(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2021-57006
(P2021-57006A)
(43)公開日　令和3年4月8日(2021.4.8)

(51)Int.Cl.			F I			テーマコード（参考）
G06F	9/38	(2006.01)	G06F	9/38	380X	5B013
G06F	9/34	(2006.01)	G06F	9/34	380	5B033
G06F	9/32	(2006.01)	G06F	9/32	380	

審査請求 未請求　請求項の数 25　ＯＬ　外国語出願　（全 57 頁）

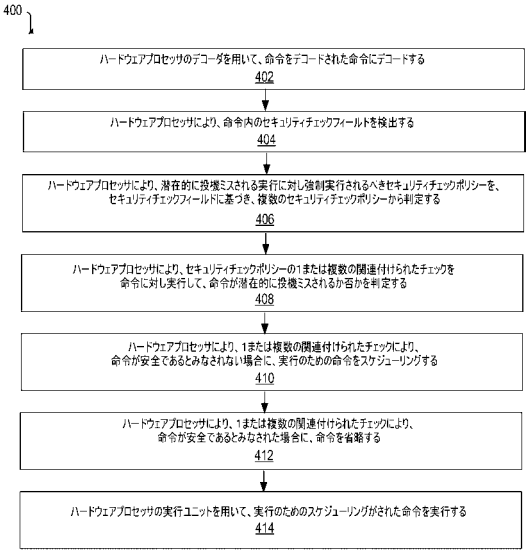
(21)出願番号　特願2020-104078 (P2020-104078)
(22)出願日　令和2年6月16日 (2020.6.16)
(31)優先権主張番号　16/585,964
(32)優先日　令和1年9月27日 (2019.9.27)
(33)優先権主張国・地域又は機関
米国 (US)

(71)出願人　591003943
インテル・コーポレーション
アメリカ合衆国　９５０５４　カリフォルニア州・サンタクララ・ミッション　カレッジ　ブーレバード・２２００
(74)代理人　110000877
龍華国際特許業務法人
(72)発明者　マイケル　ルメイ
アメリカ合衆国　９５０５４　カリフォルニア州・サンタクララ・ミッション　カレッジ　ブーレバード・２２００　インテル・コーポレーション内

Ｆターム(参考)　5B013 EE04
5B033 FA02　FA09

(54)【発明の名称】投機実行中に安全とみなされた場合に、セキュリティチェックを省略するためのハードウェア

(57)【要約】　（修正有）
【課題】投機実行におけるセキュリティチェックの省略のためのハードウェアを提供する。
【解決手段】コンピュータシステムによる方法であって、プロセッサコアの投機マネージャ及び実行ユニットを用いて、潜在的に投機ミスされる実行に対し強制実行されるべきセキュリティチェックポリシーを、セキュリティチェックフィールドに基づき、複数のセキュリティチェックポリシーから判定することと、セキュリティチェックポリシーの１または複数の関連付けられたチェックを命令に対し実行して、命令が潜在的に投機ミスされるか否かを判定することと、１または複数の関連付けられたチェックにより、命令が安全ではないとみなされた場合に、命令の実行をスケジューリングすることと、１または複数の関連付けられたチェックにより、命令が安全であるとみなされた場合に、命令を省略することと、実行のためにスケジューリングされた命令を実行するためことと、を含む。
【選択図】図４



【特許請求の範囲】
【請求項１】
命令をデコードされた命令にデコードするためのデコーダと、
投機マネージャ回路であって、
前記命令内のセキュリティチェックフィールドを検出する、
潜在的に投機ミスされる実行に対し強制実行されるべきセキュリティチェックポリシーを、前記セキュリティチェックフィールドに基づき、複数のセキュリティチェックポリシーから判定する、
前記セキュリティチェックポリシーの１または複数の関連付けられたチェックを前記命令に対し実行して、前記命令が潜在的に投機ミスされるか否かを判定する、
前記１または複数の関連付けられたチェックにより、前記命令が安全ではないとみなされた場合に、前記命令の実行をスケジューリングする、
前記１または複数の関連付けられたチェックにより、前記命令が安全であるとみなされた場合に、前記命令を省略する、投機マネージャ回路と、
実行のためのスケジューリングがされた前記命令を実行するための実行ユニットと、を備える、装置。
【請求項２】
前記セキュリティチェックフィールドは、コンパイラにより提供されるヒントである、請求項１に記載の装置。
【請求項３】
前記投機マネージャ回路は、前記セキュリティチェックポリシーの前記１または複数の関連付けられたチェックを、前記命令の関連付けられたメモリアクセスのセットに対し実行する、請求項１または２に記載の装置。
【請求項４】
前記セキュリティチェックポリシーは、メモリ安全性チェックポリシーである、請求項１から３のいずれか一項に記載の装置。
【請求項５】
前記セキュリティチェックポリシーは、型安全性チェックポリシーである、請求項１から３のいずれか一項に記載の装置。
【請求項６】
前記セキュリティチェックポリシーの前記１または複数の関連付けられたチェックは、メモリ安全性チェックおよび型安全性チェックを含む、請求項１に記載の装置。
【請求項７】
前記１または複数の関連付けられたチェックは、前記装置のアーキテクチャ仕様の完全準拠チェックより少ない、請求項１に記載の装置。
【請求項８】
前記命令は、プログラム順序において後続するメモリアクセス命令に関連付けられたセキュリティチェック命令である、請求項１から７のいずれか一項に記載の装置。
【請求項９】
ハードウェアプロセッサのデコーダを用いて、命令をデコードされた命令にデコードする段階と、
前記ハードウェアプロセッサにより、前記命令内のセキュリティチェックフィールドを検出する段階と、
前記ハードウェアプロセッサにより、潜在的に投機ミスされる実行に対し強制実行されるべきセキュリティチェックポリシーを、前記セキュリティチェックフィールドに基づき、複数のセキュリティチェックポリシーから判定する段階と、
前記ハードウェアプロセッサにより、前記セキュリティチェックポリシーの１または複数の関連付けられたチェックを前記命令に対し実行して、前記命令が潜在的に投機ミスされるか否かを判定する段階と、
前記ハードウェアプロセッサにより、前記１または複数の関連付けられたチェックによ

り前記命令が安全ではないとみなされた場合に、前記命令の実行をスケジューリングする段階と、
前記ハードウェアプロセッサにより、前記１または複数の関連付けられたチェックにより、前記命令が安全であるとみなされた場合に、前記命令を省略する段階と、
前記ハードウェアプロセッサの実行ユニットを用いて、実行のためのスケジューリングがされた前記命令を実行する段階と、を備える、方法。
【請求項１０】
前記セキュリティチェックフィールドは、コンパイラにより提供されるヒントである、請求項９に記載の方法。
【請求項１１】
前記実行する段階は、前記セキュリティチェックポリシーの前記１または複数の関連付けられたチェックを、前記命令の関連付けられたメモリアクセスのセットに対し実行する段階を含む、請求項９または１０に記載の方法。
【請求項１２】
前記セキュリティチェックポリシーは、メモリ安全性チェックポリシーである、請求項９から１１のいずれか一項に記載の方法。
【請求項１３】
前記セキュリティチェックポリシーは、型安全性チェックポリシーである、請求項９から１１のいずれか一項に記載の方法。
【請求項１４】
前記セキュリティチェックポリシーの前記１または複数の関連付けられたチェックは、メモリ安全性チェックおよび型安全性チェックを含む、請求項９に記載の方法。
【請求項１５】
前記１または複数の関連付けられたチェックは、前記ハードウェアプロセッサのアーキテクチャ仕様の完全準拠チェックより少ない、請求項９に記載の方法。
【請求項１６】
前記命令は、プログラム順序において後続するメモリアクセス命令に関連付けられたセキュリティチェック命令である、請求項９から１５のいずれか一項に記載の方法。
【請求項１７】
ハードウェアプロセッサに、
前記ハードウェアプロセッサのデコーダを用いて、命令をデコードされた命令にデコードする手順と、
前記命令内のセキュリティチェックフィールドを検出する手順と、
潜在的に投機ミスされる実行に対し強制実行されるべきセキュリティチェックポリシーを、前記セキュリティチェックフィールドに基づき、複数のセキュリティチェックポリシーから判定する手順と、
前記セキュリティチェックポリシーの１または複数の関連付けられたチェックを前記命令に対し実行して、前記命令が潜在的に投機ミスされるか否かを判定する手順と、
前記１または複数の関連付けられたチェックにより前記命令が安全ではないとみなされた場合に、前記命令の実行をスケジューリングする手順と、
前記１または複数の関連付けられたチェックにより前記命令が安全であるとみなされた場合に、前記命令を省略する手順と、
前記ハードウェアプロセッサの実行ユニットを用いて、実行のためのスケジューリングがされた前記命令を実行する手順と、を実行させるための、プログラム。
【請求項１８】
前記セキュリティチェックフィールドは、コンパイラにより提供されるヒントである、請求項１７に記載のプログラム。
【請求項１９】
前記実行する手順は、前記セキュリティチェックポリシーの前記１または複数の関連付けられたチェックを、前記命令の関連付けられたメモリアクセスのセットに対し実行する

手順を含む、請求項１７または１８に記載のプログラム。
【請求項２０】
前記セキュリティチェックポリシーは、メモリ安全性チェックポリシーである、請求項１７から１９のいずれか一項に記載のプログラム。
【請求項２１】
前記セキュリティチェックポリシーは、型安全性チェックポリシーである、請求項１７から１９のいずれか一項に記載のプログラム。
【請求項２２】
前記セキュリティチェックポリシーの前記１または複数の関連付けられたチェックは、メモリ安全性チェックおよび型安全性チェックを含む、請求項１７に記載のプログラム。
【請求項２３】
前記１または複数の関連付けられたチェックは、前記ハードウェアプロセッサのアーキテクチャ仕様の完全準拠チェックより少ない、請求項１７に記載のプログラム。
【請求項２４】
前記命令は、プログラム順序において後続するメモリアクセス命令に関連付けられたセキュリティチェック命令である、請求項１７から２３のいずれか一項に記載のプログラム。
【請求項２５】
請求項１７から２４のいずれか一項に記載のプログラムを格納した非一時的機械可読媒体。
【発明の詳細な説明】
【技術分野】
【０００１】
［背景技術］
プロセッサ、またはプロセッサのセットは、例えば、命令セットアーキテクチャ（ＩＳＡ）といった命令セットから命令を実行する。命令セットは、プログラミングに関連するコンピュータアーキテクチャの一部であり、概して、ネイティブのデータ型、命令、レジスタアーキテクチャ、アドレス指定モード、メモリアーキテクチャ、割り込みおよび例外処理、並びに外部入力および出力（Ｉ／Ｏ）を含む。本明細書において、命令という用語は、例えば、プロセッサに実行のために提供される命令であるマクロ命令、または、例えば、プロセッサのデコーダがマクロ命令をデコードした結果から得られる命令であるマイクロ命令を指してよいことに留意されたい。
【図面の簡単な説明】
【０００２】
本開示による様々な実施形態について、図面を参照して説明する。
【０００３】
【図１】本開示の実施形態による、プロセッサコアを含むコンピュータシステムを示す。
【０００４】
【図２】本開示の実施形態によるセキュリティチェック命令を省略するためのフロー図を示す。
【０００５】
【図３】本開示の実施形態により、ヒントをコード内に発出するためのフロー図を示す。
【０００６】
【図４】本開示の実施形態によるフロー図を示す。
【０００７】
【図５Ａ】本開示の実施形態による、汎用ベクトル向け命令フォーマットおよびそのクラスＡ命令テンプレートを示すブロック図である。
【０００８】
【図５Ｂ】本開示の実施形態による、汎用ベクトル向け命令フォーマットおよびそのクラスＢ命令テンプレートを示すブロック図である。

【０００９】
【図６Ａ】本開示の実施形態による、図５Ａおよび図５Ｂ中の汎用ベクトル向け命令フォーマットのためのフィールドを示すブロック図である。
【００１０】
【図６Ｂ】本開示の一実施形態による、図６Ａ中の特定ベクトル向け命令フォーマットのフルオペコードフィールドを構成するフィールドを示すブロック図である。
【００１１】
【図６Ｃ】本開示の一実施形態による、図６Ａ中の特定ベクトル向け命令フォーマットのレジスタインデックスフィールドを構成するフィールドを示すブロック図である。
【００１２】
【図６Ｄ】本開示の一実施形態による、図６Ａ中の特定ベクトル向け命令フォーマットの拡張オペレーションフィールド５５０を構成するフィールドを示すブロック図である。
【００１３】
【図７】本開示の一実施形態による、レジスタアーキテクチャのブロック図である。
【００１４】
【図８Ａ】本開示の実施形態による、例示的なインオーダパイプラインおよび例示的なレジスタリネーミング、アウトオブオーダ発行／実行パイプラインの両方を示すブロック図である。
【００１５】
【図８Ｂ】本開示の実施形態による、プロセッサに含まれるべき例示的実施形態のインオーダアーキテクチャコアおよび例示的なレジスタリネーミング、アウトオブオーダ発行／実行アーキテクチャコアの両方を示すブロック図である。
【００１６】
【図９Ａ】本開示の実施形態による、単一のプロセッサコアを、オンダイ相互接続ネットワークへ接続された状態およびそのレベル２（Ｌ２）キャッシュのローカルサブセットを有する状態と共に示すブロック図を示す。
【００１７】
【図９Ｂ】本開示の実施形態による、図９Ａ中のプロセッサコアの一部の拡大図である。
【００１８】
【図１０】本開示の実施形態による、２個以上のコアを有してよい、統合メモリコントローラを有してよい、統合グラフィックを有してよい、プロセッサのブロック図である。
【００１９】
【図１１】本開示の一実施形態による、システムのブロック図である。
【００２０】
【図１２】本開示の実施形態による、より具体的な例示的システムのブロック図である。
【００２１】
【図１３】本開示の実施形態による、第２のより具体的な例示的システムのブロック図である。
【００２２】
【図１４】本開示の実施形態による、システムオンチップ（ＳｏＣ）のブロック図が示される。
【００２３】
【図１５】本開示の実施形態による、ソース命令セット内のバイナリ命令を、ターゲット命令セット内のバイナリ命令に変換するソフトウェア命令コンバータの使用を対比するブロック図である。
【発明を実施するための形態】
【００２４】
以下の説明では、多くの具体的な詳細について記載される。しかしながら、本開示の実施形態は、これらの具体的な詳細を省いても実施可能であることを理解されたい。他の例においては、本説明の理解を不明瞭にしないために、周知の回路、構造および技術は詳細

に示されていない。

【００２５】

本明細書中の「一実施形態（ｏｎｅ　ｅｍｂｏｄｉｍｅｎｔ）」、「実施形態（ａｎ　ｅｍｂｏｄｉｍｅｎｔ）」、「例示的な実施形態（ａｎ　ｅｘａｍｐｌｅ　ｅｍｂｏｄｉｍｅｎｔ）」等への言及は、記載される実施形態は、特定の機能、構造または特性を含んでよいが、すべての実施形態が必ず特定の機能、構造または特性を含んでいなくてもよいことを示す。また、このような文言は、必ずしも同一の実施形態を指していない。さらに、ある実施形態に関連して特定の機能、構造または特性が説明されている場合は、明示の記載の有無に関わらず、他の実施形態に関し、このような機能、構造または特定に影響が及ぶことは、当業者の知識の範囲内であると考えられる。

【００２６】

（例えば、ハードウェア）プロセッサ（例えば、１または複数のコアを有する）は、命令（例えば、命令のスレッド）を実行して、データに対し処理、例えば、算術的、論理的または他の機能を実行してよい。例えば、ソフトウェアはオペレーションを要求してよく、ハードウェアプロセッサ（例えば、コアまたはハードウェアプロセッサのコア）は、当該要求に応答してオペレーションを実行してよい。ソフトウェアは、プログラム順序とは異なる命令シーケンスの実行を生じさせる１または複数の分岐（例えば、分岐命令）を含んでよい。分岐命令は、常に分岐をもたらす無条件分岐であってよく、または、特定の条件により分岐をもたらしてもよい、またはもたらさなくてよい条件分岐であってよい。特定のプロセッサは、より多くの命令がより高速に完了することを可能にするためにパイプライン化される。これは、概して、前の命令が完了するのを待機せず、命令（例えば、投機的）の実行を開始することを意味する。しかしながら、条件分岐により、このアプローチに関する問題が生じる。特に、プロセッサが条件分岐に遭遇した場合に、当該条件の結果がまだ計算されていないときは、プロセッサは分岐を取るべきか否かがわからない。分岐予測とは、特定のプロセッサが、条件分岐を取るか否かを決定するために用いるものである。不正確な予測（例えば、予測ミス）は、特定のプロセッサに対し、実行する必要のなかったすべての命令を破棄させ、正しい命令セットでやり直すことを生じさせ、例えば、このプロセスが深くパイプラインされたプロセッサを用いている場合に特にコスト高となるので、この情報を可能な限り正確に取得することが重要である。一実施形態において、プロセッサの分岐予測器は積極的に投機を行い、例えば、アウトオブオーダの深さおよび幅を増大させ、大きな性能を取得する。

【００２７】

特定の実施形態において、投機実行（例えば、分岐予測を含む）をサポートするハードウェアプロセッサにより実行されるべきコードは、１または複数のセキュリティチェック（例えば、セキュリティチェック命令）を含む。特定のセキュリティチェック（例えば、メモリ安全性チェックまたは型安全性チェックを強制実行する）は、静的解析（例えば、コンパイラによる）を用いて省略されてよいが、その解析は、投機実行を考慮に入れるように拡張されるべきである。所望のセキュリティポリシーを、投機ミスの実行パスを含むすべての可能性のある実行パスに対し強制実行するには、追加のセキュリティチェックとなり得る。しかしながら、特定の実施形態において、投機ミスはまれにしか生じないものであり、そのため、これらの実施形態において、これらのセキュリティチェックは、通常は不要である追加のオーバーヘッドを課すこととなる。

【００２８】

特定の実施形態において、メモリ安全性チェックは、メモリアクセスエラー、例えば、バッファオーバーフローおよびダングリングポインタ等をチェックする。一実施形態において、メモリ安全性チェックは、配列の境界およびポインタの参照外しをチェックする。

【００２９】

特定の実施形態において、型安全性チェックは、プログラムの定数、変数およびメソッド（例えば、関数）のための異なるデータ型間の矛盾をチェックする（例えば、メモリアクセス要求のために）ものであり、例えば、整数（ｉｎｔ）であるデータを、浮動小数点

の数値（ｆｌｏａｔ）として扱うエラーがある。

【００３０】

本明細書における特定の実施形態は、プロセッサのアーキテクチャを拡張して、チェックされているオペレーションが潜在的に投機ミスされ得る場合にのみ、特定のセキュリティチェックが必要となるというヒント（例えば、コンパイラにより発出される）を読み取らせる。プロセッサが、オペレーションが安全とみなされないと判定する場合、チェックが実行されてよく、且つ投機が継続されてよく、これにより、最適化の機会を向上させる。一方で、これらの実施形態において、プロセッサが、オペレーションの実行準備が整うまでに、かかるオペレーションが安全であると判定した場合は、関連付けられたセキュリティチェックをスキップして、オーバーヘッドを減少させてよい。本明細書における特定の実施形態は、命令が投機ミスされ得る場合にのみ、セキュリティチェックが実行されるべきであることを指定する方法を提供する。将来のコンパイラは、コンパイラにより発出されるセキュリティチェック命令の数を増やす可能性があり、その結果、チェックが潜在的な投機制御フローに対し、安全性（例えば、型）を強制実行する。このような場合に、本明細書の特定の実施形態は、正しく予測されている／アーキテクチャ上正しいとわかっている制御フローに対し、スキップ可能なセキュリティチェックはどれかに関する情報をマイクロアーキテクチャに伝達（例えば、コンパイラから）することで、有利な効率性を提供する。

【００３１】

フェンスを用いて、先行するセキュリティチェックが完了するまで、当該チェックにより制限されているオペレーションの投機実行を制限できる。投機実行におけるセキュリティチェックの省略のための本明細書の実施形態とは異なり、フェンスは、セキュリティチェックに依存していないオペレーションの投機実行も制限する可能性があり、故に、プロセッサの性能を最適化する能力を制限する。

【００３２】

特定の実施形態において、ハードウェアプロセッサにおけるハードウェア投機攻撃に対し防御するための戦略は、データキャッシュ階層内で投機を不可視にすることであり、例えば、情報をリークさせ得るような態様でマイクロアーキテクチャの状態に影響を及ぼすことなく、且つ、投機実行の継続が許可される時点よりも前の状態変更をバッファすることなく、ロードを安全に進められる時点をより正確に判定することである。一実施形態において、当該戦略は、例えば、安全でない投機ロードがデータを投機バッファに読み込んでしまうような投機ロードを理由に、マルチプロセッサデータキャッシュ階層を修正することなく、マルチプロセッサのデータキャッシュ階層を通るマイクロアーキテクチャのカバートチャネル（隠れ通信路）およびサイドチャネルをブロックする。ロードが安全になった時点で、この戦略はそれらをシステムの残部に対し可視にする。この戦略は、メモリ整合性を害した可能性のあるロードを識別し、この時点で、これらのロードに対し、検証ステップの実行を強制する。投機実行におけるセキュリティチェックの省略のための本明細書の実施形態と異なり、上記の戦略は、もし開発者が、当該戦略が強制するキャッシュアクセス可視性のアーキテクチャ仕様の、より厳格な完全準拠チェックよりも、緩いセキュリティポリシー（メモリ安全性または型安全性等）を強制実行することにのみ関心がある場合は、ロードを安全に実行可能な時点を判定するにおいて、過度に悲観的である。この戦略は、またバッファ領域に対する要求も増大させる。バッファロジックはまた、特に、最終的にロードの実行が許容される際に、当該戦略はバッファされたロードを再発行するので、追加のオーバーヘッドと複雑性をもたらす。投機実行におけるセキュリティチェックの省略のための本明細書の実施形態は、開発者が所望する特定のセキュリティポリシーを強制実行するセキュリティチェックの使用（例えば、コンパイラにより挿入され）を可能にし、これらのチェック自体を投機的に実行し、これらのチェックが行われなかった場合よりも早期に、オペレーション（例えば、ロード）が安全であると宣言できるようにする。投機実行におけるセキュリティチェックの省略のための本明細書の実施形態は、バッファと共に用いられてよいことを留意されたい。

【００３３】
投機実行におけるセキュリティチェックの省略のための本明細書の実施形態は、開発者により所望されるセキュリティポリシーを強制実行するセキュリティ手段を含む新しいタイプの命令（および以前に定義された命令への修飾子）を定義し、同時に、顕著な動的最適化を可能にする。例えば、当該命令は、チェックされているアドレスからのロードが安全ではないとみなされる場合にのみ、境界チェック命令（例えば、ポインタ命令またはメモリ保護拡張（ＭＰＸ）境界チェック命令（例えば、下限または上限のチェック）のための機能）または暗黙の境界チェックおよび型チェックのセットを、必要とマークすることを可能にする。プロセッサがその命令に到達したとき、安全性基準が既に満たされている（例えば、先行する分岐の正しい方向が決定済み）場合は、プロセッサは、その命令をスキップしてよい。特定の実施形態において、この文脈における機能は、ポインタストレージ内に埋め込まれた関連付けられたセキュリティメタデータを持つポインタを指す。
【００３４】
データリークをもたらし得る共通のセキュリティ問題を回避すべく、タイプセーフ言語が、開発者により彼らのプログラム内に用いられてよい。特定の実施形態において、メモリ安全性は、同程度に厳格ではない（例えば、型の取り違えを防止しない）が、実行にあたりオーバーヘッドをより少なく課す可能性がある。投機実行におけるセキュリティチェックの省略のための本明細書の実施形態は、プロセッサに対し、投機ミスがされ得る制御フローに対してのみ必要なセキュリティチェックは何かを通知するヒント（例えば、コンパイラにより提供される）を用いることで、実行のアーキテクチャのパスおよび投機ミスのパスの両方に沿って、セキュリティポリシーの真部分集合（例えば、メモリ安全性および／または型安全性）をより効率的に強制実行する方法を提供する。特定の実施形態において、潜在的に投機ミスされる命令についてのみ必要なチェックを示すヒントは、コード内でプログラムに対し可視である。
【００３５】
投機実行におけるセキュリティチェックの省略のための本明細書の実施形態の基礎をなす基本的概念は、一部のプログラムは、プログラムを実行するプロセッサのアーキテクチャ仕様に完全準拠するよりも、ポリシーの真部分集合（例えば、全部より少ない）（例えば、メモリ安全性および／または型安全性ポリシー）を強制実行するだけでよいという点にあり、それにより、より厳格なポリシーが強制実行される場合に特定のセットのプロセッサリソースにとって可能となったであろうものよりも、より大きな投機実行の機会を提供する。
【００３６】
故に、投機実行におけるセキュリティチェックの省略のための本明細書の実施形態の一態様は、完全なアーキテクチャの準拠を要求する代わりに、境界チェックおよび型チェック等の特定の狭いタイプのセキュリティチェックを用いて、投機実行をゲートする。これは、関連するチェック（例えば、ポインタチェック、ＭＰＸチェックまたは他の定義されたチェックの機能）が完了する前に、セキュリティセンシティブなオペレーション（例えば、メモリロード）により誘発される、敵対者に可視となるマイクロアーキテクチャの状態変更が生じるのを防止することで実現されてよい。例えば、これは、チェックが完了するまで、さらなる投機実行をブロックする、または、もたらされる状態変更をバッファし、チェックが失敗した場合にそれらをロールバックするという形態を取ってよい。
【００３７】
第２に、これは、投機ミスの可能性のある制御フローをカバーするために、さらに多くのセキュリティチェックが必要となり得ることを示唆する。これに対し、以前は、コンパイラがアーキテクチャ上あり得ないフローに対しチェックを静的に省略したであろう。これは、特定の実施形態のコンパイラが、すべての可能な投機制御フローを識別可能なように、投機実行のモデルを有することを必要とする。
【００３８】
第３に、効率性の理由で、特定の実施形態のコンパイラは、プロセッサに対し、チェッ

