(19)日本国特許庁(JP)

(12)特許公報(B2)

(11)特許番号
特許第7000326号
(P7000326)

(45)発行日　令和4年1月19日(2022.1.19)

(24)登録日　令和3年12月27日(2021.12.27)

(51)国際特許分類		FI		
G06F 21/53 (2013.01)		G06F	21/53	
G06F 9/30 (2018.01)		G06F	9/30	370
G06F 12/0875(2016.01)		G06F	12/0875	100
G06F 21/55 (2013.01)		G06F	21/55	
G06F 21/56 (2013.01)		G06F	21/56	360

請求項の数　20　（全205頁）　最終頁に続く

(21)出願番号　特願2018-531549(P2018-531549)
(86)(22)出願日　平成28年12月12日(2016.12.12)
(65)公表番号　特表2019-504403(P2019-504403A)
(43)公表日　平成31年2月14日(2019.2.14)
(86)国際出願番号　PCT/US2016/066188
(87)国際公開番号　WO2017/106101
(87)国際公開日　平成29年6月22日(2017.6.22)
審査請求日　令和1年12月12日(2019.12.12)
(31)優先権主張番号　15/168,689
(32)優先日　平成28年5月31日(2016.5.31)
(33)優先権主張国・地域又は機関
米国(US)
(31)優先権主張番号　62/268,639
(32)優先日　平成27年12月17日(2015.12.17)
最終頁に続く

(73)特許権者　591044474
ザ・チャールズ・スターク・ドレイパー・ラボラトリー・インコーポレイテッド
アメリカ合衆国 02139 マサチューセッツ州、ケンブリッジ、テクノロジー・スクエア 555
(74)代理人　100108453
弁理士　村山 靖彦
(74)代理人　100110364
弁理士　実広 信哉
(74)代理人　100133400
弁理士　阿部 達彦
(72)発明者　アンドレ・デホン
アメリカ合衆国・ペンシルベニア・19139・フィラデルフィア・サンサム・
最終頁に続く

(54)【発明の名称】　メタデータ処理のための技法

(57)【特許請求の範囲】

【請求項１】

コード実行領域及びメタデータ処理領域を含むコンピュータシステムのプロセッサによって命令を処理する方法であって、

メタデータ処理のために、関連するメタデータタグとともに現在の命令を受信するステップであって、前記メタデータ処理が前記現在の命令を含む前記コード実行領域から隔離された前記メタデータ処理領域において実行される、ステップと、

前記メタデータ処理領域において、かつ前記メタデータタグおよび前記現在の命令に従って、ルールキャッシュに前記現在の命令のためのルールが存在するかどうかを判定するステップであって、前記ルールキャッシュが、前記メタデータ処理により使用されるメタデータについてのルールを含み、前記ルールは前記現在の命令のために許容される動作を定義する、ステップと、

前記ルールキャッシュに前記現在の命令のためのルールが存在しないと判定したことに応答して、前記メタデータ処理領域においてルールキャッシュミス処理を実行するステップであって、

前記現在の命令の実行が許容されるか禁止されるかを判定するステップと、

前記現在の命令が前記コード実行領域において実行されることが許容されると判定したことに応答して、前記現在の命令のための新しいルールを生成するステップと、

生成された前記新しいルールのためのタグ値をレジスタに書き込むステップと、

前記メタデータ処理領域における前記レジスタへの書込みに応答して、前記新しいルール

を前記ルールキャッシュに挿入するステップと
を備える、ステップとを備え、
前記現在の命令の実行が許容されることを判定するステップが、
前記現在の命令の特権レベルを判定するステップと、
前記メタデータ処理領域の複数のセキュリティモードのなかから実行されるべき現行セキュリティモードを決定するためにタグモード制御ステータスレジスタにアクセスするステップと、
前記現在の命令の前記特権レベルが、決定された前記現行セキュリティモードで実行されることが許容されると判定するステップと、
を備える、方法。

【請求項２】
前記現在の命令のための前記ルールを選択するために使用される第1のメタデータが、前記メタデータ処理により使用される複数の制御ステータスレジスタの少なくとも一つの第1の部分に記憶され、前記複数の制御ステータスレジスタの少なくとも一つの前記第1の部分が、前記現在の命令のための複数のメタデータタグを前記メタデータ処理領域に伝えるために使用され、前記複数のメタデータタグが前記メタデータ処理領域においてメタデータとして使用される、請求項1に記載の方法。

【請求項３】
前記レジスタが、前記メタデータ処理により使用される前記複数の制御ステータスレジスタの第1の制御ステータスレジスタであり、前記複数の制御ステータスレジスタの前記第1の部分が、前記メタデータ処理領域から前記ルールキャッシュに前記複数のメタデータタグを伝えるために使用される、請求項2に記載の方法。

【請求項４】
前記複数のメタデータタグが前記現在の命令のためのものである、請求項3に記載の方法。

【請求項５】
別のメタデータタグを前記第1の制御ステータスレジスタに書き込んだことに応答して、前記新しいルールが前記ルールキャッシュへと挿入され、前記別のメタデータタグが前記現在の命令の結果に付けられ、前記結果が宛先レジスタまたはメモリ位置のいずれかである、請求項4に記載の方法。

【請求項６】
前記複数の制御ステータスレジスタが、
すべての他の生成されるメタデータタグがそこから導出される初期メタデータタグを含む、ブートストラップタグ制御ステータスレジスタと、
デフォルトメタデータタグを指定するデフォルトタグ制御ステータスレジスタと、
公開であり信頼されないものとして分類される命令およびデータをタグ付けするために使用される公開信頼不可メタデータタグを指定する、公開信頼不可制御ステータスレジスタと、
オペグループについての情報と様々なオペコードのためのケア情報とを含むテーブルに書き込まれるデータを含む、オペグループ値制御ステータスレジスタと、
前記オペグループ値制御ステータスレジスタのデータが書き込まれる前記テーブルの中の位置を指定する、オペグループアドレス制御ステータスレジスタと、
pumpフラッシュ制御ステータスレジスタであって、前記pumpフラッシュ制御ステータスレジスタへの書込みが前記ルールキャッシュのフラッシュをトリガする、pumpフラッシュ制御ステータスレジスタと
のうちの任意の1つまたは複数を含む、請求項2に記載の方法。

【請求項７】
前記複数の制御ステータスレジスタが、実行されるべき前記現行セキュリティモードを示す前記タグモード制御ステータスレジスタを含む、請求項2に記載の方法。

【請求項８】
前記タグモード制御ステータスレジスタが、メタデータ処理がいつ不関与であり、それに

より、1つまたは複数の定義されたポリシーのルールがメタデータ処理によって実施されないかを示す、請求項7に記載の方法。
【請求項９】
前記タグモード制御ステータスレジスタが、メタデータ処理の前記現在のモードを示すために前記複数のセキュリティモードのうちの1つに設定され、前記複数のセキュリティモードが、メタデータ処理がすべての結果にデフォルトタグを書き込むオフ状態と、命令が1つまたは複数の指定された特権レベルで前記コード実行領域において実行されるときにメタデータ処理が関与しており動作可能であることを示す関与状態とを含む、請求項8に記載の方法。
【請求項１０】
前記ルールキャッシュミス処理が、メタデータ処理が不関与である前記複数のセキュリティモードのうちの第1のモードにおいて実行される、請求項9に記載の方法。
【請求項１１】
前記複数のセキュリティモードが、命令がユーザ特権レベルで前記コード実行領域において実行するときにのみメタデータ処理が関与していることを示す第1の状態と、命令がユーザ特権レベルまたはスーパーバイザー特権レベルで前記コード実行領域において実行するときにのみメタデータ処理が関与していることを示す第2の状態と、命令がユーザ特権レベル、スーパーバイザー特権レベル、またはハイパーバイザー特権レベルで前記コード実行領域において実行するときにのみメタデータ処理が関与していることを示す第3の状態と、命令がユーザ特権レベル、スーパーバイザー特権レベル、ハイパーバイザー特権レベル、またはマシン特権レベルで前記コード実行領域において実行するときにのみメタデータ処理が関与していることを示す第4の状態とを含む、請求項9に記載の方法。
【請求項１２】
前記メタデータ処理が関与しているか不関与であるかが、前記現在の命令の前記特権レベルと前記現行セキュリティモードとに従って前記現在の命令の実行が許容されるか禁止されるかどうかに従って判定され、メタデータ処理が不関与であるときに1つまたは複数の定義されるポリシーのルールが実施されず、メタデータ処理が関与しているときに前記ルールが実施される、請求項7に記載の方法。
【請求項１３】
テーブルが、命令セットのオペコードを対応するオペグループおよびビットベクトル情報にマッピングする情報を含み、前記オペグループが、前記メタデータ処理領域によって同様に扱われる関連するオペコードのグループを示し、前記ビットベクトル情報が、前記メタデータ処理領域に関する特定の入力および出力が前記オペコードを処理することに関連して使用されるかどうかを示し、前記テーブルが、許容可能なオペコードビットの最大の数より少ないオペコードビットの第1の部分を使用してインデクシングされ、前記最大の数が前記命令セットのオペコードのビットの数の上限を示す、請求項2に記載の方法。
【請求項１４】
前記複数の制御ステータスレジスタの前記第1の部分が、前記現在の命令のための追加のオペコードビットがもしあればそれを含む、拡張オペコード制御ステータスレジスタを含み、前記現在の命令が可変長オペコードを有する前記命令セットに含まれ、前記命令セットの各オペコードが、任意選択で前記追加のオペコードビットを含み、前記拡張オペコード制御ステータスレジスタが、前記現在の命令のための前記追加のオペコードビットがもしあればそれを含む、請求項13に記載の方法。
【請求項１５】
前記テーブルを使用してマッピングされる各オペコードに対して、前記各オペコードに対応する結果ビットベクトルがあり、前記結果ビットベクトルが、もしあれば、前記拡張オペコード制御ステータスレジスタの中の前記追加のオペコードビットのどの部分がメタデータ処理のために前記各オペコードとともに使用されるかを示す、請求項14に記載の方法。
【請求項１６】

前記現在の命令が、単一のメタデータタグと関連付けられるメモリの単一のワードに記憶される複数の命令のうちの1つであり、前記単一のメタデータタグが、前記単一のワードに含まれる前記複数の命令と関連付けられる、請求項2に記載の方法。
【請求項１７】
前記複数の制御ステータスレジスタが、前記単一のワードに記憶されている前記複数の命令のいずれが前記現在の命令であるかを示す、サブ命令制御ステータスレジスタを含む、請求項16に記載の方法。
【請求項１８】
前記単一のメタデータタグが、前記単一のワードの中の前記複数の命令の各々に対する異なるメタデータタグを含む第1のメモリ位置への第1のポインタである、請求項17に記載の方法。
【請求項１９】
前記複数の命令のうちの第1の命令のための前記第1のメモリ位置に記憶されている少なくとも第1のメタデータタグが、前記第1の命令のためのメタデータタグ情報を含む第2のメモリ位置への第2のポインタを含む、請求項18に記載の方法。
【請求項２０】
前記第1の命令のための前記メタデータタグ情報が複雑な構造を含み、前記複雑な構造が、少なくとも1つのスカラーデータフィールドと、第3のメモリ位置への少なくとも1つのポインタフィールドとを備える、請求項19に記載の方法。
【発明の詳細な説明】
【技術分野】
【０００１】
関連出願の相互参照
本出願は、2015年12月17日に出願された米国仮出願第62/268,639号、SOFTWARE DEFINED METADATA PROCESSING、および2015年12月21日に出願された米国仮出願第62/270,187号、SOFTWARE DEFINED METADATA PROCESSINGの優先権を主張する、2016年5月31日に出願された米国出願第15/168,689号の優先権を主張し、これらのすべての全体が参照によって本明細書に組み込まれる。
【０００２】
本出願は全般にデータ処理に関し、より具体的には、メタデータ処理のためのプログラム可能ユニットに関する。
【背景技術】
【０００３】
今日のコンピュータシステムは、セキュアにするのが難しいことで有名である。従来のプロセッサアーキテクチャは、たとえば、バッファオーバーフローおよびポインタ偽造などの、高水準の抽象化に違反する様々な挙動を許容する。プログラミング言語とハードウェアとのギャップを埋めることはソフトウェア任せであることがあり、完璧な抽象化を実施することのコストはしばしば高すぎると見なされる。
【０００４】
実行の間にメタデータを伝播し、安全性の侵害および悪意のある攻撃をそれらが発生するにつれて捉えるポリシーを実施することの価値を、一部の最近の試みが示している。これらのポリシーは、ソフトウェアにおいて実施され得るが、性能および/またはコストなどにおいて望ましくない大きなオーバーヘッドを通常は招き、このことは、ポリシーの展開を妨げ、または他には、より保護の程度が低い粗悪な類似物を用いる動機を与える。不変のポリシーに対するハードウェアのサポートは、オーバーヘッドを許容可能なレベルに減らし、悪意のあるコードまたはマルウェア攻撃により実行され得るような望ましくないコードの侵入の大部分を防ぐことができる。たとえば、Intelは最近、境界チェックおよび隔離のためのハードウェアを発表した。これらは今日の攻撃の多くを軽減するが、完全にシステムをセキュアにするには、メモリの安全性および隔離以上のものが必要である。攻撃は、あらゆる形態の残存する脆弱性を利用するように高速に進化する。

