【請求項４】
請求項１に記載の方法において、
前記コンピュータ・プログラムのダイナミックに変化する効率を調べる工程
を更に備えることを特徴とする方法。
【請求項５】
請求項４に記載の方法において、
コンピュータ・プログラムをコンパイルできる前記複数の方法のうちの現在実行中の１つの方法に関する特定のデータを、ランタイムにおいて集める工程
を更に備えることを特徴とする方法。
【請求項６】
請求項１に記載の方法において、
前記第２の方法が前記第１の方法と異なる際、前記バイトコード命令を前記第２の方法でリコンパイルする工程
を更に備えることを特徴とする方法。
【請求項７】
コンピュータによって実行される、プログラム内の浮動小数点オペレーションを実行する方法であって、
浮動小数点オペレーションが浮動小数点アンダフローを形成し得るか否かを決定する工程と、
前記浮動小数点オペレーションが浮動小数点アンダフローを形成し得るとき、特定の浮動小数点オペレーションがアンダフローを何回引き起こしたかを決定するために、特定のインジケータをチェックする工程と、
前記特定のインジケータが所定の基準を満たした時、前記特定の浮動小数点オペレーションを第１の方法を用いてコンパイルし、それ以外の時は、前記特定の浮動小数点オペレーションを第２の方法を用いてコンパイルする工程と、
を備える方法。
【請求項８】
請求項７に記載の方法において、
前記特定の浮動小数点オペレーションをコンパイルする第１及び第２の方法に関するデータを、ランタイムにおいてダイナミックに形成し、格納する工程
を更に備えることを特徴とする方法。
【請求項９】
請求項７に記載の方法において、
前記第１の方法はトラップ・ルーチンであり、前記第２の方法は明示的チェックであり、前記明示的チェックはインライン・コードを前記プログラムへ挿入することによってインプリメントされることを特徴とする方法。
【請求項１０】
請求項９に記載の方法において、
前記特定の浮動小数点オペレーションが明示的チェックを用いてコンパイルされた時点から所定時間が経過したか否かをチェックする工程と、
前記所定時間が経過している場合、リコンパイルされるべき前記特定の浮動小数点オペレーションを含むモジュールをマークする工程と、
を更に備えることを特徴とする方法。

【請求項１１】
請求項９に記載の方法において、
前記トラップ・ルーチンを開始する工程
を更に備えることを特徴とする方法。
【請求項１２】
請求項９に記載の方法において、
前記特定の浮動小数点オペレーションが明示的チェックを用いてコンパイルされる場合、タイマをセットする工程
を更に備えることを特徴とする方法。
【請求項１３】
請求項９に記載の方法において、
前記特定のインジケータは、前記浮動小数点アンダフローがトラップ・ルーチンを用いて処理される度に、インクリメントされるカウンタであることを特徴とする方法。
【請求項１４】
請求項９に記載の方法において、
どの浮動小数点オペレーションが前記トラップ・ルーチンの実行を引き起こしたかを決定する工程と、
カウンタをインクリメントする工程と、
前記カウンタが所定値を超えている場合、リコンパイルされるべき前記特定の浮動小数点オペレーションを含むモジュールをマークする工程と、
を備えることを特徴とする方法。
【請求項１５】
請求項１４に記載の方法において、
前記モジュールがリコンパイルすべくマークされている場合、前記モジュールをリコンパイル・キューへ加える工程を更に含む請求項１９に記載の方法。
【請求項１６】
請求項１４に記載の方法において、
前記方法がリコンパイルされるべくマークされている場合、前記方法に関連するカウンタをリセットする工程
を更に備えることを特徴とする方法。
【請求項１７】
コンピュータ・プログラム内の浮動小数点アンダフローを形成する浮動小数点オペレーションに関連するバイトコード命令をコンパイルする方法をランタイムにおいて決定するコンピュータ・プログラムを格納するコンピュータ読み取り可能媒体であって、
複数の方法でコンパイルされ得る前記浮動小数点オペレーションに関連するバイトコード命令を取り出す機能と、
前記浮動小数点アンダフローに関する判定を受けて、前記バイトコード命令を第１の方法でコンパイルする機能と、
前記浮動小数点オペレーションが浮動小数点アンダフローを形成し得るとき、前記バイトコード命令をコンパイルする第２の方法が望ましいことを、ランタイムにおいて決定する機能と、前記第２の方法は、前記浮動小数点アンダフローに関連する予め定められた条件に基づいて決定されることと、
前記バイトコード命令を前記第２の方法でリコンパイルする機能と、
をコンピュータによって実現させるコンピュータ読み取り可能媒体。
【請求項１８】
コンピュータによって実現される、プログラム内の浮動小数点命令を実行するバーチャルマシンシステムであって、
浮動小数点命令が浮動小数点アンダフローを形成し得るか否かを決定する命令インタプリタと、
特定の浮動小数点オペレーションが浮動小数点アンダフローを引き起こした回数を保持

する浮動小数点アンダフロー・インジケータと、
前記特定のインジケータ部が所定の基準を満たした時、前記特定の浮動小数点オペレーションを第１の方法を用いてコンパイルし、それ以外の時は、前記特定の浮動小数点オペレーションを第２の方法を用いてコンパイルするコンパイラ
を備え、
前記特定の浮動小数点オペレーションがアンダフローを何回引き起こしたかを決定するために、前記浮動小数点アンダフロー・インジケータがチェックされることを特徴とするバーチャルマシンシステム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

一般的に、本発明はコンピュータ・ソフトウェア及びソフトウェア・ポータビリティの分野に関する。特に、本発明はプログラムをプラットフォーム固有の要件に基づいてコンパイルする方法に関する。

【０００２】

【従来の技術】

ジャバ（Ｊａｖａ（商標名））バーチャル・マシン（ＪＶＭ）は、様々なコンピュータ・アーキテクチャ上でインプリメント可能であり、 異なるマイクロプロセッサからそれぞれ生じる異なる仕様及び規格に適応可能である。この種の規格の１つの例としては、いくつかのマイクロプロセッサが浮動小数点数を扱う際に利用する拡張精度浮動小数点フォーマットが挙げられる。この種のマイクロプロセッサの１つとしては、拡張精度（８０ビット）浮動小数点演算を使用するインテル社のＩＡ－３２マイクロプロセッサ・アーキテクチャが挙げられる。一般的に、他のプロセッサは単（３２ビット）精度浮動小数点演算または倍（６４ビット）精度浮動小数点演算を使用する。

【０００３】

拡張精度フォーマットで算出された数値を単精度フォーマットまたは倍精度フォーマットへ変換する際、問題が発生する。浮動小数点オペレーションが結果をＩＥＥＥ７５４の指定するレンジ及び精度で形成すべきことは、ジャバ（商標名）言語によって指定されている。このＩＥＥＥ７５４の内容は、この開示をもって本明細書中に開示したものとする。その一方、カリフォルニア州サンタクララに所在するインテル社が製造するインテルＩＡ－３２プロセッサは結果をより広いレンジ及びより高い精度で形成する。これによって、前記の問題が発生する。これらのより広い結果はＩＥＥＥ７５４単精度フォーマット及び倍精度フォーマットへ正確に丸める必要がある。ＩＡ－３２マイクロプロセッサの場合、この種の丸めを実施する少なくとも２つの方法があり、これらの方法はそれぞれ異なるコスト（コード・サイズ及び実行速度）を伴う。スタティック・コンパイラ（またはワンタイム・ダイナミック・コンパイラ）は、１つのインプリメンテーションを選択する必要があり、この選択は全ての状況下で最適な選択とはならない。前記の問題を図１に示す。

【０００４】

図１は倍精度浮動小数点の一般的なフォーマットと、拡張精度浮動小数点のフォーマットとを示すブロック図である。フォーマット１０２は、インテルＩＡ－３２アーキテクチャの倍精度浮動小数点フォーマットとは対照的なインテルＩＡ－３２アーキテクチャの拡張精度浮動小数点数フォーマットを示す。符号ビット１０４は、この浮動小数点数が正及び負のいずれであるかを示す。浮動小数点数の指数の値を表すための指数値を示すビット１０６が、これに続いて配置されている。

【０００５】

ビット１０８は仮数を保持するためのビットを含む。仮数部は浮動小数点数の整数部を表すビットを最高で６４ビット保持できる。従って、８０（１＋１５＋６４）ビットが、拡張精度浮動小数点フォーマット内に存在する。一般的に、浮動小数点オペレーションは浮動小数点ユニットによって処理される。仮数及び指数を操作することによって、このユニットは浮動小数点数を必要とする複雑なオペレーションを効率的に実施できる。当該技術