クされている値がそれらのアーキテクチャの定義に準拠している場合に、プロセッサがそれらのチェックをスキップできるように、どのセキュリティチェックが追加（さもなければ省略されたであろう）されているかについて通知できる必要がある。それらのチェックをスキップ（例えば、それらのチェックの実行ユニットへのディスパッチをスキップ）することにより、プロセッサのエネルギー、帯域幅および他のリソースを節約し、故に、プロセッサ（例えば、プロセッサを含むコンピュータ）への改善となる。

【００３９】

以下のコードは、この第３の態様に関し作成されたものである。投機実行におけるセキュリティチェックの省略のための本明細書の実施形態は、チェックにより制限されているオペレーションが何らかの点で安全であるとまだみなされていない場合に、どのセキュリティチェックのみが必要であるかに関するヒント（例えば、コンパイラによって提供される）をプロセッサが使用することを可能にする。例えば、以下のサンプル擬似コードについて考えてみよう。

【表１】

```
// 複数のソケットタイプをサポートするソケットライブラリを持つネットワーキングアプリケーション。
TCP／IPソケットはローカルドメインソケットの構造よりも大きい構造で表される。
これは小さいソケット構造があたかもより大きな構造のようにアクセスされる型の取り違えを
生じさせ得る。もし当該小さいソケット構造の直後にSSLキーが格納されている場合は、
これによりキー情報のリークをもたらし得る。
class Socket {

};


// ローカルドメインソケットを表わすより小さい構造
__attribute__((packed))

class LocalDomainSocket : Socket {

  uint8_t *path;

};

// 可能なソケット状態を列挙

enum SocketState {

  invalid, closed, waiting, connected

};

// IPソケットを表わすより大きな構造

class TCPIPSocket : Socket {

  uint8_t ip[4];

  uint16_t port;

  // 各フィールドは32ビット境界に整列されているので、sttはLocalDomainSocket
  構造体の終了位置より突き出すと仮定

  SocketState stt;

};


Socket *lookupSocket(int id) {

  // 簡潔にするため、ソケット初期化を省略

  static Socket *sockets[] = { new LocalDomainSocket(), new
TCPIPSocket() };

return sockets[id];

}

struct Service {

  // この例において、Listnerの定義は重要ではない

  Listener *listener;

  char name[56]; // 説明目的のために、これがServiceのサイズをキャッシュラインのサイズと
  合致させるために選択される

};

static Service tcpipServices[65536];

static Service domainServices[128];

// 以下のルーチンは、指定されたアドレスにサービスがリスナーとして登録されるか否かを
チェックする

Listener *lookupListener(Socket *socket) {

  if (TCPIPSocket *ts = dynamic_cast<TCPIPSocket>(socket)) {

return tcpipServices[socket->port];

  } else {

    LocalDomainSocket *ds = static_cast<LocalDomainSocket>(socket);

    // ローカルドメインソケットのルックアップは関係がないため詳細は省略する

  }

}

int networkProcessingLoop() {

  Socket *socket = NULL;

  // . . . ソケット変数を初期化して、ソケットの確立をもたらすコード. . .


  Listener *listener = lookupListener(socket);

  // . . . 着信するネットワーク要求を満たすためのコード. . .

  return 0;

  }
```

【００４０】

特定の実施形態において、以下のアクセスのための境界チェックは無条件で実行される必要がある。というのは、"ｉｄ"の任意の値がこの機能に提供され得るからである（この例について、インライン化の可能性を除く）。コンパイラは、ソケットのアドレスはプロセッサにより投機されないことを前提に、このアクセスの型安全性を静的に検証可能であるので、型チェックは必要でなくてよい。そうではなく、そのレベルの投機が可能な場合は、動的型安全性チェックにより、投機の実行をさらに進めつつ、同時に型安全性が強制実行されることを保証することを可能にしてよい。

【００４１】

しかしながら、特定の実施形態においては、たとえアドレス投機が可能であったとしても、コンパイラは、ソケットのアドレスがそのアーキテクチャの定義に合致する場合、型安全性チェックは不要であると静的に決定してよいことに留意されたい。例えば、該当する場合、ソケットのアドレスのロード元のメモリ位置をアリアスする可能性のある先行分岐または先行ストア等について、プロセッサ（例えば、中央処理装置（ＣＰＵ））が既に任意の条件を解決済みである、または、ソケットのアドレスのプロセッサのアーキテクチャ定義からの逸脱（例えば、投機ミス）を潜在的に生じさせた可能性のある命令をリタイア済みである場合に、この後者の条件は満たされる。

【００４２】

投機実行におけるセキュリティチェックの省略のための本明細書の実施形態は、コンパイラが、プロセッサに、アクセスの実行準備が整った時点で、以下のアクセスに入力されたアドレスがアーキテクチャ上のものであるとプロセッサ（例えば、ＣＰＵ）により既に判定された場合は、型安全性チェックは省略可能であることを示すヒントを渡すことを可能にする。

【００４３】

一実施形態において、上記の擬似コード内のｄｙｎａｍｉｃ＿ｃａｓｔルーチンは、パラメータがキャストされるポインタのタイプを指定するテンプレートパラメータを受け取る。ｄｙｎａｍｉｃ＿ｃａｓｔルーチンは、入力パラメータのタイプが、テンプレートパラメータのサブクラスであるか否か、または、それらが正確に合致するか否かを判定するための動的チェックを実行する。是である場合、ルーチンの結果は、入力ポインタと同一アドレスを持つ新しいタイプのオブジェクトへのポインタである。それ以外の場合、ｄｙｎａｍｉｃ＿ｃａｓｔはヌルを返す。

【００４４】

１つの例として、この"ｉｆ"ステートメントは、コンパイラに、条件制御フローを実行するための分岐命令を生成させる。当該ｉｆステートメントのこの分岐内の命令が実行されるまでに、前の分岐が解決済みである場合、型チェックは通過される。これは、ｔｓ－＞ｐｏｒｔへの後続のアクセスも、ｔｓの境界の範囲内にあることを示唆する。故に、このｔｓ－＞ｐｏｒｔアクセスに、アクセスされているアドレスがそのアーキテクチャの定義に合致することが既に判明しており、すべての先行する分岐が既に解決済みである場合は、型チェックおよび境界チェック（例えば、境界チェック命令）は省略されてよいことを示すヒントが供給されてよい。これは、ｌｏｏｋｕｐＳｏｃｋｅｔ内の最後のステートメントから生成されるアクセスに指定されたものよりも、さらに強い制限であることに留意されたい。各タイプの制限について、複数のタイプのヒント（例えば、異なる命令プレフィックスまたは異なる命令バリアント）が定義されてよい。

【００４５】

上記の擬似コードの別の例として、分岐が投機ミスされ、その結果、あたかもそれがＴＣＰＩＰＳｏｃｋｅｔであるかのようにＬｏｃａｌＤｏｍａｉｎＳｏｃｋｅｔがアクセスされる場合、ＬｏｃａｌＤｏｍａｉｎＳｏｃｋｅｔオブジェクトの末尾のすぐの箇所に格納された１６ビットのデータが用いられて、ｔｃｐｉｐＳｅｒｖｉｃｅｓアレイへインデックスされる。これが、何らかの機密データ、例えば、サーバの秘密キーの一部が格納された場所について生じる場合、その情報は、アレイ全体が最初にキャッシュされないことを前提に、ｔｃｐｉｐＳｅｒｖｉｃｅｓアレイ内のどのエントリがキャッシュに入れられ

るかを判定できる可能性がある。敵対者が、例えば、各サービスへの接続を試み、接続が確立されるまでの所要時間を計時する等により、ｔｃｐｉｐＳｅｒｖｉｃｅｓアレイへの後続のアクセスを計時できる場合、それは機密データの値をリークする可能性がある。最も早く接続するサービスは、機密データと等しいポート番号を持つものであってよい。暗号キーは、例示の１６ビットより大きい可能性があるが、何であれ秘密キーをリークすることは潜在的に問題である。また、キーのリーク以上のものをもたらし得る、この例で説明したものよりも複雑なシナリオも存在し得る。

【００４６】

アクセスの実行準備が整ったが、アクセスは、上で定義された基準により、安全であるとまだみなされていない場合、境界チェックおよび型チェックが実行されてよい。分岐が投機ミスされると、境界チェックおよび型チェックの両方がこの場合において失敗する。しかしながら、他の場合においては、１種類のチェックのみが失敗し得る。

【００４７】

いくつかの場合においては、オブジェクトの開始へのポインタと、オブジェクト内の他のオフセットへのポインタとの間を区別することが有用であってよい。その結果、ポインタがオブジェクトのベースを指しており、静的チェックで十分であろう場合であっても、後者のタイプのポインタのサブタイプを用いて、アクセスへの動的境界チェックが実行されてよい。ポインタがオブジェクトの開始をポイントするか否かの表記をポインタ内に指定することで効率性を高めてよい。例えば、ポインタ（またはｆａｔポインタ）がオブジェクトベースに対し初期化される場合、予約ビットがポインタ内に設定されてよく、後続のポインタ更新が、得られるポインタがそのオブジェクトベースをポイントするか否かに応じて、そのビットを更新してよい。

【００４８】

図１は、本開示の実施形態によるプロセッサコア１０９を含むコンピュータシステム１００を示す。プロセッサコア１０９は、本明細書で説明する投機機能の制御を管理するための投機マネージャ１１０を含んでよい。スケジューラ１４７は、本明細書で説明する投機実行機能におけるセキュリティチェック省略を含む投機マネージャ１１０を含んでよい。特定の実施形態において、投機マネージャ１１０は、特定のセキュリティチェックポリシーのために実行されるべき１または複数の関連付けられたチェックを格納するためのセキュリティチェックポリシー１１２を含む。例えば、第１のセキュリティチェックポリシーは、メモリ安全性チェックポリシー１１４であってよく、第２のセキュリティチェックポリシーは、型安全性チェックポリシー１１６であってよい。特定の実施形態において、投機マネージャ１１０は、コード（例えば、追加のヒントを持つコード１０６）（例えば、ヒントが追加された命令、またはコード１０６からのその命令のデコードされた命令）にアクセスし、どのポリシーを適用するかを判定し、その後、１または複数の関連付けられたチェックを省略することが安全か安全でないかを判定する。一実施形態において、投機マネージャ１１０は、例えば、省略されるべき命令（例えば、マイクロコード）のスケジューリングを停止（故に実行を停止）するための、コア１０９のスケジューラ１４７へのパス１１０Ａを含む。パス１１０Ａは、一方向通信または双方向通信を可能にしてよい。一実施形態において、スケジューラ１４７は、１または複数の実行ユニット１５４上の命令（例えば、マイクロコード）の実行をスケジューリングする。

【００４９】

図示されたコンピュータシステム１００は、本開示の実施形態による、パイプライン化されたプロセッサコア１０９（１）～１０９（Ｎ）内に分岐予測器１２０および分岐アドレス計算器１４２（ＢＡＣ）を含む。図１を参照すると、パイプライン化されたプロセッサコア（例えば、１０９（１））は、命令ポインタ生成（ＩＰ　Ｇｅｎ）ステージ１１１、フェッチステージ１３０、デコードステージ１４０および実行ステージ１５０を含む。特定の実施形態において、リタイヤメントステージ（例えば、リオーダバッファ（ＲＯＢ）を含む）が実行ステージ１５０の後にくる。一実施形態において、コンピュータシステム（例えば、プロセッサ）１００は、複数のコア１０９（１～Ｎ）を含み、Ｎは任意の正

の整数である。別の実施形態において、コンピュータシステム（例えば、プロセッサ）１００は、シングルコアを含む。特定の実施形態において、各プロセッサコア１０９（１～Ｎ）インスタンスは、マルチスレッディング（例えば、２または２より多いオペレーションまたはスレッドの並行セットを第1の論理コアおよび第２の論理コアに実行）をサポートし、これは、タイムスライスマルチスレッディング、同時マルチスレッディング（例えば、単一の物理コアが、当該物理コアが同時マルチスレッディングを行っているスレッドの各々に、論理コアを提供する場合）、またはこれらの組み合わせ（例えば、タイムスライスフェッチおよびデコーディング並びにその後の同時マルチスレッディング）を含む様々な方法でそのように実行されてよい。示される実施形態においては、各単一のプロセッサコア１０９（１）から１０９（Ｎ）は、分岐予測器１２０のインスタンスを含む。分岐予測器１２０は、分岐ターゲットバッファ（ＢＴＢ）１２４を含んでよい。

【００５０】

特定の実施形態において、分岐ターゲットバッファ１２４は、複数の分岐命令（例えば、複数回実行されたコードのセクションの分岐命令）の各々に対応する予測されたターゲット命令を格納（例えば、分岐予測器アレイ内に）する。示される実施形態においては、分岐アドレス計算器（ＢＡＣ）１４２は、ＢＡＣがアクセス（例えば、含む）するリターンスタックバッファ１４４（ＲＳＢ）を含む。特定の実施形態においては、リターンスタックバッファ１４４は、任意のＣＡＬＬ命令（例えば、スタックにそれらの戻りアドレスをプッシュする）の戻りアドレスを格納（例えば、後入れ先出し（ＬＩＦＯ）のスタックデータ構造で）する。

【００５１】

分岐アドレス計算器（ＢＡＣ）１４２は、特定のタイプの分岐命令のアドレスを計算する、および／または、分岐予測器（例えば、ＢＴＢ）によりなされた分岐予測を検証するために用いられる。特定の実施形態において、分岐アドレス計算器は、分岐ターゲットの、および／または次のシーケンシャルの、線形アドレス計算を実行する。特定の実施形態において、分岐アドレス計算器は、アドレス計算に基づき、分岐に対し、静的予測を実行する。

【００５２】

特定の実施形態において、分岐アドレス計算器１４２は、ＣＡＬＬ命令の戻りアドレスを記録するためのリターンスタックバッファ１４４を含む。一実施形態において、分岐アドレス計算器は、分岐予測ミスのペナルティを低減すべく、分岐予測器１２０によってなされた不適切な予測を訂正することを試みる。１つの例として、分岐アドレス計算器は、分岐命令および命令ポインタから唯一に判定可能なターゲットを持つ分岐に対する分岐予測を検証する。

【００５３】

特定の実施形態において、分岐アドレス計算器１４２は、リターンスタックバッファ１４４を、戻り命令のターゲットアドレスを判定するための分岐予測メカニズムとして用いられるように維持する。例えば、リターンスタックバッファ１４４は、すべての"呼び出しサブルーチン"および"サブルーチンからの戻り"分岐命令をモニタリングすることで動作する。一実施形態において、分岐アドレス計算器が、"呼び出しサブルーチン"分岐命令を検出すると、分岐アドレス計算器は、次の命令のアドレスをリターンスタックバッファにプッシュし、例えば、スタックポインタの最上部が、リターンスタックバッファの最上部をマークする。この実施形態では、各"呼び出しサブルーチン"命令の直後のアドレスをリターンスタックバッファにプッシュすることで、リターンスタックバッファは、リターンアドレスのスタックを含む。後で分岐アドレス計算器が"サブルーチンからの戻り"分岐命令を検出したとき、分岐アドレス計算器は、例えば、分岐予測器１２０により予測されたリターンアドレスを検証すべく、リターンスタックバッファから最上部のリターンアドレスを取り出す。一実施形態において、直接分岐タイプについては、分岐アドレス計算器は、（例えば、常に）条件分岐を取ると予測し、例えば、分岐予測器が直接分岐を取ると予測していない場合、分岐アドレス計算器は、分岐予測器の予測ミスまたは不適切な予測

を上書きする。

【００５４】

図１中のコア１０９は、分岐予測器１２０によってなされた分岐予測を検証するための回路を含む。各分岐予測器１２０エントリは（例えば、ＢＴＢ１２４内にある）、さらに、分岐予測器１２０によってなされた分岐予測の精度を高め且つ検証をするために用いられる有効フィールドおよびバンドルアドレス（ＢＡ）フィールドを含み得、これについては後でより詳しく説明する。一実施形態において、有効フィールドおよびＢＡフィールドはそれぞれ、１ビットフィールドで構成されている。しかしながら、他の実施形態においては、有効フィールドおよびＢＡフィールドのサイズは異なってよい。一実施形態において、フェッチされた命令は、デコードされるべく、デコーダ１４６に送信（例えば、ＢＡＣ１４２により、ライン１３７から）され、デコードされた命令は、実行ユニット１５４の１または複数における実行をスケジューリングすべく、スケジューラ１４７に送信される。

【００５５】

図示されたコンピュータシステム１００は、ネットワークデバイス１０１、入／出力（Ｉ／Ｏ）回路１０３（例えば、キーボード）、ディスプレイ１０５およびシステムバス（例えば、相互接続）１０７を含む。

【００５６】

一実施形態において、分岐予測器１２０内に格納された分岐命令は、取得される分岐命令として、コンパイラによって事前選択される。特定の実施形態においては、図１のメモリ内１０２に格納された図示されるようなコンパイラコード１０４が、実行時に高水準言語で記述されたプログラムのソースコードを、実行可能な機械コードに変換するコードシーケンスを含む。一実施形態において、コンパイラコード１０４は、さらに、追加のヒントを持つコード１０６を形成するためのヒントを含む。コンパイラコードは、さらに、分岐命令（例えば、取得される可能性のある分岐命令（例えば、予め選択された分岐命令））のターゲット命令を予測してよい。分岐予測器１２０（例えば、そのＢＴＢ１２４）は、その後、分岐命令のターゲット命令で更新される。一実施形態において、ソフトウェアがハードウェアＢＴＢを管理し、例えば、当該ソフトウェアが予測モードを指定する、または、ＢＴＢに書き込みを行う命令のモードによって暗黙に定義された予測モードがエントリ内にモードビットも設定する。

【００５７】

以下に説明するように、図示されたコア（例えば、その分岐予測器１２０）は、１または複数のレジスタへのアクセスを含む。特定の実施形態において、コアは、１または複数の汎用レジスタ１０８を含む。

【００５８】

特定の実施形態において、分岐予測器１２０の各エントリ（例えば、そのＢＴＢ１２４内にある）は、タグフィールドおよびターゲットフィールドを含む。一実施形態において、ＢＴＢストア内の各エントリのタグフィールドは、分岐命令を識別する命令ポインタ（例えば、メモリアドレス）の少なくとも一部を格納する。一実施形態において、ＢＴＢストア内の各エントリのタグフィールドは、コード内の分岐命令を識別する命令ポインタ（例えば、メモリアドレス）を格納する。一実施形態において、ターゲットフィールドは、同一エントリのタグフィールド内で識別された分岐命令のターゲットのための命令ポインタの少なくとも一部を格納する。また、他の実施形態において、分岐予測器１２０のエントリ（例えば、そのＢＴＢ１２４内にある）は、１または複数の他のフィールドを含む。特定の実施形態において、エントリは、例えば、分岐命令が存在する（例えば、ＢＴＢ内に）場合は、当該分岐命令が取得されるとみなされるといったように、分岐命令が取得されるかどうかの予測を支援するための別個のフィールドは含まない。

【００５９】

図１に示されるように、ＩＰ生成ステージ１１１のＩＰ　Ｇｅｎ　ｍｕｘ１１３は、ライン１１４Ａから命令ポインタを受信する。ライン１１５Ａを介して提供される命令ポイ

ンタは、インクリメント回路１１５によって生成され、インクリメント回路１１５は、パス１１３Ａから最新の命令ポインタのコピーを受信する。インクリメント回路１１５は、コアによって現在実行されているプログラムシーケンスから次のシーケンシャル命令を取得すべく、命令ポインタを予め定められた量だけインクリメントしてよい。

【００６０】

一実施形態において、ＩＰ　Ｇｅｎ　ｍｕｘ１１３からＩＰを受信するとすぐに、分岐予測器１２０は、ＩＰの一部を、分岐予測器１２０（例えば、ＢＴＢ１２４）内の各エントリのタグフィールドと比較する。この実施形態において、ＩＰと分岐予測器１２０のタグフィールドとの間に合致が見つからない場合、ＩＰ　Ｇｅｎ　ｍｕｘは続けて、次のシーケンシャルＩＰを、フェッチされるべき次の命令として選択する。反対に、合致が検出された場合、分岐予測器１２０は、ＩＰと合致した分岐予測器エントリの有効フィールドを読み取る。この実施形態において、有効フィールドが設定されていない（例えば、論理値０を有する）場合、分岐予測器１２０は、それぞれのエントリを"無効"とみなし、ＩＰとそれぞれのエントリのタグとの間の合致を無視し、そして、例えば、それぞれのエントリの分岐ターゲットは、ＩＰ　Ｇｅｎ　Ｍｕｘに転送されない。一方で、この実施形態において、合致するエントリの有効フィールドが設定されている（例えば、論理値１を有する）場合、分岐予測器１２０は続けて、命令ポインタ（ＩＰ）の予め定められた部分と、合致する分岐予測器エントリの分岐アドレス（ＢＡ）フィールドとの間の論理比較を実行する。"許容可能な条件"が存在する場合、合致するエントリの分岐ターゲットは、ＩＰ　Ｇｅｎ　ｍｕｘに転送される。そうでない場合、分岐予測器１２０はＩＰと分岐予測器エントリのタグとの間の合致を無視する。いくつかの実施形態において、エントリインジケータが現在の分岐ＩＰだけでなく、グローバル履歴の少なくとも一部からも形成される。

【００６１】

より具体的には、一実施形態において、ＢＡフィールドは、それぞれの分岐命令がキャッシュメモリ１３２のライン内のどこに格納されているかを示す。特定の実施形態において、プロセッサは、クロックサイクルごとに、複数の命令の実行を開始でき、ここで命令は、相互依存しておらず、同一の実行リソースを使用しない。

【００６２】

例えば、図１中に示される命令キャッシュ１３２の各ラインは、複数の命令（例えば、６つの命令）を含む。また、この実施形態において、フェッチユニット１３４によるフェッチオペレーションに応答して、命令キャッシュ１３２は、キャッシュのフルラインを、フェッチユニット１３４に提供することで、応答する（例えば、"ヒット"の場合）。キャッシュのライン内の命令は、別個の"バンドル"としてグループ化されてよい。例えば、図１に示されるように、キャッシュライン１３３内の第１の３つの命令は、バンドル０としてアドレス指定されてよく、第２の３つの命令は、バンドル１としてアドレス指定されてよい。バンドル内の命令の各々は互いに独立している（例えば、実行のために同時発行可能）。特定の実施形態において、分岐予測器１２０エントリ内に設けられたＢＡフィールドは、それぞれのエントリに対応する分岐命令のバンドルアドレスを識別するために用いられる。例えば、一実施形態において、ＢＡは、分岐命令が特定のキャッシュラインの第１のバンドルまたは第２のバンドルのいずれに格納されているかを識別する。

【００６３】

一実施形態において、分岐予測器１２０は、合致するエントリのＢＡフィールドと、ＩＰの予め定められた部分との間の論理比較を実行して、"許容可能な条件"が存在するかどうかを判定する。例えば、一実施形態において、ＩＰの５番目のビット位置（例えば、ＩＰ［４］）が、合致する（例えば、ＢＴＢ）エントリのＢＡフィールドと比較される。一実施形態において、ＩＰ［４］がＢＡより大きくない場合に、許容可能な条件が存在する。このような許容可能な条件は、分岐命令の明らかに不要な予測を防止することに役立ち、このような分岐命令は実行されなくてよい。すなわち、分岐予測器１２０のタグとの比較を行う際にＩＰの全部より少ないものが考慮される場合、タグとの合致を有することは

可能であるが、それは真の合致ではない可能性がある。にもかかわらず、ＩＰと分岐予測器のタグとの間の合致は、キャッシュの特定のラインを示し、当該特定のラインは、それぞれの分岐予測器エントリに対応する分岐命令を含み、まさに実行される予定であってよい。具体的に言うと、ＩＰのバンドルアドレスが合致する分岐予測器エントリのＢＡフィールドより大きくない場合は、それぞれのキャッシュライン内の分岐命令はすぐに実行される予定である。従って、特定の実施形態において、分岐命令のターゲットのフェッチに進むことで、性能の利益が得られてよい。

【００６４】

上記の通り、この例では、"許容可能な条件"が存在する場合、合致するエントリの分岐ターゲットは、ＩＰ　Ｇｅｎ　ｍｕｘに転送されることになる。そうでない場合、分岐予測器は、ＩＰとタグとの間の合致を無視することになる。一実施形態において、分岐予測器から転送された分岐ターゲットは、最初に分岐予測（ＢＰ）ｒｅｓｔｅｅｒ　ｍｕｘ１２８に送信されてから、ＩＰ　Ｇｅｎ　ｍｕｘに送信される。図１に示されるように、ＢＰ　ｒｅｓｔｅｅｒ　ｍｕｘ１２８は、他の分岐予測デバイスからの命令ポインタも受信してよい。一実施形態において、ＢＰ　ｒｅｓｔｅｅｒ　ｍｕｘによって受信される入力ラインは、どの入力ラインが、ＢＰ　ｒｅｓｔｅｅｒ　ｍｕｘからＩＰ　Ｇｅｎ　ｍｕｘへと渡されることが可能かを決定するために優先順位付けされる。

【００６５】

分岐ターゲットをＢＰ　ｒｅｓｔｅｅｒ　ｍｕｘに転送することに加え、ＩＰと分岐予測器のタグとの間の合致が検出されるとすぐに、合致する分岐予測器エントリのＢＡが分岐アドレス計算器（ＢＡＣ）１４２に転送される。図１中、ＢＡＣ１４２は、デコードステージ１４０に位置するように図示されているが、他のステージに位置してもよい。ＢＡＣは、フェッチユニット１３４からライン１３７を介してキャッシュラインを受信してもよい。

【００６６】

この例において、ＩＰ　Ｇｅｎ　ｍｕｘにより選択されるＩＰも、データライン１３５を介してフェッチユニット１３４に転送される。ＩＰがフェッチユニット１３４に受信されると、当該ＩＰに対応するキャッシュラインが、命令キャッシュ１３２からフェッチされる。命令キャッシュから受信されるキャッシュラインは、データライン１３７を介してＢＡＣに転送される。