【先行技術文献】
【非特許文献】
【０００５】
【文献】K. Mai、R. Ho、E. Alon、D. Liu、Y. Kim、D. Patil、およびM. Horowitz、Architecture and Circuit Techniques for a 1.1GHz 16-kb Reconfigurable Memory in 0.18um-CMOS、IEEE J. Solid-State Circuits、40(1):261-275、2005年1月
PROGRAMMING THE PUMP, Hardware-Assisted Micro-Policies for Security
The RISC-V Instruction Set Manual Vol. I, User-Level ISA, Version 2.0、2014年5月6日、Waterman、Andrew他
The RISC-V Instruction Set Manual Volume II: Privileged Architecture, Version 1.7、2015年5月9日
【発明の概要】
【発明が解決しようとする課題】
【０００６】
したがって、この変化し続ける状況に迅速に適応できる、フレキシブルなセキュリティアーキテクチャが必要である。そのようなアーキテクチャに、最小限のオーバーヘッドで、ソフトウェアにより定義されるメタデータの処理をサポートさせることが望ましい。そのようなアーキテクチャは、メタデータに割り振られるビット数に対する目に見える厳しい制限を課すことなく、あらゆる数およびタイプのポリシーを全般にサポートし実施するように拡張可能であることが望ましい。ポリシーを実施し、たとえば悪意のあるコードまたはマルウェア攻撃などの、そのようなポリシーの侵害を捉えるために、実行の間にメタデータが伝播され得る。
【課題を解決するための手段】
【０００７】
本明細書において説明される技法の一態様に従うのは、メタデータ処理のために、関連するメタデータタグとともに現在の命令を受信するステップであって、前記メタデータ処理が現在の命令を含むコード実行領域から隔離されたメタデータ処理領域において実行される、ステップと、メタデータ処理領域において、かつメタデータタグおよび現在の命令に従って、現在の命令のためのルールキャッシュにルールが存在するかどうかを判定するステップであって、前記ルールキャッシュが、許容される動作を定義するために前記メタデータ処理によって使用されるメタデータについてのルールを含む、ステップと、現在の命令のためのルールキャッシュにルールが存在しないと判定したことに応答して、メタデータ処理領域においてルールキャッシュミス処理を実行するステップであって、現在の命令の実行が許容されるかどうかを判定するステップを備える、ステップと、現在の命令がコード実行領域において実行されることが許容されると判定したことに応答して、現在の命令のための新しいルールを生成するステップと、レジスタに書き込むステップと、レジスタへの書込みに応答して、新しいルールをルールキャッシュに挿入するステップとを備える、命令を処理する方法である。現在の命令のためのルールを選択するために使用される第1のメタデータは、メタデータ処理により使用される複数の制御ステータスレジスタの第1の部分に記憶されてよく、複数の制御ステータスレジスタの第1の部分は、現在の命令のための複数のメタデータタグをメタデータ処理領域に伝えるために使用されてよく、前記複数のメタデータタグは、メタデータ処理領域においてデータとして使用されてよい。レジスタは、メタデータ処理によって使用される複数の制御ステータスレジスタの第1の制御ステータスレジスタであってよく、複数の制御ステータスレジスタの第1の部分は、メタデータ処理領域からルールキャッシュに複数のメタデータタグを伝えるために使用されてよい。複数のメタデータタグは、現在の命令のためのものであり得る。別のメタデータタグを第1の制御ステータスレジスタに書き込んだことに応答して、新しいルールがルールキャッシュに挿入されてよく、別のメタデータタグが現在の命令の結果に対して付けられてよく、この結果は宛先レジスタまたはメモリ位置のいずれかであってよい。複数の制御ステータスレジスタは、すべての他の生成されるメタデータタグがそこから導出され

る初期メタデータタグを含む、ブートストラップタグ制御ステータスレジスタ、デフォルトのメタデータタグを指定するデフォルトタグ制御ステータスレジスタ、公開であり信頼されないものとして分類される命令およびデータをタグ付けするために使用される公開信頼不可メタデータタグを指定する、公開信頼不可制御ステータスレジスタ、オペグループについての情報および様々なオペコードのためのケア情報を含むテーブルに書き込まれるデータを含む、オペグループ値制御ステータスレジスタ、オペグループ値制御ステータスレジスタのデータが書き込まれるテーブルの中の位置を指定するオペグループアドレス制御ステータスレジスタ、ならびに、pumpフラッシュ制御ステータスレジスタのうちの、任意の1つまたは複数を含むことがあり、pumpフラッシュ制御ステータスレジスタへの書込みはルールキャッシュのフラッシュをトリガする。複数の制御ステータスレジスタが、メタデータ処理の現在のモードを示すタグモード制御ステータスレジスタを含み得る。タグモード制御ステータスレジスタは、メタデータ処理がいつ不関与であり、それにより、1つまたは複数の定義されたポリシーのルールがメタデータ処理によって実施されないかを、示し得る。タグモード制御ステータスレジスタは、メタデータ処理の現在のモードを示すために、許容される状態の定義された集合のうちの1つに設定され得る。許容される状態は、メタデータ処理がすべての結果にデフォルトタグを書き込む状態であるオフ状態、および命令が1つまたは複数の指定された特権レベルでコード領域において実行されるときにメタデータ処理が関与しており動作可能であることを示す状態のうちの、いずれかを含み得る。ルールキャッシュミス処理は、メタデータ処理が不関与である許容される状態の定義された集合のうちの第1の状態において実行され得る。許容される状態は、命令がユーザ特権レベルでコード領域において実行するときにのみメタデータ処理が関与していることを示す第1の状態と、命令がユーザ特権レベルまたはスーパーバイザー特権レベルでコード領域において実行するときにのみメタデータ処理が関与していることを示す第2の状態と、命令がユーザ特権レベル、スーパーバイザー特権レベル、またはハイパーバイザー特権レベルでコード領域において実行するときにのみメタデータ処理が関与していることを示す第3の状態と、命令がユーザ特権レベル、スーパーバイザー特権レベル、ハイパーバイザー特権レベル、またはマシン特権レベルでコード領域において実行するときにのみメタデータ処理が関与していることを示す第4の状態とを含み得る。メタデータ処理が関与しているか不関与であるかは、タグモード制御ステータスレジスタの現在のタグモードと、コード領域において実行するコードの現在の特権レベルとの組合せに従って判定されることがあり、メタデータ処理が不関与であるときに1つまたは複数の定義されるポリシーのルールは実施されないことがあり、メタデータ処理が関与しているときにルールは実施されることがある。テーブルは、命令セットのオペコードを対応するオペグループおよびビットベクトル情報にマッピングする情報を含み得る。オペグループは、メタデータ処理領域によって同様に扱われる関連するオペコードのグループを示し得る。ビットベクトル情報は、メタデータ処理領域に関する特定の入力および出力がオペコードを処理することに関連して使用されるかどうかを示し得る。テーブルは、許容可能なオペコードビットの最大の数より少ないオペコードビットの第1の部分を使用してインデクシングされることがあり、この最大の数は命令セットのオペコードのビット数に対する上限を示すことがある。複数の制御ステータスレジスタの第1の部分は、現在の命令のための追加のオペコードビットがもしあればそれを含む、拡張オペコード制御ステータスレジスタを含むことがあり、現在の命令は可変長オペコードを有する命令セットに含まれることがあり、命令セットの各オペコードは追加のオペコードビットを任意選択で含むことがあり、拡張オペコード制御ステータスレジスタは現在の命令のための追加のオペコードビットがもしあればそれを含む。テーブルを使用してマッピングされる各オペコードに対して、前記各オペコードに対応する結果ビットベクトルがあり、結果ビットベクトルは、もしあれば、拡張オペコード制御ステータスレジスタの中の追加のオペコードビットのどの部分がメタデータ処理のために前記各オペコードとともに使用されるかを示し得る。現在の命令は、単一のメタデータタグと関連付けられるメモリの単一のワードに記憶される複数の命令のうちの1つであることがあり、前記単一のメタデータタグは、単一のワードに含まれる

複数の命令と関連付けられることがある。複数の制御ステータスレジスタは、単一のワードに記憶されている複数の命令のいずれが現在の命令であるかを示す、サブ命令制御ステータスレジスタを含み得る。単一のメタデータタグは、単一のワードの中の複数の命令の各々に対する異なるメタデータタグを含む第1のメモリ位置への第1のポインタであり得る。少なくとも、複数の命令のうちの第1の命令のための第1のメモリ位置に記憶される第1のメタデータタグは、第1の命令のためのメタデータタグ情報を含む第2のメモリ位置への第2のポインタを含み得る。第1の命令のためのメタデータタグ情報は複雑な構造を含み得る。複雑な構造は、少なくとも1つのスカラーデータフィールドと、第3のメモリ位置への少なくとも1つのポインタフィールドとを含み得る。

【０００８】

本明細書の技法の別の態様に従うのは、実行されると、命令を処理する方法を実行するコードを備える非一時的コンピュータ可読媒体であり、この方法は、メタデータ処理のために、関連するメタデータタグとともに現在の命令を受信するステップであって、前記メタデータ処理が現在の命令を含むコード実行領域から隔離されたメタデータ処理領域において実行される、ステップと、メタデータ処理領域において、かつメタデータタグおよび現在の命令に従って、現在の命令のためのルールキャッシュにルールが存在するかどうかを判定するステップであって、前記ルールキャッシュが許容される動作を定義するために前記メタデータ処理によって使用されるメタデータについてのルールを含む、ステップと、現在の命令のためのルールキャッシュにルールが存在しないと判定したことに応答して、メタデータ処理領域においてルールキャッシュミス処理を実行するステップであって、現在の命令の実行が許容されるかどうかを判定するステップを備える、ステップと、現在の命令がコード実行領域において実行されることが許容されると判定したことに応答して、現在の命令のための新しいルールを生成するステップと、レジスタに書き込むステップと、レジスタへの書込みに応答して、新しいルールをルールキャッシュに挿入するステップとを備える。