分野でよく知られているように、浮動小数点数を整数及び指数で表すことにより、浮動小数点数の演算は遙かに容易になる。

【０００６】

更に、図１はＩＥＥＥ７５４に開示されている倍精度浮動小数点フォーマット１１２を示す。このフォーマットは拡張精度フォーマットとレイアウトの点で類似しているが、指数フィールド及び仮数フィールド内のビット数の点で異なる。前記のように、インテルＩＡ－３２プロセッサは結果を拡張精度フォーマットで形成する。前記の問題はジェームズ・ゴスリン、ビル・ジョイ及びガイ・スチールによる“ジャバ（商標名）言語詳説”（ＩＳＢＮ０－２０１－６３４５１－１）に由来しており、この文献の内容はこの開示をもって本明細書に開示したものとする。この詳説では、結果をＩＥＥＥ７５４単精度フォーマットまたは倍精度フォーマットで形成する必要がある。フォーマット１１２の説明へ戻り、符号ビット１０４は、倍精度フォーマットまたは単精度フォーマットと対照をなす拡張精度フォーマットにおける符号ビットと同じである。指数ビット１１４はビット１０６と同じ機能を有するが、拡張精度フォーマットにおける１５ビットとは対照的に１１ビットを保持する。仮数ビット１１６は拡張精度フォーマットにおける６４ビットとは対照的に５２ビットを保持する。従って、倍精度浮動小数点フォーマットは６４ビットを保持可能である。仮数長の違いを、図１の破線領域１１８で示す（指数長における４ビットの違いは図中にこれと同様に示されていない）。

【０００７】

拡張精度の結果が例えばＩＡ－３２プロセッサから与えられた際、単精度フォーマットまたは倍精度フォーマットの結果を必要とするジャバ言語では、問題が発生する。拡張指数が単精度または倍精度のレンジの外側に位置する場合、オーバフローまたはアンダフローが発生し得る。ＩＡ－３２では、オーバフローはハードウェアによって処理されるが、本発明の方法で解決に努めることができる。指数を減らすべく、仮数が（右側へ）シフトするので、アンダフローは処理が更に困難である。しかしながら、このシフトによって仮数のビットが失われ、これによって、結果の精度が失われる。正しく、かつ、精度が更に低い仮数を演算するためには、幾つかの命令が必要であり、一般的に、オペレーションは必要な時に呼び出されるように、別個のサブルーチンに置かれる。前記の問題は結果を正しく丸めることではなく、むしろ、訂正が生ずべきことを検出することにある。

【０００８】

図２（ａ）及び図２（ｂ）は浮動小数点アンダフローを検出する２つの方法を示す。図２（ａ）に示す１つの方法では、問題を訂正するトラップ・ルーチン２０６を呼び出すために、プログラム・コード２０２内のトラップ・ハンドラ２０４を使用して、トラップを実施することによって、プログラム・コード２０２は、その問題を検出する。図２（ｂ）に示す別の方法では、プログラム・コード２０８はコード２１０を含んでおり、そのコード２１０は、乗算オペレーション及び除算オペレーションなど、問題を潜在的に引き起こし得る方法で、浮動小数点が使用される度、その問題を検出する。

【０００９】

トラップ・ハンドラ２０６を利用することによって、問題の解決に努めている間、全てのオペレーションは中止される。トラップを呼び出す際、トラップ・ルーチンの実行前に、トラップを引き起こした命令のロケーションを含むマシンの状態を格納する。しかし、トラップを呼び出さない場合、オペレーション毎のオーバーヘッドは存在しない。そして、トラップ・ハンドラをセットアップするための、ワンタイム・オーバーヘッドがスレッド毎に存在するのみである。そして、このセットアップされたトラップ・ハンドラは全ての浮動小数点オペレーションを監視する。その一方、プログラム・コード２１０は、浮動小数点アンダフローの問題を処理するために、コードをプログラムへ挿入する技術を示す。この方法では、正しく丸められた結果を形成するために、必要に応じて、問題をチェックし、サブルーチンを呼び出す目的で、インライン・コードがアンダフローの問題を引き起こし得る各オペレーションの後に続けられる。この方法は、発生しない各アンダフローに対する多くの不必要なプロセッサ・オペレーションを必要とする。しかし、その問題が検

出された際、この問題は、全てのオペレーションを一時停止させることと、トラップを処理するために、プロセス・コンテキストまたはスレッド・コンテキストを保存することを要することなく、解決される。前記のように、浮動小数点アンダフローは、任意のプラットフォーム上における選択可能な解決策を有する問題の１つの例である。他の問題が発生し得る。この場合、ジャバ・バーチャル・マシンは特定の問題を解決するいくつかのインプリメンテーションを利用可能であり、各インプリメンテーションはいくらかのケースでより効率的である。

【００１０】

【発明が解決しようとする課題】

従って、プラットフォーム固有のバリエーションに起因して生じる問題の解決に使用するために、インプリメンテーションをインテリジェントに、ダイナミックに選択することが望ましい。浮動小数点アンダフローの問題を単なる例として使用した場合、例えば、必要なインライン・コードの量を減少し、さらに問題を訂正するサブルーチンをディスパッチするための、トラップ・ハンドラの使用のオーバーヘッドを防止するとともに、浮動小数点アンダフローを検出し、訂正することが望ましい。ダイナミック・ランタイム・コンパイラが１つのインプリメンテーションを選択し、その効率を監視し、要求されれば、そのインプリメンテーションを変更できることが望ましい。

【００１１】

【課題を解決するための手段およびその作用・効果】

バーチャル・マシンによって、プログラム実行中に、プログラムに関連するバイトコードをコンパイルする方法を決定する本発明に従う方法、装置及びコンピュター・プログラム製品を開示する。本発明の１つの態様では、複数の方法でランタイムにおいてコンパイルできるプログラム内の命令を取り出し、特定の方法（一般的には、デフォルトの方法）でコンパイルする。次いで、バーチャル・マシンは命令をコンパイルする別の方法がより望ましいことをランタイムにおいて決定し、バイトコード命令はこの別の方法でリコンパイルされる。

【００１２】

１つの形態では、リコンパイルされるバイトコード命令を含む実行可能なコードは、リコンパイルされる他の命令と一緒にキューへ加えられる。バーチャル・マシンはプログラムの実行時に発生したプログラムの全ての変化する要件を調べる。この場合、これらの要件は、プログラムをコンパイルできる複数の方法のそれぞれのプロフィール・データに由来している。別の形態では、命令をコンパイルする第１の方法、即ち、デフォルトの方法とは異なる方法で、前記の特定のバイトコード命令をバーチャル・マシンによってリコンパイルする。

【００１３】

本発明の別の態様では、単一のプログラムから、実行可能な命令の異なるセットを形成する方法を提供する。バイトコード命令の１つのセットを形成するために、プログラムを特定の方法（デフォルトの方法など）でランタイムにおいてコンパイルする。次いで、バーチャル・マシンはプログラムを別の方法でコンパイルすることが望ましいか否かをランタイムにおいて決定する。そして、バーチャル・マシンは、そのようにして、ネイティブ命令の別のセットを形成し、この別のセットは第１のセットと置換される。

【００１４】

１つの形態では、プログラムをリコンパイルする方法を決定するために、バーチャル・マシンは、プログラムを実行できる複数の方法のそれぞれのダイナミックに形成されたプロフィール・データを調べる。プロフィール・データは、プログラムを特定の方法で実行した回数を格納するカウンタを含む。プログラムのダイナミックに変化する要件をより効率的に処理するために、バーチャル・マシンは、ネイティブ命令の特定のセットをネイティブ命令の別のセットと置換すべきか否かを決定する。

【００１５】

本発明の別の態様では、プログラム内の浮動小数点命令を実行するシステムを開示する。

システムは、特定の命令が浮動小数点アンダフローを形成し得るか否かを決定する。次いで、浮動小数点オペレーションがアンダフローを引き起こした回数を決定するために、システムはインジケータをチェックする。インジケータが所定値未満である場合、バーチャル・マシンは浮動小数点オペレーションを１つの方法でランタイムにおいてコンパイルする。インジケータが所定値を越えている場合、バーチャル・マシンは浮動小数点オペレーションを別の方法でランタイムにおいてコンパイルする。

【００１６】

本発明の更に別の態様では、トラップ・ルーチンまたは明示的チェックを使用して浮動小数点アンダフローを検出するための命令を形成する方法を開示する。プログラム内のオペレーションが浮動小数点アンダフローを形成し得るか否かを決定する。次いで、特定の浮動小数点オペレーションがアンダフローを引き起こした回数を決定するために、カウンタをチェックする。カウンタが所定値未満である場合、オペレーションをトラップ・ルーチンを用いてランタイムにおいてコンパイルする。カウンタが所定値を越えている場合、オペレーションを明示的インライン・チェックを用いてリコンパイルする。トラップ・ルーチンに関連するデータ及び明示的インライン・チェックに関連するデータを、ランタイムにおいてダイナミックに形成し、格納する。

【００１７】

本発明は添付図面に基づく以下の説明によって更によく理解できる。

【００１８】

【発明の実施の形態】

本発明の特定の実施例を詳述する。この実施例は添付図面に示されている。本発明を特定の実施例に関連して詳述するが、これは本発明を１つの実施例に限定することを意図するものではない。逆に、添付の請求の範囲によって定義される本発明の精神及び範囲内に含まれる別例、変更例及び等価なものを包含することを意図している。

【００１９】

本発明は、特定の種類のオペレーションを任意のアーキテクチャ上で実施する選択可能な複数のインプリメンテーションの選択に取り組む。一般的に、従来の方法はコードをコンパイル・タイムにおいてスタティックに一度形成するか、またはランタイムにおいてダイナミックに一度形成する。ここで開示され、クレームされた発明は、バーチャル・マシンが、ランタイム・コンパイラによって形成するコード・セグメントを、複数の可能なコード・セグメントの中からランタイム・パフォーマンス・データに基づいてランタイムで選択することを可能にする。これによって、ダイナミック・コンパイラは１つのインプリメンテーションを選択でき、要求されれば、その効率を監視し、インプリメンテーションを変更する。