【００６７】

この例において、ＢＡを受信するとすぐに、ＢＡＣはＢＡを読み取り、予め選択された分岐命令（例えば、合致する分岐予測器エントリ内で識別された）が、ＢＡＣにより読み取られるべき次のキャッシュライン内のどこ（例えば、キャッシュラインの第１のバンドルまたは第２のバンドル）に位置するかを判定する。一実施形態において、分岐命令がキャッシュラインのバンドル内のどこに位置するかは予め決まっていてよい（例えば、３つの命令の１つのバンドル内といったように。分岐命令は、第２の命令として格納される）。

【００６８】

代替的な実施形態においては、ＢＡは、キャッシュライン内の分岐命令のアドレスをより具体的に識別するための追加ビットを含む。従って、分岐命令は、バンドル内の特定の命令位置に限定されない。

【００６９】

ＢＡＣがキャッシュライン内の予め選択された分岐命令のアドレスを判定し、且つ、フェッチユニット１３４からそれぞれのキャッシュラインを受信した後、ＢＡＣは、分岐命令に真に対応するＩＰを検証すべく、それぞれの命令をデコードする。受信されたキャッシュライン内のＢＡでアドレス指定された命令が分岐命令である場合、その分岐予測に対する訂正は不要である。反対に、キャッシュライン内のそれぞれの命令が分岐命令でない（すなわち、ＩＰが分岐命令に対応していない）場合、同一分岐予測器エントリでの同様の予測ミスを防止すべく、ＢＡＣはそれぞれの分岐予測器エントリを無効にするためのメ

ッセージを分岐予測器に送信する。その後、無効にされた分岐予測器エントリは、新しい分岐予測器エントリで上書きされる。

【００７０】

また、一実施形態においてにおいて、ＢＡＣは、ＩＰを予め定められた量だけインクリメントして、インクリメントされたＩＰをＢＰ　ｒｅｓｔｅｅｒ　ｍｕｘ１２８にデータライン１４５を介して転送する。例えば、ＢＡＣから出るデータライン１４５は、分岐予測器からのデータラインより優先する。その結果、当該ＩＰの後にシーケンシャルに続く命令をフェッチすることで、分岐予測ミスを訂正すべく、インクリメントされたＩＰはＩＰ　Ｇｅｎ　ｍｕｘに転送され、フェッチユニットに渡される。

【００７１】

投機実行におけるセキュリティチェックの省略のための本明細書の実施形態は、特定のセキュリティポリシー（例えば、メモリ安全性ポリシーおよび／または型安全性ポリシー）を強制実行するための様々なアーキテクチャを用いてインスタンス化されてよい。この１つの例示が、インテル（登録商標）メモリ保護拡張（ＭＰＸ）を用いるものであり、ＭＰＸは、ペアのレジスタ（例えば、６４ビット）により示されるメモリ範囲に対しポインタをチェックする命令（例えば、ＢＮＤＣＬ／Ｎ／Ｕ）を定義する。故に、これらの命令は、メモリ安全性を強制実行するために用いられてよい。投機実行におけるセキュリティチェックの省略のための本明細書の実施形態は、プロセッサに対し、安全であるとまだみなされていないアクセスのポインタをチェックする場合にのみ、チェックは必要であるというヒントを示すべく、命令プレフィックス（例えば、セグメント上書きプレフィックス）（さもなければ命令（例えば、ＭＰＸ命令）の文脈において無意味となる）を再定義してよい。プロセッサは、どの命令がポインタを含むレジスタを使用するかを列挙することで、関連するアクセスを識別してよい。しかしながら、例えば、実際のアクセスは関連付けられたＭＰＸ命令に従うので、命令自体をあたかもアクセス命令であるとみなし、且つ、アクセス命令ベースで安全と判定することも可能である。

【００７２】

以下の擬似コードシーケンスは、ＭＰＸ命令が投機ミスされ得る場合にのみ、そのセキュリティチェックが必要となることを示すヒントを用いて、どのようにＭＰＸ命令を修正できるかを示す。

【表２】

```
int foo(int *x) {
  return *x; // この例は、セキュリティチェック省略のヒントの説明を簡略化するために、
構築された
}

セキュリティチェックを無条件で実行するアセンブリコード:
foo:
// パラメータxがレジスタRDIに渡され、xがポイントするオブジェクトの境界値が、
レジスタBND0に渡される:
BNDCL BND0, RDI //  この境界チェック命令は、RDIがBND0レジスタ内の下限値未満の場合、
例外を生成する
BNDCU BND0, [RDI+3] //  この境界チェック命令は、RDI＋3（後続のmove（MOV）命令に
よりアクセスされる最上位バイトのアドレス）がBND0レジスタの
上限値を超えている場合に、例外を生成する
MOV EAX, [RDI]

投機ミスされ得るアクセスに対し、条件的にセキュリティチェックを
実行するアセンブリコード:
foo:
//パラメータxがレジスタRDIに渡され、xがポイントするオブジェクトの境界値が、
レジスタBND0に渡される
ES BNDCL BND0, RDI //  この境界チェック命令は、RDIがBND0レジスタ内の
下限値未満である場合、例外を生成する。しかしながら、アクセスが安全であると判明している場合は、
境界チェックはスキップされる
ES BNDCU BND0, [RDI+3] // この境界チェックは、RDI＋3（後続のMOV命令によりアクセスされる
最上位バイトのアドレス）が、BND0レジスタ内の上限値を超えている場合、例外を生成する。しかしながら、
アクセスが安全であると判明している場合は、境界チェックはスキップされる。
MOV EAX, [RDI]
```

【００７３】

この擬似コードの例は、ＥＳセグメント上書きプレフィックスを用いて、関連付けられたアクセスが投機ミスされないと判明している場合は、メモリ安全性チェックは不要（例えば、省略可能）であることを示す。例えば、この文脈では、ＥＳプレフィックスの元の目的は重要ではなく、故にここでヒントに用いるために利用可能な命令プレフィックスであるとみなす。

【００７４】

特定の実施形態において、ヒントを持つ命令は、それが既に安全とみなされている場合は、オペレーションなし（ｎｏｐ）として扱われる。この一例が図２に示されている。

【００７５】

図２は、本開示の実施形態による、セキュリティチェック命令を省略するためのフロー図２００を示す。示されたフロー図２００は、ヒント（例えば、セキュリティチェックフィールド）を持つセキュリティチェック命令を実行するための要求を受信する２０２段階と、セキュリティチェック命令のすべての関連付けられたアクセスが既に安全であるか否かを判定する２０４段階と、是であれば、セキュリティチェック命令の実行をスキップ（例えば、省略）する段階２０６と、否であれば、セキュリティチェック命令を実行する２０８段階と、を含む。

【００７６】

特定の実施形態において、省略により、命令の機能（例えば、セキュリティチェック）の真部分集合が除去されるが、機能のすべてを除去することはしない（例えば、部分的に省略された命令のセキュリティチェック機能以外は除去しない）。この例のための擬似コ

ードについて以下に説明する。

【００７７】

プロセッサは、エンコードされたインライン機能（ＥＩＣ）を、別個のセキュリティチェック命令を活用する別のセキュリティアーキテクチャとして有してよく、ＥＩＣは、セキュリティチェックの省略のための本明細書の実施形態の文脈において、ＭＰＸ命令と幾分同様に扱われてよい。特定の実施形態において、ＭＰＸとＥＩＣとの間の差異は、ＥＩＣの境界チェック命令は、ＭＰＸ命令と同様の境界チェックを実行するのみでなく、入力ポインタ値が後続のメモリアクセスによって使用できるように入力ポインタ値もデコードする。ＭＰＸの実施形態が、境界をポインタとは別個に格納する一方、ＥＩＣの実施形態では、境界情報をエンコードされたポインタ（例えば、６４ビット）に直接埋め込む。これは、特定の実施形態において、ポインタを介してメモリへのアクセスを試行する前に、その追加情報を除去するために、ポインタがデコードされることを意味する。故に、たとえ、アクセスが安全であると判明しており、且つ、潜在的に投機ミスされた場合にのみセキュリティチェックが必要であるとマークされていたとしても、特定の実施形態のＥＩＣセキュリティチェック命令は、ポインタを依然としてデコードし、後続のメモリアクセスが続行できるようにする。

【００７８】

ＥＩＣセキュリティチェック命令の１つの例（例えば、チェック機能（ＣｈｅｃｋＣａｐ）と称される）は、エンコードされたポインタおよび実行されるべきアクセスのサイズを指定する入力オペランドを取得し、例えば、その結果、命令をデコードおよび実行することで、境界チェックが行われ、境界チェックが成功した場合は、エンコードされたポインタがデコードされたポインタで上書きされる。

【００７９】

これは、ＭＰＸベースの例で用いられた上記の擬似コードに基づく、以下のコードシーケンスで示される。

【表３】

```
int foo(int *x) {
  return *x;
}

セキュリティチェックを無条件に実行するアセンブリコード:
foo:
// パラメータxがレジスタRDIに渡されると仮定:
CheckCap RDI, 4
// この命令は、RDIがRDIにエンコードされた下限値未満の場合、または、RDI＋3がRDIに
エンコードされた上限値を超えている場合、例外を生成する。それ以外の場合、この命令は、
RDI内のポインタをデコードし、RDIの元の値を、デコードされたポインタで上書きする。
MOV RAX, [RDI]

投機ミスされ得るアクセスに対して、セキュリティチェックを条件的に
実行するアセンブリコード:
foo:
// パラメータxがレジスタRDIに渡されると仮定:
ES CheckCap RDI, 4
// この命令は、RDIがRDIにエンコードされた下限値未満の場合、または、RDI＋3がRDIに
エンコードされた上限値を超えている場合、例外を生成する。命令が安全であると判明している場合、
境界値チェックはスキップされる。いずれの場合も、命令が例外を生成しない場合は、この命令は、
RDI内のポインタをデコードし、RDIの元の値を、デコードされたポインタで上書きする。
MOV RAX, [RDI]
```

【００８０】

ＭＰＸベースの例と同様、ＥＩＣの例は、特定の実施形態において、命令が潜在的に投

機ミスされる場合にのみ境界チェックが実行されることが必要であることをプロセッサに示すべく、セグメント上書きプレフィックスＥＳを再定義する。

【００８１】

境界および型情報のストレージを管理するためのアプリケーションまたは言語ランタイムを用い、且つ必要時に実際のセキュリティチェックを実行する他の命令も再定義可能である。例えば、ＲＢＮＤＣＬ／Ｎ／Ｕ命令は、ＭＰＸ境界レジスタではなく、汎用レジスタ（ＧＰＲ）内に格納された境界に対しオペレーションを実行してよく、それらは、本開示のこの文脈において同様に修正されてよい。さらに、型チェック（ＴＹＰＥＣＨＫ）命令は、２つのタイプの識別値（ＩＤ）が等しいかを比較し、且つ、合致しない場合は例外を生成するように定義されてよい。これらの命令も、命令が安全な場合に、セキュリティチェックをスキップするためのＥＳプレフィックス等のようなヒントで修正されてよい。

【００８２】

一部のアーキテクチャは、セキュリティチェックを実行するための別個の命令に依存しなくてよく、代わりに、メモリにアクセスする前に自動的にセキュリティチェックを実行してよい。例えば、タグ値を含むポインタがメモリアクセスで用いられる場合は、メモリタグが自動的にチェックされてよい。すべてのアクセスされるメモリ位置のタグ値がタグストレージからロードされ、ポインタ内のタグと比較されてよい。これらの比較のうちのいずれかのものが不一致を呈する場合、例外が生成されてよい。セキュリティチェック命令のためにヒントが供給可能であるのと同様に、自動的なセキュリティチェックがスキップされるべきであることを示すヒントがメモリアクセスに供給されてよい。以下は、上記の例に基づく例示的な擬似コードである。

【表４】

```
int foo(int *x) {
  return *x;
}

セキュリティチェックを無条件に実行するアセンブリコード:
foo:
// パラメータxがレジスタRDIに渡されると仮定:
MOV RAX, [RDI]
// この命令は、RDIのタグチェックがミスマッチを示す場合、例外を生成する

セキュリティチェックを条件的に実行するアセンブリコード:
foo:
// パラメータxがレジスタRDIに渡されると仮定:
ES MOV RAX, [RDI]
// この命令は、RDIのタグチェックがミスマッチを示す場合、例外を生成する。
しかしながら、プロセッサがこの命令が安全であるとわかっている場合は、
プロセッサはタグチェックを省略してよい。これは、タグ値をロードする必要性すら回避することで、
オーバーヘッドを大きく低減し得る
```

【００８３】

しかしながら、ヒントとして、特定のセグメント上書きプレフィックスを用いることが、メモリアクセスに望ましくない影響を及ぼすことがあり得る。例えば、本開示によるヒントを提供するためにＦＳプレフィックスおよびＧＳプレフィックスを使用することで、メモリアクセス内に指定されたアドレスへのセグメントベースアドレスの追加が生じる可能性があり、これにより、誤ったメモリ位置へのアクセスが生じる可能性がある。例えば、これらのプレフィックスがプロセッサアーキテクチャ内に既に定義されているときに、この種のアドレス計算が生じる。他のモード（例えば、３２ビット線形アドレスを用いる）では、さらに多くのセグメント上書きプレフィックスにより、意図しないセグメントベ

ースへのアドレスの追加が生じる可能性がある。故に、特定の実施形態において、意図しない効果を生じさせない他のプレフィックスを選択する、または、ヒントのための新しいプレフィックス若しくは他のエンコーディングを定義する必要があり得る。

【００８４】

いくつかの命令は、同一命令内に複数のタイプのチェックを組み合わせる。一実施形態において、命令は、境界および型情報の両方をチェックする。これを説明するために、ＭＰＸ境界レジスタおよび対応する境界テーブルエントリが型情報を含むように拡張され得ること、およびＭＰＸ命令がその型情報をチェックするように拡張され得ることを考えてみよう。例えば、境界チェック（例えば、下限チェック）命令の１つの変形例は、単一レジスタオペランド内のチェックされるべきアドレスを受け付けるので、タイプＩＤを指定する即値オペランドを受け付けるように拡張可能である。一例として、ＴＹＰＥＢＮＤＣＬと呼ばれる、この拡張された命令（例えば、ＢＮＤＣＬ）は、その即値オペランド内のタイプＩＤを、境界レジスタ内のタイプＩＤと比較し、それらが合致しない場合に例外を生成してよい。故に、コンパイラは、命令が安全であると判明している場合に、境界チェックおよび型チェックのうちの一方または両方を選択的にキャンセルするためのヒントを供給し得る。例えば、ＥＳプレフィックスは、このような場合に境界チェックをキャンセルしてよく、ＦＳセグメント上書きプレフィックスは、このような場合に型チェックをキャンセルしてよい。これら両方のチェックをキャンセルすべく、ＥＳプレフィックスとＧＳセグメント上書きプレフィックスの両方のプレフィックスが供給されてよく、または、その目的のために第３のヒント、例えば、ＧＳプレフィックスが定義されてもよい。

【００８５】

セキュリティポリシーを強制実行するためのすべてのアーキテクチャが別個のセキュリティチェック命令を定義するわけではなく、命令を修正して、命令プレフィックス付きのアクセスを生成するアプローチがここでも適用されてよい。例えば、メモリタギングは、アクセスされているストレージ位置のアドレスによりインデックスされている境界メタデータに対し、アクセスを暗黙にチェックする。特定のアーキテクチャは、複合命令セットコンピューティング（ＣＩＳＣ）であるので、多数の命令がメモリアクセスを生成してよく、例えば、すべての命令がメモリオペランド付きである。また他のセキュリティアーキテクチャも、例えば、ｆａｔポインタを用いたメモリおよび型安全性を強制実行する機能メカニズムのような暗黙的なチェックを実行してよい。上記のサンプルコードに示される通り、場合によっては、安全なアクセスに対し、境界チェックおよび型チェックのうちの一方または他方のみを実行することが必要であり、よって、これらのタイプのチェックの各々のための別個の修飾子が単一の命令内で両方をサポートするアーキテクチャ用に定義されてよい。

【００８６】

以下に、ヒント（例えば、セキュリティチェックフィールド）を挿入する例について説明する。特定の実施形態において、コンパイラが、メモリアクセス、および任意の関連付けられたセキュリティチェック命令（該当する場合は、複数の関連付けられたセキュリティチェック命令）を発出する各時点において、コンパイラは本明細書で説明するような、１または複数のヒントも発出するか否かを決定する。そうするために、これらの実施形態のうちの特定の実施形態において、コンパイラは、アクセスが安全とみなされる場合にセキュリティチェックを省略してよいか否かを考慮する。いくつかの例示的なシナリオについて上記したが、これらは網羅的ではない。

【００８７】

図３を参照して、これらの判定をするための例示的なフローについて以下に説明する。このフローは、アクセスごとに複数回実行されてよく、例えば１回目はメモリ安全性チェックポリシー、別の回は型安全性チェックポリシーといったように異なるタイプのポリシーのための異なる判定をする。特定の実施形態において、これらの決定は最終的に上記のようなマージされたヒントをもたらし、例えば、アクセスが安全であるとみなされる場合、メモリ安全性チェックおよび型安全性チェックの両方が省略されることを示す。

【００８８】
図３は、本開示の実施形態によりコード内にヒントを発出するためのフロー図３００を示す。示されるフロー３００は、メモリアクセスおよび関連付けられたセキュリティチェック命令を含むコード（例えば、コンパイラによる、またはコンパイラからの）を受信する３０２段階と、アクセスが安全とみなされる場合、適用可能なセキュリティポリシーが害される可能性があるか否かを判定する３０４段階と、否の場合、ヒント（例えば、セキュリティチェック命令のセキュリティチェックフィールドとして）をコード内に発出する３０６段階と、是の場合、ヒント（例えば、セキュリティチェック命令のセキュリティチェックフィールドとして）をコード内に発出しない（例えば、その結果、セキュリティチェック命令が実行されるように）３０８段階と、を含む。
【００８９】
以下は、潜在的なタイプの投機および関連付けられたチェックの例を含む。
【００９０】
一実施形態において、命令が潜在的に投機ミスされる可能性がある場合（例えば、のみ）に実行されるセキュリティチェックに関するプロセッサへのヒントは、潜在的な投機ミスの場合に強制実行されるべきポリシーのタイプを指定するものである。すると、例えば、プロセッサは、プロセッサがサポートする投機のタイプに依存する条件をチェックして、命令が潜在的に投機ミスされるか否かを判定する。
【００９１】
以下に、プロセッサによってサポートされ得る潜在的な投機のタイプ、およびその特定のタイプの投機について、命令が潜在的に投機ミスされるか否かを判定するためにプロセッサが実行し得る関連付けられたチェックに関する、非網羅的な例を説明する。プロセッサが複数のタイプの投機をサポートする場合、プロセッサは、すべてのサポートされたタイプの投機に対するチェックを実行して、命令が特定のセキュリティポリシー（例えば、メモリ安全性または型安全性）を害し得る態様で投機ミスされないことを判定してよい。
【００９２】
プロセッサが他のタイプの投機に関連する命令の態様が正しく投機されるかを判定していなくても、適用可能なセキュリティポリシーを害することがないためにセキュリティチェックがスキップされてよい、サポートされた他のタイプの投機が存在してよいことに留意されたい。例えば、投機可能なデータ値が考えられる。たとえストアのデータ値が投機ミスされる場合であっても、それは、特定の実施形態のメモリ安全性ポリシーまたは型安全性ポリシーを害することはない。例えば、格納されるべきデータ値は、ストアにどのアドレスが用いられるかには影響を及ぼさないからである。
【００９３】
故に、特定の実施形態は、プロセッサが、命令の実行のいくつかの態様の投機ミスが適用可能なセキュリティポリシーを弱体化させないと判定した命令を言及するにあたり、"正しく投機される"命令ではなく、"安全"である命令（例えば、セキュリティチェック命令の関連付けられたメモリアクセス）を区別する。
［分岐予測］
【００９４】
プロセッサは、前の分岐および他の要因からの履歴に基づき、分岐がどの方向を取るかについて予測を試みてよい。特定の実施形態において、これは、分岐が解決される（例えば、正しい方向が完全に決定される）前に、予測される分岐ターゲットを投機的に実行するために用いられる。特定の実施形態において、命令が正確に投機されたパスにあることを判定すべく、プロセッサはまず、プログラム内の先行する分岐を解決する必要がある。
［ストアバッファバイパス投機］
【００９５】
プロセッサは、ロードおよびストアを並べ替えてよく、場合によっては、ロードより古いストア（例えば、プログラム内のロードに先行する）により上書きされることになる位置からデータを取得するロードが発生することがある。そのデータは、そのロードに依存

するオペレーションにより投機的に用いられてよい。例えば、そのデータは、ポインタアドレスまたは整数値若しくは浮動小数点値等として解釈されてよい。プロセッサが正確な値がロードされていることを判定するために、プロセッサは、メモリ曖昧性除去回路を用いて、ロードが、既にバッファ済みの古い（例えば、プログラム順序において）ストアオペレーションによっても参照されない位置を参照していることを検証してよい。

［他の可能なタイプのセキュリティハードニング］

【００９６】

セキュリティハードニングの他の可能なアプローチが存在する。これは、例えば、敵対者に可視となる可能性のあるキャッシュ状態への変更を行うメモリアクセスを許容する前に、すべての先行する分岐が解決されるまで待機することである。これらは、上で示した例よりも、さらに厳格なポリシーを強制実行することができる。そうしなければ、メモリ安全性チェックまたは型安全性チェックは完了した後ではあるが、これらのポリシーの強制実行の一環として、未だチェックされていない投機ミスの形態があり得るといった場合に、敵対者に可視となり得るキャッシュ状態への変更を許容することになり得る。例えば、以下の例示的な擬似コードを検討してみよう。

【表５】

```
int foo(bool *priv) {
  int priv_array[] = { … /* 5 elements */ };
  int unpriv_array[] = { … /* 10 elements */ };
  int *arr = NULL;
  int idx = 9;
  if (*priv) { // *privの値が真であることは、特権エンティティが当該機能を呼び出しており、
 priv_array内のデータにアクセスすることが承認されることを示す。そうでない場合、*privが偽の場合、
 呼び出し元は、unpriv_arrayからデータを受信することを承認されるのみである。
    arr = priv_array;
    idx = 4;
  } else {
    arr = unpriv_array;
  }
  // 例えば、ifブロック内のidxへの更新により生成されるストアが以下の戻りステートメントから生成される
 idxのロードによりバイパスされることにより、プロセッサがarrにインデックスすべく投機ミスのidxの値を
 用い得ると仮定すると、コンパイラが、アクセスが安全であるとみなされればセキュリティチェックは
 不要であるとのヒントをメモリ安全性チェックに挿入することを選択する例である。プロセッサがarrが誤った型の
 メモリをポイントするという投機ミスをする可能性がある場合は、型安全性チェックも実行可能である。
  return arr[idx];
 }
```

【００９７】

ｐｒｉｖによりポイントされたメモリからの読み取りは遅延する可能性があることを考慮すると、プロセッサは、ｉｆブロック内またはｅｌｓｅブロック内のいずれのコードを実行すべきかを投機ミスする可能性がある。メモリ安全性チェックおよび／または型安全性チェックは、ａｒｒ変数を通して参照されるアレイへのアクセスに依然適用可能である。しかしながら、本開示で説明された特定のヒントがこれらのチェックに適用されるか否かに関わらず、ｆｏｏが特権のない呼び出し元から呼び出された場合に、これらのチェックではｐｒｉｖ＿ａｒｒａｙへのアクセスを必ずしも防止しない。その際、悪意の呼び出し元が潜在的に当該機能からの戻り値を用いて、敵対者に可視となるような態様でキャッシュの状態に影響を及ぼす可能性があることを考慮されたい。

【００９８】

このような状況のために、上記の擬似コードで説明したもののような、より厳格なポリシーを選択的に強制実行する別のタイプのヒントがここで定義される。例えば、サンプルコード内の戻りステートメントにより生成されるメモリアクセスは、別のセグメント上書きプレフィックス、例えばＧＳを用いて修正され、当該メモリアクセスは、さらに厳格な

ポリシーにより調整される必要があることを示してよい。しかしながら、いくつかの適用または適用のうちの一部については、メモリ安全性および型安全性で十分である可能性があり、そのコードのユーザは、それらのより緩いポリシーのみが強制実行される場合に可能となる追加の最適化から利益を得るであろう。上記のようなさらに厳格なポリシーを選択的に強制実行できることは、既定では、より緩いポリシーを強制適用して追加の最適化の機会を提供しつつ、プログラムにおけるクリティカルポイントでは、さらに厳格なポリシーをも強制実行する点において有用であろう。

【００９９】

データキャッシュ階層において投機を不可視にする戦略は、上記のようなこのヒントにより選択され得るさらに厳格なタイプのポリシーを強制実行するためのメカニズムの一例である。

【０１００】

図４は、本開示の実施形態によるフロー図４００を示す。示されるフロー４００は、ハードウェアプロセッサのデコーダを用いて、命令をデコードされた命令にデコードする４０２段階と、ハードウェアプロセッサにより、命令内のセキュリティチェックフィールドを検出する４０４段階と、ハードウェアプロセッサにより、潜在的に投機ミスされる実行に対し強制実行されるべきセキュリティチェックポリシーを、セキュリティチェックフィールドに基づき、複数のセキュリティチェックポリシーから判定する４０６段階と、ハードウェアプロセッサにより、セキュリティチェックポリシーの１または複数の関連付けられたチェックを命令に対し実行して、命令が潜在的に投機ミスされるか否かを判定する４０８段階と、ハードウェアプロセッサにより、１または複数の関連付けられたチェックにより命令が安全ではないとみなされた場合に、命令の実行をスケジューリングする４１０段階と、ハードウェアプロセッサにより、１または複数の関連付けられたチェックにより、命令が安全であるとみなされた場合に、命令を省略する４１２段階と、ハードウェアプロセッサの実行ユニットを用いて、実行のためのスケジューリングがされた命令を実行する４１４段階と、を含む。