【０００９】

本明細書の技法の別の態様に従うのは、プロセッサと、プロセッサによって実行されると命令を処理する方法を実行するコードを記憶したメモリとを備えるシステムであり、この方法は、メタデータ処理のために、関連するメタデータタグとともに現在の命令を受信するステップであって、前記メタデータ処理が現在の命令を含むコード実行領域から隔離されたメタデータ処理領域において実行される、ステップと、メタデータ処理領域において、かつメタデータタグおよび現在の命令に従って、現在の命令のためのルールキャッシュにルールが存在するかどうかを判定するステップであって、前記ルールキャッシュが許容される動作を定義するために前記メタデータ処理によって使用されるメタデータについてのルールを含む、ステップと、現在の命令のためのルールキャッシュにルールが存在しないと判定したことに応答して、メタデータ処理領域においてルールキャッシュミス処理を実行するステップであって、現在の命令の実行が許容されるかどうかを判定するステップを備える、ステップと、現在の命令がコード実行領域において実行されることが許容されると判定したことに応答して、現在の命令のための新しいルールを生成するステップと、レジスタに書き込むステップと、レジスタへの書込みに応答して、新しいルールをルールキャッシュに挿入するステップとを備える。プロセッサは、縮小命令セットコンピューティングアーキテクチャにおけるパイプラインプロセッサであり得る。

【００１０】

本明細書の技法の別の態様に従うのは、現在の命令を含むコード実行領域から隔離されたメタデータ処理領域において実行されるメタデータ処理のための現在の命令を受信するステップと、現在の命令のためのメタデータに関連するメタデータ処理領域によって、1つまたは複数のポリシーの集合に従った現在の命令の実行を許容するかどうかを判定するステップとを備える、命令を処理する方法であって、現在の命令は第1のルーチンのスタックフレームの第1の位置にアクセスし、現在の命令およびスタックフレームの位置は関連するメタデータタグを有し、1つまたは複数のポリシーの集合は、スタック保護を提供し

第1のルーチンのスタックフレームの記憶位置を含むスタック記憶位置への不適切なアクセスを防ぐ、スタック保護ポリシーを含む。スタック保護ポリシーは、第1のルーチンのスタックフレームの第1の位置にアクセスする現在の命令のメタデータ処理において使用される第1のルールを含み得る。第1のルールは、第1の位置が第1のルーチンのスタック位置であることを示すメタデータを第1の位置が有し、現在の命令が第1のルーチンに含まれる場合、現在の命令の実行を許容し得る。現在の命令は、第1のルーチンの特定の呼出しインスタンスによって使用されることがあり、スタック保護ポリシーは現在の命令のメタデータ処理において使用される第1のルールを含むことがある。第1のルールは、現在の命令が第1のルーチンに含まれ第1のルーチンの特定の呼出しインスタンスによっても使用される場合、現在の命令の実行を許容し得る。第1のルールは、第1のルーチンの特定の呼出しインスタンスによる現在の命令の実行を許容するかどうかを判定するために、プログラムカウンタと関連付けられるとともに権限および能力のいずれかを示すメタデータを、調査することを含み得る。スタック保護ポリシーは、オブジェクトレベルの保護のいずれかを提供することができ、単一のスタックフレームの中の異なるオブジェクトは異なる色メタデータタグを有し、階層的なオブジェクトのための階層的なオブジェクト保護は複数のサブオブジェクトを含み、ここで単一のスタックフレームの複数のサブオブジェクトの各々は異なるメタデータタグを有する。方法は、新しいルーチン呼出しのために新しいスタックフレームを作成するステップと、厳密なオブジェクト初期化または遅延オブジェクト色付けに従って新しいスタックフレームのメモリ位置をタグ付けまたは色付けするステップとを含むことがあり、厳密なオブジェクト初期化は、新しいスタックフレームに情報を記憶する前に新しいスタックフレームの各メモリ位置を最初にタグ付けする1つまたは複数のルールのメタデータ処理をトリガする1つまたは複数の命令を実行する初期化処理を実行することを含み、遅延オブジェクト色付けは、特定のメモリ位置にデータを記憶する命令に応答してトリガされるルールのメタデータ処理に関連して新しいスタックフレームのその特定のメモリ位置をタグ付けする。1つまたは複数のポリシーは、特定のリターン位置へのリターンが特定の呼出しの後に行われるときにのみ有効であることを保証する、動的制御フロー整合性ポリシーの実施のためのルールの集合を含み得る。第1の位置は、リターン命令を含む呼び出されたルーチンに制御を移転する呼出し命令を含むことがあり、第2の位置は第2の命令を含むことがあり、ここで前記第2の位置は、呼び出されたルーチンのリターン命令を実行した結果として制御が移転されるリターンターゲット位置を示し得る。方法は、呼出し命令を含む第1の位置を第1のコードタグでタグ付けするステップと、リターンターゲット位置を示す第2の位置を第2のコードタグでタグ付けするステップと、第1のコードタグでタグ付けされる呼出し命令のために集合の第1のルールのメタデータ処理を実行するステップであって、第1のコードタグでタグ付けされた呼出し命令のための第1のルールのメタデータ処理が、リターンアドレスレジスタが第2の位置のための有効なリターンアドレスを含むことを示す有効リターンアドレスタグでリターンアドレスレジスタをタグ付けするステップを含み、呼出し命令の実行が、第2の位置に戻る能力を示すようにリターンアドレスレジスタのタグを更新する、ステップと、リターンアドレスレジスタが有効なリターンアドレス能力タグでタグ付けされる場合に、リターンアドレスレジスタに記憶されているリターンアドレスへ制御を移転するためにリターン命令の実行を許容する、呼び出されたルーチンのリターン命令のための集合の第2のルールのメタデータ処理を実行するステップであって、第2のルールが、リターン命令のランタイム実行に続く次の命令のために使用されるプログラムカウンタタグにリターンアドレスレジスタの有効なリターンアドレス能力タグを伝播する、ステップと、リターン命令のランタイム実行に続く第2の命令のための集合の第3のルールのメタデータ処理を実行するステップであって、第2の命令が第2のコードタグに等しいコードタグを有し、かつプログラムカウンタタグが有効なリターンアドレス能力タグである場合に、第3のルールのメタデータ処理が第2の命令の実行を許容し、第3のルールが、第2の命令のランタイム実行に続く次の命令のために使用されるプログラムカウンタタグをクリアする、ステップとを含み得る。

【００１１】

本明細書の技法の別の態様に従うのは、現在の命令を含むコード実行領域から隔離されたメタデータ処理領域において実行されるメタデータ処理のために現在の命令を受信するステップと、現在の命令のためのメタデータに関連するメタデータ処理領域によって、1つまたは複数のポリシーの集合に従って現在の命令の実行を許容するかどうかを判定するステップとを備える、命令を処理する方法であって、1つまたは複数のポリシーは、命令の完全なシーケンスの実行を、完全なシーケンスの最初の命令から完全なシーケンスの最後の命令まで指定された順序で実施するルールの集合を含む。方法は、第1の共有される物理ページを第1のプロセスの第1の仮想アドレス空間へとマッピングするステップと、第1の共有される物理ページを第2のプロセスのための第2の仮想アドレス空間へとマッピングするステップとを含むことがあり、前記第1の共有される物理ページが複数のメモリ位置を含み、複数のメモリ位置の各々がメタデータ処理領域におけるルール処理に関連して使用される複数のグローバルメタデータタグのうちの1つと関連付けられる。複数のグローバルメタデータタグは、少なくとも第1のプロセスおよび第2のプロセスを含む複数のプロセスによって共有されるメタデータタグの集合を示すことがあり、第1のプロセスと第2のプロセスの両方のために同じポリシーがメタデータ処理領域によって実施されることがある。メタデータ処理領域による同じポリシーの実施は、そうされなければ第2のプロセスのために同じポリシーによって許容されない動作を第1のプロセスが実行することを許容するためにメタデータを使用することがあり、プログラムカウンタは関連するプログラムカウンタタグを有することがあり、関連するプログラムカウンタタグの異なる値が、そうされなければ第2のプロセスのための同じポリシーによって許容されない動作を第1のプロセスが実行することを許容するために、同じポリシーのルールによって使用されることがある。方法は、アプリケーションの割振りルーチンによって第1の処理を実行して、アプリケーションの現在の色を使用してアプリケーションの次の色を生成するステップを含むことがあり、アプリケーションの現在の色はアプリケーションのためのアプリケーション固有の色シーケンスの現在の状態を示し、次の色はアプリケーションのためのアプリケーション固有の色シーケンスの次の状態を示し、現在の色は第1のアトムの第1のメタデータタグに記憶される。第1の処理は、第1の1つまたは複数の命令を実行することを含むことがあり、第1の1つまたは複数の命令は、メタデータ処理領域による1つまたは複数のルールを使用したメタデータ処理をトリガし、メタデータ処理領域による1つまたは複数のルールを使用したメタデータ処理は、現在の色を使用して次の色を生成し、第1のアトムの第1のメタデータタグに次の色を記憶することによってアプリケーションのためのアプリケーション固有の色シーケンスの現在の状態を更新する。第1の1つまたは複数の命令はアプリケーションの割振りルーチンに含まれてよく、第1のアトムはレジスタおよびメモリ位置のいずれかであってよい。アプリケーション固有の色シーケンスは、アプリケーションによる使用が可能な異なる色の無限のシーケンスであってよく、次の色はアプリケーションにより使用される1つまたは複数のメモリ位置の各々のためのタグ値として記憶されてよく、1つまたは複数のメモリ位置は割振りルーチンによって割り振られてよい。ルールの集合は第1のルールおよび第2のルールを含むことがあり、命令の完全なシーケンスは第1の命令および第2の命令を含むことがあり、第2の命令は第1の命令の直後に実行されることがある。方法は、第1の命令のための第1のルールのメタデータ処理を実行するステップであって、第1のルールのメタデータ処理が、第1の命令のランタイム実行に続く次の命令のために使用されるプログラムカウンタのプログラムカウンタタグを特別なタグ値に設定することを含む、ステップと、第2の命令のための第2のルールのメタデータ処理を実行するステップであって、第2のルールのメタデータ処理が、第2の命令のためのプログラムカウンタのプログラムカウンタタグが特別なタグに等しいときにのみ第2の命令の実行が許容されることを保証することを含む、ステップとを含み得る。

【００１２】

本発明の別の態様に従うのは、実行されると、命令を処理する方法を実行するコードを記憶した非一時的コンピュータ可読媒体であり、この方法は、現在の命令を含むコード実行領域から隔離されたメタデータ処理領域において実行されるメタデータ処理のための現在

の命令を受信するステップと、現在の命令のためのメタデータに関連するメタデータ処理領域によって、1つまたは複数のポリシーの集合に従った現在の命令の実行を許容するかどうかを判定するステップとを備え、現在の命令は第1のルーチンのスタックフレームの第1の位置にアクセスし、現在の命令およびスタックフレームの位置は関連するメタデータタグを有し、1つまたは複数のポリシーの集合は、スタック保護を提供し第1のルーチンのスタックフレームの記憶位置を含むスタック記憶位置への不適切なアクセスを防ぐ、スタック保護ポリシーを含む。

【００１３】

本明細書の技法の別の態様に従うのは、プロセッサと、プロセッサによって実行されると命令を処理する方法を実行するコードを記憶したメモリとを備える、システムであり、この方法は、現在の命令を含むコード実行領域から隔離されたメタデータ処理領域において実行されるメタデータ処理のための現在の命令を受信するステップと、現在の命令のためのメタデータに関連するメタデータ処理領域によって、1つまたは複数のポリシーの集合に従った現在の命令の実行を許容するかどうかを判定するステップとを備え、現在の命令は第1のルーチンのスタックフレームの第1の位置にアクセスし、現在の命令およびスタックフレームの位置は関連するメタデータタグを有し、1つまたは複数のポリシーの集合は、スタック保護を提供し第1のルーチンのスタックフレームの記憶位置を含むスタック記憶位置への不適切なアクセスを防ぐ、スタック保護ポリシーを含む。

【００１４】

本明細書の技法の別の態様に従うのは、実行されると、命令を処理する方法を実行するコードを記憶した非一時的コンピュータ可読媒体であり、この方法は、現在の命令を含むコード実行領域から隔離されたメタデータ処理領域において実行されるメタデータ処理のために現在の命令を受信するステップと、現在の命令のためのメタデータに関連するメタデータ処理領域によって、1つまたは複数のポリシーの集合に従った現在の命令の実行を許容するかどうかを判定するステップとを備え、1つまたは複数のポリシーは、命令の完全なシーケンスの実行を、完全なシーケンスの最初の命令から完全なシーケンスの最後の命令まで指定された順序で実施するルールの集合を含む。