【００２０】

前記のように、浮動小数点アンダフローを検出して修正する２つの一般的な方法が存在する。このうちの一方の方法は、高速で短いコードであって、正常なプログラムの実行の中断をトラップの処理中に必要とするコードを使用することを含むトラップ法を使用する。他方の方法は、アンダフローを各浮動小数点オペレーション後に検出するために、コードをプログラム内に挿入することを含む。これは、コードを毎回実行することを必要とするが、プログラムを順番に続行することを可能にする。

【００２１】

拡張浮動小数点フォーマットの結果を単精度浮動小数点フォーマットまたは倍精度浮動小数点フォーマットで格納する必要が生じた際、アンダフローの問題が発生する。これは、例えば、結果をインテル・アーキテクチャ・コンピュータ上で算出し、次いで、この結果を単精度フォーマットまたは倍精度フォーマットで格納する際、発生し得る。より具体的には、この問題は、結果の指数が宛先において表示可能な最小の指数（ＩＥＥＥ７５４単精度フォーマットまたは倍精度フォーマット）より小さく、仮数が正確な結果を丸めた結果である際など、非常に小さな数を使用した演算を実施する際に、発生する。最も近い表現を宛先内に格納するために、仮数が不正確であること（その結果、丸められたこと）と

、仮数を丸めた方法及び理由と、を知る必要がある。この検出及び訂正は、例えば数値がゼロに近づく速度を測定すべく、非常に小さな数の正確さを維持するために、重要である。

【００２２】

図３は本発明の一実施例に従う、ジャバ・ソース・コードからネイティブ命令を形成することに関連した入力／出力及び実行ソフトウェア／システムを示すブロック図である。他の実施例では、本発明を、別の言語のためのバーチャル・マシンを用いて実施するか、またはジャバ・クラス・ファイル以外のクラス・ファイルを用いて実施できる。図の左側から始めると、第１入力は、カリフォルニア州マウンテンビューに所在するサン・マイクロシステムズによって開発されたジャバ（商標名）プログラム言語で書かれたジャバ・ソース・コード３０１である。ジャバ・ソース・コード３０１をバイトコード・コンパイラ３０３へ入力する。本質的に、バイトコード・コンパイラ３０３はソース・コード３０１をバイトコードへコンパイルするプログラムである。バイトコードは１つ以上のジャバ・クラス・ファイル３０５に含まれる。ジャバ・クラス・ファイル３０５はジャバ・バーチャル・マシン（ＪＶＭ）を有する任意のコンピュータ上で実行できるので、ポータブルといえる。バーチャル・マシンの複数のコンポーネントを図４により詳細に示す。ジャバ・クラス・ファイル３０５はＪＶＭ３０７へ入力される。ＪＶＭ３０７は任意のコンピュータ上に存在可能である。従って、ＪＶＭ３０７は、バイトコード・コンパイラ３０３を有する同一のコンピュータ上に存在する必要はない。ＪＶＭ３０７はインタプリタまたはコンパイラなどの幾つかの役割のうちの１つとしての動作が可能である。それがコンパイラとして動作する場合、それは“ジャスト・イン・タイム（ＪＩＴ）”コンパイラまたはアダプティブ・コンパイラとしてさらに動作し得る。インタプリタとして動作する際、ＪＶＭ３０７はジャバ・クラス・ファイル３０５に含まれる各バイトコード命令をインタプリトする。

【００２３】

図４は後で記述する図１１のコンピュータ・システム１０００によってサポートできるＪＶＭ３０７などのバーチャル・マシン３１１を示す図である。前記のように、コンピュータ・プログラム（例：ジャバ（商標名）プログラム言語で書かれたプログラム）をソースからバイトコードへ翻訳する際、ソース・コード３０１はコンパイルタイム環境３０３内のバイトコード・コンパイラ３０３へ提供される。バイトコード・コンパイラ３０９は、ソース・コード３０１をバイトコード３０５へ翻訳する。一般的に、ソフトウェア開発者がソース・コード３０１を形成した時点において、ソース・コード３０１はバイトコード３０５へ翻訳される。

【００２４】

一般的に、バイトコード３０５はネットワーク（例：図１１のネットワーク・インターフェース１０２４）を通じて複製、ダウンロード若しくは配布されるか、または図１１の一次ストレージ１００４などのストレージ・デバイス上へ格納され得る。本実施例では、バイトコード３０３はプラットフォームから独立している。即ち、バイトコード３０３は、適切なバーチャル・マシン３１１を実行している実質的に全てのコンピュータ・システム上で実行可能である。バイトコードをコンパイルすることによって形成されたネイティブ命令は、後からＪＶＭで使用するために保持できる。この結果、インタプリトされたコードより優れた速度の効果をネイティブ・コードへ提供するために、翻訳のコストは複数の実行を通じて償却される。例えば、ジャバ（商標名）環境では、バイトコード３０５はＪＶＭを実行しているコンピュータ・システム上で実行可能である。

【００２５】

バイトコード３０５はバーチャル・マシン３１１を含むランタイム環境３１３へ提供される。一般的に、ランタイム環境３１３は図１１のＣＰＵ１００２などのプロセッサを使用して実行できる。バーチャル・マシン３１１はコンパイラ３１５、インタプリタ３１７及びランタイム・システム３１９を含む。一般的に、バイトコード３０５はコンパイラ３１５またはインタプリタ３１７へ提供可能である。

【００２６】
バイトコード３０５をコンパイラ３１５へ提供した際、バイトコード３０５に含まれるメソッドはネイティブ・マシン命令（図示せず）へコンパイルされる。その一方、バイトコード３０５をインタプリタ３１７へ提供した際、バイトコード３０５は１バイトコードずつインタプリタ３１７内へ読み込まれる。そして、各バイトコードがインタプリタ３１７内へ読み込まれることにより、インタプリタ３１７は各バイトコードによって定められたオペレーションを実施する。一般的に、インタプリタ３１７は実質的に連続してバイトコード３０５を処理し、バイトコード３０５に関連するオペレーションを実施する。
【００２７】
オペレーティング・システム３２１がメソッドを呼び出す際、このメソッドをインタプリトされたメソッドとして呼び出すことを決定した場合、ランタイム・システム３１９はメソッドをインタプリタ３１７から獲得できる。その一方、メソッドをコンパイルされたメソッドとして呼び出すことを決定した場合、ランタイム・システム３１９はコンパイラ３１５を起動する。次いで、コンパイラ３１５はネイティブ・マシン命令をバイトコード３０５から形成し、マシン言語命令を実行する。一般的に、バーチャル・マシン３１１を終了する際、マシン言語命令は廃棄される。バーチャル・マシン、より詳細には、ジャバ（商標名）バーチャル・マシンのオペレーションはティム・リンドホルム及びフランク・イェリンによる“ジャバ（商標名）バーチャル・マシン詳説”（ＩＳＢＮ０－２０１－６３４５２－Ｘ）と称される文献に更に詳細に開示されており、この文献の内容はこの開示をもって本明細書中に開示したものとする。
【００２８】
前記のように、ジャバ・プログラム内の命令は時には１より多い方法でコンパイルできる。先の例の続きを説明する。乗算（ＦＭＵＬ）オペレーションまたは除算（ＦＤＩＶ）オペレーションなどのアンダフローを潜在的に引き起こし得る浮動小数点オペレーションは、少なくとも２つの方法（明示的チェックを伴う方法またはトラップを伴う方法）でコンパイル可能である。図５はジャバ・プログラムからネイティブ命令の異なるバージョンがどのように形成され得るかを示す流れ図である。バイトコード・コンパイラによってジャバ・クラス・ファイル内にコンパイルされた後、ブロック４０３（図３のシステム３０７）において、ジャバ・プログラム４０１（図３及び図４の入力３０１）は、ＪＶＭによってバイトコードからネイティブ・マシン命令へランタイムでコンパイルされる。ＪＶＭは例示を目的として使用しているだけである。当業者に知られているように、バーチャル・マシンは任意の入力表現からネイティブ命令セットへの一般的な翻訳に用いる。この場合、インプリメンテーションの選択が存在する。ＪＶＭによるジャバ・クラス・ファイルのコンパイレーションの方法を図６に基づいて以下に記述する。
【００２９】
前記のように、ＪＶＭは２つ役割、即ち、クラス・ファイルに含まれるジャバ・バイトコードをインタプリトすることと、クラス・ファイルをコンパイルし、これによって、ＪＶＭを有する同一コンピュータ上で実行されるネイティブ命令セット（即ち、これらはポータブルでない）を形成することのうち、いずれか一方の役割を担うことが可能である。従って、コンパイラとして動作するＪＶＭの場合、ブロック４０５に示すように、ＪＶＭがバイトコードをどのようにコンパイルするかに依存して、様々なネイティブ命令セットが同一のジャバ・プログラムから形成され得る。浮動小数点オペレーションを例として使用した場合、ネイティブ命令４０７はその全てのＦＭＵＬ及びＦＤＩＶにおける明示的チェック（即ち、インライン）を含むことが可能な一方、ネイティブ命令４０９はこれら同じ浮動小数点オペレーションのためのトラップのみを含むことが可能であり、また、ネイティブ命令４１１はこれら両方の組み合わせを含むことが可能である。
【００３０】
どのコンパイレーション・ルートを採用するか（即ち、ジャバ・ソース・コードをコンパイルすることと、ジャバ・ソース・コードをインタプリトすることと、どのようにもしくはいつジャバ・ソース・コードを実行するかについて他のオペレーションを実施すること