【０１０１】

以下に、上記の内容が用いられてよい例示的なアーキテクチャ、システム等について説明する。

【０１０２】

本開示の技術の少なくともいくつかの実施形態には、以下の例に照らし説明できる。

［例１］

命令をデコードされた命令にデコードするためのデコーダと、

投機マネージャ回路であって、

上記命令内のヒント（例えば、セキュリティチェックフィールド）を検出する、

潜在的に投機ミスされる実行に対し強制実行されるべきセキュリティチェックポリシーを、上記ヒント（例えば、上記セキュリティチェックフィールド）に基づき、複数のセキュリティチェックポリシーから判定する、

上記セキュリティチェックポリシーの１または複数の関連付けられたチェックを上記命令に対し実行して、上記命令が潜在的に投機ミスされるか否かを判定する、

上記１または複数の関連付けられたチェックにより、上記命令が安全ではないとみなされた場合に、上記命令の実行をスケジューリングする、

上記１または複数の関連付けられたチェックにより、上記命令が安全であるとみなされた場合に、上記命令を省略する、投機マネージャ回路と、

実行のためのスケジューリングがされた上記命令を実行するための実行ユニットと、を備える、装置。

［例２］

上記ヒント（例えば、上記セキュリティチェックフィールド）はコンパイラにより提供されるヒントである、例１に記載の装置。

［例３］

上記投機マネージャ回路は、上記セキュリティチェックポリシーの上記１または複数の関連付けられたチェックを、上記命令の関連付けられたメモリアクセスのセットに対し実行する、例１に記載の装置。

［例４］

上記セキュリティチェックポリシーは、メモリ安全性チェックポリシーである、例１に記載の装置。

［例５］

上記セキュリティチェックポリシーは、型安全性チェックポリシーである、例１に記載の装置。

［例６］

上記セキュリティチェックポリシーの上記１または複数の関連付けられたチェックは、メモリ安全性チェックおよび型安全性チェックを含む、例１に記載の装置。

［例７］

上記１または複数の関連付けられたチェックは、上記装置のアーキテクチャ仕様の完全準拠チェックより少ない、例１に記載の装置。

［例８］

上記命令は、プログラム順序において後続するメモリアクセス命令に関連付けられたセキュリティチェック命令である、例１に記載の装置。

［例９］　ハードウェアプロセッサのデコーダを用いて、命令をデコードされた命令にデコードする段階と、

上記ハードウェアプロセッサにより、上記命令内のセキュリティチェックフィールドを検出する段階と、

上記ハードウェアプロセッサにより、潜在的に投機ミスされる実行に対し強制実行されるべきセキュリティチェックポリシーを、上記セキュリティチェックフィールドに基づき、複数のセキュリティチェックポリシーから判定する段階と、

上記ハードウェアプロセッサにより、上記セキュリティチェックポリシーの１または複数の関連付けられたチェックを上記命令に対し実行して、上記命令が潜在的に投機ミスされるか否かを判定する段階と、

上記ハードウェアプロセッサにより、上記１または複数の関連付けられたチェックにより上記命令が安全ではないとみなされた場合に、上記命令の実行をスケジューリングする段階と、

上記ハードウェアプロセッサにより、上記１または複数の関連付けられたチェックにより、上記命令が安全であるとみなされた場合に、上記命令を省略する段階と、

上記ハードウェアプロセッサの実行ユニットを用いて、実行のためのスケジューリングがされた上記命令を実行する段階と、を備える、方法。

［例１０］

上記セキュリティチェックフィールドは、コンパイラにより提供されるヒントである、例９に記載の方法。

［例１１］

上記実行する段階は、上記セキュリティチェックポリシーの上記１または複数の関連付けられたチェックを、上記命令の関連付けられたメモリアクセスのセットに対し実行する段階を含む、例９に記載の方法。

［例１２］

上記セキュリティチェックポリシーは、メモリ安全性チェックポリシーである、例９に記載の方法。

［例１３］

上記セキュリティチェックポリシーは、型安全性チェックポリシーである、例９に記載の方法。

［例１４］

上記セキュリティチェックポリシーの上記１または複数の関連付けられたチェックは、

メモリ安全性チェックおよび型安全性チェックを含む、例９に記載の方法。
［例１５］
上記１または複数の関連付けられたチェックは、上記ハードウェアプロセッサのアーキテクチャ仕様の完全準拠チェックより少ない、例９に記載の方法。
［例１６］
上記命令は、プログラム順序において後続するメモリアクセス命令に関連付けられたセキュリティチェック命令である、例９に記載の方法。
［例１７］
機械により実行されると、機械に対し、方法を実行させるためのコードを格納する非一時的機械可読媒体であって、上記方法は、
ハードウェアプロセッサのデコーダを用いて、命令をデコードされた命令にデコードする手順と、
上記ハードウェアプロセッサにより、上記命令内のセキュリティチェックフィールドを検出する段階と、
上記ハードウェアプロセッサにより、潜在的に投機ミスされる実行に対し強制実行されるべきセキュリティチェックポリシーを、上記セキュリティチェックフィールドに基づき、複数のセキュリティチェックポリシーから判定する段階と、
上記ハードウェアプロセッサにより、上記セキュリティチェックポリシーの１または複数の関連付けられたチェックを上記命令に対し実行して、上記命令が潜在的に投機ミスされるか否かを判定する段階と、
上記ハードウェアプロセッサにより、上記１または複数の関連付けられたチェックにより上記命令が安全ではないとみなされた場合に、上記命令の実行をスケジューリングする段階と、
上記ハードウェアプロセッサにより、上記１または複数の関連付けられたチェックにより上記命令が安全であるとみなされた場合に、上記命令を省略する段階と、
上記ハードウェアプロセッサの実行ユニットを用いて、実行のためのスケジューリングがされた上記命令を実行する段階と、を備える、非一時的機械可読媒体。
［例１８］
上記セキュリティチェックフィールドは、コンパイラにより提供されるヒントである、例１７に記載の非一時的機械可読媒体。
［例１９］
上記実行する段階は、上記セキュリティチェックポリシーの上記１または複数の関連付けられたチェックを、上記命令の関連付けられたメモリアクセスのセットに対し実行する段階を含む、例１７に記載の非一時的機械可読媒体。
［例２０］
上記セキュリティチェックポリシーは、メモリ安全性チェックポリシーである、例１７に記載の非一時的機械可読媒体。
［例２１］
上記セキュリティチェックポリシーは、型安全性チェックポリシーである、例１７に記載の非一時的機械可読媒体。
［例２２］
上記セキュリティチェックポリシーの上記１または複数の関連付けられたチェックは、メモリ安全性チェックおよび型安全性チェックを含む、例１７に記載の非一時的機械可読媒体。
［例２３］
上記１または複数の関連付けられたチェックは、上記ハードウェアプロセッサのアーキテクチャ仕様の完全準拠チェックより少ない、例１７に記載の非一時的機械可読媒体。
［例２４］
上記命令は、プログラム順序において後続するメモリアクセス命令に関連付けられたセキュリティチェック命令である、例１７に記載の非一時的機械可読媒体。

【０１０３】
さらなる別の実施形態において、装置は、ハードウェアプロセッサにより実行されると、ハードウェアプロセッサに、本明細書で開示された任意の方法を実行させるコードを格納するデータストレージデバイスを備える。装置は、詳細な説明中に説明されたようなものであってよい。方法は、詳細な説明中に説明されたようなものであってよい。
【０１０４】
命令セットは、１または複数の命令のフォーマットを含んでよい。特定の命令フォーマットは、指定する様々なフィールド（例えば、ビット数、ビット位置）、とりわけ、実行されるべきオペレーション（例えば、オペコード）およびそのオペレーションが実行されるオペランドおよび／または他のデータフィールド（例えば、マスク）を定義してよい。いくつかの命令フォーマットはさらに、命令テンプレート（またはサブフォーマット）の定義により細分化されてよい。例えば、特定の命令フォーマットの命令テンプレートは、命令フォーマットのフィールド（含まれるフィールドは通常同一の順序であるが、少なくともいくつかは、含まれるフィールドがより少ないので、異なるビット位置を有する）の異なるサブセットを有するように定義されてよく、および／または、解釈上異なる特定のフィールドを有するように定義されてよい。故に、ＩＳＡの各命令は、特定の命令フォーマット（および定義される場合は、その命令フォーマットの複数の命令テンプレートのうちの特定のもの）を用いて表現され、且つ、オペレーション及びオペランドを指定するためのフィールドを含む。例えば、例示的なＡＤＤ命令は、特定のオペコード並びにそのオペコードを指定するためのオペコードフィールドおよびオペランドを選択するためのオペランドフィールド（ソース１／デスティネーションおよびソース２）を含む命令フォーマット有する。命令ストリーム内のこのＡＤＤ命令の出現箇所は、特定のオペランドを選択するオペランドフィールド内に特定のコンテンツを有することになる。Ａｄｖａｎｃｅｄ Ｖｅｃｔｏｒ Ｅｘｔｅｎｓｉｏｎｓ（ＡＶＸ）（ＡＶＸ１およびＡＶＸ２）と称されるＳＩＭＤ拡張のセットおよびＶｅｃｔｏｒ Ｅｘｔｅｎｓｉｏｎｓ（ＶＥＸ）コーディングスキームの使用が、リリースおよび／または公開されている（例えば、２０１８年１１月のインテル（登録商標）６４およびＩＡ‐３２アーキテクチャソフトウェア開発者のマニュアルを参照、および、２０１８年１０月のインテル（登録商標）アーキテクチャ命令セット拡張プログラミングリファレンスを参照）。
［例示的な命令フォーマット］
【０１０５】
本明細書で説明される命令の実施形態は、異なるフォーマットで具現化されてよい。追加で、以下に、例示的システム、アーキテクチャおよびパイプラインについて説明する。命令の実施形態は、このようなシステム、アーキテクチャおよびパイプライン上で実行されてよいが、説明されたものに限定はされない。
［汎用ベクトル向け命令フォーマット］
【０１０６】
ベクトル向け命令フォーマットとは、ベクトル命令に適した命令フォーマットのことである（例えば、ベクトルオペレーションに特有の特定のフィールドが存在する）。ベクトル向け命令フォーマットにより、ベクトル演算およびスカラ演算の両方がサポートされる実施形態について説明されているが、代替的な実施形態は、ベクトル向け命令フォーマットを通して、ベクトルオペレーションのみを用いる。
【０１０７】
図５Ａ～５Ｂは、本開示の実施形態による、汎用ベクトル向け命令フォーマットおよびその命令テンプレートを示すブロック図である。図５Ａは、本開示の実施形態による、汎用ベクトル向け命令フォーマットおよびそのクラスＡ命令テンプレートを示すブロック図である一方、図５Ｂは、本開示の実施形態による、汎用ベクトル向け命令フォーマットおよびそのクラスＢ命令テンプレートを示すブロック図である。具体的に言うと、汎用ベクトル向け命令フォーマット５００には、クラスＡ命令テンプレートおよびクラスＢ命令テンプレートが定義され、それらの両方が、非メモリアクセス５０５命令テンプレートおよ

びメモリアクセス５２０命令テンプレートを含む。ベクトル向け命令フォーマットの文脈における汎用という用語は、いずれの特定の命令セットにも結合されていない命令フォーマットを指す。

【０１０８】

本開示は、ベクトル向け命令フォーマットが、３２ビット（４バイト）または６４ビット（８バイト）のデータ要素幅（またはサイズ）（故に、６４バイトベクトルは、１６ダブルワードサイズの要素または代替的に８クワッドワードサイズの要素で構成される）を持つ６４バイトのベクトルオペランド長（またはサイズ）、１６ビット（２バイト）または８ビット（１バイト）のデータ要素幅（またはサイズ）を持つ６４バイトのベクトルオペランド長（またはサイズ）、３２ビット（４バイト）、６４ビット（８バイト）、１６ビット（２バイト）または８ビット（１バイト）のデータ要素幅（またはサイズ）を持つ３２バイトベクトルオペランド長（またはサイズ）、および３２ビット（４バイト）、６４ビット（８バイト）、１６ビット（２バイト）、または８ビット（１バイト）のデータ要素幅（またはサイズ）を持つ１６バイトのベクトルオペランド長（またはサイズ）をサポートする実施形態について説明される一方で、代替的な実施形態は、さらに多くの、さらに少ない、または異なるデータ要素幅（例えば、１２８ビット（１６バイト）データ要素幅）を持つ、さらに多くの、さらに少ない、および／または異なるベクトルオペランドサイズ（例えば、２５６バイトベクトルオペランド）をサポートしてよい。

【０１０９】

図５Ａ中のクラスＡ命令テンプレートは、１）非メモリアクセス５０５命令テンプレート内に示される、非メモリアクセス、フルラウンド制御タイプオペレーション５１０命令テンプレートと、非メモリアクセス、データ変換タイプオペレーション５１５命令テンプレートと、並びに２）メモリアクセス５２０命令テンプレート内に示されるメモリアクセス、一時的５２５命令テンプレートと、メモリアクセス非一時的５３０命令テンプレートとを含む。図５Ｂ中のクラスＢ命令テンプレートは、１）非メモリアクセス５０５命令テンプレート内に示される、非メモリアクセス、書き込みマスク制御、パーシャルラウンド制御タイプ５１２命令テンプレートと、非メモリアクセス、書き込みマスク制御、ｖサイズタイプオペレーション５１７命令テンプレートと、並びに２）メモリアクセス５２０命令テンプレート内に示されるメモリアクセス、書き込みマスク制御５２７命令テンプレートを含む。

【０１１０】

汎用ベクトル向け命令フォーマット５００は、以下に示されるフィールドを図５Ａ～５Ｂに示される順序で含む。

【０１１１】

フォーマットフィールド５４０：このフィールド内の特定の値（命令フォーマット識別子の値）は、ベクトル向け命令フォーマットを一意に識別し、故に、命令ストリーム内のベクトル向け命令フォーマット内の命令の出現を一意に識別する。そのため、このフィールドは、汎用ベクトル向け命令フォーマットのみを有する命令セットには不要であるという意味においてオプションである。

【０１１２】

基本オペレーションフィールド５４２：その内容により、異なる基本オペレーションを区別する。

【０１１３】

レジスタインデックスフィールド５４４：その内容は、ソースオペランドおよびデスティネーションオペランドがレジスタ内にあるか、メモリ内にあるかに関わらず、それらの位置を直接に、または、アドレス生成を通して指定する。これらは、ＰｘＱ（例えば、３２×５１２、１６×１２８、３２×１０２４、６４×１０２４）レジスタファイルからＮ個のレジスタを選択するための、十分な数のビットを含む。一実施形態において、Ｎは最大で３つのソースレジスタおよび１つのデスティネーションレジスタであってよい一方、代替的な実施形態は、それより多いまたは少ない数のソースおよびデスティネーションレ

ジスタをサポートしてよい（例えば、最大２つのソースをサポートしてよく、この場合、これらソースのうちの１つはデスティネーションとしても動作する。最大３つのソースをサポートしてよく、この場合、これらのソースのうちの１つはデスティネーションとして動作する。最大２つのソースと１つのデスティネーションをサポートしてよい）。

【０１１４】

修飾子フィールド５４６：その内容により、汎用ベクトル命令フォーマットにおけるメモリアクセスを指定する命令の出現を、メモリアクセスを指定しない命令から区別する。すなわち、非メモリアクセス５０５命令テンプレートと、メモリアクセス５２０命令テンプレートとの間を区別する。メモリアクセスオペレーションは、メモリ階層への読み取りおよび／または書き込みを行う（場合により、レジスタ内の値を用いて、ソースアドレスおよび／またはデスティネーションアドレスを指定する）一方、非メモリアクセスオペレーションはこれを行わない（例えば、ソースおよびデスティネーションはレジスタである）。一実施形態において、このフィールドはまた、メモリアドレス計算を実行するために３つの異なる方法から選択する一方、代替的な実施形態は、メモリアドレス計算を実行するために、より多い、より少ない、または異なる方法をサポートしてよい。

【０１１５】

拡張オペレーションフィールド５５０：その内容で、基本オペレーションに加え、様々な異なるオペレーションのうちのどれが実行されるかを区別する。このフィールドはコンテキスト固有である。本開示の一実施形態において、このフィールドは、クラスフィールド５６８、アルファフィールド５５２およびベータフィールド５５４に分割される。拡張オペレーションフィールド５５０は、共通のオペレーションのグループが、２、３または４個の命令ではなく、単一の命令内で実行されることを可能にする。

【０１１６】

スケールフィールド５６０：その内容は、メモリアドレス生成のための（例えば、2^{scale}＊インデックス＋基本を用いるアドレス生成のための）インデックスフィールドの内容のスケーリングを可能にする。

【０１１７】

変位フィールド５６２Ａ：その内容は、メモリアドレス生成（例えば、2^{scale}＊インデックス＋基本＋変位を用いるアドレス生成のための）の一部として用いられる。

【０１１８】

変位係数フィールド５６２Ｂ（変位係数フィールド５６２Ｂの直接上に変位フィールド５６２Ａが併置されていることは、一方または他方が用いられることを示すことに留意されたい）：その内容は、アドレス生成の一部として用いられ、それは、メモリアクセス（Ｎ）のサイズによりスケーリングされるべき変位係数を指定する。ここで、Ｎは、メモリアクセスにおけるバイト数である（例えば、2^{scale}＊インデックス＋基本＋スケーリングされた変位を用いるアドレス生成のための）。冗長下位ビットは無視され、従って、変位係数フィールドの内容は、有効アドレスを計算する際に用いられる最終的な変位を生成すべく、メモリオペランドの合計サイズ（Ｎ）で乗じられる。Ｎの値は、ランタイム時にプロセッサハードウェアにより、フルオペコードフィールド５７４（本明細書で後述する）およびデータ操作フィールド５５４Ｃに基づき決定される。変位フィールド５６２Ａおよび変位係数フィールド５６２Ｂは、それらが、非メモリアクセス５０５命令テンプレートで用いられない、および／または異なる実施形態は、これらの２つのうちの一方のみを実装してよい、またはいずれも実装しなくてよいという意味においてオプションである。

【０１１９】

データ要素幅フィールド５６４：その内容により、複数のデータ要素幅のどれが用いられるかを区別する（いくつかの実施形態においては、すべての命令について、他の実施形態においては、命令のうちの一部のみについて）。このフィールドは、１つのデータ要素幅のみがサポートされる、および／または、オペコードのいくつかの態様を用いて、データ要素幅がサポートされる場合は、このフィールドが不要という意味においてオプション

である。
【０１２０】
書き込みマスクフィールド５７０：その内容により、データ要素位置ベースごとに、デスティネーションベクトルオペランド内のそのデータ要素位置が、基本オペレーションおよび拡張オペレーションの結果に反映されるか否かを制御する。クラスＡ命令テンプレートは、マージング書き込みマスクをサポートする一方で、クラスＢ命令テンプレートは、マージング書き込みマスクおよびゼロ化書き込みマスクの両方をサポートする。マージングの場合に、ベクトルマスクは、デスティネーションの任意の要素セットが、オペレーションの実行中に更新されないように保護されることを可能にする（基本オペレーションおよび拡張オペレーションで指定）。他の一実施形態においては、対応するマスクビットが０を有する場合、デスティネーションの各要素の古い値を保持する。対照的に、ゼロ化の場合、ベクトルマスクは、デスティネーションの任意の要素セットが、オペレーションの実行中にゼロ化されることを可能にする（基本オペレーションおよび拡張オペレーションにおいて指定される）。一実施形態においては、対応するマスクビットが０値を有する場合、デスティネーションの要素は０に設定される。この機能のサブセットは、実行されているオペレーションのベクトル長（すなわち、修正される要素の第１のものから最後のものまでのスパン）を制御する能力である。しかしながら、修正される要素は連続的であることは必要ではない。故に、書き込みマスクフィールド５７０は、ロード、ストア、算術、論理等を含む部分的なベクトルオペレーションを可能にする。書き込みマスクフィールドの５７０の内容が、用いられるべき書き込みマスクを含む複数の書き込みマスクレジスタのうちの１つを選択する本開示の実施形態が説明される（故に、書き込みマスクフィールド５７０の内容は、実行されるべきマスクを間接的に識別する）一方で、代替的な実施形態は、代替的にまたは追加的に、書き込みマスクフィールド５７０の内容が、実行されるべきマスクを直接的に指定することを可能にする。
【０１２１】
即値フィールド５７２：その内容は、即値の指定を可能にする。このフィールドは、即値をサポートしない汎用ベクトルフォーマットの実装においては、これは存在しない、および、即値を用いない命令内にはこれは存在しないという意味においてオプションである。
【０１２２】
クラスフィールド５６８：その内容により、異なるクラスの命令間を区別する。図５Ａ～図５Ｂを参照すると、このフィールドの内容は、クラスＡ命令とクラスＢ命令との間で選択する。図５Ａ～図５Ｂ中、角の丸められた四角は、特定の値がフィールド内に存在することを示すために用いられる（例えば、図５Ａ～図５Ｂ中のそれぞれのクラスフィールド５６８に対するクラスＡ　５６８ＡおよびクラスＢ　５６８Ｂ）。
［クラスＡの命令テンプレート］
【０１２３】
クラスＡの非メモリアクセス５０５命令テンプレートの場合、アルファフィールド５５２はＲＳフィールド５５２Ａとして解釈され、その内容により、異なる拡張オペレーションタイプのうちのどれが実行されるべき（例えば、ラウンド５５２Ａ．１およびデータ変換５５２Ａ．２が、非メモリアクセス、ラウンドタイプオペレーション５１０および非メモリアクセス、データ変換タイプオペレーション５１５命令テンプレートのそれぞれに対し指定される）かを区別する一方、ベータフィールド５５４により、指定されたタイプのオペレーションのうちのいずれが実行されるべきかを区別する。非メモリアクセス５０５命令テンプレートでは、スケールフィールド５６０、変位フィールド５６２Ａおよび変位スケールフィールド５６２Ｂは存在しない。
［非メモリアクセス命令テンプレート‐フルラウンド制御タイプオペレーション］
【０１２４】
非メモリアクセスフルラウンド制御タイプオペレーション５１０命令テンプレートでは、ベータフィールド５５４がラウンド制御フィールド５５４Ａとして解釈され、その内容

が静的ラウンディングを提供する。本開示の説明される実施形態において、ラウンド制御フィールド５５４Ａは、すべての浮動小数点例外を抑制（ＳＡＥ）フィールド５５６およびラウンドオペレーション制御フィールド５５８を含む一方、代替的な実施形態は、これらのコンセプトの両方の同一フィールドへのエンコードをサポートしてよく、または、これらのコンセプト／フィールドのうち一方または他方のみを有することをサポートしてよい（例えば、ラウンドオペレーション制御フィールド５５８のみを有してよい）。

【０１２５】

ＳＡＥフィールド５５６：その内容により、例外イベント報告を無効にするか否かを区別する。ＳＡＥフィールド５５６の内容が、抑制が有効になっていることを示す場合、特定の命令は浮動小数点例外フラグを報告せず、浮動小数点例外ハンドラを掲げない。

【０１２６】

ラウンドオペレーション制御フィールド５５８：その内容により、一群のラウンドオペレーション（例えば、切り上げ、切り下げ、ゼロへの丸めおよび最近似値への丸め）のうちのいずれが実行されるかを区別する。故に、ラウンドオペレーション制御フィールド５５８は、命令ベースごとに、ラウンドモードの変更を可能にする。プロセッサがラウンドモードを指定するための制御レジスタを含んでいる本開示の一実施形態においては、ラウンドオペレーション制御フィールド５５０の内容は、そのレジスタ値を上書きする。

［非メモリアクセス命令テンプレート‐データ変換タイプオペレーション］

【０１２７】

非メモリアクセスデータ変換タイプオペレーション５１５命令テンプレートでは、ベータフィールド５５４は、データ変換フィールド５５４Ｂとして解釈され、データ変換フィールド５５４Ｂの内容により、複数のデータ変換（例えば、データ変換なし、スウィズル、ブロードキャスト）のうちのいずれが実行されるかを区別する。

【０１２８】

クラスＡのメモリアクセス５２０命令テンプレートの場合、アルファフィールド５５２はエビクションヒントフィールド５５２Ｂとして解釈され、エビクションヒントフィールド５５２Ｂの内容により、どのエビクションヒントが用いられるかを区別する（図５Ａでは、一時５５２Ｂ．１および非一時５５２Ｂ．２がメモリアクセス、一時５２５命令テンプレートおよびメモリアクセス、非一時５３０命令テンプレートにそれぞれ指定される）一方、ベータフィールド５５４はデータ操作フィールド５５４Ｃとして解釈され、データ操作フィールド５５４Ｃの内容により、複数のデータ操作オペレーション（プリミティブとしても知られる）（例えば、操作なし、ブロードキャスト、ソースのアップコンバージョン、デスティネーションのダウンコンバージョン）のうちのいずれが実行されるかを区別する。メモリアクセス５２０命令テンプレートは、スケールフィールド５６０を含み、オプションで、変位フィールド５６２Ａまたは変位スケールフィールド５６２Ｂを含む。

【０１２９】

ベクトルメモリ命令は、変換サポートにより、メモリからのベクトルロードおよびメモリへのベクトルストアを実行する。通常のベクトル命令の場合と同様、ベクトルメモリ命令は、データ要素全体でデータをメモリから／メモリへ転送し、実際に転送される要素は、書き込みマスクとして選択されるベクトルマスクの内容によって記述されている。

［メモリアクセス命令テンプレート‐一時］

【０１３０】

一時的データとは、キャッシングによる恩恵を得るのに十分早く再使用される可能性が高いデータである。しかしながら、これはヒントであり、異なるプロセッサは、ヒントを完全に無視することを含め、それを異なる方法で実装してよい。

［メモリアクセス命令テンプレート‐非一時］

【０１３１】

非一時的データとは、第１のレベルのキャッシュ内にキャッシュすることによる恩恵を得るのに十分早く再使用される可能性が低いデータであり、これにはエビクションの優先度が与えられるべきである。しかしながら、これはヒントであり、異なるプロセッサは、