【００１５】

本明細書の技法の別の態様に従うのは、プロセッサと、プロセッサによって実行されると命令を処理する方法を実行するコードを記憶したメモリとを備える、システムであり、この方法は、現在の命令を含むコード実行領域から隔離されたメタデータ処理領域において実行されるメタデータ処理のために現在の命令を受信するステップと、現在の命令のためのメタデータに関連するメタデータ処理領域によって、1つまたは複数のポリシーの集合に従った現在の命令の実行を許容するかどうかを判定するステップとを備え、1つまたは複数のポリシーは、命令の完全なシーケンスの実行を、完全なシーケンスの最初の命令から完全なシーケンスの最後の命令まで指定された順序で実施するルールの集合を含む。

【００１６】

本明細書の技法の別の態様に従うのは、コード実行領域から隔離されたメタデータ処理領域において使用される複数の指定されたレジスタのうちの第1の指定されたレジスタにブートストラップタグを記憶するステップと、ブートストラップタグから1つまたは複数の追加のメタデータタグを導出するために第1の処理を実行するステップとを備える、メタデータタグを生成して使用する方法であって、前記第1の処理は、メタデータ処理領域における1つまたは複数のルールのメタデータ処理をトリガする1つまたは複数の命令をコード実行領域において実行するステップを含む。ブートストラップタグは、メタデータ処理領域により使用されるすべての他のメタデータタグがそこから導出される、初期シードタグとして使用され得る。ブートストラップタグは、ハードワイヤリングされ、または読取り専用メモリの一部分に記憶され得る。この記憶および第1の処理は、メタデータ処理領域およびコード実行領域を含むシステムをブートするときにブートストラッププログラムの第1のコード部分を実行することにより実行される処理に含まれ得る。方法は、第1の指定されたレジスタに記憶されているブートストラップタグからデフォルトタグを導出する

ステップと、複数の指定されたレジスタのうちの第2の指定されたレジスタにデフォルトタグを記憶するステップと、コード実行領域により使用される複数のメモリ位置の各々のためのメタデータタグとして第2の指定されたレジスタからデフォルトタグを書き込むメタデータ処理領域におけるルールのメタデータ処理をトリガする、命令シーケンスを実行するステップとを含み得る。第1の処理は、ブートストラップタグから導出されたメタデータタグの初期集合を生成するステップを含むことがあり、初期集合のメタデータタグの各々は、メタデータ処理領域におけるルールキャッシュミス処理をトリガするコード実行領域における現在の命令を実行することによって生成されることがあり、これにより、現在の命令のためのルールキャッシュにルールが存在せず、ルールキャッシュは許容される動作を定義するためにメタデータ処理領域によって使用されるメタデータについてのルールを含む。ルールキャッシュミス処理は、メタデータ処理領域において実行するルールキャッシュミスハンドラによって、現在の命令のための新しいルールを計算するステップを含むことがあり、新しいルールはメタデータタグの初期集合の結果メタデータタグを含む。初期集合の各メタデータタグは、他のメタデータタグを導出するためにさらに使用され得るタグ生成源であり得る。1つまたは複数の指定された命令のうちの第1の集合の実行は、1つまたは複数の他のメタデータタグのシーケンスを生成するために使用されるタグ生成源として示される各メタデータタグを生成する、メタデータ処理領域におけるルールおよびルールキャッシュミス処理をトリガすることがあり、1つまたは複数の指定された命令の第2の集合の実行は、追加のメタデータタグをさらに生成するために使用できない非生成タグとして示される各メタデータタグを生成する、メタデータ処理領域におけるルールおよびルールキャッシュミス処理をトリガし得る。ブートストラッププログラムはさらに、拡張された特権、能力、または権限をタグ付けされた1つまたは複数の命令に与えるために、指定されたコード部分の1つまたは複数の命令に1つまたは複数の特別なメタデータコードタグを書き込む、メタデータ処理領域において処理されるルールをトリガする命令を含み得る。指定されたコード部分は、カーネルコードおよびローダコードのうちの1つまたは複数を含み得る。1つまたは複数の特別なメタデータコードタグは、メタデータタグの初期集合の第1のメタデータタグから導出され、第1のメタデータタグは特別な命令タグ生成源である。メタデータタグの初期集合は、命令をタグ付けするために使用される1つまたは複数のコードタグのシーケンスを生成するために使用されるタグ生成源である初期命令メタデータタグと、1つまたは複数の他のmallocタグ生成源のシーケンスを生成するために使用されるタグ生成源である初期mallocメタデータタグであって、1つまたは複数の他のmallocタグ生成源の各々が、割り振られたメモリセルおよび異なるアプリケーションにより使用される割り振られたメモリセルへのポインタのいずれかを色付けすることに関連して異なるアプリケーションのための1つまたは複数の他のメタデータタグのシーケンスを生成するために使用される、初期mallocメタデータタグと、1つまたは複数の他の制御フロー整合性タグ生成源のシーケンスを生成するために使用されるタグ生成源である初期制御フロー整合性タグであって、1つまたは複数の他の制御フロー整合性タグ生成源の各々が、異なるアプリケーションの制御移転ターゲットをタグ付けすることに関連して異なるアプリケーションのための1つまたは複数の他のメタデータタグのシーケンスを生成するために使用される、初期制御フロー整合性タグと、1つまたは複数の他のテイントタグ生成源のシーケンスを生成するために使用されるタグ生成源である初期テイントタグであって、1つまたは複数の他のテイントタグ生成源の各々が、データアイテムを産生または修正したコードに基づいて、異なるアプリケーションにより使用されるデータアイテムをメタデータテイントタグでタグ付けすることに関連して異なるアプリケーションのための1つまたは複数の他のメタデータテイントタグのシーケンスを生成するために使用される、初期テイントタグのうちの任意の1つまたは複数を含み得る。メタデータタグのシーケンスは、メタデータ処理領域におけるルールの他の処理をトリガする命令を実行することによって生成され得る。他の処理は、シーケンスの中の現在のメタデータタグを使用してシーケンスの中の次のメタデータタグを生成するステップであって、現在のメタデータタグが、シーケンスの現在の状態を示し、アトムと関連付けられるメタデータタグ

として記憶され、アトムがレジスタまたはメモリ位置のいずれかである、ステップと、アトムと関連付けられるメタデータタグとして次のメタデータタグを保存することによってシーケンスの現在の状態を更新するステップとを含み得る。

【００１７】

本明細書の技法の別の態様に従うのは、プロセッサによる実行のためにアプリケーションをロードするローダを実行するステップであって、ローダを実行する前記ステップが、メタデータ処理領域における1つまたは複数のルールの第1の集合のメタデータ処理をトリガする1つまたは複数の命令を含む第1のコード部分を実行するステップを含み、1つまたは複数のルールの第1の集合の前記メタデータ処理が、メタデータ処理領域にアクセス可能でありコード実行領域にアクセス可能ではないアプリケーションメタデータとしてアプリケーションのための制御フロー情報を収集して記憶するステップを含む、ステップと、コード実行領域においてアプリケーションの命令を実行するステップであって、アプリケーションの前記命令を実行する前記ステップが、アプリケーションにおける制御の第1のソース位置から第1のターゲット位置への移転を許容するかどうかを判定するために制御フロー情報の少なくとも一部分を使用する、制御フローポリシーのルールの第2の集合のメタデータ処理をトリガする、ステップとを備える、アプリケーションのための制御フロー情報を取得する方法である。第1のターゲット位置は、制御を第1のターゲット位置に移転することが許容される1つまたは複数の許容可能なソース位置の集合を有し得る。アプリケーションメタデータとしてアプリケーションのための制御フロー情報を収集して記憶するステップはさらに、メタデータ処理領域が他の処理を実行するステップを備え得る。他の処理は、1つまたは複数の許容可能なソース位置の集合を特定する第1のメタデータで第1のターゲット位置をタグ付けするステップを含むことがあり、第1のメタデータはアプリケーションメタデータの制御フロー情報の一部分として記憶される。アプリケーションの第1の命令は、制御を第1のソース位置から第1のターゲット位置に移転することができ、第1の命令は、制御を第1のターゲット位置に移転することが許容される1つまたは複数の許容可能なソース位置の集合に第1のソース位置が含まれるかどうかを判定することによって第1の命令の実行を許容するかどうかを判定するために第1のメタデータを使用する、制御フローポリシーの1つまたは複数のルールのメタデータ処理をトリガすることができる。他の処理も、固有のソースメタデータタグで集合の各々の許容可能なソース位置をタグ付けするステップを含み得る。各々の許容可能なソース位置の各々の固有のソースメタデータタグは、アプリケーションのためのソースメタデータタグの第1のシーケンスに含まれることがあり、第1のシーケンスは、制御フロー生成源タグから生成されるソースメタデータタグの固有のシーケンスであり得る。制御フロー生成源タグは、初期ブートストラップタグから導出された初期制御フロー生成源タグから生成され得る。初期制御フロー生成源タグは、複数の追加の制御フロー生成源タグを生成するために使用されることがあり、追加の制御フロー生成源タグの各々は、異なるアプリケーションのための固有のソースメタデータタグのシーケンスを生成するために使用されることがある。

【００１８】

本明細書の技法の別の態様に従うのは、実行されると、メタデータタグを生成して使用する方法を実行するコードを記憶した非一時的コンピュータ可読媒体であって、この方法は、コード実行領域から隔離されたメタデータ処理領域において使用される複数の指定されたレジスタのうちの第1の指定されたレジスタにブートストラップタグを記憶するステップと、ブートストラップタグから1つまたは複数の追加のメタデータタグを導出するために第1の処理を実行するステップとを備え、前記第1の処理は、メタデータ処理領域における1つまたは複数のルールのメタデータ処理をトリガする1つまたは複数の命令をコード実行領域において実行するステップを含む。

【００１９】

本明細書の技法の別の態様に従うのは、プロセッサと、実行されるとメタデータタグを生成して使用する方法を実行するコードを記憶したメモリとを備える、システムであって、この方法は、コード実行領域から隔離されたメタデータ処理領域において使用される複数

の指定されたレジスタのうちの第1の指定されたレジスタにブートストラップタグを記憶するステップと、ブートストラップタグから1つまたは複数の追加のメタデータタグを導出するために第1の処理を実行するステップとを備え、前記第1の処理は、メタデータ処理領域における1つまたは複数のルールのメタデータ処理をトリガする1つまたは複数の命令をコード実行領域において実行するステップを含む。

【００２０】

本明細書の技法の別の態様に従うのは、実行されると、アプリケーションのための制御フロー情報を取得する方法を実行するコードを記憶した非一時的コンピュータ可読媒体であり、この方法は、プロセッサによる実行のためにアプリケーションをロードするローダを実行するステップであって、ローダを実行する前記ステップが、メタデータ処理領域における1つまたは複数のルールの第1の集合のメタデータ処理をトリガする1つまたは複数の命令を含む第1のコード部分を実行するステップを含み、1つまたは複数のルールの第1の集合の前記メタデータ処理が、メタデータ処理領域にアクセス可能でありコード実行領域にアクセス可能ではないアプリケーションメタデータとしてアプリケーションのための制御フロー情報を収集して記憶するステップを含む、ステップと、コード実行領域においてアプリケーションの命令を実行するステップであって、アプリケーションの前記命令を実行する前記ステップが、アプリケーションにおける制御の第1のソース位置から第1のターゲット位置への移転を許容するかどうかを判定するために制御フロー情報の少なくとも一部分を使用する、制御フローポリシーのルールの第2の集合のメタデータ処理をトリガする、ステップとを備える。