のうちのどれか）をランタイムにおいて決定するＪＶＭを、図５が示していないことは注目に値する。これに代えて、ＪＶＭの採用したコンパイレーション・ルートが、そのコードをランタイムにおいてコンパイルすることである場合、それを異なる“方法”で実行し、これによって、ネイティブ命令の異なるセットを形成することを、図５は示している。このプロセスを図６に関連して詳述する。

【００３１】

図６は本発明の一実施例に従うジャバ・バイトコードをネイティブ・マシン命令へコンパイルするジャバ・バーチャル・マシンのプロセスを示すフローチャートである。ステップ５０１では、ＪＶＭは１つ以上のバイトコード命令をジャバ・クラス・ファイルから取り出す。ステップ５０３では、ＪＶＭは特定の命令を複数の方法でコンパイルできるか否かを決定する。複数の方法でコンパイルできるバイトコード命令の特定の例は図８に基づいて後で詳述する。ＪＶＭが、ＩＡＤＤオペレーションまたはＬＳＵＢオペレーションのように、１つの方法でしか命令をコンパイルできないことを決定した場合、ＪＶＭはバイトコードをステップ５０５でコンパイルする。以前に取り出したバイトコードをコンパイルした後、ＪＶＭは残されたバイトコードが存在するか否かをステップ５１５で決定する。

【００３２】

バイトコードをコンパイルする複数の方法が存在することを、ＪＶＭが決定した場合、ＪＶＭはどの方法でバイトコードをコンパイルするかをステップ５０７で決定すべく処理を続行する。記述した実施例では、図７に詳細を示すように、この決定を行うために、ＪＶＭはメカニズムを使用する。このメカニズムは、バイトコード命令をコンパイルできる異なる方法のそれぞれのダイナミックに形成されたプロフィール情報を使用することを含む。ステップ５０９では、ＪＶＭは、バイトコードをデフォルトの方法を用いてコンパイルするか否かを決定する。そのデフォルトの方法は、一般的に、ランタイム・コンパイラのライタが、その時点で利用可能なオプションを検討した後、最も効率的または論理的な方法であると確信する方法である。

【００３３】

ステップ５０９で、ＪＶＭがバイトコードを第１の方法でコンパイルすることを決定した場合、ステップ５１３で、ＪＶＭはこれを実施して、図５のネイティブ命令セットＡなどの第１ネイティブ命令セットを形成する。次いで、ＪＶＭは、他のバイトコードがクラス・ファイル内に存在するか否かをステップ５１５で決定する。存在する場合、ＪＶＭは、次のバイトコードを取り出すためにステップ５０１へ戻る。バイトコードが１つも存在しない場合、プロセスは完了する。ステップ５０９で、ＪＶＭが、バイトコードをコンパイルする方法が第１の方法、即ち、デフォルトの方法でないことを決定した場合、ＪＶＭはバイトコード命令を別のコンパイレーション技術を使用してステップ５１１でコンパイルする。次いで、ＪＶＭはステップ５１３からの場合と同様に処理を続行し、コンパイルする残りのバイトコードが存在するか否かをステップ５１５において決定する。簡単にするために、２つの異なる方法のみを図６に示すが、本発明はバイトコードをコンパイルする３つ以上の方法へ効果的に適用できる。

【００３４】

図７は本発明の一実施例に従うダイナミックに形成されたプロフィール・データを含むネイティブ・マシン命令をどのように形成するかを示すブロック図である。ＪＶＭによってバイトコード命令をコンパイルし得る異なる方法のそれぞれに関する情報を含む点を除けば、図７は図５に類似している。その情報は、図６のステップ５０３に示すように、バイトコードをネイティブ命令へコンパイルする複数の方法が存在することが決定されると、形成される。ジャバ・プログラム６０１が図７のトップに位置している。バイトコード・コンパイラによってバイトコードへコンパイルされた後、ジャバ・プログラム６０１はジャバ・バーチャル・マシン６０３へ入力される。次いで、ＪＶＭは、バイトコードをネイティブ命令へコンパイルし得る異なる方法に基づいて、幾つかの異なるネイティブ命令セット６０５を出力可能である。ネイティブ命令セット６０５は、ランタイムにおいてダイナミックに集められたデータ６０７を格納するデータ・スペースを更に含むことができる

。この情報は、カウンタ、タイミング・データ、及びバイトコードをコンパイルする特定の方法の効率に関する他の情報のようなプロフィール情報を含んでよい。

【００３５】

ダイナミックに集められたデータは、ネイティブ命令と一緒に格納可能であり、そして、バイトコードをＪＶＭによってコンパイルする間に更新できる。１つの実施例では、図６のステップ５０７で最初に説明したように、ＪＶＭは、バイトコードをどの方法でコンパイルするかを決定するために、この情報を調べる。ダイナミック・プロフィール・データは、例えば、コンパイルの特定の方法が効率的であり続けるか否か、バイトコードがその方法で何回実行されたか、または特定の時間を経過したか否かを決定するために、ＪＶＭによって使用され得る。ＪＶＭは、現在の命令がバイトコードの最も効率的なインプリメンテーションであるか否かを決定するために、コンパイル中におけるデータへのクエリーが可能である。効率的に実行されていないことがＪＶＭによって確認された任意のバイトコードを、ＪＶＭによってリコンパイルできる。ＪＶＭは、データ６０７へのクエリーによってどのようにバイトコードがコンパイルされるべきかを決定すると、図６のステップ５０９に示すように、ＪＶＭはこれが第１の方法（デフォルトの方法）であるべきか、または他の方法のうちの１つであるべきかを決定することによって、処理を継続し得る。

【００３６】

図８は本発明の記述した実施例に従う、浮動小数点オペレーションをコンパイルし、アンダフローが発生した場合、このアンダフローを訂正する方法を決定するジャバ・バーチャル・マシンを示すフローチャートである。前記のように、浮動小数点オペレーションのコンパイルは、プログラムを幾つかの方法でコンパイルする方法を決定する特定の例である。より一般的には、コンパイレーションをヒューリスティックス、即ち、コード（例：テーブルスイッチ命令のコンパイル）の平均的振る舞いに関する仮定によってガイドする任意のアプリケーションは、前記のようにプログラムを幾つかの方法でコンパイルする方法を決定する方法を利用できる。ステップ７０１では、ＪＶＭはバイトコードをジャバ・クラス・ファイルから取り出す。ステップ７０３では、ＪＶＭは、バイトコード命令がアンダフローを形成し得るか否かを決定する。アンダフローを形成し得る一般的な２つの浮動小数点オペレーションは乗算及び除算である。記述した実施例では、ＪＶＭは、特定の命令がアンダフローの問題を形成できないことを決定すると、ステップ７０５に示すように、ＪＶＭはバイトコードをコンパイルすべく処理を続行する。

【００３７】

命令がアンダフローの問題を潜在的に形成し得る場合、ＪＶＭは、どのようにアンダフローが検出され、訂正されるを決定するプロセスを開始する。前記のように、記述した実施例では、ＪＶＭはアンダフローを検出するための明示的チェック（即ち、インライン・コード）またはトラップを使用し得る。別の実施例では、前記の方法に代えて、または前記の方法に加えて、アンダフローを検出する別の方法を使用できる。

【００３８】

ステップ７０７では、バーチャル・マシンは、アンダフローを検出するためのトラップに関連するカウンタが所定の閾値を越えたか否かをチェックする。トラップの一部として、各トラッピング命令に関連するカウンタをインクリメントするための命令が含まれている。任意のカウンタがある閾値を越えた場合、バイトコード・トランスレータを再呼び出しするための命令が更に含まれている。カウンタは、特定のバイトコード（この例では、浮動小数点命令）をどの方法でコンパイルするのかを決定するためにチェックできる情報、即ち、プロフィール・データの１つの例である。図７に示すように、カウンタ及びこれに類する情報６０７は、ネイティブ命令セットと一緒に維持可能である。別の実施例では、ＪＶＭがどの方法でバイトコードをコンパイルするために使用されるべきかを決定するために、カウンタに代えて、またはカウンタと一緒に、タイマなどの別の種類のデータを使用できる。

【００３９】

ＪＶＭ上でのジャバ・クラス・ファイルの１回の実行中に複数回実施される特定の浮動小

数点オペレーションを実行するために、特定の方法が使用される度に、カウンタが更新され得る。例えば、特定の命令を条件付きループ内に有することによって、カウンタの更新を行い得る。トラップを使用して訂正するアンダフローを特定の浮動小数点オペレーションが引き起こす度に、カウンタはインクリメントされる。図６に示すステップ５０９で説明した“第１の方法”は、命令をコンパイルするトラップ法に該当し得る。記述した実施例では、一般的に採用される実行のパスにおけるオペレーションの数を減少させることが好ましいため、第１の方法でコンパイルする際、カウンタを避けることが望ましい。図８に示す特定の実施例では、コンパイルする（そして、アンダフローを潜在的に形成する）特定の浮動小数点オペレーションのためのカウンタが閾値へ達していない場合、ＪＶＭは第１の方法（この例では、命令をコンパイルするトラップ法）の使用を継続する。カウンタが閾値を越えている場合、“第２の方法”でのリコンパイレーションを行うために、フラグをこの命令／方法に対して立てる。前記のように、各トラッピング命令に関連するカウンタをインクリメントするための命令がトラップの一部として含まれている。