ヒントを完全に無視することを含め、それを異なる方法で実装してよい。
［クラスＢの命令テンプレート］
【０１３２】
クラスＢの命令テンプレートの場合、アルファフィールド５５２は、書き込みマスク制御（Ｚ）フィールド５５２Ｃとして解釈され、書き込みマスク制御（Ｚ）フィールド５５２Ｃの内容により、書き込みマスクフィールド５７０により制御される書き込みマスクが、マージングであるべきかゼロ化であるべきかを区別する。
【０１３３】
クラスＢの非メモリアクセス５０５命令テンプレートの場合、ベータフィールド５５４の一部はＲＬフィールド５５７Ａとして解釈され、その内容により、異なる拡張オペレーションタイプのうちのいずれが実行されるかを区別する（例えば、ラウンド５５７Ａ．１およびベクトル長ＶＳＩＺＥ５５７Ａ．２が、非メモリアクセス、書き込みマスク制御、パーシャルラウンド制御タイプオペレーション５１２命令テンプレートおよび非メモリアクセス書き込みマスク制御ＶＳＩＺＥタイプオペレーション５１７命令テンプレートにそれぞれ指定される）一方、ベータフィールド５５４の残部により、指定されたタイプのオペレーションのうちいずれが実行されるかを区別する。非メモリアクセス５０５命令テンプレートには、スケールフィールド５６０、変位フィールド５６２Ａ、および変位スケールフィールド５６２Ｂは存在しない。
【０１３４】
非メモリアクセス、書き込みマスク制御、パーシャルラウンド制御タイプオペレーション５１０命令テンプレートでは、ベータフィールド５５４の残部は、ラウンドオペレーションフィールド５５９Ａとして解釈され、例外イベント報告が無効にされる（特定の命令は、あらゆる種類の浮動小数点例外フラグを報告せず、浮動小数点例外ハンドラを掲げない）。
【０１３５】
ラウンドオペレーション制御フィールド５５９Ａ：ラウンドオペレーション制御フィールド５５８と全く同様、その内容により、一群のラウンドオペレーションのうちのいずれが実行されるかを区別する（例えば、切り上げ、切り下げ、ゼロへの丸めおよび最近似値への丸め）。故に、ラウンドオペレーション制御フィールド５５９Ａは、命令ベースごとに、ラウンドモードの変更を可能にする。プロセッサがラウンドモードを指定するための制御レジスタを含んでいる本開示の一実施形態においては、ラウンドオペレーション制御フィールド５５０の内容は、そのレジスタ値を上書きする。
【０１３６】
非メモリアクセス書き込みマスク制御ＶＳＩＺＥタイプオペレーション５１７命令テンプレートでは、ベータフィールド５５４の残部はベクトル長フィールド５５９Ｂとして解釈され、その内容により、複数のデータベクトル長（例えば、１２８、２５６または５１２バイト）のうちのいずれが実行されるかを区別する。
【０１３７】
クラスＢのメモリアクセス５２０命令テンプレートの場合、ベータフィールド５５４の一部はブロードキャストフィールド５５７Ｂとして解釈され、その内容により、ブロードキャストタイプデータ操作オペレーションが実行されるか否かを区別する一方、ベータフィールド５５４の残部はベクトル長フィールド５５９Ｂとして解釈される。メモリアクセス５２０命令テンプレートは、スケールフィールド５６０、およびオプションで変位フィールド５６２Ａまたは変位スケールフィールド５６２Ｂを含む。
【０１３８】
汎用ベクトル向け命令フォーマット５００に関しては、フルオペコードフィールド５７４は、フォーマットフィールド５４０、基本オペレーションフィールド５４２およびデータ要素幅フィールド５６４を含むように図示されている。一実施形態はフルオペコードフィールド５７４がすべてのこれらのフィールドを含むように図示されているが、これらのすべてをサポートしていない実施形態においては、フルオペコードフィールド５７４は、

これらのフィールドの全部より少ない数を含む。フルオペコードフィールド５７４は、オペレーションコード（オペコード）を提供する。

【０１３９】

拡張オペレーションフィールド５５０、データ要素幅フィールド５６４、および書き込みマスクフィールド５７０は、汎用ベクトル向け命令フォーマットにおいて、これらの機能が命令ベースごとに指定されることを可能にする。

【０１４０】

書き込みマスクフィールドおよびデータ要素幅フィールドを組み合わせると、それらが、異なるデータ要素幅に基づき、マスクが適用されることを可能にするという意味においてタイプ別の命令を形成する。

【０１４１】

クラスＡおよびクラスＢ内で見つかる様々な命令テンプレートは、異な状況において有益である。本開示のいくつかの実施形態において、異なるプロセッサまたはプロセッサ内の異なるコアは、クラスＡのみ、クラスＢのみ、またはこれら両方のクラスをサポートしてよい。例えば、汎用コンピューティング向け高性能汎用アウトオブオーダコアはクラスＢのみをサポートしてよく、主にグラフィックおよび／または科学（スループット）コンピューティング向けコアは、クラスＡのみをサポートしてよく、およびこれらの両方向けのコアはこれらの両方をサポートしてよい（もちろん、両方のクラスからのすべてのテンプレートおよび命令ではなく、両方のクラスからのテンプレートおよび命令のいくつかの混在を有するコアは、本開示の範囲内に属する）。また、単一のプロセッサは複数のコアを含んでよく、これらのすべてが、同一クラスをサポートするか、または、異なるコアは異なるクラスをサポートする。例えば、別個のグラフィックコアおよび汎用コアを持つプロセッサでは、主にグラフィックコンピューティングおよび／または科学コンピューティング向けのグラフィックコアのうちの１つがクラスＡのみをサポートしてよい一方、汎用コアのうちの１または複数は、クラスＢのみをサポートする汎用コンピューティング向けアウトオブオーダ実行およびレジスタリネーミングを持つ高性能汎用コアであってよい。別個のグラフィックコアを有していない別のプロセッサは、クラスＡおよびクラスＢの両方をサポートする１または複数の汎用インオーダまたはアウトオブオーダコアを含んでよい。もちろん、本開示の異なる実施形態においては、１つのクラスに属する諸機能が、他のクラスに実装されてもよい。高水準言語で記述されたプログラムは、様々な異なる実行可能な形態に編成される（例えば、実行時コンパイルまたは静的コンパイル）。このような形態としては、１）実行のためのターゲットプロセッサによりサポートされるクラスの命令のみを有する形態、または２）すべてのクラスの異なる組み合わせを用いて記述された代替ルーチンを有し、且つ、コードを現在実行中のプロセッサによりサポートされる命令に基づき、実行すべきルーチンを選択する制御フローコードを有する形態が含まれる。

［例示的な特定ベクトル向け命令フォーマット］

【０１４２】

図６Ａ～図６Ｄは、本開示の実施形態による、例示的な特定ベクトル向け命令フォーマットを示すブロック図である。図６Ａ～図６Ｄは、特定ベクトル向け命令フォーマット６００を示す。特定ベクトル向け命令フォーマット６００は、場所、サイズ、解釈およびフィールド順序に加え、これらのフィールドの一部の値を指定するという意味において特定的である。特定ベクトル向け命令フォーマット６００を用いて、ｘ８６命令セットを拡張してよく、故に、フィールドの一部は、既存のｘ８６命令セットおよびその拡張（例えば、ＡＶＸ）で用いられるものと同様または同一である。このフォーマットは、拡張機能付きの既存のｘ８６命令セットのプレフィックスエンコーディングフィールド、リアルオペコードバイトフィールド、ＭＯＤ　Ｒ／Ｍフィールド、ＳＩＢフィールド、変位フィールドおよび即値フィールドと整合している。図５Ａ～図５Ｂのフィールドが図６Ａ～図６Ｄのどのフィールドにマッピングされるかが図示されている。

【０１４３】

本開示の実施形態は、説明目的で、汎用ベクトル向け命令フォーマット５００の文脈の

中で、特定ベクトル向け命令フォーマット６００に関し説明されているが、本開示は、特許請求される場合を除き、特定ベクトル向け命令フォーマット６００に限定はされないことを理解されたい。例えば、汎用ベクトル向け命令フォーマット５００は様々なフィールドの様々な可能性のあるサイズを想定している一方、特定ベクトル向け命令フォーマット６００は、特定のサイズのフィールドを有するように図示されている。特定の例として、データ要素幅フィールド５６４は、特定ベクトル向け命令フォーマット６００では１ビットフィールドとして示されている一方、本開示はこのようには限定されない（すなわち、汎用ベクトル向け命令フォーマット５００は、データ要素幅フィールド５６４の他のサイズを想定している）。

【０１４４】

汎用ベクトル向け命令フォーマット５００は、以下に示されるフィールドを図６Ａに示される順序で含む。

【０１４５】

ＥＶＥＸプレフィックス（バイト０－３）６０２：４バイト形式でエンコードされる。

【０１４６】

フォーマットフィールド５４０（ＥＶＥＸバイト０、ビット［７：０］）：第１のバイト（ＥＶＥＸバイト０）はフォーマットフィールド５４０であり、０ｘ６２（本開示の一実施形態において、ベクトル向け命令フォーマットを区別するために用いられる一意の値）を含む。

【０１４７】

第２～第４バイト（ＥＶＥＸバイト１－３）は、特定の機能を提供する複数のビットフィールドを含む。

【０１４８】

ＲＥＸフィールド６０５（ＥＶＥＸバイト１、ビット［７‐５］）：ＥＶＥＸ．Ｒビットフィールド（ＥＶＥＸバイト１、ビット［７］‐Ｒ）、ＥＶＥＸ．Ｘビットフィールド（ＥＶＥＸバイト１、ビット［６］‐Ｘ）および５５７ＢＥＸバイト１、ビット［５］‐Ｂで構成される。ＥＶＥＸ．Ｒ、ＥＶＥＸ．ＸおよびＥＶＥＸ．Ｂビットフィールドは、対応するＶＥＸビットフィールドと同一の機能を提供し、１の補数形式を用いてエンコードされる。すなわちＺＭＭ０は１１１１Ｂとしてエンコードされ、ＺＭＭ１５は０００Ｂとしてエンコードされる。命令の他のフィールドは、本技術分野で周知のように、レジスタインデックスの下位３ビット（ｒｒｒ、ｘｘｘおよびｂｂｂ）をエンコードする。そのため、ＥＶＥＸ．Ｒ、ＥＶＥＸ．ＸおよびＥＶＥＸ．Ｂを追加することで、Ｒｒｒｒ、ＸｘｘｘおよびＢｂｂｂが形成されてよい。

【０１４９】

ＲＥＸ'フィールド５１０：これは、ＲＥＸ'フィールド５１０の第１の部分であり、拡張３２レジスタセットの上位１６または下位１６のいずれかをエンコードするために用いられるＥＶＥＸ．Ｒ'ビットフィールド（ＥＶＥＸバイト１、ビット［４］‐Ｒ'）である。本開示の一実施形態において、このビットは、以下に示す他のものと共に、ＢＯＵＮＤ命令から区別（周知のｘ８６　３２ビットモードで）するためにビット反転形式で格納され、ＢＯＵＮＤ命令のリアルオペコードバイトは６２であるが、ＭＯＤ　Ｒ／Ｍフィールド（後述）では、ＭＯＤフィールド内の値１１を受け入れない。本開示の代替的な実施形態は、これおよび以下に示される他のビットを反転形式でストアしない。値１が用いられて、下位の１６個のレジスタをエンコードする。換言すると、ＥＶＥＸ．Ｒ'、ＥＶＥＸ．Ｒおよび他のフィールドからの他のＲＲＲを組み合わせて、Ｒ'Ｒｒｒｒが形成される。

【０１５０】

オペコードマップフィールド６１５（ＥＶＥＸバイト１、ビット［３：０］‐ｍｍｍｍ）：その内容により、暗示された先頭オペコードバイト（０Ｆ、０Ｆ　３８または０Ｆ３）をエンコードする。

【０１５１】

データ要素幅フィールド５６４（ＥＶＥＸバイト２、ビット［７］－Ｗ）：表記ＥＶＥＸ．Ｗにより表される。ＥＶＥＸ．Ｗは、データ型の粒度（サイズ）を定義するために用いられる（３２ビットデータ要素または６４ビットデータ要素のいずれか）。

【０１５２】

ＥＶＥＸ．ｖｖｖｖ６２０（ＥＶＥＸバイト２、ビット［６：３］－ｖｖｖｖ）：ＥＶＥＸ．ｖｖｖｖの役割は、以下を含んでよい。１）ＥＶＥＸ．ｖｖｖｖは、反転（１の補数）形式で指定された第１のソースレジスタオペランドをエンコードし、２つ以上のソースオペランドを有する命令について有効である。２）ＥＶＥＸ．ｖｖｖｖは、特定のベクトルシフトについて１の補数形式で指定されたデスティネーションレジスタオペランドをエンコードする。または、３）ＥＶＥＸ．ｖｖｖｖは、いずれのオペランドもエンコードせず、当該フィールドは予約されており、１１１１ｂを含むはずである。故に、ＥＶＥＸ．ｖｖｖｖフィールド６２０は、反転（１の補数）形式で格納された第１のソースレジスタ指定子の４つの下位ビットをエンコードする。命令に応じて、追加の異なるＥＶＥＸビットフィールドが使用され、指定子サイズを３２個のレジスタに拡張する。

【０１５３】

ＥＶＥＸ．Ｕ５６８クラスフィールド（ＥＶＥＸバイト２、ビット［２］－Ｕ）：ＥＶＥＸ．Ｕ＝０の場合、これは、クラスＡまたはＥＶＥＸ．Ｕ０を示す。ＥＶＥＸ．Ｕ＝１の場合、これは、クラスＢまたはＥＶＥＸ．Ｕ１を示す。

【０１５４】

プレフィックスエンコーディングフィールド６２５（ＥＶＥＸバイト２、ビット［１：０］－ｐｐ）：基本オペレーションフィールドに対し追加ビットを提供する。ＥＶＥＸプレフィックスフォーマットのレガシＳＳＥ命令へのサポートを提供することに加え、これはまたＳＩＭＤプレフィックスを圧縮する利益も有する（ＳＩＭＤプレフィックスを表すために１バイトを必要とせず、ＥＶＥＸプレフィックスは２ビットのみを必要とする）。一実施形態において、レガシフォーマットおよびＥＶＥＸプレフィックスフォーマットの両方でＳＩＭＤプレフィックス（６６Ｈ、Ｆ２Ｈ、Ｆ３Ｈ）を使用するレガシＳＳＥ命令をサポートするために、これらのレガシＳＩＭＤプレフィックスは、ＳＩＭＤプレフィックスエンコーディングフィールドにエンコードされ、ランタイム時にデコード回路のＰＬＡに提供される前に、レガシＳＩＭＤプレフィックスに拡張される（そのため、ＰＬＡは、修正することなく、これらのレガシ命令のレガシフォーマットとＥＶＥＸフォーマットの両方を実行できる）。より新しい命令は、ＥＶＥＸプレフィックスエンコーディングフィールド内の内容を直接オペコード拡張として使用できるが、特定の実施形態は整合性のために同様の形式で拡張するが、これらのレガシＳＩＭＤプレフィックスにより指定される異なる手段を可能にする。代替的な実施形態は、２ビットＳＩＭＤプレフィックスエンコーディングをサポートし、従って、拡張を必要としないようにＰＬＡを再設計することができる。

【０１５５】

アルファフィールド５５２（ＥＶＥＸバイト３、ビット［７］－ＥＨ：ＥＶＥＸ．ＥＨ、ＥＶＥＸ．ｒｓ、ＥＶＥＸ．ＲＬ、ＥＶＥＸ．書き込みマスク制御およびＥＶＥＸ．Ｎとしても知られ、αでも示される）：上記したように、このフィールドはコンテキスト固有である。

【０１５６】

ベータフィールド５５４（ＥＶＥＸバイト３、ビット［６：４］－ＳＳＳ、ＥＶＥＸｓ$_{2-0}$、ＥＶＥＸｒ$_{2-0}$、ＥＶＥＸ．ｒｒ１、ＥＶＥＸ．ＬＬ０、ＥＶＥＸ．ＬＬＢとしても知られ、βββでも示される）。上記したように、このフィールドはコンテキスト固有である。

【０１５７】

ＲＥＸ'フィールド５１０：これは、ＲＥＸ'フィールドの残部であり、拡張３２レジスタセットの上位１６個または下位１６個のいずれかをエンコードするために用いられ得るＥＶＥＸ．Ｖ'ビットフィールド（ＥＶＥＸバイト３、ビット［３］－Ｖ'）である。この

ビットは、ビット反転フォーマットで格納される。下位の１６個のレジスタをエンコードするために値１が用いられる。換言すると、ＥＶＥＸ．Ｖ'、ＥＶＥＸ．ｖｖｖｖを組み合わせて、Ｖ'ＶＶＶＶが形成される。

【０１５８】

書き込みマスクフィールド５７０（ＥＶＥＸバイト３、ビット［２：０］‐ｋｋｋ）：その内容は、上記の通り、書き込みマスクレジスタのレジスタのインデックスを指定する。本開示の一実施形態において、特定の値ＥＶＥＸ．ｋｋｋ＝０００は、特定の命令に書き込みマスクが用いられないことを暗示する特別な挙動を有する（これは、すべて１にハードワイヤードされる書き込みマスクまたはマスキングハードウェアをバイパスするハードウェアの使用を含み、様々な方法で実装されてよい）。

【０１５９】

リアルオペコードフィールド６３０（バイト４）は、オペコードバイトとしても知られる。オペコードの一部は、このフィールドで指定される。

【０１６０】

ＭＯＤ　Ｒ／Ｍフィールド６４０（バイト５）は、ＭＯＤフィールド６４２、Ｒｅｇフィールド６４４およびＲ／Ｍフィールド６４６を含む。上記の通り、ＭＯＤフィールド６４２の内容により、メモリアクセスオペレーションと非メモリアクセスオペレーションとの間を区別する。Ｒｅｇフィールド６４４の役割は、２つの状況に要約できる。デスティネーションレジスタオペランドまたはソースレジスタオペランドのいずれかをエンコードする、または、オペコード拡張として扱われ、命令オペランドをエンコードするために用いられない。Ｒ／Ｍフィールド６４６の役割は、メモリアドレスを参照する命令オペランドをエンコードすること、または、デスティネーションレジスタオペランドまたはソースレジスタオペランドのいずれかをエンコードすることを含んでよい。

【０１６１】

スケール、インデックス、基本（ＳＩＢ）バイト（バイト６）：上記の通り、スケールフィールド５５０の内容は、メモリアドレス生成に用いられる。ＳＩＢ．ｘｘｘ６５４およびＳＩＢ．ｂｂｂ６５６：これらのフィールドの内容は、レジスタインデックスＸｘｘｘおよびＢｂｂｂに関して記載済みである。

【０１６２】

変位フィールド５６２Ａ（バイト７－１０）：ＭＯＤフィールド６４２が１０を含む場合、バイト７‐１０が変位フィールド５６２Ａであり、変位フィールド５６２Ａはレガシ３２ビットの変位（ｄｉｓｐ３２）と同じように動作し、バイト粒度で動作する。

【０１６３】

変位係数フィールド５６２Ｂ（バイト７）：ＭＯＤフィールド６４２が０１を含む場合、バイト７が変位係数フィールド５６２Ｂである。このフィールドの位置は、レガシｘ８６命令セット８ビット変位（ｄｉｓｐ８）の位置と同一であり、バイト粒度で動作する。ｄｉｓｐ８は符号拡張されるので、－１２８と１２７バイトのオフセット間でのみアドレス指定できる。６４バイトのキャッシュラインの観点では、ｄｉｓｐ８は４つの現実的に有用な値－１２８、－６４、０、および６４にのみ設定できる８ビットを使用し、より大きな範囲がしばしば必要とされるので、ｄｉｓｐ３２が使用されるが、ｄｉｓｐ３２は４バイトを必要とする。ｄｉｓｐ８およびｄｉｓｐ３２と対照的に、変位係数フィールド５６２Ｂはｄｉｓｐ８を再解釈したものであり、変位係数フィールド５６２Ｂを使用するとき、実際の変位は、メモリオペランドアクセスのサイズ（Ｎ）で乗算された変位係数フィールドの内容によって決定される。このタイプの変位は、ｄｉｓｐ８＊Ｎと称される。これは、平均命令長を低減する（変位に使用される単一バイトであるが、はるかに広い範囲を有する）。このような圧縮された変位は、実効変位がメモリアクセスの粒度の倍数であるという仮定に基づいており、従って、アドレスオフセットの冗長な下位ビットはエンコードされる必要がない。換言すると、変位係数フィールド５６２Ｂは、レガシｘ８６命令セットの８ビット変位を置き換える。故に、変位係数フィールド５６２Ｂは、ｄｉｓｐ８がｄｉｓｐ８＊Ｎにオーバーロードされる点のみを除き、ｘ８６命令セットの８ビット変

位と同じ方法でエンコードされる（そのため、ＭｏｄＲＭ／ＳＩＢエンコーディング規則における変更はない）。換言すると、エンコーディング規則またはエンコーディング長に変更はないが、ハードウェア（バイト単位のアドレスオフセットを取得するために、メモリオペランドのサイズによって変位をスケーリングする必要がある）による変位値の解釈のみに変更がある。即値フィールド５７２は、上記の通り動作する。

［フルオペコードフィールド］

【０１６４】

図６Ｂは、本開示の一実施形態によるフルオペコードフィールド５７４を構成する特定ベクトル向け命令フォーマット６００のフィールドを示すブロック図である。具体的に言うと、フルオペコードフィールド５７４は、フォーマットフィールド５４０、基本オペレーションフィールド５４２、およびデータ要素幅（Ｗ）フィールド５６４を含む。基本オペレーションフィールド５４２は、プレフィックスエンコーディングフィールド６２５、オペコードマップフィールド６１５およびリアルオペコードフィールド６３０を含む。

［レジスタインデックスフィールド］

【０１６５】

図６Ｃは、本開示の一実施形態による、レジスタインデックスフィールド５４４を構成する特定ベクトル向け命令フォーマット６００のフィールドを示すブロック図である。具体的に言うと、レジスタインデックスフィールド５４４は、ＲＥＸフィールド６０５、ＲＥＸ'フィールド６１０、ＭＯＤＲ／Ｍ．ｒｅｇフィールド６４４、ＭＯＤＲ／Ｍ．ｒ／ｍフィールド６４６、ＶＶＶＶフィールド６２０、ｘｘｘフィールド６５４およびｂｂｂフィールド６５６を含む。

［拡張オペレーションフィールド］

【０１６６】

図６Ｄは、本開示の一実施形態による、拡張オペレーションフィールド５５０を構成する特定ベクトル向け命令フォーマット６００のフィールドを示すブロック図である。クラス（Ｕ）フィールド５６８が０を含む場合、それは、ＥＶＥＸ．Ｕ０（クラスＡ　５６８Ａ）を意味し、１を含むとき、それはＥＶＥＸ．Ｕ１（クラスＢ　５６８Ｂ）を意味する。Ｕ＝０であり且つＭＯＤフィールド６４２が（非メモリアクセスオペレーションを意味する）１１を含むとき、アルファフィールド５５２（ＥＶＥＸバイト３、ビット［７］－ＥＨ）はｒｓフィールド５５２Ａとして解釈される。ｒｓフィールド５５２Ａが１（ラウンド５５２Ａ．１）を含むとき、ベータフィールド５５４（ＥＶＥＸバイト３、ビット［６：４］－ＳＳＳ）はラウンド制御フィールド５５４Ａとして解釈される。ラウンド制御フィールド５５４Ａは、１ビットのＳＡＥフィールド５５６および２ビットのラウンドオペレーションフィールド５５８を含む。ｒｓフィールド５５２Ａが０を含む場合（データ変換５５２Ａ．２）、ベータフィールド５５４（ＥＶＥＸバイト３、ビット［６：４］‐ＳＳＳ）は３ビットのデータ変換フィールド５５４Ｂとして解釈される。Ｕ＝０で且つＭＯＤフィールド６４２が００、０１または１０を含む場合（メモリアクセスオペレーションを意味）、アルファフィールド５５２（ＥＶＥＸバイト３、ビット［７］‐ＥＨ）は、エビクションヒント（ＥＨ）フィールド５５２Ｂとして解釈され、ベータフィールド５５４（ＥＶＥＸバイト３、ビット［６：４］‐ＳＳＳ）は３ビットのデータ操作フィールド５５４Ｃとして解釈される。

【０１６７】

Ｕ＝１の場合、アルファフィールド５５２（ＥＶＥＸバイト３、ビット［７］－ＥＨ）は、書き込みマスク制御（Ｚ）フィールド５５２Ｃとして解釈される。Ｕ＝１であり且つＭＯＤフィールド６４２が１１を含む（非メモリアクセスオペレーションを表わす）場合、ベータフィールド５５４の一部（ＥＶＥＸバイト３、ビット［４］－$Ｓ_0$）は、ＲＬフィールド５５７Ａとして解釈される。１を含む場合（ラウンド５５７Ａ．１）、ベータフィールド５５４の残部（ＥＶＥＸバイト３、ビット［６‐５］－$Ｓ_{2-1}$）は、ラウンドオペレーションフィールド５５９Ａとして解釈される一方、ＲＬフィールド５５７Ａが０を含む場合（ＶＳＩＺＥ　５５７．Ａ２）、ベータフィールド５５４の残部（ＥＶＥＸバ

イト３、ビット［６‐５］－S_{2-1}）は、ベクトル長フィールド５５９Ｂ（ＥＶＥＸバイト３、ビット［６‐５］‐L_{1-0}）として解釈される。Ｕ＝１であり且つＭＯＤフィールド６４２が００、０１、または１０を含む（メモリアクセスオペレーションを意味する）場合、ベータフィールド５５４（ＥＶＥＸバイト３、ビット［６：４］－ＳＳＳ）は、ベクトル長フィールド５５９Ｂ（ＥＶＥＸバイト３、ビット［６－５］－L_{1-0}）およびブロードキャストフィールド５５７Ｂ（ＥＶＥＸバイト３、ビット［４］－Ｂ）として解釈される。