【００２１】

本明細書の技法の別の態様に従うのは、プロセッサと、実行されるとアプリケーションのための制御フロー情報を取得する方法を実行するコードを記憶したメモリとを備える、システムであり、この方法は、プロセッサによる実行のためにアプリケーションをロードするローダを実行するステップであって、ローダを実行する前記ステップが、メタデータ処理領域における1つまたは複数のルールの第1の集合のメタデータ処理をトリガする1つまたは複数の命令を含む第1のコード部分を実行するステップを含み、1つまたは複数のルールの第1の集合の前記メタデータ処理が、メタデータ処理領域にアクセス可能でありコード実行領域にアクセス可能ではないアプリケーションメタデータとしてアプリケーションのための制御フロー情報を収集して記憶するステップを含む、ステップと、コード実行領域においてアプリケーションの命令を実行するステップであって、アプリケーションの前記命令を実行する前記ステップが、アプリケーションにおける制御の第1のソース位置から第1のターゲット位置への移転を許容するかどうかを判定するために制御フロー情報の少なくとも一部分を使用する、制御フローポリシーのルールの第2の集合のメタデータ処理をトリガする、ステップとを備える。

【００２２】

本明細書の技法の別の態様に従うのは、プロセッサ上で、タグ付けされていないデータソースから第1のデータをロードする第1の命令を実行するステップであって、前記タグ付けされていないデータソースが関連するメタデータタグを有しないメモリ位置を含む、ステップと、第1のハードウェアによって、第1のデータが信頼されず公開のデータソースからのものであることを示す第1のメタデータタグで第1のデータをタグ付けするステップであって、第1のメタデータタグを有する第1のデータが第1のバッファに記憶される、ステップと、プロセッサ上で、第1の1つまたは複数のルールを使用するメタデータ処理をトリガする第1のコードを実行するステップであって、第1の1つまたは複数のルールを使用するメタデータ処理が、第1のデータが信頼されることを示す第2のメタデータタグを有するように第1のデータを再タグ付けする再タグ付けを実行する、ステップとを備える、タグ付けされたデータソースとタグ付けされていないデータソースとの間でプロセッサにより仲介されるデータ転送を実行するための方法である。第2のメタデータタグは追加で、第1のデータが公開のソースからのものであることを示し得る。第2のメタデータタグを有する第1のデータは、関連するメタデータタグを各々有するメモリ位置を含むタグ付けされた

データソースであるメモリに記憶され得る。メモリは、1つまたは複数の信頼されるデータソースからのデータを含む信頼されるメモリであり得る。メタデータ処理は、第1のコードを含むコード実行領域から隔離されたメタデータ処理領域において実行され得る。第1の1つまたは複数のルールは、許容された動作を定義するためにメタデータ処理によって使用されるメタデータについてのルールであり得る。第1のコードは1つまたは複数の命令を含むことがあり、1つまたは複数の命令の各々は、第2のメタデータタグを有するように第1のデータを再タグ付けする1つまたは複数のルールを呼び出す権限を前記各々の命令が有することを示す、特別な命令タグを有し得る。第1のメタデータタグを有する第1のデータは暗号化されることがあり、方法は、プロセッサ上で1つまたは複数の命令を実行することによって、第1のメタデータタグを有する第1のデータを復号し、第1のメタデータタグを有する第1のデータの復号された形式を生成するステップと、プロセッサ上で1つまたは複数の追加の命令を実行することによって検証処理を実行するステップとを含むことがあり、前記検証処理は第1のデータの復号された形式が有効であることを確実にするためにデジタル署名を使用し、前記再タグ付けは第1のデータの検証処理の成功の後で実行される。第2のメタデータタグを有する第1のデータは、タグ付けされたメモリの第1のメモリ位置に復号された形式で記憶されることがあり、方法は、暗号化された形式の第1のデータを産生するために第1のデータを暗号化し、第1のデータに従ってデジタル署名を生成するステップであって、暗号化する前記ステップおよび生成する前記ステップがプロセッサ上で追加のコードを実行することによって実行される、ステップと、プロセッサ上で、タグ付けされたメモリの第1のメモリ位置からの第1のデータの暗号化された形式をタグ付けされていないメモリの宛先位置に記憶する第2の命令を実行するステップであって、第1のデータの暗号化された形式が関連するメタデータタグなしで宛先位置に記憶され、第2のメタデータタグが宛先位置に第1のデータの暗号化された形式を記憶する前に第2のハードウェアによって除去される、ステップとを含み得る。第1の時点において、第1のデータが、タグ付けされていないメモリ部分の第1の位置に記憶されることがあり、第2の時点において、第1のメタデータタグを有し、第1のデータが信頼されず公開のデータソースからのものであることを示す第1のデータが、タグ付けされたメモリ部分の第2の位置に記憶されることがある。タグ付けされていないメモリ部分および前記タグ付けされたメモリ部分は、同じメモリコントローラによってサービスされる同じメモリに含まれることがあり、第2のメタデータ処理ルールは、データが公開であることを示す関連するメタデータタグを有するデータをタグ付けされていないメモリ部分に書き込む動作を実行することのみをプロセッサに許容することがあり、タグ付けされていないデータに対して動作する外部のタグ付けされていないソースからの直接のメモリ動作は、同じメモリのタグ付けされていないメモリ部分にアクセスすることのみを許容されることがある。第2のメタデータ処理ルールの少なくとも一部分はさらに、データが公開であり、加えて信頼されないことを示す関連するメタデータタグを有するデータをタグ付けされていないメモリ部分に書き込む動作を実行することのみをプロセッサに許容することがある。タグ付けされていないデータソースは、タグ付けされていないデータソースのみを含む第1のインターコネクトファブリックに接続されることがあり、第2のメタデータタグを伴う第1のデータは、タグ付けされたデータソースのみを含む第2のインターコネクトファブリックに接続されるメモリの位置に記憶され得る。第2のプロセッサは、第1のインターコネクトファブリックに接続されることがあり、タグ付けされていないデータソースからのタグ付けされていないデータを使用して他の命令を実行することがある。他の命令は、メタデータ処理を実行することなく、かつ許容可能な動作を実施するためにメタデータについてのルールを使用することなく、実行されることがあり、前記第2のプロセッサによる前記他の命令の実行は、第1のインターコネクトファブリックのタグ付けされていないデータソースからデータを読み取ること、および第1のインターコネクトファブリックのタグ付けされていないデータソースにデータを書き込むことのうちのいずれかを含む1つまたは複数の動作を実行することを含み得る。

【００２３】

本明細書の技法の別の態様に従うのは、プロセッサと、1つまたは複数のタグ付けされたメモリであって、1つまたは複数のタグ付けされたメモリの各メモリ位置が関連するメタデータタグを有する、メモリと、第1のタグ付けされていないメモリを含む1つまたは複数のタグ付けされていないメモリであって、1つまたは複数のタグ付けされていないメモリのメモリ位置が関連するメタデータタグを有しない、メモリと、命令に関連して許容される動作を定義するためにメタデータ処理を実行する際に使用されるメタデータについてのルールを含むルールキャッシュであって、プロセッサによって現在の命令を実行する前に、現在の命令の実行が許容されるかどうかを判定するためにルールキャッシュの1つまたは複数のルールを使用したメタデータ処理が実行される、ルールキャッシュと、プロセッサによって実行されると、第1のタグ付けされていないメモリからプロセッサにより使用されるデータキャッシュへと第1のデータをロードする第1の命令であって、データキャッシュに記憶されている第1のデータが関連する第1のメタデータタグを有する、第1の命令と、プロセッサによって実行されると、データキャッシュからの第2のデータを第1のタグ付けされていないメモリに記憶する第2の命令であって、データキャッシュに記憶されている第2のデータが関連する第2のメタデータタグを有する、第2の命令と、プロセッサによってシステムにおいて使用されるタグ付けされたデータにタグ付けされていないデータを変換する第1のハードウェア構成要素であって、第1の命令の実行に応答して、第1のハードウェア構成要素が、第1のタグ付けされていないメモリから、どのような関連するメタデータタグも伴わずに第1のデータを受信し、関連する第1のメタデータタグを有する第1のデータを出力する、第1のハードウェア構成要素と、タグ付けされたデータをタグ付けされていないデータに変換する第2のハードウェア構成要素であって、第2の命令の実行に応答して、第2のハードウェア構成要素が、関連する第2のメタデータタグを有する第2のデータを受信し、どのような関連するメタデータタグも伴わずに第2のデータを出力する、第2のハードウェア構成要素とを備える、システムである。どのような関連するメタデータタグも伴わない第1のデータは暗号化されることがあり、第1のハードウェア構成要素は、第1のデータを復号された形式に変換することができ、デジタル署名を使用して第1のデータの検証処理を実行することができ、検証処理が成功すると、第1のデータが信頼されることを示す関連する第1のメタデータタグを有するように第1のデータをタグ付けすることができる。第2の関連するメタデータタグを有する第2のデータは、復号された形式であることがあり、第2のハードウェア構成要素は、第2のデータを暗号化された形式に変換することができ、第2のデータに従ってデジタル署名を生成する。第1のハードウェア構成要素は、第1のデータが信頼されることを示すとともに、第1のデータが公開のソースからのものであることを特定する、関連する第1のメタデータタグを有するように第1のデータをタグ付けすることができる。1つまたは複数の暗号鍵集合は、ハードウェアにおいて符号化されるか、メモリに記憶されるかのいずれかであり得る。1つまたは複数の暗号鍵集合は、復号および検証処理を実行することに関連して第1のハードウェア構成要素によって使用されることがあり、暗号化を実行することおよびデジタル署名を作成することに関連して第2のハードウェア構成要素によって使用されることがある。第1のデータは、第1のデータを復号するために第1のハードウェア構成要素によって使用される暗号鍵集合のうちの特定の1つを特定することができ、第2のデータの関連する第2のメタデータタグは、第2のデータを暗号化してそれに署名するために第2のハードウェア構成要素によって使用される暗号鍵集合のうちの特定の1つを特定することができる。

【００２４】

本明細書の技法の別の態様に従うのは、メタデータ処理のために現在の命令を受信するステップと、現在の命令を含むコード実行領域から隔離されたメタデータ処理領域において現在の命令のためのメタデータ処理を実行するステップであって、前記現在の命令がメタデータ処理において使用される第1のメタデータタグを有する第1のメモリ位置を参照し、現在の命令のための前記メタデータ処理が、メモリから第1のメタデータタグを取り出すための処理を実行するステップを含む、ステップと、メモリから第1のメモリ位置のための第1のメタデータタグを受信する前に、第1のメモリ位置の第1のメタデータタグの予測