【００４０】

トラップ法を使用した命令のコンパイルにおける第１工程は、ステップ７０９に示すように、トラップ・ハンドラがセットアップされているか否かを決定することである。ジャバ・クラス・ファイルがトラップを初めて呼び出した際、トラップ・ハンドラが形成される。トラップ・ハンドラを必要とするコードをコンパイルすることを、コンパイラが初めて決定した際（トラップ・ハンドラはこれ以前に必要となり得ない）、トラップ・ハンドラが（ＪＶＭによって）セットアップされる。記述した実施例及び殆どのジャバ・プログラムでは、１つのトラップ・ハンドラがプログラム内の各スレッドに対して存在する。スレッドの詳細な説明は“ジャバ言語詳説”に開示されており、この文献の内容はこの開示をもって本明細書に開示したものとする。トラップ・ハンドラを形成する場合、これはステップ７１１で行われる。トラップ・ハンドラが特定のスレッドに対して既にセットアップされている場合、ステップ７１３に示すように、ＪＶＭはトラップ法を使用して命令をコンパイルする。トラップ・ハンドラを使用して命令をコンパイルするこのプロセスは、図９において更に詳述する。コンパイルを終えると、ＪＶＭは他のバイトコードがジャバ・クラス・ファイル内に存在するか否かを見るためにステップ７１５においてチェックする。存在する場合、ＪＶＭはステップ７０１へ戻り、プロセスを繰り返す。

【００４１】

ステップ７０７では、ＪＶＭは、カウンタが所定値を超えたか否か（即ち、命令が特定の方法で所定回数を超えて実行されたか否か）をチェックする。超えている場合、ＪＶＭはバイトコードを次の方法でコンパイルする。この例では、その方法は、浮動小数点アンダフローを検出し、訂正するために、ステップ７１７に示すように、明示的チェック（インライン・コード）を使用する。別の実施例では、バイトコード・トランスレータがコンパイルされたバイトコードを特定の方法で実行し続けるべきか否かを決定するために、カウンタ以外の基準を使用することができる。バイトコード命令をコンパイルする明示的チェック法は図１０において詳述する。ステップ７１９では、バーチャル・マシンは明示的チェックと一緒に使用するタイマをセットする。その時間は、明示的チェック法を使用した時間的な長さを測定するために、使用される。次いで、ステップ７１５では、バーチャル・マシンはそれ以上のバイトコードが存在するか否かをチェックする。何も存在しない場合、ジャバ・クラス・ファイル内のバイトコードをコンパイルするプロセスは完了する。

【００４２】

図９は浮動小数点命令からのアンダフローを処理するために、トラップを使用する図８のステップ７１３のプロセスの詳細を示すフローチャートである。ステップ８０１では、ＪＶＭは、ジャバ・クラス・ファイル内のどのバイトコード命令がトラップを呼び出しているかを決定する。バーチャル・マシンは、どの命令がトラップを呼び出しているかを決定すると、バーチャル・マシンは浮動小数点オペレーションに関連するカウンタをステップ８０３でインクリメントする。図７で述べたように、カウンタをネイティブ命令と一緒に維持し得る。カウンタがインクリメントされると、バーチャル・マシンはカウンタの値を

ステップ８０５でチェックする。カウンタが閾値を越えている場合、ステップ８０７において、浮動小数点命令を含むモジュールに対して、リコンパイルするためにフラグを立てる。

【００４３】

記述した実施例では、モジュールを即座にリコンパイルしない。その代わりに、そのリソース及びアクティビティのレベルに基づいてバーチャル・マシンによって決定された時点において、モジュールは、リコンパイルするために、キューへ加えられる。別の実施例では、モジュールを即座にまたは設定時間にコンパイルし得る。モジュールをリコンパイルする実際の時間とは無関係に、記述した実施例では、カウンタに基づいて、バーチャル・マシンはリコンパイルすることを決定する。別の実施例では、バーチャル・マシンは図７で述べたネイティブ命令と一緒に格納可能なダイナミックに形成されたプロフィール・データから得られた別の印を使用できる。ステップ８０７で、モジュールに対して、リコンパイルするために、フラグを立てるか、別の方法でマークすると、バーチャル・マシンは、ジャバ・クラス・ファイル内にそれ以上のバイトコードが存在するか否かをチェックするために、図８のステップ７１５へ戻る。カウンタがステップ８０５で所定値を超えていない場合、ＪＶＭは、ステップ８０９で、浮動小数点アンダフローを処理するために、トラップを使用することによって処理を継続する。次いで、バーチャル・マシンは図８のステップ７１５へ戻る。

【００４４】

図１０は図８のステップ７１７で述べた浮動小数点アンダフローを検出し、訂正するための明示的チェック・ルーチンを示すフローチャートである。前記のように、明示的チェックは、浮動小数点アンダフローを検出し、訂正するために、ＪＶＭによってジャバ・クラス・ファイルから形成されたネイティブ命令セット内へ挿入されたコードである。図１０は、バーチャル・マシンが、明示的チェック法（明示的チェック法は図６のステップ５１１で述べた“次の方法”に該当し得る）でコンパイルする時、または図９のステップ８０７同様に、リコンパイルするために浮動小数点命令を含むモジュールに対してフラグを立てる時を、どのように決定するかを示す。ステップ９０１では、バーチャル・マシンは、浮動小数点アンダフローを訂正するために、明示的チェック法を使用する。前記のように、明示的チェックによってアンダフローが何回検出されたかを追跡するために、明示的カウンタを挿入し得るか、またはタイマを使用し得る。カウンタをこのパスで使用することによる１つの潜在的問題としては、カウンタが処理の期間で比較的コスト高となり得る点が挙げられる。別の実施例では、タイマ及びカウンタの組み合わせを使用可能である。次いで、バーチャル・マシンは明示的チェックを最初に使用した時点から何時間経過したかをチェックする。図８のステップ７１９において思い起こすと、記述した実施例では、命令を明示的チェックとしてコンパイルした後、バーチャル・マシンはタイマをセットする。所定時間を経過したか否かを決定するために、これと同じタイマをステップ９０３で使用する。記述した実施例では、所定時間を経過した場合、ステップ９０５において、ＪＶＭはコンパイルするためにモジュールに対してフラグを立てるか、またはマークをつける。

【００４５】

１つの実施例では、特定の時間を経過した後、ＪＶＭはバイトコードをリコンパイルする。これは、現在コンパイルしている方法をデフォルトの方法へリセットすることによって、ジャバ・クラス・ファイルの実行が新たな状況へ適合せずに潜在的に非効率的になることを防止するために、行われる。バーチャル・マシンが決定した時点で、リコンパイルするためにモジュールに対してフラグを立て、キューへ加えると、この特定の浮動小数点命令に対応するカウンタ及び他のプロフィール・データをリセットまたはリフレッシュし、これによって、新しいプロフィール情報を維持できる。カウンタをステップ９０７でリセットする。次いで、バーチャル・マシンは図８のステップ７１９へ戻る。

【００４６】

本発明はコンピュータ・システム内に格納された情報を使用する様々なコンピュータ実装

オペレーションを使用する。これらのオペレーションは物理量の物理操作を必要とするオペレーションを含む（但し、同オペレーションに限定されない）。一般的に、必ずしも必要でないが、これらの量は格納、転送、結合、比較及び他の操作が可能な電気信号または磁気信号の形態をなす。本発明の一部を構成するここで記述するオペレーションは、有用なマシン・オペレーションである。実施する操作は形成、識別、実行、決定、比較、実行、ダウンロードまたは検出等の用語で示されることが多い。主に共通の用法を得る理由で、これらの電気信号または磁気信号をビット、値、エレメント、変数、キャラクター等として示すと時に都合が良い。しかし、これらの用語またはこれらに類似する用語の全ては適切な物理量に関連づけるべきであり、かつ、これらの量に適用された都合の良いラベルにすぎない点を覚えておく必要がある。

【００４７】

更に、本発明は前記のオペレーションを実施するためのデバイス、システムまたは装置に関する。システムは要求された目的のために特別に構築可能であり、または、システムは、そのコンピュータに格納されたコンピュータ・プログラムによって選択的に作動または構成される汎用コンピュータとすることが可能である。前記のプロセスは特定のコンピュータまたは他のコンピューティング装置に本質的には関連しない。特に、様々な汎用コンピュータを、ここで開示されていることに基づいて記述されたプログラムと併用してよく、あるいは、これに代えて、要求されたオペレーションを実施するために、より特別なコンピュータ・システムを形成することはより都合が良い。