［例示的レジスタアーキテクチャ］

【０１６８】

図７は、本開示の一実施形態による、レジスタアーキテクチャ７００のブロック図である。図示された実施形態においては、５１２ビット幅である３２個のベクトルレジスタ７１０が存在し、これらのレジスタは、ｚｍｍ０からｚｍｍ３１と称される。下位１６個のｚｍｍレジスタの下位２５６ビットは、ｙｍｍ０～１６レジスタ上に重なっている。下位１６個のｚｍｍレジスタの下位１２８ビット（ｙｍｍレジスタの下位１２８ビット）は、ｘｍｍ０～１５レジスタ上に重なっている。特定ベクトル向け命令フォーマット６００は、以下の表に示されるように、これらの重なったレジスタファイルを処理する。

【表６】

調整可能なベクトル長	クラス	オペレーション	レジスタ
ベクトル長フィールド559Bを含まない命令テンプレート	A(図5A; U=0)	510, 515, 525, 530	zmmレジスタ(ベクトル長は64バイト)
	B(図5B; U=1)	512	zmmレジスタ(ベクトル長は64バイト)
ベクトル長フィールド559Bを含む命令テンプレート	B(図5B; U=1)	517, 527	ベクトル長フィールド559Bにより、zmm、ymmまたはxmmレジスタ(ベクトル長は64バイト、32バイト、または16バイト)

【０１６９】

換言すると、ベクトル長フィールド５５９Ｂは、最大長と、１または複数の他のより短い長さとの間で選択し、このようなより短い長さの各々は先行する長さの半分の長さであり、ベクトル長フィールド５５９Ｂのない命令テンプレートは、最大ベクトル長を処理する。さらに、一実施形態において、特定ベクトル向け命令フォーマット６００のクラスＢ命令テンプレートは、パックされたまたはスカラの単精度／倍精度浮動小数点データおよびパックされたまたはスカラの整数データに対し処理を行う。スカラ演算とは、ｚｍｍ／ｙｍｍ／ｘｍｍレジスタ内の最下位のデータ要素の位置で実行される演算である。実施形態に応じ、より上位のデータ要素の位置は、命令前と同じに保持されるか、ゼロにされるかのいずれかである。

【０１７０】

書き込みマスクレジスタ７１５：図示された実施形態では、各々が６４ビットサイズの８つの書き込みマスクレジスタ（ｋ０からｋ７）が存在する。代替的な実施形態においては、書き込みマスクレジスタ７１５は１６ビットのサイズである。前述したように、本開示の一実施形態において、ベクトルマスクレジスタｋ０は、書き込みマスクとして用いることはできず、通常ｋ０を示すエンコーディングが書き込みマスクに用いられる場合、ベクトルマスクレジスタは、０ｘＦＦＦＦのハードワイヤされた書き込みマスクを選択することで、その命令に対する書き込みマスキングを実質無効にする。

【０１７１】

汎用レジスタ７２５：図示された実施形態では、メモリオペランドをアドレス指定するための既存のｘ８６アドレス指定モードと共に使用される１６個の６４ビット汎用レジスタが存在する。これらのレジスタは、ＲＡＸ、ＲＢＸ、ＲＣＸ、ＲＤＸ、ＲＢＰ、ＲＳＩ

、ＲＤＩ、ＲＳＰおよびＲ８～Ｒ１５という名称で参照される。
【０１７２】
示される実施形態において、ＭＭＸパックされた整数フラットレジスタファイル７５０でアリアスされた部分のスカラ浮動小数点スタックレジスタファイル（ｘ８７スタック）７４５では、ｘ８７スタックは、ｘ８７命令セット拡張を用いて、３２／６４／８０ビット浮動小数点データに対し、スカラ浮動小数点演算を行うために用いられる８要素から成るスタックである。一方で、ＭＭＸレジスタは、６４ビットのパックされた整数データに対し演算を行うために用いられ、また、ＭＭＸレジスタとＸＭＭレジスタとの間で行われるいくつかの演算のオペランドを保持するために用いられる。
【０１７３】
本開示の代替的な実施形態は、より広い、またはより狭いレジスタを用いてよい。また、本開示の代替的な実施形態は、より多い、より少ない、または異なるレジスタファイルおよびレジスタを用いてよい。
［例示的なコアアーキテクチャ、プロセッサ、およびコンピュータアーキテクチャ］
【０１７４】
プロセッサコアは、異なる方法で、異なる目的のために、異なるプロセッサにおいて実装されてよい。例えば、このようなコアの実装形態には、１）汎用コンピューティング向けの汎用インオーダコア、２）汎用コンピューティング向けの高性能汎用アウトオブオーダコア、３）主にグラフィックおよび／または科学（スループット）コンピューティング向けの専用コアが含まれてよい。異なるプロセッサの実装形態には、１）汎用コンピューティング向けの１または複数の汎用インオーダコアおよび／または汎用コンピューティング向け１または複数の汎用アウトオブオーダコアを含むＣＰＵ、および、２）主にグラフィックおよび／または科学（スループット）向けの１または複数の専用コアを含むコプロセッサが含まれてよい。このような異なるプロセッサは、異なるコンピュータシステムアーキテクチャをもたらし、このようなコンピュータシステムアーキテクチャには、１）ＣＰＵとは別のチップ上のコプロセッサ、２）ＣＰＵと同じパッケージ内の別のダイ上のコプロセッサ、３）ＣＰＵと同じダイ上のコプロセッサ（この場合、このようなコプロセッサは、統合グラフィックおよび／または科学（スループット）ロジック等の専用ロジック、または、専用コアと呼ばれることがある）、および４）同じダイ上に（アプリケーションコアまたはアプリケーションプロセッサと呼ばれることもある）説明されたＣＰＵ、上述したコプロセッサ、および追加の機能を含んでよいシステムオンチップが含まれてよい。次に、例示的なコアアーキテクチャについて説明し、その後に例示的なプロセッサおよびコンピュータアーキテクチャの説明が続く。
［例示的なコアアーキテクチャ］
［インオーダおよびアウトオブオーダコアのブロック図］
【０１７５】
図８Ａは、本開示の実施形態による、例示的なインオーダパイプラインおよび例示的なレジスタリネーミング、アウトオブオーダ発行／実行パイプラインの両方を示すブロック図である。図８Ｂは、本開示の実施形態によるプロセッサに含まれる例示的実施形態のインオーダアーキテクチャコア、および例示的なレジスタリネーミング、アウトオブオーダ発行／実行アーキテクチャコアの両方を示すブロック図である。図８Ａ～図８Ｂ中の実線のボックスがインオーダパイプラインおよびインオーダコアを示す一方、破線のボックスがオプションの追加のレジスタリネーミング、アウトオブオーダ発行／実行パイプライン及びコアを示す。インオーダの態様はアウトオブオーダの態様のサブセットであることを想定して、アウトオブオーダの態様について説明する。
【０１７６】
図８Ａ中、プロセッサパイプライン８００は、フェッチステージ８０２、長さデコードステージ８０４、デコードステージ８０６、割り当てステージ８０８、リネーミングステージ８１０、スケジューリング（ディスパッチまたは発行としても知られる）ステージ８１２、レジスタ読み取り／メモリ読み取りステージ８１４、実行ステージ８１６、書き戻

し／メモリ書き込みステージ８１８、例外処理ステージ８２２およびコミットステージ８２４を含む。

【０１７７】

図８Ｂは、実行エンジンユニット８５０に結合されたフロントエンドユニット８３０を含むプロセッサコア８９０を示し、これら両方がメモリユニット８７０に結合されている。コア８９０は、縮小命令セットコンピューティング（ＲＩＳＣ）コア、複合命令セットコンピューティング（ＣＩＳＣ）コア、超長命令語（ＶＬＩＷ）コア、またはハイブリッドコアタイプまたは代替コアタイプであってよい。さらに別のオプションとして、コア８９０は、例えば、ネットワークコアまたは通信コア等の専用コア、圧縮エンジン、コプロセッサコア、汎用コンピューティンググラフィック処理装置（ＧＰＧＰＵ）コアまたはグラフィックコア等であってよい。

【０１７８】

フロントエンドユニット８３０は、命令キャッシュユニット８３４に結合された分岐予測ユニット８３２を含み、命令キャッシュユニット８３４は命令変換ルックアサイドバッファ（ＴＬＢ）８３６に結合され、命令変換ルックアサイドバッファ（ＴＬＢ）８３６は命令フェッチユニット８３８に結合され、命令フェッチユニット８３８はデコードユニット８４０に結合されている。デコードユニット８４０（例えば、デコード回路）は命令（例えば、マクロ命令）をデコードし、且つ、１または複数のマイクロオペレーション、マイクロコードエントリポイント、マイクロ命令、他の命令または他の制御信号を出力として生成してよく、これらは元の命令からデコードされ、あるいは元の命令を反映し、あるいは元の命令から導出される。デコードユニット８４０は、様々な異なるメカニズムを用いて実装されてよい。好適なメカニズムの例としては、限定されるものではないが、ルックアップテーブル、ハードウェア実装、プログラマブルロジックアレイ（ＰＬＡ）、マイクロコードリードオンリメモリ（ＲＯＭ）等が含まれる。一実施形態において、コア８９０は、マイクロコードＲＯＭ、または特定のマクロ命令のためのマイクロコードを格納（例えば、デコードユニット８４０内に、あるいは、フロントエンドユニット８３０内に）する他の媒体を含む。デコードユニット８４０は、実行エンジンユニット８５０内のリネーム／アロケータユニット８５２に結合される。

【０１７９】

実行エンジンユニット８５０は、リタイアメントユニット８５４に結合されたリネーム／アロケータユニット８５２および１または複数のスケジューラユニットのセット８５６を含む。スケジューラユニット８５６は、予約ステーション、中央命令ウィンドウ等を含む任意の数の異ななるスケジューラ回路を表わす。スケジューラユニット８５６は、物理レジスタファイルユニット８５８に結合される。物理レジスタファイルユニット８５８の各々は、１または複数の物理レジスタセットを表わし、それらのうちの異なる物理レジスタセットは、スカラ整数、スカラ浮動小数点、パックされた整数、パックされた浮動小数点、ベクトル整数、ベクトル浮動小数点等の１または複数の異なるデータ型、ステータス（例えば、実行されるべき次の命令のアドレスである命令ポインタ）等を格納する。一実施形態において、物理レジスタファイルユニット８５８は、ベクトルレジスタユニット、書き込みマスクレジスタユニット、およびスカラレジスタユニットを含む。これらのレジスタユニットは、アーキテクチャベクトルレジスタ、ベクトルマスクレジスタおよび汎用レジスタを提供してよい。物理レジスタファイルユニット８５８は、リタイアメントユニット８５４に重ねられて、レジスタリネーミングおよびアウトオブオーダ実行が実装されてよい様々な態様（例えば、リオーダバッファおよびリタイアメントレジスタファイルを用いる、将来のファイル、履歴バッファ、およびリタイアメントレジスタファイルを用いる、レジスタマップおよびレジスタプールを用いる等）を示す。リタイアメントユニット８５４および物理レジスタファイルユニット８５８は、実行クラスタ８６０に結合されている。実行クラスタ８６０は、１または複数の実行ユニット８６２のセット（例えば、実行回路）および１または複数のメモリアクセスユニット８６４のセットを含む。実行ユニット８６２は、様々な演算（例えば、シフト、加算、減算、乗算）を、様々なデータ型（

例えば、スカラ浮動小数点、パックされた整数、パックされた浮動小数点、ベクトル整数、ベクトル浮動小数点）に対し実行してよい。いくつかの実施形態が、特定の機能または機能セットに専用の複数の実行ユニットを含んでよい一方、他の実装は、１つの実行ユニットのみ、またはすべての機能をすべて実行する複数の実行ユニットを含んでよい。スケジューラユニット８５６、物理レジスタファイルユニット８５８および実行クラスタ８６０は複数の可能性があるものとして図示されているが、これは、特定の実施形態において、特定のデータ型／演算のために別個のパイプラインが形成されるからである（例えば、スカラ整数パイプライン、スカラ浮動小数点／パックされた整数／パックされた浮動小数点／ベクトル整数／ベクトル浮動小数点のパイプライン、および／またはメモリアクセスパイプライン。これらパイプラインの各々は自身のスケジューラユニット、物理レジスタファイルユニットおよび／または実行クラスタを有し、別個のメモリアクセスパイプラインの場合は、このパイプラインの実行クラスタのみがメモリアクセスユニット８６４を有するといった特定の実装が実装される）。また別個のパイプラインが用いられる場合は、これらのパイプラインのうちの１または複数は、アウトオブオーダ発行／実行であってよく、残りがインオーダであってよいことも理解されたい。

【０１８０】

メモリアクセスユニット８６４のセットがメモリユニット８７０に結合され、レベル２（Ｌ２）キャッシュユニット８７６に結合されたデータキャッシュユニット８７４に結合されたデータＴＬＢユニット８７２を含む。１つの例示的実施形態において、メモリアクセスユニット８６４は、ロードユニット、ストアアドレスユニット、およびストアデータユニットを含んでよく、これらの各々が、メモリユニット８７０のデータＴＬＢユニット８７２に結合されている。命令キャッシュユニット８３４は、さらに、メモリユニット８７０内のレベル２（Ｌ２）キャッシュユニット８７６に結合されている。Ｌ２キャッシュユニット８７６は、１または複数の他のレベルのキャッシュに結合され、最終的にメインメモリに結合される。

【０１８１】

例示として、例示的レジスタリネーミング、アウトオブオーダ発行／実行コアアーキテクチャは、以下のようなパイプライン８００を実装してよい。１）命令フェッチ８３８は、フェッチステージ８０２および長さデコーディングステージ８０４を実行する。２）デコードユニット８４０は、デコードステージ８０６を実行する。３）リネーム／アロケータユニット８５２は、割り当てステージ８０８およびリネームステージ８１０を実行する。４）スケジューラユニット８５６は、スケジューリングステージ８１２を実行する。５）物理レジスタファイルユニット８５８およびメモリユニット８７０は、レジスタ読み取り／メモリ読み取りステージ８１４を実行し、実行クラスタ８６０は、実行ステージ８１６を実行する。６）メモリユニット８７０および物理レジスタファイルユニット８５８は、書き戻し／メモリ書き込みステージ８１８を実行する。７）様々なユニットが例外処理ステージ８２２に関与してよい。８）リタイアメントユニット８５４および物理レジスタファイルユニットは、コミットステージ８２４を実行する。

【０１８２】

コア８９０は、本明細書で説明した命令を含む、１または複数の命令セット（例えば、ｘ８６命令セット（より新しいバージョンが追加されたいくつかの拡張がなされたもの）、カリフォルニアのサニーベールにあるＭＩＰＳ　Ｔｅｃｈｎｏｌｏｇｉｅｓ　ＭＩＰＳ命令セット、カリフォルニアのサニーベールにあるＡＲＭ　Ｈｏｌｄｉｎｇｓ　ＡＲＭ命令セット（ＮＥＯＮ等のオプションの拡張機能が追加された））をサポートしてよい。一実施形態において、コア８９０は、パックされたデータ命令セット拡張（例えば、ＡＶＸ１、ＡＶＸ２）をサポートするロジックを含み、これにより、多くのマルチメディアアプリケーションにより用いられるオペレーションがパックされたデータを用いて実行されることを可能にする。

【０１８３】

コアは、マルチスレッディング（２または２より多いオペレーションまたはスレッドの

並列セットを実行）をサポートしてよく、タイムスライスマルチスレッディング、同時マルチスレッディング（単一の物理コアが、物理コアが同時マルチスレッディングをしているスレッドの各々に対し論理コアを提供する）、またはこれらの組み合わせ（例えば、インテル（登録商標）ハイパースレッディングテクノロジのように、タイムスライスフェッチおよびデコード並びにその後の同時マルチスレッディング）を含む様々な方法でそのように実行してよいことを理解されたい。

【０１８４】

レジスタリネーミングはアウトオブオーダ実行の文脈で説明されているが、レジスタリネーミングはインオーダアーキテクチャで用いられてよいことを理解されたい。プロセッサの示された実施形態は、別個の命令およびデータキャッシュユニット８３４／８７４および共有Ｌ２キャッシュユニット８７６を含む一方、代替的な実施形態は、例えば、レベル１（Ｌ１）内部キャッシュまたは複数のレベルの内部キャッシュ等の命令およびデータ両方のための単一の内部キャッシュを有してよい。いくつかの実施形態において、システムは、内部キャッシュと、コアおよび／またはプロセッサの外部にある外部キャッシュとの組み合わせを含んでよい。代替的に、すべてのキャッシュは、コアおよび／またはプロセッサの外部に存在してよい。

［具体的な例示的インオーダコアアーキテクチャ］

【０１８５】

図９Ａ～図９Ｂは、より具体的な例示的インオーダコアアーキテクチャのブロック図を示し、このコアは、チップ内のいくつかの論理ブロック（同じタイプおよび／または異なるタイプの他のコアを含む）のうちの１つとなる。論理ブロックは、適用に応じ、高帯域幅の相互接続ネットワーク（例えば、リングネットワーク）を通して、いくつかの固定機能ロジック、メモリＩ／Ｏインタフェースおよび他の必要なＩ／Ｏロジックと通信する。

【０１８６】

図９Ａは、本開示の実施形態による、単一のプロセッサコアを、オンダイ相互接続ネットワーク９０２へのその接続およびレベル２（Ｌ２）キャッシュ９０４のそのローカルサブセットと共に示すブロック図を示す。一実施形態において、命令デコードユニット９００は、パックされたデータ命令セット拡張を持つｘ８６命令セットをサポートする。Ｌ１キャッシュ９０６は、スカラユニットおよびベクトルユニットに入るキャッシュメモリへの低レイテンシのアクセスを可能にする。一実施形態において、（設計の単純化のために）スカラユニット９０８およびベクトルユニット９１０は、別個のレジスタセット（それぞれ、スカラレジスタ９１２およびベクトルレジスタ９１４）を用い、これらの間で転送されるデータはメモリに書き込まれた後、レベル１（Ｌ１）キャッシュ９０６から再読み取りされる一方、本開示の代替的な実施形態は、異なるアプローチ（例えば、単一のレジスタセットを用いる、または書き込みおよび再読み取りを行うことなく、２つのレジスタファイル間でデータ転送を可能とする通信パスを含む）を用いてよい。

【０１８７】

Ｌ２キャッシュ９０４のローカルサブセットは、プロセッサコア当たりに１つのサブセットとなるように、別個のローカルサブセットに分割されたグローバルＬ２キャッシュの一部である。各プロセッサコアは、Ｌ２キャッシュ９０４のその独自のローカルサブセットへの直接アクセスパスを有する。プロセッサコアにより読み取られたデータは、そのＬ２キャッシュサブセット９０４に格納され、それらの独自のローカルＬ２キャッシュサブセットにアクセスする他のプロセッサコアと並行して、迅速にアクセスされてよい。プロセッサコアにより書き込まれたデータは、その独自のＬ２キャッシュサブセット９０４内に格納され、必要な場合は、他のサブセットからフラッシュされる。リングネットワークは、共有データのコヒーレンシを保証する。リングネットワークは、プロセッサコア、Ｌ２キャッシュおよび他の論理ブロック等のエージェントが、チップ内で互いに通信できるように双方向である。各リングデータパスは、各方向で１０１２ビット幅である。

【０１８８】

図９Ｂは、本開示の実施形態による図９Ａ中のプロセッサコアの一部の拡大図である。

図９Ｂは、Ｌ１キャッシュ９０４の一部であるＬ１データキャッシュ９０６Ａと、ベクトルユニット９１０およびベクトルレジスタ９１４に関する詳細とを含む。具体的に言うと、ベクトルユニット９１０は、１６幅ベクトル処理ユニット（ＶＰＵ）（１６幅ＡＬＵ９２８を参照）であり、ＶＰＵは、整数、単精度および倍精度の浮動命令のうちの１または複数を実行する。ＶＰＵは、スウィズルユニット９２０を用いてレジスタ入力をスウィズルすること、数値変換ユニット９２２Ａ－Ｂを用いて数値変換すること、およびレプリケーションユニット９２４を用いてメモリ入力にレプリケーションを行うことをサポートする。書き込みマスクレジスタ９２６は、得られるベクトル書き込みを予測することを可能にする。

【０１８９】

図１０は、本開示の実施形態による、２つ以上のコアを有してよい、統合メモリコントローラを有してよい、統合グラフィックを有してよいプロセッサ１０００のブロック図である。図１０の実線のボックスは、シングルコア１００２Ａ、システムエージェント１０１０、１または複数のバスコントローラユニット１０１６から成るセットを有するプロセッサ１０００を示す一方、破線のボックスの追加オプションは、マルチコア１００２Ａ～Ｎ、システムエージェントユニット１０１０内の１または複数の統合メモリコントローラユニット１０１４から成るセット、および専用ロジック１００８を有する代替のプロセッサ１０００を示す。

【０１９０】

故に、プロセッサ１０００の異なる実装は、１）統合されたグラフィックおよび／または科学（スループット）ロジック（１または複数のコアを含んでよい）である専用ロジック１００８と、１または複数の汎用コア（例えば、汎用インオーダコア、汎用アウトオブオーダコア、これら２つの組み合わせ）であるコア１００２Ａ～Ｎとを持つＣＰＵ、２）主にグラフィックおよび／または科学（スループット）向けの多数の専用コアであるコア１００２Ａ～Ｎを持つコプロセッサ、および、３）多数の汎用インオーダコアであるコア１００２Ａ～Ｎを持つコプロセッサを含んでよい。故に、プロセッサ１０００は、汎用プロセッサ、コプロセッサ、または、例えば、ネットワークプロセッサ若しくは通信プロセッサ、圧縮エンジン、グラフィックプロセッサ、ＧＰＧＰＵ（汎用グラフィック処理装置）、高スループットＭＩＣ（Ｍａｎｙ　Ｉｎｔｅｇｒａｔｅｄ　Ｃｏｒｅ）コプロセッサ、（３０以上のコアを含む）または組み込みプロセッサ等の専用プロセッサであってよい。プロセッサは、１または複数のチップ上に実装されてよい。プロセッサ１０００は、例えば、ＢｉＣＭＯＳ、ＣＭＯＳまたはＮＭＯＳ等の任意の数のプロセス技術を用いた、１または複数の基板の一部であってよく、および／または、当該基板上に実装されてよい。

【０１９１】

メモリ階層は、コア内に１または複数のレベルのキャッシュ、共有キャッシュユニットのセットまたは１若しくは複数の共有キャッシュユニット１００６、および統合メモリコントローラユニット１０１４のセットに結合された外部メモリ（不図示）を含む。共有キャッシュユニットのセット１００６は、１または複数の中間レベルのキャッシュ、例えば、レベル２（Ｌ２）、レベル３（Ｌ３）、レベル４（Ｌ４）または他のレベルのキャッシュ、ラストレベルキャッシュ（ＬＬＣ）および／またはこれらの組み合わせを含んでよい。一実施形態において、リングベースの相互接続ユニット１０１２が、統合グラフィックロジック１００８、共有キャッシュユニットのセット１００６、およびシステムエージェントユニット１０１０／統合メモリコントローラユニット（１０１４）を相互接続する一方、代替的な実施形態は、このようなユニットを相互接続するための任意の数の周知の技術を用いてよい。一実施形態において、１または複数のキャッシュユニット１００６と、コア１００２－Ａ－Ｎとの間のコヒーレンシが維持される。

【０１９２】

いくつかの実施形態において、コア１００２Ａ－Ｎのうちの１または複数がマルチスレッディングを可能である。システムエージェント１０１０は、コア１００２Ａ－Ｎを調整および操作するこれらのコンポーネントを含む。システムエージェントユニット１０１０

は、例えば、電力制御ユニット（ＰＣＵ）およびディスプレイユニットを含んでよい。ＰＣＵは、コア１００２Ａ－Ｎおよび統合グラフィックロジック１００８の電力状態を調整するために必要なロジックおよびコンポーネントであってよい、またはこれらを含んでよい。ディスプレイユニットは、１または複数の外部接続されたディスプレイを駆動させるためのものである。

【０１９３】

コア１００２Ａ－Ｎは、アーキテクチャ命令セットの観点から同種または異種であってよく、すなわち、コア１００２Ａ－Ｎのうちの２または２より多いものは、同一命令セットを実行可能であって良い一方、他のものは、その命令セットのサブセットまたは異なる命令セットを実行可能であってよい。

［例示的なコンピュータアーキテクチャ］

【０１９４】

図１１～１４は、例示的なコンピュータアーキテクチャのブロック図である。ラップトップ、ハンドヘルドＰＣ、パーソナルデジタルアシスタント、エンジニアリングワークステーション、サーバ、ネットワークデバイス、ネットワークハブ、スイッチ、埋め込みプロセッサ、デジタル信号プロセッサ（ＤＳＰ）、グラフィックデバイス、ビデオゲームデバイス、セットトップボックス、マイクロコントローラ、携帯電話、ポータブルメディアプレーヤ、ハンドヘルドデバイス、および様々な他の電子デバイス用に当該技術分野で既知の他のシステム設計および構成もまた好適である。一般に、本明細書で開示されたプロセッサおよび／または他の実行ロジックを組み込み可能な膨大な数の様々なシステムおよび電子デバイスが、概して好適である。

【０１９５】

ここで図１１を参照すると、本開示の一実施形態による、システム１１００のブロック図が示されている。システム１１００は、１または複数のプロセッサ１１１０、１１１５を含んでよく、当該プロセッサはコントローラハブ１１２０に結合される。一実施形態において、コントローラハブ１１２０は、グラフィックメモリコントローラハブ（ＧＭＣＨ）１１９０および入／出力ハブ（ＩＯＨ）１１５０（別個のチップ上に存在してよい）を含む。ＧＭＣＨ１１９０は、メモリ１１４０およびコプロセッサ１１４５が結合されたメモリコントローラおよびグラフィックコントローラを含む。ＩＯＨ１１５０は、入／出力（Ｉ／Ｏ）デバイス１１６０を、ＧＭＣＨ１１９０に結合する。代替的に、メモリコントローラおよびグラフィックコントローラのうちの一方または両方は、プロセッサ内に統合されてよく（本明細書で説明したように）、メモリ１１４０およびコプロセッサ１１４５は、プロセッサ１１１０およびＩＯＨ１１５０内の単一チップにあるコントローラハブ１１２０に直接結合される。メモリ１１４０は、例えば、実行時にプロセッサに、本開示の任意の方法を実行させるコードを格納するための投機マネージャコード１１４０Ａを含んでよい。