される値を判定するステップと、第1のメモリ位置の第1のメタデータタグの予測される値を使用して、現在の命令の結果オペランドのための第1の結果メタデータタグを判定するステップと、メモリから第1のメタデータタグを受信するステップと、第1のメタデータタグが第1のメタデータタグの予測される値と一致するかどうかを判定するステップと、第1のメタデータタグが第1のメタデータタグの予測される値と一致すると判定したことに応答して、結果オペランドのための最終的な結果メタデータタグとして第1の結果メタデータタグを使用するステップとを備える、現在の命令を処理する方法である。現在の命令のためのメタデータ処理は、現在の命令および現在の命令のための入力メタデータタグの集合に従って、現在の命令のための第1のルールを判定するステップであって、前記第1のルールが、第1のメモリ位置の第1のメタデータタグの予測される値を含み、第1の結果メタデータタグを含み、前記第1のルールがメタデータ処理領域におけるメタデータ処理のために使用されるルールキャッシュに含まれる、ステップと、第1のメタデータタグが第1のメタデータタグの予測される値と一致しないと判定したことに応答して、現在の命令のためのメタデータ処理領域においてルールキャッシュミス処理を実行するステップとを含み得る。現在の命令のためのメタデータ処理領域におけるルールキャッシュミス処理は、コード実行領域における現在の命令の実行が許容されるかどうかを判定するステップと、コード実行領域における現在の命令の実行が許容されると判定したことに応答して、現在の命令のための新しいルールを生成するステップであって、前記新しいルールが現在の命令、入力メタデータタグの集合、および第1のメタデータタグに従って生成される、ステップと、メタデータ処理領域におけるメタデータ処理のために使用されるルールキャッシュへと新しいルールを挿入するステップとを含み得る。他の入力メタデータタグの集合は、現在の命令のための複数の他のメタデータタグを含むことがあり、他のメタデータ入力タグの前記集合は、プログラムカウンタ、現在の命令、および現在の命令の入力オペランドのうちのいずれかのためのメタデータタグを含むことがある。結果オペランドは、現在の命令を実行したことの結果を記憶する宛先メモリ位置または宛先レジスタであり得る。命令は、第1の段階および第2の段階を含む複数の段階に従って処理されることがあり、第1の段階は第2の段階の前に発生することがある。第1のメモリ位置の第1のメタデータタグの予測される値は第1の段階において判定されることがあり、第2の段階は、第1のメタデータタグが第1のメタデータタグの予測される値と一致するかどうかを判定する前記ステップを実行することを含むことがあり、第2の段階はまた、第1のメタデータタグが第1のメタデータタグの予測される値と一致しないと判定したことに応答して、現在の命令のためのメタデータ処理領域における前記ルールキャッシュミス処理を実行することを含むことがある。ルールキャッシュは、予測セレクタモードに従って、予測モードまたは通常処理モードのいずれかで動作するように構成可能であり得る。ルールキャッシュは、現在の命令のための前記メタデータ処理を実行するとき、予測モードで動作するように構成され得る。ルールキャッシュが前記予測モードで動作するように構成されるとき、ルールキャッシュは第1のルールに従って第1の出力を生成し得る。第1の出力は、次の命令のプログラムカウンタのためのメタデータタグ、現在の命令の結果オペランドのための第1の結果メタデータタグ、および第1のメタデータタグの予測される値を、第1の段階の出力として含むことがある。ルールキャッシュが前記通常処理モード動作するように構成されるとき、ルールキャッシュは、第1のルールとは異なる第2のルールに従って第2の出力を生成することがあり、第2の出力は第1のメタデータタグの予測される値を含まないことがあり、第2の出力は現在の命令の結果オペランドのための、および次の命令のプログラムカウンタのためのメタデータタグを含むことがある。ルールキャッシュは、予測モードにおいて動作するときに第1のポリシーのルールの第1のバージョンを使用することができ、そうではなく通常の処理モードで動作するときに第1のポリシーのルールの第2のバージョンを使用することができ、第1のルールはルールの第1のバージョンに含まれることがあり、第2のルールはルールの第2のバージョンに含まれることがある。

【００２５】

本明細書の技法の別の態様に従うのは、複数のパイプライン段階を含むパイプラインプロ

セッサであって、前記複数の段階がメモリ段階およびライトバック段階を含む、パイプラインプロセッサと、メモリ段階の完了の前に動作する統合されたメタデータ処理のためのプログラム可能ユニット(PUMP:programmable unit for metadata processing)であって、PUMPがメタデータ処理において使用される第1のメタデータタグを有する第1のメモリ位置を参照する現在の命令のためのメタデータ処理を実行し、PUMPが現在の命令のための第1のメタデータタグを含む第1の入力を受信し、PUMPがライトバック段階への入力として与えられる第1の出力を生成し、第1の出力が第1のメモリ位置の第1のメタデータタグの予測される値および現在の命令の結果オペランドのための第1の結果メタデータタグを含み、第1の結果メタデータタグが第1のメモリ位置のための第1のメタデータタグの予測される値に従ってPUMPによって判定される、PUMPと、第1のメモリ位置のための第1のメタデータタグが第1のメタデータタグの予測される値と一致するかどうかを判定し、第1のメタデータタグが第1のメタデータタグの予測される値と一致するときに結果オペランドのための最終的な結果メタデータタグとして第1の結果メタデータタグを使用する、前記ライドバック段階のハードウェア構成要素とを備える、システムである。PUMPは、メモリ段階と同時に動作し、さらに予測モードで動作する第1のPUMPであることがあり、第1のメモリ位置の第1のメタデータタグの予測される値を判定することができ、システムは、通常の非予測モードで動作する第2のPUMPを含むことがあり、第1のメモリ位置の第1のメタデータタグのためのいずれの予測される値も判定しないことがある。第2のPUMPは、メモリ段階とライトバック段階との間の別の段階として統合されることがある。第1のPUMPは、予測モードで動作するときに使用するための第1のポリシーのルールの第1のバージョンを使用することがあり、第2のPUMPは、通常の非予測モードで動作するときに使用するための第1のポリシーのルールの第2のバージョンを使用することがある。第1のPUMPは、第1のバージョンからの第1のルールに従って第1の出力を判定することがあり、第2のPUMPは、第2のバージョンからの第2のルールに従って第2の出力を判定することがある。第2の出力は、第1のメモリ位置のための第2の結果メタデータタグを含むことがあり、前記第2の出力は、ライトバック段階への入力として与えられることがある。ライトバック段階のハードウェア構成要素は追加で、第1のメタデータが予測される値と一致しないとき、結果オペランドのための最終的な結果メタデータタグとして第2の結果メタデータタグを使用することがある。

【００２６】

本明細書の技法の別の態様に従うのは、実行されると、タグ付けされたデータソースとタグ付けされていないデータソースとの間でプロセッサにより仲介されるデータ転送の方法を実行するコードを記憶した、非一時的コンピュータ可読媒体であり、この方法は、プロセッサ上で、タグ付けされていないデータソースから第1のデータをロードする第1の命令を実行するステップであって、前記タグ付けされていないデータソースが、関連するメタデータタグを有しないメモリ位置を含む、ステップと、第1のハードウェアによって、第1のデータが信頼されず公開されているデータソースからのものであることを示す第1のメタデータタグで第1のデータをタグ付けするステップであって、第1のメタデータタグを有する第1のデータが第1のバッファに記憶される、ステップと、プロセッサ上で、第1の1つまたは複数のルールを使用したメタデータ処理をトリガする第1のコードを実行するステップであって、第1の1つまたは複数のルールを使用したメタデータ処理が、第1のデータが信頼されることを示す第2のメタデータタグを有するように第1のデータを再タグ付けする再タグ付けを実行する、ステップとを備える。

【００２７】

本明細書の別の態様に従うのは、実行されると、現在の命令を処理する方法を実行するコードを記憶した、非一時的コンピュータ可読媒体であり、この方法は、メタデータ処理のために現在の命令を受信するステップと、現在の命令を含むコード実行領域から隔離されたメタデータ処理領域において現在の命令のためのメタデータ処理を実行するステップであって、前記現在の命令がメタデータ処理において使用される第1のメタデータタグを有する第1のメモリ位置を参照し、現在の命令のための前記メタデータ処理が、メモリから

第1のメタデータタグを取り出すための処理を実行するステップを含む、ステップと、メモリから第1のメモリ位置のための第1のメタデータタグを受信する前に、第1のメモリ位置の第1のメタデータタグの予測される値を判定するステップと、第1のメモリ位置の第1のメタデータタグの予測される値を使用して、現在の命令の結果オペランドのための第1の結果メタデータタグを判定するステップと、メモリから第1のメタデータタグを受信するステップと、第1のメタデータタグが第1のメタデータタグの予測される値と一致するかどうかを判定するステップと、第1のメタデータタグが第1のメタデータタグの予測される値と一致すると判定したことに応答して、結果オペランドのための最終的な結果メタデータタグとして第1の結果メタデータタグを使用するステップとを備える。

【００２８】

本明細書の技法の特徴および利点は、添付の図面とともに引用される技法の例示的な実施形態の以下の詳細な説明からより明らかになるであろう。

【図面の簡単な説明】

【００２９】

【図１】プロセッサパイプラインの中のパイプライン段階として統合されるPUMPキャッシュの例を示す概略図である。

【図２】PUMP評価フレームワークを示す概略図である。

【図３Ａ】図2に示される評価フレームワークを使用した簡単な実装形態を用いて単一ランタイムポリシーの実行結果を示すグラフである。

【図３Ｂ】簡単な実装形態を用いて単一エネルギーポリシーの実行結果を示すグラフである。

【図４Ａ】64bのタグを伴う簡単な実装形態の合成ポリシーのランタイムオーバーヘッドを示す一連の棒グラフであり、合成ポリシーは、(i)空間的および時間的なメモリ安全性、(ii)テイント追跡、(iii)制御フロー整合性、および(iv)コードとデータの分離というポリシーを同時に実施する。