【００４８】

図１１は本発明の一実施例に従う処理の実施に適した汎用コンピュータ・システム１０００のブロック図である。例えば、ＪＶＭ３０７、バーチャル・マシン３１１またはバイトコード・コンパイラ３０３を汎用コンピュータ・システム１０００上で実行できる。図１１は汎用コンピュータ・システムの一実施例を示す。本発明の処理を実施するために、他のコンピュータ・システム・アーキテクチャ及び構成を使用することができる。以下に記述する様々なサブシステムからなるコンピュータ・システム１０００は、少なくとも１つのマイクロプロセッサ・サブシステム（中央処理装置、即ち、ＣＰＵとも称される）１００２を含む。即ち、ＣＰＵ１００２はシングルチップ・プロセッサまたはマルチプル・プロセッサによって実現し得る。ＣＰＵ１００２はコンピュータ・システム１０００のオペレーションを制御する汎用デジタル・プロセッサである。メモリから取り出した命令を使用して、ＣＰＵ１００２は入力情報の受信及び操作と、出力デバイス上での情報の出力及び表示とを制御する。

【００４９】

ＣＰＵ１００２は、メモリ・バス１００８を介して、一般的にランダム・アクセス・メモリ（ＲＡＭ）からなる第１の一次ストレージ１００４に双方向接続され、一般的にリード・オンリ・メモリ（ＲＯＭ）からなる第２の一次ストレージ領域１００６に単方向接続されている。当該技術分野でよく知られているように、一次ストレージ１００４は汎用ストレージ領域及び作業メモリとして使用可能であり、さらには入力データ及び処理済みデータを格納するためにも使用できる。更に、ＣＰＵ１００２上で処理されるプロセスのためのデータ及び命令を格納する以外に、一次ストレージ１００４はプログラミング命令及びデータを格納可能であり、そして、一般的に、データ及び命令を、メモリ・バス１００８の間を双方向で高速転送するために、使用される。同様に、当該技術分野でよく知れられているように、一次ストレージ１００６は、一般的に、ＣＰＵ１００２がその機能を果たすために使用する基本オペレーティング命令、プログラム・コード、データ及びオブジェクトを含む。一次ストレージ・デバイス１００４，１００６は、例えば、データ・アクセスが双方向または単方向のいずれを必要とするかに依存して、以下に詳述する適切なコンピュータ読み取り可能ストレージ媒体を含み得る。ＣＰＵ１００２は、キャッシュ・メモリ１０１０において、頻繁に必要となるデータを超高速で直接取り出し、そして、格納できる。

【００５０】

取り外し可能大容量ストレージ・デバイス１０１２はコンピュータ・システム１０００のための別のデータ・ストレージ容量を提供し、ペリフェラル・バス１０１４を介してＣＰＵ１００２に双方向または単方向のいずれかで接続されている。例えば、ＣＤ－ＲＯＭとして知られている特定の取り外し可能大容量ストレージ・デバイスは、一般的にデータを単方向でＣＰＵ１００２へ送信する。その一方、フロッピー・ディスクはデータを双方向でＣＰＵ１００２へ送信し得る。ストレージ１０１２は、磁気テープ、フラッシュ・メモリ、搬送波に具現化された信号、スマート・カード、ポータブル大容量ストレージ・デバイス及び他のストレージ・デバイス等のコンピュータ読み取り可能媒体を更に含み得る。また、固定大容量ストレージ１０１６は、別のデータ・ストレージ容量を提供し、ペリフェラル・バス１０１４を介してＣＰＵ１００２に双方向で接続されている。一般的に、これらの媒体へのアクセスは一次ストレージ１００４，１００６へのアクセスより遅い。大容量ストレージ１０１２，１０１６は、一般的に、ＣＰＵ１００２が頻繁に使用しない他のプログラミング命令及びデータ等を格納する。必要に応じて、大容量ストレージ１０１２，１０１６内に保持された情報は、一次ストレージ１００４（例：ＲＡＭ）の一部を構成するバーチャル・メモリとして標準的に組込み可能である。

【００５１】

ストレージ・サブシステムへのＣＰＵ１００２のアクセスを提供する以外に、ペリフェラル・バス１０１４は、同様に、他のサブシステム及びデバイスへのアクセスを提供するために使用される。記述した実施例では、これらは、ディスプレイ・モニタ１０１８及びアダプタ１０２０、プリンタ・デバイス１０２２、ネットワーク・インターフェース１０２４、補助入力／出力装置インターフェース１０２６、サウンド・カード１０２８及びスピーカ１０３０、並びに必要とされる他のサブシステムを含む。

図示するように、ネットワーク接続を使用することにより、ネットワーク・インターフェース１０２４はＣＰＵ１００２を別のコンピュータ、コンピュータ・ネットワークまたは通信ネットワークへ接続可能にする。前記の方法のステップを実行するうえで、ネットワーク・インターフェース１０２４を通じることで、ＣＰＵ１００２が、オブジェクト、プログラム命令またはバイトコード命令などの情報を別のネットワーク内のコンピュータから受信するか、または情報を別のネットワーク内のコンピュータに出力することを意図している。ＣＰＵで実行する命令のシーケンスとしてしばしば表される情報は、例えば、搬送波に具現化されたコンピュータ・データ信号の形態で別のネットワークに対して送受信可能である。インターフェース・カードまたはこれに類似するデバイスと、ＣＰＵ１００２によって実行される適切なソフトウェアとは、コンピュータ・システム１０００を外部ネットワークへ接続し、標準プロトコルに従ってデータを転送するために使用できる。即ち、本発明で具現化される方法はＣＰＵ１００２上で単独で実行してよいし、または、処理の一部を共有する遠隔ＣＰＵと協動することにより、インターネット、イントラネットワーク若しくはローカル・エリア・ネットワーク等のネットワークを通じて実行してよい。また、別の大容量ストレージ・デバイス（図示せず）をネットワーク・インターフェース１０２４を通じてＣＰＵ１００２へ接続してよい。

【００５２】

補助入力／出力装置インターフェース１０２６は、ＣＰＵ１００２に他のデバイスにデータを送信させ、より一般的に、そのデバイスからのデータを受信させ得る汎用及びカスタム・インターフェースを表す。キーボード１０３６またはポインタ・デバイス１０３８からの入力を受信し、さらにはデコードしたシンボルをキーボード１０３６またはポインタ・デバイス１０３８からＣＰＵ１００２へ送信するために、キーボード・コントローラ１０３２がローカル・バス１０３４を通じてＣＰＵ１００２へ接続されている。ポインタ・デバイスは、マウス、スタイラス、トラック・ボールまたはタブレットであってよく、そして、グラフィカル・ユーザ・インターフェースとの相互作用に有用である。

【００５３】

加えて、本発明の実施例は、さらに、様々なコンピュータ実行オペレーションを実施するためのプログラム・コードを含むコンピュータ読み取り可能媒体を有するコンピュータ

・ストレージ製品に関する。コンピュータ読み取り可能媒体は、コンピュータ・システムによって後からの読み取りが可能なデータを格納し得る任意のデータ・ストレージ・デバイスである。コンピュータ読み取り可能媒体の例としては、ハード・ディスクと、フレキシブル・ディスクと、特定用途向け集積回路（ＡＳＩＣ）またはプログラム可能論理回路（ＰＬＤ）などの特別に形成されたハードウェア・デバイスと、を含めた前記の全ての媒体が挙げられる（但し、これらに限定されない）。

【００５４】

前記のハードウェア・エレメント及びソフトウェア・エレメントが標準的な設計及び構成を有することを当業者は理解し得る。本発明を使用するのに適した他のコンピュータ・システムは別のサブシステムまたは少数のサブシステムを含み得る。更に、メモリ・バス１００８、ペリフェラル・バス１０１４及びローカル・バス１０３４はサブシステムをリンクするのに役立つ任意の相互接続方式の実例である。例えば、ローカル・バスはＣＰＵを固定大容量ストレージ１０１６及びディスプレイ・アダプタ１０２０へ接続するために使用可能である。しかし、図１１に示すコンピュータ・システムは本発明を使用するのに適したコンピュータ・システムの例である。サブシステムの別の構成を有する他のコンピュータ・アーキテクチャを利用し得る。

【００５５】

以上、理解を容易にする目的で、本発明をある程度詳しく説明したが、本発明の請求の範囲内において、特定の変更及び修正を実施しても良い。例えば、トラップ法及び明示的インライン・チェック法を浮動小数点アンダフローに関連して詳述したが、アンダフローを検出する他のツールも使用し、本発明に組み入れることができる。別の例では、命令をコンパイルする２つの方法を開示したが、プログラムをコンパイルする更に多くの方法を利用可能な場合、本発明の方法及び装置はこれら２つより多い方法に対応可能である。更に、本発明を浮動小数点アンダフロー・オペレーションを使用して説明したが、プラットフォーム固有のバリエーションに起因する問題の解決に使用するために、本発明はインプリメンテーションをインテリジェントで、ダイナミックに選択できる。浮動小数点アンダフローは、この種の問題の１つに過ぎない。更に、本発明の方法及び装置の両方をインプリメントするための代わりの方法が存在することを認識する必要がある。従って、本実施例は例示を目的とするものであって、限定を目的としない。更に、本発明はここで与えられた詳細部分に限定されることなく、添付の請求の範囲及びそれに等価なものの範囲内で変更できる。