【０１９６】

追加のプロセッサ１１１５のオプションの性質が図１１に破線で示されている。各プロセッサ１１１０、１１１５は、本明細書で説明された処理コアのうちの１または複数を含んでよく、プロセッサ１０００の何らかのバージョンであってよい。

【０１９７】

メモリ１１４０は、例えば、ダイナミックランダムアクセス（ＤＲＡＭ）、相変化メモリ（ＰＣＭ）またはこれら２つの組み合わせであってよい。少なくとも１つの実施形態では、コントローラハブ１１２０は、フロントサイドバス（ＦＳＢ）等のマルチドロップバス、Ｑｕｉｃｋｐａｔｈインターコネクト（ＱＰＩ）等のポイントツーポイントインタフェース、または同様の接続１１９５を介して、プロセッサ１１１０、１１１５と通信する。

【０１９８】

一実施形態において、コプロセッサ１１４５は、例えば、高スループットＭＩＣプロセッサ、ネットワークプロセッサまたは通信プロセッサ、圧縮エンジン、グラフィックプロ

セッサ、ＧＰＧＰＵまたは組み込みプロセッサ等の専用プロセッサである。一実施形態において、コントローラハブ１１２０は、統合グラフィックアクセラレータを含んでよい。

【０１９９】

物理リソース１１１０、１１１５間には、アーキテクチャ特性、マイクロアーキテクチャ特性、熱特性、電力消費特性等を含む広範な価値基準の観点から様々な差異が存在し得る。

【０２００】

一実施形態において、プロセッサ１１１０は、汎用タイプのデータプロセッシング処理を制御する命令を実行する。命令内には、コプロセッサ命令が埋め込まれてよい。プロセッサ１１１０は、これらのコプロセッサ命令を、取り付けられたコプロセッサ１１４５により実行されるべきタイプであると認識する。従って、プロセッサ１１１０は、これらのコプロセッサ命令（またはコプロセッサ命令を表わす制御信号）を、コプロセッサバスまたは他の相互接続上でコプロセッサ１１４５に対し発行する。コプロセッサ１１４５は、コプロセッサ命令を受け取り、受信したコプロセッサ命令を実行する。

【０２０１】

ここで図１２を参照すると、本開示の実施形態による、第１のより具体的な例示的システム１２００のブロック図が示されている。図１２に示される通り、マルチプロセッサシステム１２００は、ポイント‐ツーポイント相互接続システムであり、ポイント‐ツーポイント相互接続１２５０を介して結合された第１のプロセッサ１２７０および第２のプロセッサ１２８０を含む。プロセッサ１２７０および１２８０の各々は、プロセッサ１０００の特定のバージョンであってよい。本開示の一実施形態において、プロセッサ１２７０および１２８０はそれぞれプロセッサ１１１０および１１１５である一方、コプロセッサ１２３８はコプロセッサ１１４５である。別の実施形態において、プロセッサ１２７０および１２８０は、それぞれプロセッサ１１１０およびコプロセッサ１１４５である。

【０２０２】

プロセッサ１２７０および１２８０は、それぞれ統合メモリコントローラ（ＩＭＣ）ユニット１２７２および１２８２を含むように図示されている。プロセッサ１２７０は、またそのバスコントローラユニットの一部として、ポイント‐ツーポイント（Ｐ‐Ｐ）インタフェース１２７６および１２７８を含み、同様に、第２のプロセッサ１２８０は、Ｐ‐Ｐインタフェース１２８６および１２８８を含む。プロセッサ１２７０、１２８０は、Ｐ‐Ｐインタフェース回路１２７８、１２８８を用いて、ポイント‐ツーポイント（Ｐ‐Ｐ）インタフェース１２５０を介して情報を交換してよい。図１２に示される通り、ＩＭＣ１２７２および１２８２は、プロセッサをそれぞれのメモリ、すなわちメモリ１２３２およびメモリ１２３４に結合し、当該メモリは、それぞれのプロセッサにローカルに取り付けられたメインメモリの一部であってよい。

【０２０３】

プロセッサ１２７０、１２８０はそれぞれ、ポイントツーポイントインタフェース回路１２７６、１２９４、１２８６、１２９８を用いて、個々のＰ‐Ｐインタフェース１２５２、１２５４を介してチップセット１２９０と情報を交換してよい。オプションで、チップセット１２９０は、高性能インタフェース１２３９を介して、コプロセッサ１２３８と情報を交換してよい。一実施形態において、コプロセッサ１２３８は、例えば、高スループットＭＩＣプロセッサ、ネットワークプロセッサまたは通信プロセッサ、圧縮エンジン、グラフィックプロセッサ、ＧＰＧＰＵまたは埋め込みプロセッサ等の専用プロセッサである。

【０２０４】

共有キャッシュ（不図示）は、いずれかのプロセッサ内に含まれてよく、または両方のプロセッサの外部にあるが、Ｐ‐Ｐ相互接続を介してプロセッサと接続されていてよく、そのため、プロセッサが低電力モードになった場合、いずれかのプロセッサまたは両方のプロセッサのローカルキャッシュ情報が共有キャッシュに格納されてよい。

【０２０５】

チップセット１２９０は、インタフェース１２９６を介して第１のバス１２１６に結合されてよい。一実施形態において、第１のバス１２１６は、ペリフェラルコンポーネントインターコネクト（ＰＣＩ）バス、または、ＰＣＩ Ｅｘｐｒｅｓｓバス等のバス、または別の第３世代Ｉ／Ｏ相互接続バスであってよいが、本開示の実施形態はこのように限定はされない。

【０２０６】

図１２に示される通り、様々なＩ／Ｏデバイス１２１４が、第１のバス１２１６を第２のバス１２２０に結合させるバスブリッジ１２１８と共に、第１のバス１２１６に結合されてよい。一実施形態において、コプロセッサ、高スループットＭＩＣプロセッサ、ＧＰＧＰＵ、アクセラレータ（例えば、グラフィックアクセラレータまたはデジタル信号処理（ＤＳＰ）ユニット等）、フィールドプログラマブルゲートアレイ、または任意の他のプロセッサ等の１または複数の追加のプロセッサ１２１５が、第１のバス１２１６に結合される。一実施形態において、第２のバス１２２０は、ローピンカウント（ＬＰＣ）バスであってよい。例えば、一実施形態において、キーボードおよび／またはマウス１２２２、通信デバイス１２２７、および、命令／コードおよびデータ１２３０を含んでよいディスクドライブまたは他のマスストレージデバイス等のストレージユニット１２２８を含む、様々なデバイスが第２のバス１２２０に結合されてよい。さらに、オーディオＩ／Ｏ１２２４が、第２のバス１２２０に結合されてよい。他のアーキテクチャが可能であることに留意されたい。例えば、図１２のポイント‐ツーポイントアーキテクチャの代わりに、システムは、マルチドロップバスまたは他のこのようなアーキテクチャを実装してよい。

【０２０７】

ここで図１３を参照すると、本開示の実施形態による、第２のより具体的な例示的システム１３００のブロック図が示されている。図１２および図１３の同様の要素には同様の参照符号が付され、図１３の他の態様を曖昧にすることを回避するために、図１２の特定の態様は図１３から省略されている。

【０２０８】

図１３は、プロセッサ１２７０、１２８０が、それぞれ統合メモリならびにＩ／Ｏ制御ロジック（"ＣＬ"）１２７２および１２８２を含んでよいことを示す。故に、ＣＬ１２７２、１２８２は、統合メモリコントローラユニットを含み、Ｉ／Ｏ制御ロジックを含む。図１３は、メモリ１２３２、１２３４がＣＬ１２７２、１２８２に結合されるだけでなく、Ｉ／Ｏデバイス１３１４も制御ロジック１２７２、１２８２に結合されることを示す。レガシＩ／Ｏデバイス１３１５が、チップセット１２９０に結合される。

【０２０９】

ここで図１４を参照すると、本開示の実施形態によるＳｏＣ１４００のブロック図が示されている。図１０中の同様の要素には同様の参照符号が付されている。また、破線ボックスはより高度なＳｏＣ上のオプションの機能である。図１４中、相互接続ユニット１４０２は、１または複数のコア２０２Ａ－Ｎのセットおよび共有キャッシュユニット１００６を含むアプリケーションプロセッサ１４１０；システムエージェントユニット１０１０；バスコントローラユニット１０１６；統合メモリコントローラユニット１０１４、統合グラフィックロジック、イメージプロセッサ、オーディオプロセッサおよびビデオプロセッサを含んでよいコプロセッサ１４２０のセットまたは１若しくは複数のコプロセッサ１４２０；スタティックランダムアクセスメモリ（ＳＲＡＭ）ユニット１４３０；ダイレクトメモリアクセス（ＤＭＡ）ユニット１４３２；および１または複数の外部ディスプレイに結合するためのディスプレイユニット１４４０に結合される。一実施形態において、コプロセッサ１４２０は、例えば、ネットワークプロセッサまたは通信プロセッサ、圧縮エンジン、ＧＰＧＰＵ、高スループットＭＩＣプロセッサまたは埋め込みプロセッサ等の専用プロセッサを含む。

【０２１０】

本明細書で開示された実施形態（例えば、メカニズムの）は、ハードウェア、ソフトウェア、ファームウェアまたはこのような実装アプローチの組み合わせで実装されてよい。

本開示の実施形態は、コンピュータプログラムまたは少なくとも１つのプロセッサ、ストレージシステム（揮発性および不揮発性のメモリおよび／またはストレージ要素）、少なくとも１つの入力デバイス、および少なくとも１つの出力デバイスを備えるプログラム可能なシステム上で実行されるプログラムコードとして実装されてよい。

【０２１１】

図１２中に示されるコード１２３０等のプログラムコードが入力命令に適用され、本明細書に説明された機能を実行し、出力情報を生成してよい。出力情報は、１または複数の出力デバイスに既知の態様で適用されてよい。本願の目的において、処理システムは、例えば、デジタル信号プロセッサ（ＤＳＰ）、マイクロコントローラ、特定用途向け集積回路（ＡＳＩＣ）またはマイクロプロセッサ等のプロセッサを有する任意のシステムを含んでよい。

【０２１２】

プログラムコードは、処理システムと通信するために高水準の手続型またはオブジェクト指向型プログラミング言語で実装されてよい。また、必要であれば、プログラムコードは、アセンブリ言語または機械言語で実装されてもよい。実際に、本明細書で記載されたメカニズムはその範囲において、いずれの特定のプログラミング言語にも限定されない。いずれの場合においても、言語は、コンパイルされた言語または解釈された言語であってよい。

【０２１３】

少なくとも１つの実施形態の１または複数の態様が、機械可読媒体に格納された典型的な命令により実装されてよく、当該命令は、プロセッサ内の様々なロジックを表わし、機械によって読み取られると、機械に、本明細書で説明された技術を実行するためのロジックを作成させる。"ＩＰコア"として知られるこのような表現は、有形の機械可読媒体上に格納されて、様々な顧客または製造施設へ供給されて、実際にロジックまたはプロセッサを作成する製造用機械に読み込まれてよい。

【０２１４】

このような機械可読ストレージ媒体は、限定ではないが、機械または装置により製造または形成された物品の非一時的有形構成を含んでよく、このようなものとしては、ハードディスク等のストレージ媒体、フロッピーディスク、光ディスク、コンパクトディスクリードオンリメモリ（ＣＤ‐ＲＯＭ）、コンパクトディスクリライタブル（ＣＤ‐ＲＷ）および磁気‐光ディスク等の任意の他のタイプのディスク、リードオンリメモリ（ＲＯＭ）、ダイナミックランダムアクセスメモリ（ＤＲＡＭ）、スタティックランダムアクセスメモリ（ＳＲＡＭ）等のランダムアクセスメモリ（ＲＡＭ）、消去可能プログラマブルリードオンリメモリ（ＥＰＲＯＭ）、フラッシュメモリ、電気的消去可能プログラマブルリードオンリメモリ（ＥＥＰＲＯＭ）、相変化メモリ（ＰＣＭ）等の半導体デバイス、磁気または光カード、または電子命令を格納するのに好適な任意の他のタイプの媒体が含まれる。

【０２１５】

従って、本開示の実施形態はまた、命令を含む、または、本明細書で説明された構造、回路、装置、プロセッサおよび／またはシステム機能を定義するハードウェア記述言語（ＨＤＬ）等の設計データを含む非一時的な有形の機械可読媒体を含む。このような実施形態も、プログラムプロダクトと称されてよい。

［エミュレーション（バイナリ変換、コードモーフィング等を含む）］

【０２１６】

いくつかの場合において、命令コンバータを用いて、ソース命令セットからの命令をターゲット命令セットへ変換してよい。例えば、命令コンバータは、命令をコアにより処理される１または複数の他の命令へと、変換（例えば、スタティックバイナリ変換、ダイナミックコンパイルを含むダイナミックバイナリ変換を用いて）、モーフィング、エミュレート、あるいはコンバートしてよい。命令コンバータは、ソフトウェア、ハードウェア、ファームウェアまたはこれらの組み合わせで実装されてよい。命令コンバータは、プロセ

ッサ上、プロセッサ外、または一部がプロセッサ上または一部がプロセッサ外にあってよい。

【０２１７】

図１５は、本開示の実施形態によるソース命令セット内のバイナリ命令を、ターゲット命令セット内のバイナリ命令に変換するソフトウェア命令コンバータの使用を対比したブロック図である。示された実施形態では、命令コンバータはソフトウェア命令コンバータであるが、代替的に、命令コンバータはソフトウェア、ファームウェア、ハードウェア、またはこれらの様々な組み合わせで実装されてもよい。図１５は、少なくとも１つのｘ８６命令セットコア１５１６を有するプロセッサによって、ネイティブで実行可能なｘ８６バイナリコード１５０６を生成するために、高水準言語１５０２のプログラムが、ｘ８６コンパイラ１５０４を用いてコンパイルされてよいことを示す。少なくとも１つのｘ８６命令セットコア１５１６を持つプロセッサは、少なくとも１つのｘ８６命令セットコアを持つインテル（登録商標）プロセッサと実質的に同一の結果を達成すべく、（１）インテル（登録商標）ｘ８６命令セットコアの命令セットのかなりの部分、または、（２）少なくとも１つのｘ８６命令セットコアを持つインテル（登録商標）プロセッサ上で実行される対象のアプリケーションまたは他のソフトウェアのオブジェクトコードバージョンを互換的に実行するか、または、処理するかにより、少なくとも１つのｘ８６命令セットコアを持つインテル（登録商標）プロセッサと実質的に同一の機能を実行可能な任意のプロセッサを表わす。ｘ８６コンパイラ１５０４は、さらなるリンク処理を用いて、または用いることなく、少なくとも１つのｘ８６命令セットコア１５１６を有するプロセッサ上で実行可能なｘ８６バイナリコード１５０６（例えば、オブジェクトコード）を生成するように動作可能なコンパイラを表わす。同様に、図１５は、高水準言語１５０２のプログラムが、代替的な命令セットバイナリコード１５１０を生成する代替的な命令セットコンパイラ１５０８を使用してコンパイルされ得ることを示しており、当該代替的な命令セットバイナリコード１５１０は、少なくとも１つのｘ８６命令セットコアを持たないプロセッサ１５１４（例えば、カリフォルニア州サニーベールのＭＩＰＳ ＴｅｃｈｎｏｌｏｇｉｅｓのＭＩＰＳ命令セットを実行する、および／または、カリフォルニア州サニーベールのＡＲＭ Ｈｏｌｄｉｎｇｓ のＡＲＭ命令セットを実行するコアを持つプロセッサ）によってネイティブに実行されてよい。命令コンバータ１５１２は、ｘ８６バイナリコード１５０６を、ｘ８６命令セットコアを有さないプロセッサ１５１４によってネイティブで実行可能なコードに変換するために用いられる。この変換されたコードは、代替的な命令セットバイナリコード１５１０と同じである可能性が低い。なぜなら、この変換が可能な命令コンバータは、製造が難しいからである。しかしながら、変換されたコードは、一般的なオペレーションを実現し、代替的な命令セットからの複数の命令で構成される。故に、命令コンバータ１５１２は、ソフトウェア、ファームウェア、ハードウェアまたはこれらの組み合わせを表わし、それらは、エミュレーション、シミュレーションまたは任意の他の処理を介して、ｘ８６命令セットプロセッサまたはコアを有さないプロセッサまたは他の電子デバイスが、ｘ８６バイナリコード１５０６を実行できるようにする。

［他の可能な請求項］

［項目１］

命令をデコードされた命令にデコードするためのデコーダと、

投機マネージャ回路であって、

上記命令内のセキュリティチェックフィールドを検出する、

潜在的に投機ミスされる実行に対し強制実行されるべきセキュリティチェックポリシーを、上記セキュリティチェックフィールドに基づき、複数のセキュリティチェックポリシーから判定する、

上記セキュリティチェックポリシーの１または複数の関連付けられたチェックを上記命令に対し実行して、上記命令が潜在的に投機ミスされるか否かを判定する、

上記１または複数の関連付けられたチェックにより、上記命令が安全ではないとみなされた場合に、上記命令の実行をスケジューリングする、

上記１または複数の関連付けられたチェックにより、上記命令が安全であるとみなされた場合に、上記命令を省略する、投機マネージャ回路と、
実行のためのスケジューリングがされた上記命令を実行するための実行ユニットと、を備える、装置。
［項目２］
上記セキュリティチェックフィールドは、コンパイラにより提供されるヒントである、項目１に記載の装置。
［項目３］
上記投機マネージャ回路は、上記セキュリティチェックポリシーの上記１または複数の関連付けられたチェックを、上記命令の関連付けられたメモリアクセスのセットに対し実行する、項目１に記載の装置。
［項目４］
上記セキュリティチェックポリシーは、メモリ安全性チェックポリシーである、項目１に記載の装置。
［項目５］
上記セキュリティチェックポリシーは、型安全性チェックポリシーである、項目１に記載の装置。
［項目６］
上記セキュリティチェックポリシーの上記１または複数の関連付けられたチェックは、メモリ安全性チェックおよび型安全性チェックを含む、項目１に記載の装置。
［項目７］
上記１または複数の関連付けられたチェックは、上記装置のアーキテクチャ仕様の完全準拠チェックより少ない、項目１に記載の装置。
［項目８］
上記命令は、プログラム順序において後続するメモリアクセス命令に関連付けられたセキュリティチェック命令である、項目１に記載の装置。
［項目９］
ハードウェアプロセッサのデコーダを用いて、命令をデコードされた命令にデコードする段階と、
上記ハードウェアプロセッサにより、上記命令内のセキュリティチェックフィールドを検出する段階と、
上記ハードウェアプロセッサにより、潜在的に投機ミスされる実行に対し強制実行されるべきセキュリティチェックポリシーを、上記セキュリティチェックフィールドに基づき、複数のセキュリティチェックポリシーから判定する段階と、
上記ハードウェアプロセッサにより、上記セキュリティチェックポリシーの１または複数の関連付けられたチェックを上記命令に対し実行して、上記命令が潜在的に投機ミスされるか否かを判定する段階と、
上記ハードウェアプロセッサにより、上記１または複数の関連付けられたチェックにより上記命令が安全ではないとみなされた場合に、上記命令の実行をスケジューリングする段階と、
上記ハードウェアプロセッサにより、上記１または複数の関連付けられたチェックにより、上記命令が安全であるとみなされた場合に、上記命令を省略する段階と、
上記ハードウェアプロセッサの実行ユニットを用いて、実行のためのスケジューリングがされた上記命令を実行する段階と、を備える、方法。
［項目１０］
上記セキュリティチェックフィールドは、コンパイラにより提供されるヒントである、項目９に記載の方法。
［項目１１］
上記実行する段階は、上記セキュリティチェックポリシーの上記１または複数の関連付けられたチェックを、上記命令の関連付けられたメモリアクセスのセットに対し実行する

段階を含む、項目９に記載の方法。
［項目１２］
上記セキュリティチェックポリシーは、メモリ安全性チェックポリシーである、項目９に記載の方法。
［項目１３］
上記セキュリティチェックポリシーは、型安全性チェックポリシーである、項目９に記載の方法。
［項目１４］
上記セキュリティチェックポリシーの上記１または複数の関連付けられたチェックは、メモリ安全性チェックおよび型安全性チェックを含む、項目９に記載の方法。
［項目１５］
上記１または複数の関連付けられたチェックは、上記ハードウェアプロセッサのアーキテクチャ仕様の完全準拠チェックより少ない、項目９に記載の方法。
［項目１６］
上記命令は、プログラム順序において後続するメモリアクセス命令に関連付けられたセキュリティチェック命令である、項目９に記載の方法。
［項目１７］
機械により実行されると、機械に対し、方法を実行させるためのコードを格納する非一時的機械可読媒体であって、上記方法は、
ハードウェアプロセッサのデコーダを用いて、命令をデコードされた命令にデコードする手順と、
上記ハードウェアプロセッサにより、上記命令内のセキュリティチェックフィールドを検出する段階と、
上記ハードウェアプロセッサにより、潜在的に投機ミスされる実行に対し強制実行されるべきセキュリティチェックポリシーを、上記セキュリティチェックフィールドに基づき、複数のセキュリティチェックポリシーから判定する段階と、
上記ハードウェアプロセッサにより、上記セキュリティチェックポリシーの１または複数の関連付けられたチェックを上記命令に対し実行して、上記命令が潜在的に投機ミスされるか否かを判定する段階と、
上記ハードウェアプロセッサにより、上記１または複数の関連付けられたチェックにより上記命令が安全ではないとみなされた場合に、上記命令の実行をスケジューリングする段階と、
上記ハードウェアプロセッサにより、上記１または複数の関連付けられたチェックにより上記命令が安全であるとみなされた場合に、上記命令を省略する段階と、
上記ハードウェアプロセッサの実行ユニットを用いて、実行のためのスケジューリングがされた上記命令を実行する段階と、を備える、非一時的機械可読媒体。
［項目１８］
上記セキュリティチェックフィールドは、コンパイラにより提供されるヒントである、項目１７に記載の非一時的機械可読媒体。
［項目１９］
上記実行する段階は、上記セキュリティチェックポリシーの上記１または複数の関連付けられたチェックを、上記命令の関連付けられたメモリアクセスのセットに対し実行する段階を含む、項目１７に記載の非一時的機械可読媒体。
［項目２０］
上記セキュリティチェックポリシーは、メモリ安全性チェックポリシーである、項目１７に記載の非一時的機械可読媒体。
［項目２１］
上記セキュリティチェックポリシーは、型安全性チェックポリシーである、項目１７に記載の非一時的機械可読媒体。
［項目２２］

上記セキュリティチェックポリシーの上記１または複数の関連付けられたチェックは、メモリ安全性チェックおよび型安全性チェックを含む、項目１７に記載の非一時的機械可読媒体。