【図４Ｂ】64bのタグを伴う簡単な実装形態の合成ポリシーのエネルギーオーバーヘッドを示す一連の棒グラフである。

【図４Ｃ】簡単な実装形態の電力の上限をベースラインと比較して示す一連の棒グラフである。

【図５Ａ】オペグループ最適化ありの場合となしの場合のPUMPルールの数を比較する棒グラフである。

【図５Ｂ】PUMP容量に基づいて異なるオペグループ最適化のミスレートの影響を示す一連のグラフである。

【図６Ａ】大半のワードが同じタグを有することを示す、合成ポリシーを用いたgccベンチマークについての各DRAM転送のための固有のタグの分布のグラフである。

【図６Ｂ】メインメモリのタグ圧縮を示す図である。

【図７Ａ】16b L2タグと12b L1タグとの間の変換を示す概略図である。

【図７Ｂ】12b L1タグと16b L2タグとの間の変換を示す概略図である。

【図８Ａ】L1 PUMPフラッシュに対するL1タグ長の影響(log10)を示す概略グラフである。

【図８Ｂ】L1 PUMPミスレートに対するL1タグ長の影響を示す概略グラフである。

【図９Ａ】異なるポリシーのミスレートを示す一連の棒グラフである。

【図９Ｂ】4つの例示的なマイクロアーキテクチャ最適化のためのキャッシュヒットレートを示す折れ線グラフである。

【図９Ｃ】ミスサービス性能を示す折れ線グラフである。

【図９Ｄ】容量に基づいてミスハンドラヒットレートを示す折れ線グラフである。

【図９Ｅ】合成ポリシーに対する最適化の影響を示す一連の棒グラフである。

【図１０Ａ】最適化された実装形態のランタイムオーバーヘッドを示す一連のグラフである。

【図１０Ｂ】最適化された実装形態のエネルギーオーバーヘッドを示す一連の棒グラフで

ある。
【図１０Ｃ】最適化された実装形態の絶対的な電力をベースラインと比較して示す一連の棒グラフである。
【図１１Ａ】様々な代表的なベンチマークについてタグビット長およびUCP-キャッシュ($)容量のランタイムオーバーヘッドへの影響を図示する一連の影付きグラフである。
【図１１Ｂ】様々な代表的なベンチマークについてタグビット長およびUCP-$容量のエネルギーオーバーヘッドへの影響を図示する一連の影付きグラフである。
【図１２Ａ】代表的なベンチマークでの最適化のランタイムへの影響を示す一連のグラフであり、A:単純、B:A+オペグループ化、C:B+DRAM圧縮、D:C+(10b L1,14b L2)ショートタグ、E:D+(2048-UCP;512-CTAG)である。
【図１２Ｂ】代表的なベンチマークでの最適化のエネルギーへの影響を示す一連のグラフであり、A:単純、B:A+オペグループ化、C:B+DRAM圧縮、D:C+(10b L1,14b L2)ショートタグ、E:D+(2048-UCP;512-CTAG)である。
【図１３Ａ】代表的なベンチマークに対する合成におけるランタイムポリシーへの影響を示す一連のグラフである。
【図１３Ｂ】合成におけるエネルギーポリシーへの影響を示す一連のグラフである。
【図１４】調査されるポリシーの概要を与える「表1」と名付けられる第1の表である。
【図１５】タグ付け方式の分類の概要を与える「表2」と名付けられる第2の表である。
【図１６】ベースラインおよび簡単なPUMP拡張されたプロセッサに対するメモリリソースの推定の概要を与える「表3」と名付けられる第3の表である。
【図１７】実験において使用されたPUMPパラメータ範囲の概要を与える「表4」と名付けられる第4の表である。
【図１８】PUMP最適化されたプロセッサに対するメモリリソースの推定の概要を与える「表5」と名付けられる第5の表である。
【図１９】テイント追跡ミスハンドラの概要を与える「アルゴリズム1」と名付けられる第1のアルゴリズムである。
【図２０】Nポリシーミスハンドラの概要を与える「アルゴリズム2」と名付けられる第2のアルゴリズムである。
【図２１】HWサポートを伴うNポリシーミスハンドラの概要を与える「アルゴリズム3」と名付けられる第3のアルゴリズムである。
【図２２】PUMPルールキャッシュデータフローおよびマイクロアーキテクチャの概略図である。
【図２３】PUMPマイクロアーキテクチャの概略図である。
【図２４】プロセッサパイプラインおよびそのオペグループ変換においてパイプライン段階として統合される例示的なPUMPキャッシュ、UCPキャッシュ、およびCTAGキャッシュを示す、図1と同様の概略図である。
【図２５】本明細書の技法によるある実施形態における制御ステータスレジスタ(CSR)の例の図である。
【図２６】本明細書の技法によるある実施形態におけるtagmodeの例の図である。
【図２７】本明細書の技法によるある実施形態における別個のプロセッサを伴う別個のメタデータ処理サブシステム/領域を示す例の図である。
【図２８】本明細書の技法によるある実施形態におけるPUMP入力および出力を示す図である。
【図２９】本明細書の技法によるある実施形態におけるオペグループ表に関連して入力および出力を示す図である。
【図３０】本明細書の技法によるある実施形態におけるPUMPにより実行される処理を示す図である。
【図３１】本明細書の技法によるある実施形態におけるPUMP入力および出力の制御と選択に関する追加の詳細を与える図である。
【図３２】本明細書の技法によるある実施形態におけるPUMP入力および出力の制御と選

択に関する追加の詳細を与える図である。
【図３３】本明細書の技法によるある実施形態における6段階の処理パイプラインを示す例の図である。
【図３４】ある実施形態におけるサブ命令および関連する技法を示す例の図である。
【図３５】ある実施形態におけるサブ命令および関連する技法を示す例の図である。
【図３６】ある実施形態におけるサブ命令および関連する技法を示す例の図である。
【図３７】ある実施形態におけるサブ命令および関連する技法を示す例の図である。
【図３８】ある実施形態におけるサブ命令および関連する技法を示す例の図である。
【図３９】ある実施形態におけるバイトレベルのタグ付けおよび関連する技法を示す例の図である。
【図４０】ある実施形態におけるバイトレベルのタグ付けおよび関連する技法を示す例の図である。
【図４１】ある実施形態におけるバイトレベルのタグ付けおよび関連する技法を示す例の図である。
【図４２】ある実施形態におけるバイトレベルのタグ付けおよび関連する技法を示す例の図である。
【図４３】本明細書の技法によるある実施形態における可変長オペコードを示す例の図である。
【図４４】本明細書の技法によるある実施形態におけるオペコードマッピングテーブルを示す例の図である。
【図４５】本明細書の技法によるある実施形態における共有ページを示す例の図である。
【図４６】本明細書の技法によるある実施形態における制御点の移転を示す例の図である。
【図４７】本明細書の技法によるある実施形態における呼出しスタックを示す例の図である。
【図４８】本明細書の技法によるある実施形態におけるメモリ位置のタグ付けまたは色付けを示す例の図である。
【図４９】本明細書の技法によるある実施形態におけるメモリ位置のタグ付けまたは色付けを示す例の図である。
【図５０】本明細書の技法によるある実施形態におけるsetjmpおよびlongjmpを示す例の図である。
【図５１】本明細書の技法によるある実施形態における、あるランタイム挙動および関連する予防的な活動および予防的な活動を実施するために使用される機構の表である。
【図５２】本明細書の技法によるある実施形態における、異なるランタイム挙動および関連する予防的な活動および予防的な活動を実施するために使用される機構の表である。
【図５３】本明細書の技法によるある実施形態における、異なるランタイム挙動および関連する予防的な活動および予防的な活動を実施するために使用される機構の表である。
【図５４】本明細書の技法によるある実施形態におけるポリシールールを学習または判定するために実行され得る処理を示す例の図である。
【図５５】本明細書の技法によるある実施形態におけるポリシールールを学習または判定するために実行され得る処理を示す例の図である。
【図５６】本明細書の技法によるある実施形態におけるポリシールールを学習または判定するために実行され得る処理を示す例の図である。
【図５７】データの外部バージョンと内部のタグ付けされたバージョンとの間の変換に関連してある実施形態における構成要素を示す例の図である。
【図５８】データの外部バージョンと内部のタグ付けされたバージョンとの間の変換に関連してある実施形態における構成要素を示す例の図である。
【図５９】データの外部バージョンと内部のタグ付けされたバージョンとの間の変換に関連してある実施形態における構成要素を示す例の図である。
【図６０】データの外部バージョンと内部のタグ付けされたバージョンとの間の変換に関連してある実施形態における構成要素を示す例の図である。

【図６１】本明細書の技法によるある実施形態におけるタグ予測を実行することの態様を示す例の図である。
【図６２】本明細書の技法によるある実施形態におけるタグ予測を実行することの態様を示す例の図である。
【図６３】本明細書の技法によるある実施形態におけるタグ予測を実行することの態様を示す例の図である。
【図６４】ある実施形態における、割り振られたメモリについて本明細書のメモリ位置色付け技法の使用を示す図である。
【図６５】ある実施形態における、割り振られたメモリについて本明細書のメモリ位置色付け技法の使用を示す図である。
【図６６】本明細書の技法によるある実施形態における、ハードウェアルールのサポートを提供する様々な構成要素を示す図である。
【図６７】本明細書の技法によるある実施形態における、ハードウェアルールのサポートを提供する様々な構成要素を示す図である。
【図６８】PUMPが値を返す実施形態における本明細書の技法の使用を示す例の図である。
【図６９】PUMPが値を返す実施形態における本明細書の技法の使用を示す例の図である。
【図７０】PUMPが値を返す実施形態における本明細書の技法の使用を示す例の図である。
【図７１】命令のシーケンスとともにある実施形態における本明細書の技法の使用を示す例の図である。
【図７２】本明細書の技法によるある実施形態における、システムをブートすることに関連して実行され得る処理ステップのフローチャートである。
【図７３】本明細書の技法によるある実施形態における、タグ生成に関連するツリータグ階層の例の図である。
【図７４】本明細書の技法によるある実施形態における、I/O PUMPに関連する態様および特徴を示す例の図である。
【図７５】本明細書の技法によるある実施形態における、I/O PUMPに関連する態様および特徴を示す例の図である。
【図７６】本明細書の技法によるある実施形態における、I/O PUMPに関連する態様および特徴を示す例の図である。
【図７７】本明細書の技法によるある実施形態における、I/O PUMPに関連する態様および特徴を示す例の図である。
【図７８】本明細書の技法によるある実施形態における、タグ値を記憶して判定することに関連して使用される階層を示す例の図である。
【図７９】本明細書の技法によるある実施形態における、タグ値を記憶して判定することに関連して使用される階層を示す例の図である。
【図８０】本明細書の技法によるある実施形態における、タグ値を記憶して判定することに関連して使用される階層を示す例の図である。
【図８１】本明細書の技法によるある実施形態における、タグ値を記憶して判定することに関連して使用される階層を示す例の図である。
【図８２】本明細書の技法によるある実施形態における、タグ値を記憶して判定することに関連して使用される階層を示す例の図である。
【図８３】本明細書の技法によるある実施形態における、制御フロー整合性および関連する処理を示す例の図である。
【図８４】本明細書の技法によるある実施形態における、制御フロー整合性および関連する処理を示す例の図である。
【発明を実施するための形態】
【００３０】
以下の段落で説明されるのは、メタデータタグを、システムのメインメモリ、キャッシュ、およびレジスタの中の1つ1つのワードと不可分に関連付ける、メタデータ処理のためのプログラム可能ユニット(PUMP)の様々な実施形態および態様である。無限のメタデータ

をサポートするために、タグはメモリの中のデータ構造を間接参照するのに十分大きい。1つ1つの命令上で、動作が許容されるかどうかを判定するために、および許容される場合には結果のためのタグを計算するために、入力のタグが使用される。いくつかの実施形態では、タグの確認および伝播のルールはソフトウェアにおいて定義されるが、性能への影響を最小限にするために、これらのルールは、プロセッサの算術論理装置(ALU)部分と並列に動作するハードウェア構造であるPUMPルールキャッシュにキャッシュされる。いくつかの実施形態では、ソフトウェアおよび/またはハードウェアを使用して実装され得るものなどの、ミスハンドラが、現在実施されているポリシーに基づいてキャッシュミスをサービスするために使用され得る。

【００３１】

4つの異なるポリシーの合成を使用する少なくとも1つの実施形態では、PUMPに様々な方法でストレスを与え、たとえば、(1)メモリの中のデータからコードを区別するためにタグを使用し、簡単なコードインジェクション攻撃に対する保護を提供する、非実行可能データおよび非書込み可能コード(NXD+NWC)ポリシー、(2)ヒープに割り振られたメモリにおけるすべての空間的および時間的な侵害を検出し、実質的に無限の(260個の)数の色(「テイントマーク」)まで拡張する、メモリ安全性ポリシー、(3)間接的な制御の移転をプログラムの制御フローグラフの中の許容されるエッジのみに制約し、return-oriented-programmingスタイルの攻撃を防ぐ、制御フロー整合性(CFI)ポリシー(攻撃に対する脆弱性がある可能性がある粗粒度の近似ではなく、細粒度のCFIが実施される)、(4)各ワードが複数のソース(ライブラリおよびIOストリーム)により同時にテイントされる可能性があり得る、細粒度のテイント追跡ポリシー(一般化)などの、一連のセキュリティ属性を例示する、PUMPの性能への影響が測定され得る(図14参照)。

【００３２】

上記は、本明細書の技法に従った実施形態において使用され得るよく知られているポリシーの例である。その保護能力が文献において確立されているそのようなよく知られているポリシーに対して、そのようなポリシーを実施しながら、PUMPを使用してポリシーを実施することの性能への影響も減らすために、本明細書の技法が使用され得る。NXD+NWCを除き、これらのポリシーの各々は、基本的に無限の数の固有のアイテムを区別する必要があり、対照的に、メタデータビットの数が限られている解決法は、せいぜい、ひどく簡略化された近似しかサポートすることができない。

【００３３】

本明細書の他の箇所で示され説明されるように、本明細書の技法に従った1つの実施形態は、ポインタサイズ(64bまたはバイト)のタグを64bのワードに対して使用し、それによりシステムの中のすべてのメモリのサイズおよびエネルギーの使用量を少なくとも2倍にする、PUMPの簡単で直接的な実装形態を利用することがある。これに加えて、ルールキャッシュが面積とエネルギーを追加する。この特定の実施形態では、190%の面積オーバーヘッド(図16参照)が測定され、幾何平均のエネルギーオーバーヘッドは220%前後であった。その上、いくつかのアプリケーションではランタイムオーバーヘッドが300%を超えることがある。そのような大きなオーバーヘッドは、それが行うことのできる最良のことであっても、採用を妨げることがある。

【００３４】

しかしながら、以下でより詳細に説明されるように、大半のポリシーは、タグと、タグに対して定義されるルールとの両方について、空間的および時間的な局所性を示す。したがって、本明細書の技法に従ったある実施形態は、類似する(または同一ですらある)命令のグループにわたってルールを定義することにより、固有のルールの数を大きく減らすことができ、強制的なミスを減らし、ルールキャッシュの実質的な容量を増やすことができる。オフチップのメモリトラフィックは、タグにおける空間的な局所性を利用することにより減らすことができる。オンチップの面積およびエネルギーオーバーヘッドは、使用されているポインタサイズのタグの部分集合を一度に表すために少数のビットを使用することにより最小限にすることができる。合成ポリシーミスハンドラのランタイムコストは、構