【図面の簡単な説明】

【図１】従来技術で知られているような倍精度浮動小数点の一般的なフォーマットと拡張精度浮動小数点のフォーマットを示すブロック図である。

【図２】従来技術で知られているような浮動小数点アンダフローを検出する２つの方法を示す図である。

【図３】ジャバ・ソース・コードを含むジャバ（商標名）プログラムを特定のプラットフォーム、即ち、コンピュータ上で実行されるネイティブ・コードに変換することを示すブロック／プロセス図である。

【図４】図１１のコンピュータ・システム１０００によってサポートされているバーチャル・マシン３０７を示す図である。

【図５】ネイティブ命令の異なるバージョンがジャバ・プログラムからどのように形成されるのかを示すブロック図である。

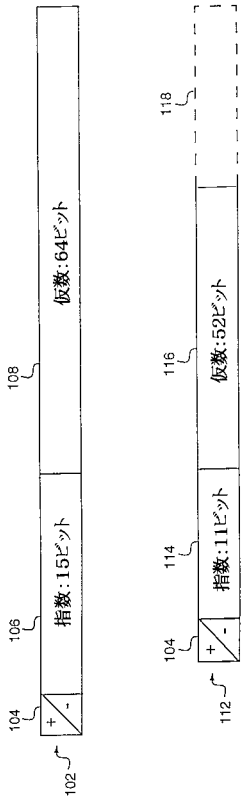
【図６】本発明の一実施例に従う、ジャバ・バイトコードをコンパイルするジャバ・バーチャル・マシンのプロセスを示すフローチャートである。

【図７】本発明の一実施例に従う、ダイナミックに形成されたプロフィール・データを含むネイティブ命令がどのように形成されるのかを示すブロック図である。

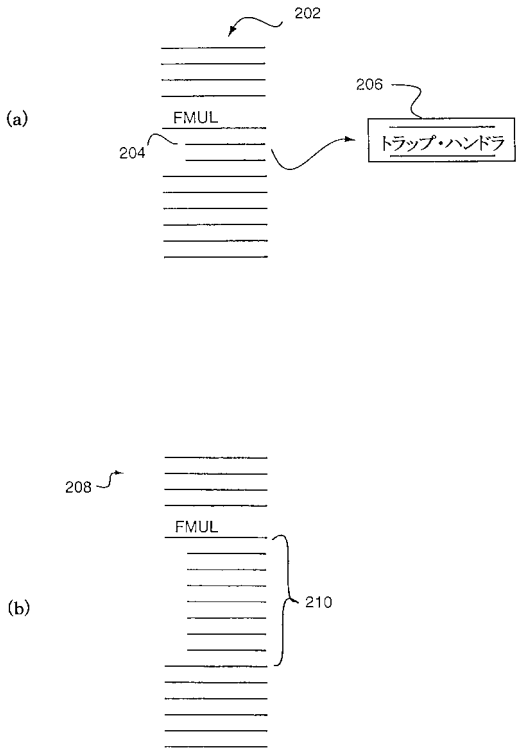
【図８】本発明の一実施例に従う、浮動小数点オペレーションをコンパイルするとともに、アンダフローが発生した場合、このアンダフローを訂正する方法を決定するジャバ・バーチャル・マシンを示すフローチャートである。

【図９】図８のステップ７１３に示す浮動小数点命令からのアンダフローを処理するトラップ・ルーチンを使用するプロセスをより詳細を示すフローチャートである。
【図１０】図８のステップ７１７に示す浮動小数点アンダフローを検出し、訂正するための明示的チェック・ルーチンを示すフローチャートである。
【図１１】本発明の一実施例を実現することに適した一般的なコンピュータ・システムのブロック図である。
【符号の説明】
１０００…コンピュータ・システム
１００２…ＣＰＵ
１００４…第１の一次ストレージ
１００６…第２の一次ストレージ領域
１００８…メモリ・バス
１０１０…キャッシュ・メモリ
１０１２…取り外し可能大容量ストレージ・デバイス
１０１４…ペリフェラル・バス
１０１６…固定大容量ストレージ
１０１８…ディスプレイ・モニタ
１０２０…アダプタ
１０２２…プリンタ・デバイス
１０２４…ネットワーク・インターフェース
１０２６…補助入力／出力装置インターフェース
１０２８…サウンド・カード
１０３０…スピーカ
１０３２…キーボード・コントローラ
１０３４…ローカル・バス
１０３６…キーボード
１０３８…ポインタ・デバイス

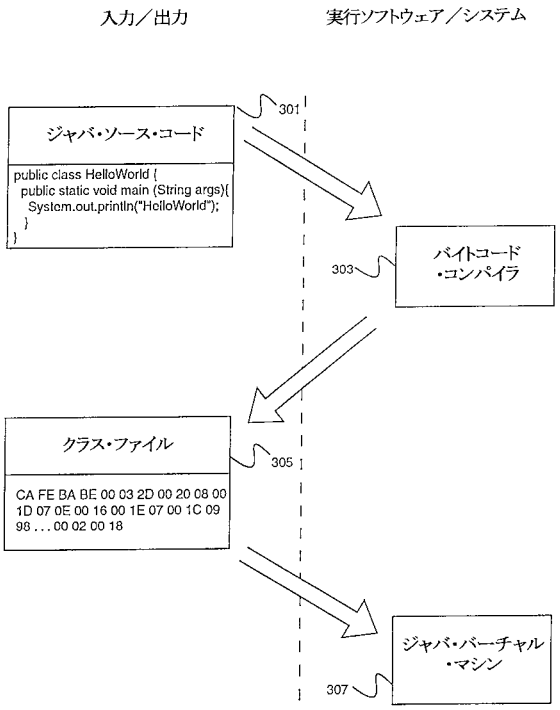
【図１】


仮数:64ビット
指数:15ビット
+
-
102
104
106
108
仮数:52ビット
指数:11ビット
112
114
116
118

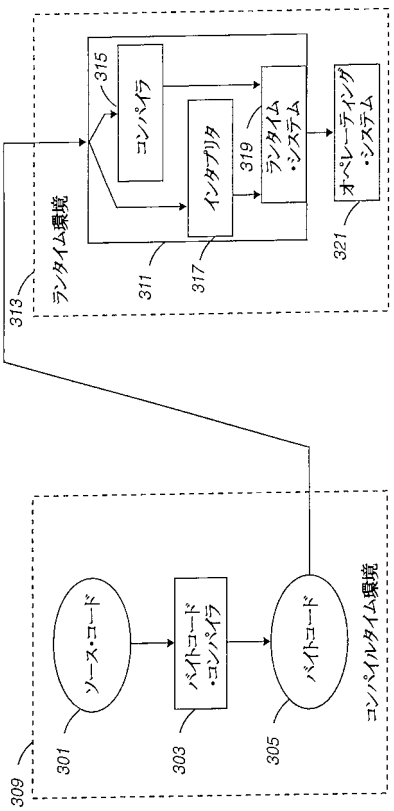
【図２】


(a)
202
FMUL
204
206
トラップ・ハンドラ
(b)
208
FMUL
210

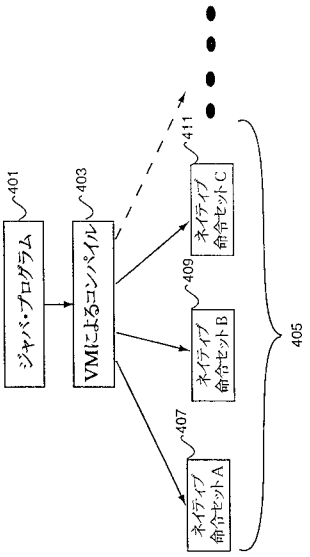
【図３】


入力／出力
実行ソフトウェア／システム
ジャバ・ソース・コード
public class HelloWorld {
public static void main (String args){
System.out.println("HelloWorld");
}
}
301
バイトコード
・コンパイラ
303
クラス・ファイル
CA FE BA BE 00 03 2D 00 20 08 00
1D 07 0E 00 16 00 1E 07 00 1C 09
98 . . . 00 02 00 18
305
ジャバ・バーチャル
・マシン
307

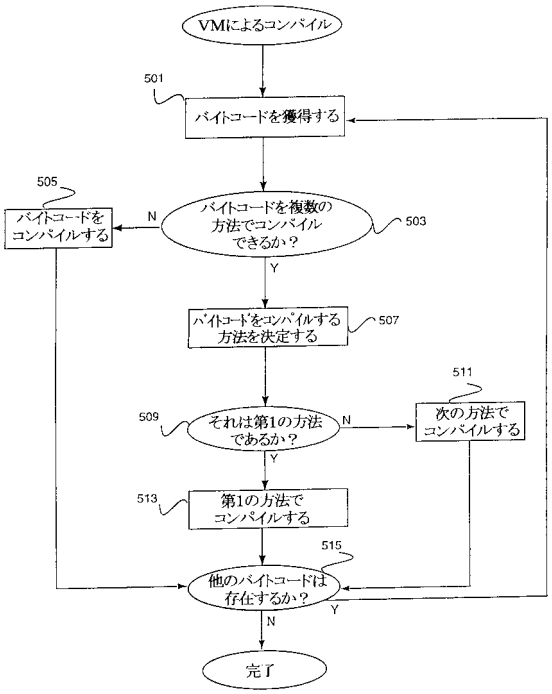
【図４】


ランタイム環境
コンパイラ
315
インタプリタ
317
ランタイム
・システム
319
オペレーティング
・システム
321
311
313
ソース・コード
301
バイトコード
・コンパイラ
303
バイトコード
305
コンパイルタイム環境
309