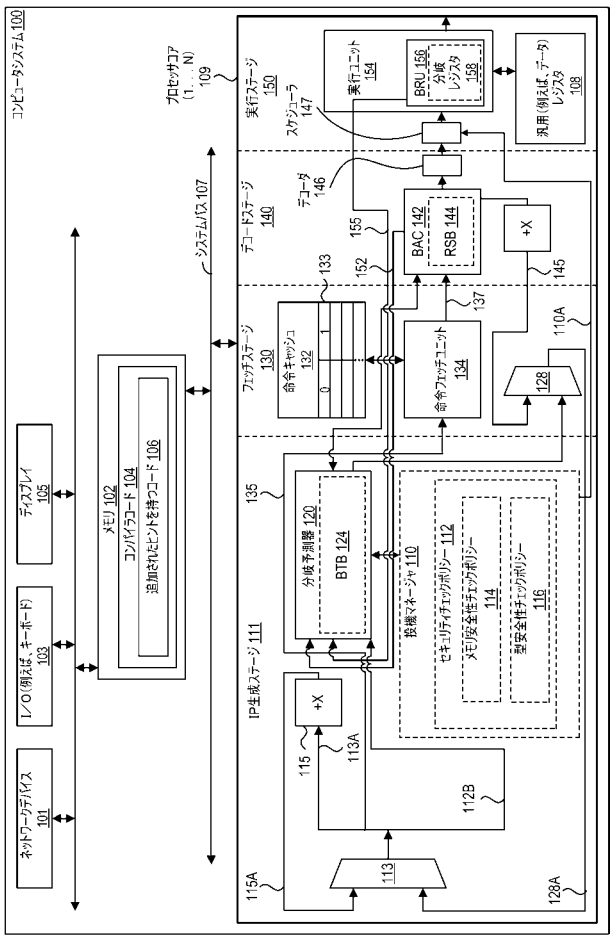
［項目２３］

上記１または複数の関連付けられたチェックは、上記ハードウェアプロセッサのアーキテクチャ仕様の完全準拠チェックより少ない、項目１７に記載の非一時的機械可読媒体。

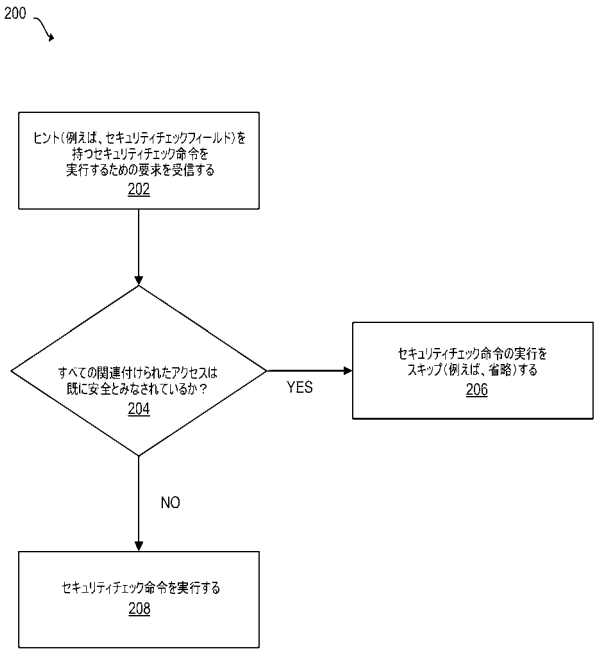
［項目２４］

上記命令は、プログラム順序において後続するメモリアクセス命令に関連付けられたセキュリティチェック命令である、項目１７に記載の非一時的機械可読媒体。

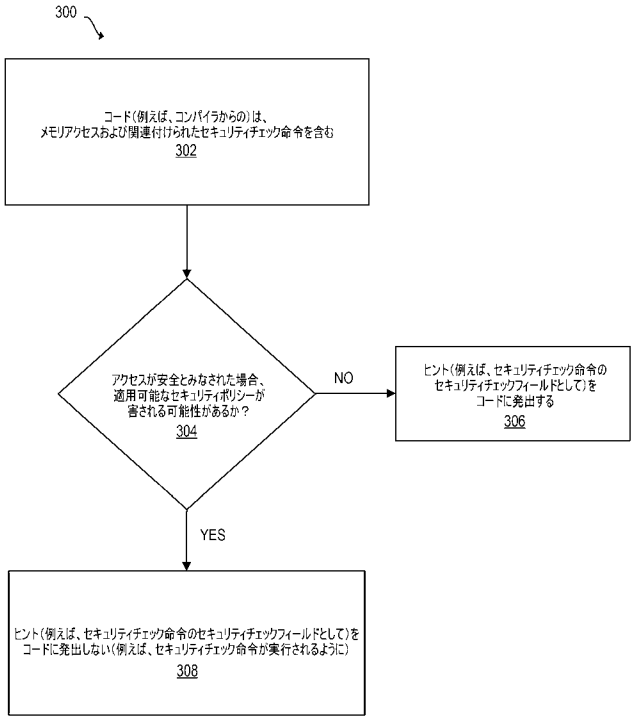
【図１】



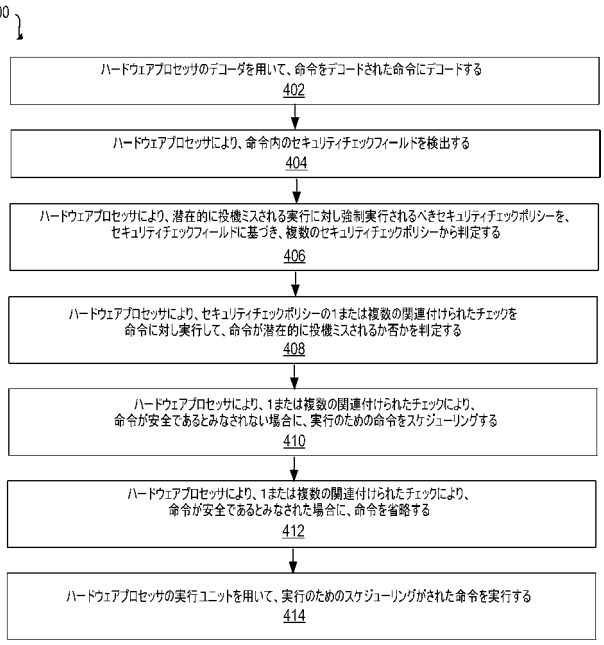
【図２】



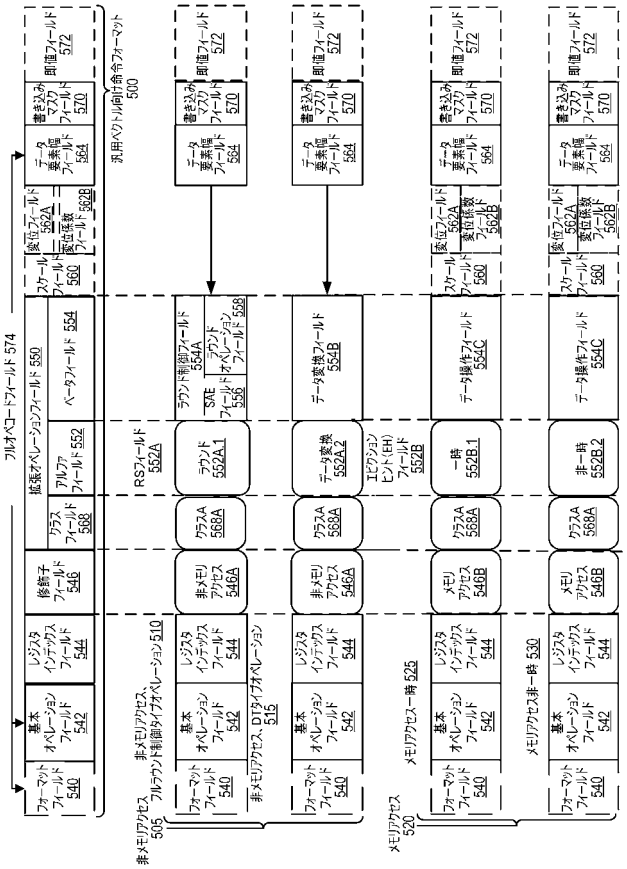
【図３】


300
コード（例えば、コンパイラからの）は、
メモリアクセスおよび関連付けられたセキュリティチェック命令を含む
302
アクセスが安全とみなされた場合、
適用可能なセキュリティポリシーが
害される可能性があるか？
304
NO
ヒント（例えば、セキュリティチェック命令の
セキュリティチェックフィールドとして）を
コードに発出する
306
YES
ヒント（例えば、セキュリティチェック命令のセキュリティチェックフィールドとして）を
コードに発出しない（例えば、セキュリティチェック命令が実行されるように）
308

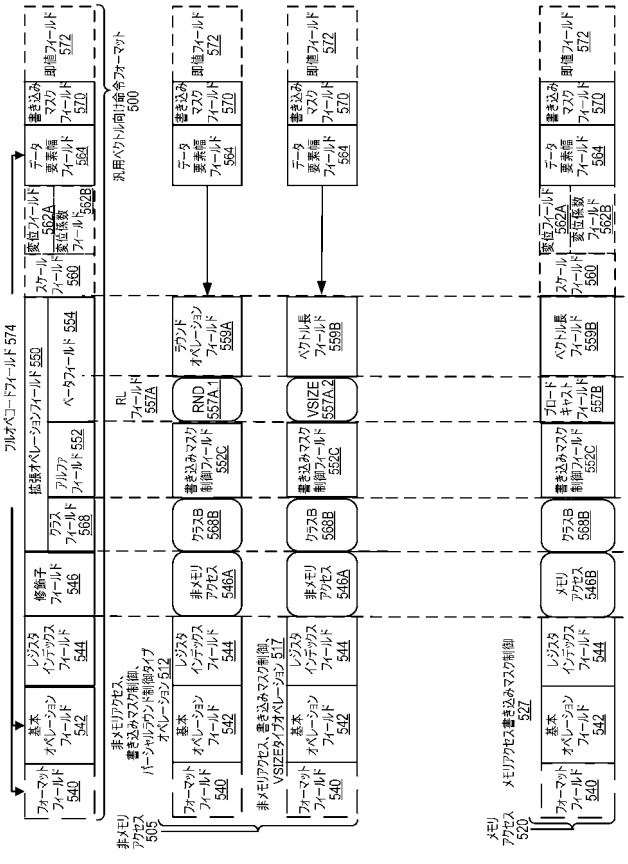
【図４】


400
ハードウェアプロセッサのデコーダを用いて、命令をデコードされた命令にデコードする
402
ハードウェアプロセッサにより、命令内のセキュリティチェックフィールドを検出する
404
ハードウェアプロセッサにより、潜在的に投機ミスされる実行に対し強制実行されるべきセキュリティチェックポリシーを、
セキュリティチェックフィールドに基づき、複数のセキュリティチェックポリシーから判定する
406
ハードウェアプロセッサにより、セキュリティチェックポリシーの1または複数の関連付けられたチェックを
命令に対し実行して、命令が潜在的に投機ミスされるか否かを判定する
408
ハードウェアプロセッサにより、1または複数の関連付けられたチェックにより、
命令が安全であるとみなされない場合に、実行のための命令をスケジューリングする
410
ハードウェアプロセッサにより、1または複数の関連付けられたチェックにより、
命令が安全であるとみなされた場合に、命令を省略する
412
ハードウェアプロセッサの実行ユニットを用いて、実行のためのスケジューリングがされた命令を実行する
414

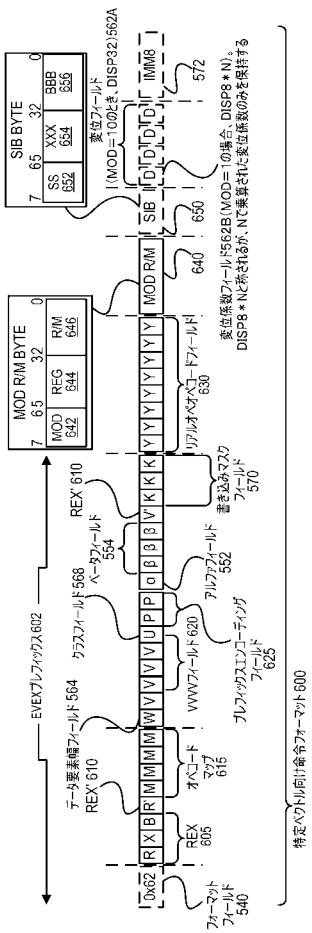
【図５Ａ】


フルオペコードフィールド 574
拡張オペレーションフィールド 550
ベータフィールド 554
アルファフィールド 552
クラスフィールド 568
修飾子フィールド 546
レジスタインデックスフィールド 544
基本オペレーションフィールド 542
フォーマットフィールド 540
スケールフィールド 560
変位フィールド 562A
変位係数フィールド 562B
データ要素幅フィールド 564
書き込みマスクフィールド 570
即値フィールド 572
汎用ベクトル向け命令フォーマット 500
RSフィールド 552A
ラウンド制御フィールド 554A
SAEフィールド 556
ラウンドオペレーションフィールド 558
ラウンド 562A.1
データ変換 562A.2
EHフィールド 552B
一時 562B.1
非一時 562B.2
データ変換フィールド 554B
データ操作フィールド 554C
クラスA 568A
クラスB 568B
非メモリアクセス 546A
メモリアクセス 546B
非メモリアクセス、フルラウンド制御タイプオペレーション 510
非メモリアクセス、DTタイプオペレーション 516
メモリアクセス－時 525
メモリアクセス非一時 530
非メモリアクセス 505
メモリアクセス 520

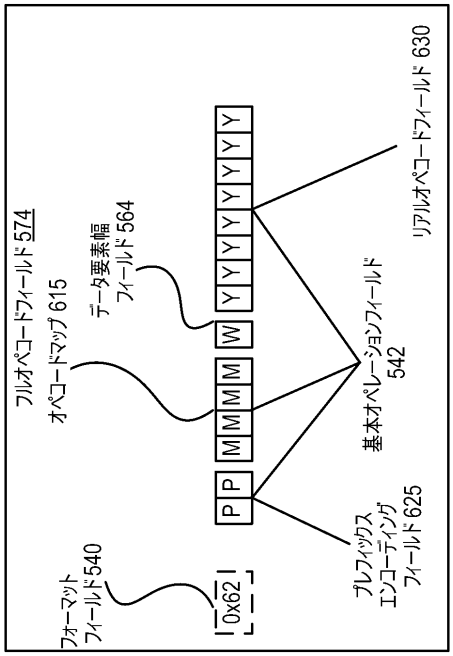
【図５Ｂ】


フルオペコードフィールド 574
拡張オペレーションフィールド 550
ベータフィールド 554
アルファフィールド 552
RLフィールド 557A
RND 557A.1
VSIZE 557A.2
ラウンドオペレーションフィールド 559A
ベクトル長フィールド 559B
ブロードキャストフィールド 557B
クラスフィールド 568
修飾子フィールド 546
レジスタインデックスフィールド 544
基本オペレーションフィールド 542
フォーマットフィールド 540
スケールフィールド 560
変位フィールド 562A
変位係数フィールド 562B
データ要素幅フィールド 564
書き込みマスクフィールド 570
即値フィールド 572
汎用ベクトル向け命令フォーマット 500
書き込みマスク制御フィールド 552C
クラスA 568A
クラスB 568B
非メモリアクセス 546A
メモリアクセス 546B
非メモリアクセス、書き込みマスク制御、パーシャルラウンド制御タイプオペレーション 512
非メモリアクセス、書き込みマスク制御、VSIZEタイプオペレーション 517
メモリアクセス、書き込みマスク制御 527
非メモリアクセス 505
メモリアクセス 520

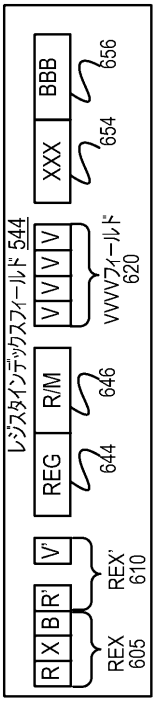
【図６Ａ】



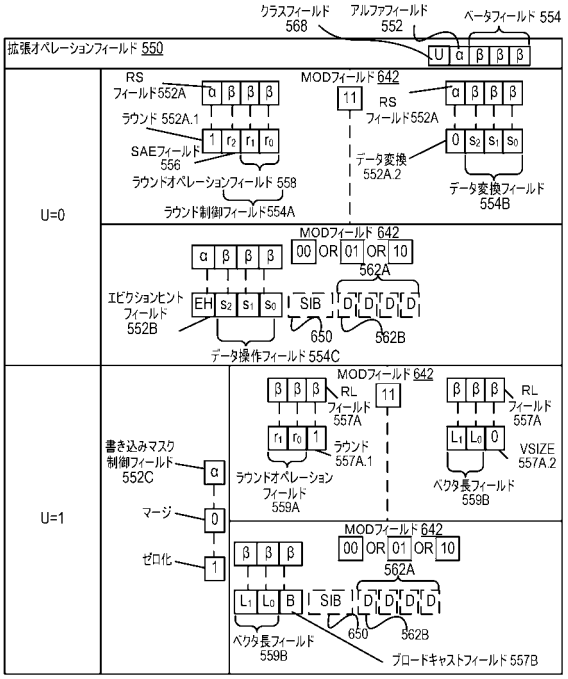
【図６Ｂ】



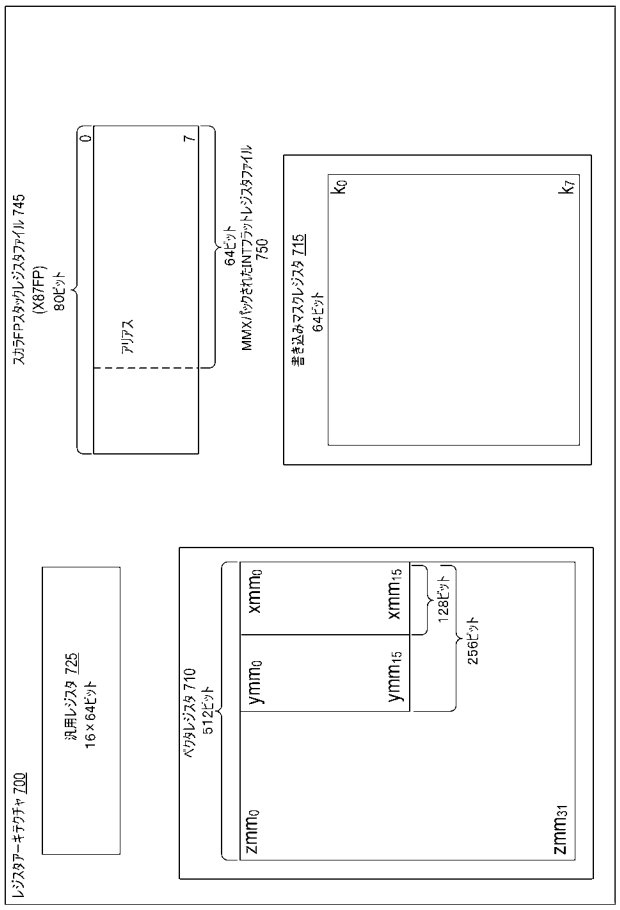
【図６Ｃ】



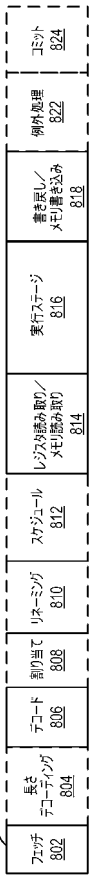
【図６Ｄ】



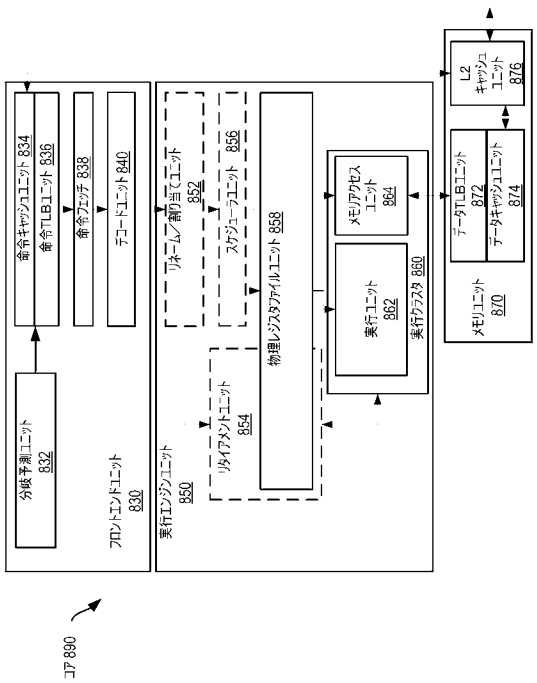
【図７】



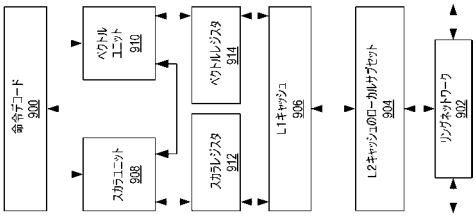
【図８Ａ】



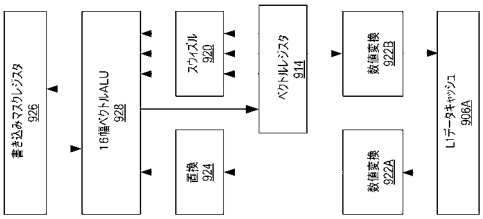
【図８Ｂ】



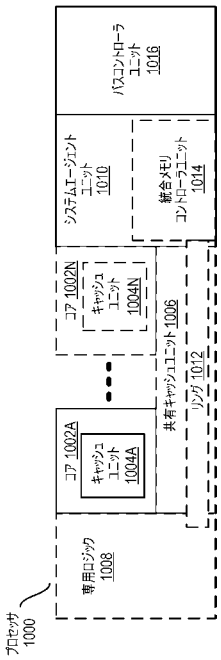
【図９Ａ】



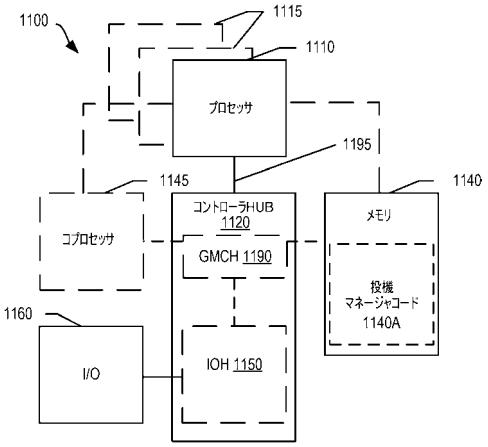
【図９Ｂ】



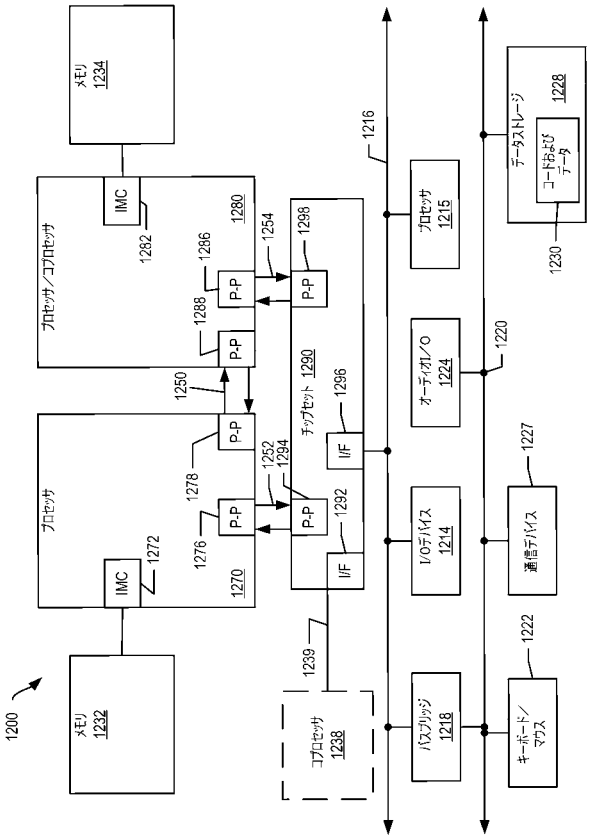
【図１０】



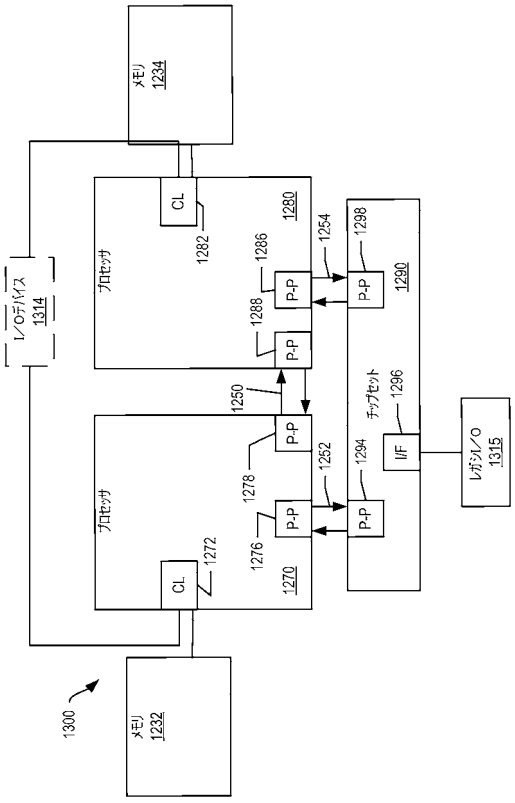
【図１１】



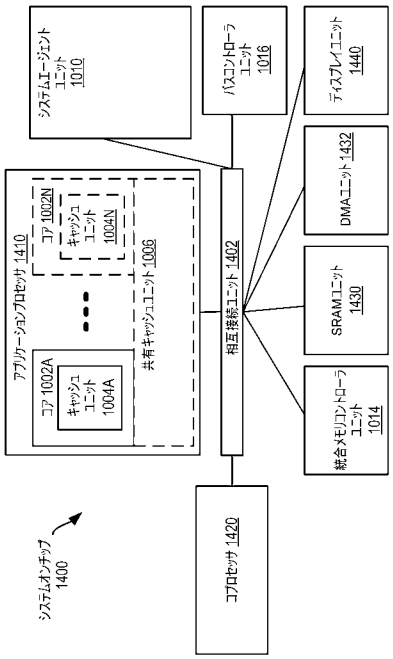
【図１２】



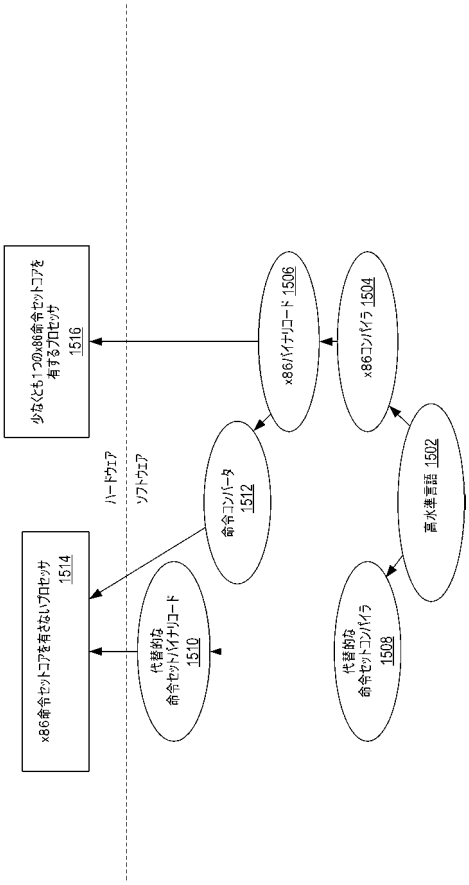
【図１３】



【図１４】


システムオンチップ
1400
アプリケーションプロセッサ 1410
コア 1002A
キャッシュユニット 1004A
コア 1002N
キャッシュユニット 1004N
共有キャッシュユニット 1006
相互接続ユニット 1402
コプロセッサ 1420
システムエージェントユニット 1010
バスコントローラユニット 1016
統合メモリコントローラユニット 1014
SRAMユニット 1430
DMAユニット 1432
ディスプレイユニット 1440

【図１５】


ハードウェア
ソフトウェア
少なくとも1つのx86命令セットコアを有するプロセッサ 1516
x86バイナリコード 1506
x86コンパイラ 1504
命令コンバータ 1512
高水準言語 1502
x86命令セットコアを有さないプロセッサ 1514
代替的な命令セットバイナリコード 1510
代替的な命令セットコンパイラ 1508

【外国語明細書】
2021057006000001.pdf