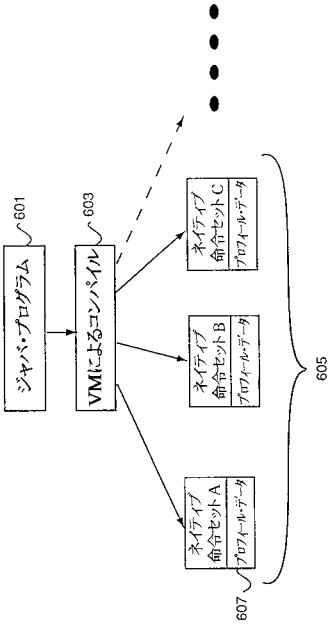
【図５】


401
ジャバ・プログラム
403
VMによるコンパイル
411
ネイティブ命令セットC
409
ネイティブ命令セットB
407
ネイティブ命令セットA
405

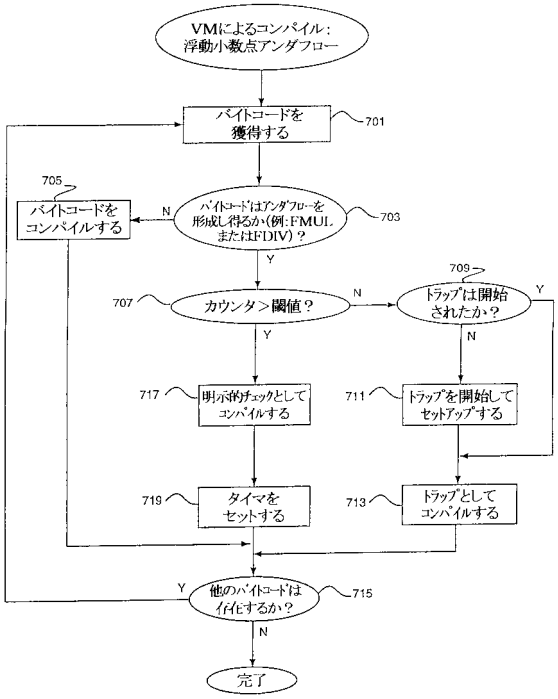
【図６】


VMによるコンパイル
501
バイトコードを獲得する
505
バイトコードをコンパイルする
N
バイトコードを複数の方法でコンパイルできるか？
503
Y
バイトコードをコンパイルする方法を決定する
507
511
次の方法でコンパイルする
509
それは第1の方法であるか？
N
Y
513
第1の方法でコンパイルする
515
他のバイトコードは存在するか？
Y
N
完了

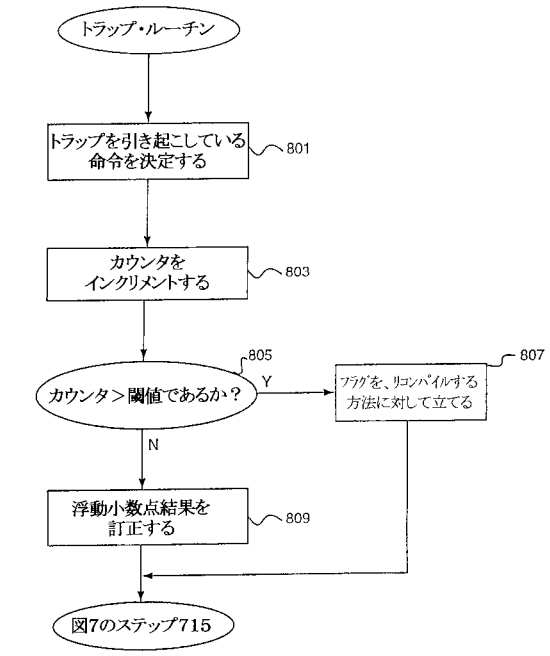
【図７】


601
ジャバ・プログラム
603
VMによるコンパイル
ネイティブ命令セットC
プロファイル・データ
ネイティブ命令セットB
プロファイル・データ
ネイティブ命令セットA
プロファイル・データ
605
607

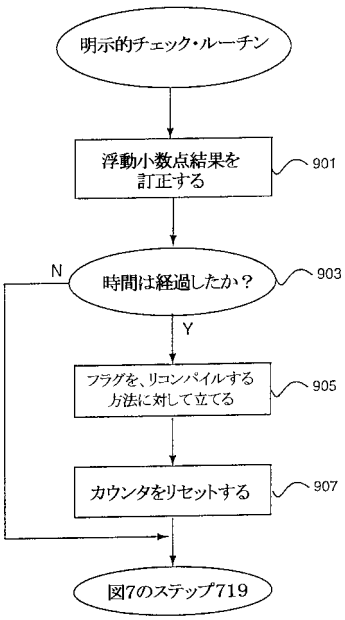
【図８】


VMによるコンパイル：浮動小数点アンダフロー
バイトコードを獲得する
701
705
バイトコードをコンパイルする
N
バイトコードはアンダフローを形成し得るか（例：FMULまたはFDIV）？
703
Y
709
トラップは開始されたか？
N
Y
707
カウンタ＞閾値？
N
Y
717
明示的チェックとしてコンパイルする
711
トラップを開始してセットアップする
719
タイマをセットする
713
トラップとしてコンパイルする
Y
他のバイトコードは存在するか？
715
N
完了

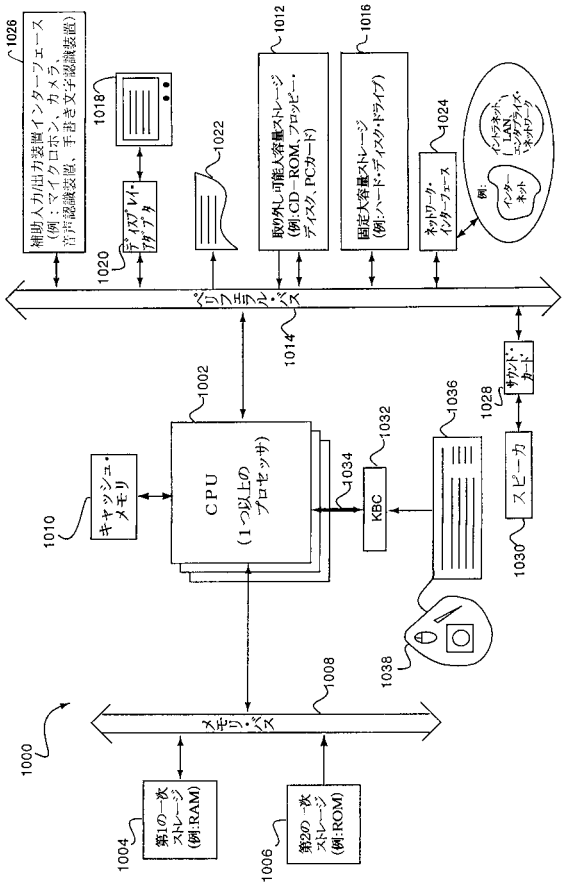
【図９】


トラップ・ルーチン
トラップを引き起こしている命令を決定する
801
カウンタをインクリメントする
803
805
カウンタ＞閾値であるか？
Y
N
807
フラグを、リコンパイルする方法に対して立てる
浮動小数点結果を訂正する
809
図7のステップ715

【図１０】


明示的チェック・ルーチン
浮動小数点結果を訂正する
901
N
時間は経過したか？
903
Y
フラグを、リコンパイルする方法に対して立てる
905
カウンタをリセットする
907
図7のステップ719

【図１１】


1026
補助入力出力装置インターフェース（例：マイクロホン、カメラ、音声認識装置、手書き文字認識装置）
1018
1022
1012
取り外し可能大容量ストレージ（例:CD−ROM、フロッピー・ディスク、PCカード）
1016
固定大容量ストレージ（例:ハード・ディスク・ドライブ）
1024
ネットワーク・インターフェース
インターネット、LAN、エンタープライズ・ネットワーク
例
ディスプレイ・アダプタ
1020
システム・バス
1014
1002
CPU（1つ以上のプロセッサ）
1010
キャッシュ・メモリ
1032
KBC
1034
1036
1028
サウンド・カード
スピーカ
1030
1038
1008
メモリ・バス
1000
第1の一次ストレージ（例:RAM）
1004
第2の一次ストレージ（例:ROM）
1006

フロントページの続き

(56)参考文献 特開平１０－２０７６９３（ＪＰ，Ａ）
特開平０４－２７３５３３（ＪＰ，Ａ）
特開平０１－１１８９３１（ＪＰ，Ａ）
特開平１０－０４０１１２（ＪＰ，Ａ）
特開平０９－２１２３３７（ＪＰ，Ａ）
小野寺民也，オブジェクト指向言語におけるメッセージ送信の高速化技法，情報処理，日本，社団法人情報処理学会，１９９７年 4月１５日，第３８巻，第４号，ｐ．３０２，３０７
中村輝雄，企業ユーザーのためのＪａｖａ入門 4回：Ｊａｖａはどこまで速くなるか？，日経コンピュータ，日本，日経ＢＰ社，１９９８年 5月２５日，ｎｏ．４４４，ｐ．１７９

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 9/45