成要素のポリシーをキャッシュすることのサポートをハードウェアに与えることによって減らすことができる。したがって、本明細書の技法に従ったある実施形態は、そのような最適化を含むことで、PUMPがそのリッチなポリシーモデルを犠牲にすることなくより小さいオーバーヘッドを達成することを可能にし得る。

【００３５】

本明細書の技法に従ったある実施形態は、独立してまたは同時に実施され得る任意の数のセキュリティポリシーを符号化するために使用され得るメタデータを用いて、メモリワードおよび内部プロセッサ状態を強化することができる。本明細書の技法に従ったある実施形態は、「従来の」プロセッサ(たとえば、RISC-CPU、GPU、ベクトルプロセッサなど)に、ポリシーの任意の集合を実施するためにデータフローと並列に動作するメタデータ処理ユニット(PUMP)を追加することによって、上記のことを達成することができ、本開示の技法は特に、メタデータを無制限にし、ソフトウェアをプログラム可能にするので、本明細書の技法は、広範囲のメタデータ処理ポリシーに対して適応され適用され得る。たとえば、PUMPは、従来の(RISC)プロセッサの新しい/別個のパイプライン段階として統合されることがあり、または、「ホスト」プロセッサと並列に動作するハードウェアのスタンドアロンの断片として統合されることがある。前者の場合、設計を特徴付けるために、命令レベルシミュレータ、精緻化されたポリシー、実装の最適化およびリソースの推定、ならびに広範囲のシミュレーションがあり得る。

【００３６】

粒度レベルが細かいポリシー(すなわち、命令)を実施しようとする既存の解決法は、ポリシーの任意の集合を実施できない。一般に、命令レベルでは少数の固定されたポリシーしか実施できない。より細かい粒度レベル(すなわち、スレッド)でポリシーを実施すると、あるクラスのリターン指向プログラミング攻撃を防ぐことができず、したがってそのタイプの実施は有用性の点で限られたものになる。対照的に、本明細書の技法に従った実施形態は、命令レベルにおいて単一でまたは同時に実施され得る、無限の数のポリシーの表現を許容する(唯一の制限はサイズアドレス空間であり、それはメタデータがあらゆる任意のデータ構造を指すことができるアドレスポインタに関して表現されるからである)。

【００３７】

以下の段落において説明される様々な図は、本明細書で説明される技法の様々な態様の様々な例、方法、および他の例示的な実施形態を示すことに留意されたい。そのような図面において、示される要素の境界(たとえば、ボックス、ボックスのグループ、または他の形状)は、一般に境界の一例を表すことを理解されたい。いくつかの例では、1つの要素が複数の要素として設計され得ること、または複数の要素が1つの要素として設計され得ることを、当業者は理解するであろう。いくつかの例では、別の要素の内部構成要素として示される要素が外部構成要素として実装されることがあり、その逆であることもある。さらに、要素は縮尺通りに描かれていないことがある。

【００３８】

図1を参照すると、メタデータ処理のためのプログラム可能ユニット(PUMP)10は、エネルギー消費を意識するアプリケーションに対して適切なin-orderの実装形態および5段階のパイプラインを伴う、従来の縮小命令セットコンピューティングまたはコンピュータ(RISC)プロセッサ12へと統合され、これは実質的に、PUMP10が追加された6段階のパイプラインに変換される。第1の段階はフェッチ段階14であり、第2の段階は復号段階16であり、第3の段階は実行段階18であり、第4の段階はメモリ段階20であり、第5の段階はライトバック段階22である。PUMP10は、メモリ段階20とライトバック段階22の間に挟まれる。

【００３９】

様々な実施形態は、ポリシー実施およびメタデータ伝播を行う機構である電子論理回路を使用して、PUMP10を実装し得る。PUMP10の実施形態は、(i)4つの異なるポリシーおよびそれらの組合せのもとでのベンチマークの標準的な集合に対する、PUMP10の簡単な実装形態のランタイム、エネルギー、電力上限、および面積への影響の経験的な評価、(ii)マ

イクロアーキテクチャ最適化の集合、および(iii)これらの最適化による利益の測定結果により特徴付けることができ、この測定結果は、オンチップメモリ構造のために110%の追加の面積を使用することによる、10%を下回る典型的なランタイムオーバーヘッド、10%の電力上限への影響、および60%を下回る典型的なエネルギーオーバーヘッドを示している。

【００４０】

コンピューティングにおいて、ベンチマーキングは、普通は数回の標準的なテストおよびそれに対する試行を実行することにより、対象の相対的な性能を評価するために、コンピュータプログラム、プログラムの集合、または他の動作を実行する活動として特徴付けられ得る。本明細書で使用される「ベンチマーク」という用語は、ベンチマークプログラム自体を指す。本出願および図面全体で使用されるベンチマークプログラムのタイプは、GemsFDTD、astar、bwaves、bzip2、cactusADM、calculix、deall、gamess、gcc、gobmk、gromacs、h264ref、hmmer、Ibm、leslie3d、libquantum、mcf、milc、namd、omnetpp、perlbench、sjeng、specrand、sphinx3、wrf、zeusmp、およびmeanである。たとえば、図10A、図10B、および図10Cを参照されたい。

【００４１】

本明細書で使用される「論理」は、限定はされないが、機能もしくは活動を実行するための、かつ/または、別の論理、方法、および/もしくはシステムからの機能もしくは活動を引き起こすための、ハードウェア、ファームウェア、ソフトウェア、および/または各々の組合せを含む。たとえば、所望のアプリケーションまたは必要性に基づいて、論理は、ソフトウェアで制御されるマイクロプロセッサ、プロセッサ(たとえば、マイクロプロセッサ)のようなディスクリート論理、特定用途向け集積回路(ASIC)、プログラムされた論理デバイス、命令を含むメモリデバイス、メモリを有する電子デバイスなどを含み得る。論理は、1つまたは複数のゲート、ゲートの組合せ、または他の回路構成要素を含み得る。論理はまた、ソフトウェアとして完全に具現化され得る。複数の論理が説明される場合、1つの物理的な論理へと複数の論理を組み込むことが可能であり得る。同様に、単一の論理が説明される場合、その単一の論理を複数の物理的な論理の間で分散させることが可能であり得る。

【００４２】

本明細書の技法に従った少なくとも1つの実施形態では、PUMP10は、従来のRISCプロセッサ12への拡張として特徴付けられ得る。以下の段落は、PUMP10のハードウェアインターフェース層を構成するISA(命令セットアーキテクチャ)レベルの拡張、基本的なマイクロアーキテクチャの変更、および本明細書の技法に従った実施形態において使用され得る付随する低水準ソフトウェアの、さらなる詳細を与える。

【００４３】

本明細書の技法に従ったある実施形態において、PUMPで強化されたシステムにおける各ワードは、ポインタサイズのタグと関連付けられ得る。これらのタグはハードウェアレベルでは解釈されない。ソフトウェアレベルにおいて、タグは、ポリシーにより定義されるように、無制限のサイズおよび複雑さのメタデータを表し得る。数ビットのメタデータしか必要としないより簡単なポリシーは、メタデータをタグに直接記憶し得る。より多くのビットが必要とされる場合、メタデータをデータ構造として、タグとして使用されるこの構造のアドレスとともにメモリに記憶するために、間接参照が使用される。特に、これらのポインタサイズのタグは、本開示の1つの例示的な態様であり、限定するものと見なされるべきではない。基本的なアドレス指定可能なメモリワードはタグを用いて不可分に拡張され、メモリ、キャッシュ、およびレジスタを含むすべての値スロットを、適度により広くする。プログラムカウンタ(PC)もタグ付けされる。ソフトウェアで定義されるメタデータ、およびポインタサイズのタグとしてのそのメタデータを表現するというこの観念は、タグのために数ビットしか使用されない、および/または固定された解釈にタグがハードワイヤリングされる、従来のタグ付け手法を拡張する。タグ付け方式のいくつかの例示的な分類が表2に表されており、これは図15において再現されている。

【００４４】
メタデータタグはユーザプログラムによってアドレス指定可能ではない。むしろ、メタデータタグは、以下で説明されるようなルールキャッシュミスに際して呼び出されるポリシーハンドラによってアドレス指定される。タグへのすべての更新はPUMP10のルールを通じて実現される。
【００４５】
無制限のメタデータ以外では、本明細書の技法に従ったPUMP10の実施形態の別の特徴は、メタデータに対する単一サイクルのコモンケース計算を、ハードウェアがサポートすることである。これらの計算は、形式オペコードのルールに関して定義される:(PC,CI,OP1,OP2,MR)⇒(PC_{new},R)。これは、「現在のオペコードがopcodeであり、プログラムカウンタの現在のタグがPCであり、現在の命令のタグがCIであり、その入力オペランド(もしあれば)のタグがOP1およびOP2であり、メモリ位置のタグ(ロード/記憶の場合)がMRである場合、次のマシン状態におけるプログラムカウンタのタグはPC_{new}であるべきであり、命令の結果のタグ(もしあれば、宛先レジスタまたはメモリ位置)はRであるべきである」と読まれるべきである。2つの出力タグが最大で5つの入力タグから計算されることを可能にするこのルールフォーマットは、通常は1つの出力を最大で2つの入力から計算する(図15の表2参照)従来の成果において検討されたものよりも明らかにフレキシブルである。データタグ(OP1,OP2,MR,R)のみを追跡する従来の解決法を超えて、本開示は、コードブロックの起源、整合性、および使用法を追跡して実施するために使用され得る現在の命令タグ(CI)、ならびに、実行履歴、ambient authority、および暗黙的な情報フローを含む「制御状態」を記録するために使用され得るPCタグを提供する。CFIは、間接参照のジャンプのソースを記録するためのPCタグと、ジャンプのターゲットを特定するためのCIタグとを利用し、NXD+NWCは、データが実行可能ではないことを強制するためにCIを利用し、テイント追跡は、データを産生したコードに基づいてデータをテイントするためにCIを使用する。
【００４６】
一般的な場合の単一のサイクルにおけるルールを解決するために、本明細書の技法に従ったある実施形態は、直近に使用されたルールのハードウェアキャッシュを使用し得る。命令およびポリシーに応じて、所与のルールにおける入力スロットのうちの1つまたは複数は使用されないことがある。使用されないスロットのすべてのあり得る値についてのルールでキャッシュを汚染するのを避けるために、ルールキャッシュルックアップ論理は、各々の入力されるスロット-オペコードのペアに対して「don't-care」(図1参照)ビットを格納するビットベクトルを参照し、このビットベクトルは、対応するタグがルールキャッシュのルックアップにおいて実際に使用されるかどうかを判定する。これらの「don't care」入力を効率的に扱うために、それらは、入力をPUMP10に提示する前にマスクされる。don't-careビットベクトルは、ミスハンドラのインストールの一部として特権命令により設定される。
【００４７】
図1は全般に、PUMP10ハードウェアを組み込む修正された5段階のプロセッサ12のパイプラインを伴う、本明細書の技法に従った1つの実施形態を示す。ルールキャッシュのルックアップは、PUMP10の段階がプロセッサパイプラインにおける追加のストールを生み出さないように、追加の段階およびバイパスタグおよびデータとして独立に追加される。
【００４８】
PUMP10を別個の段階として(メモリ段階20とライトバック段階22の間に)置くことの動機は、PUMP10への入力として、メモリから読み取られる(ロード)、またはメモリに上書きされる(記憶)ことになるワードにタグを提供することの必要性である。書き込まれているメモリ位置の既存のタグに依存してルールが許容されるので、書込み動作は、読取り-修正-書込み動作になる。既存のタグは、読取りルールのようにメモリ段階20の間に読み取られ、読取りルールはPUMP10段階において確認され、書込みはライトバック段階22とも呼ばれ得るコミット段階の間に実行される。あらゆるキャッシュ方式のように、キャッ

シュの複数のレベルがPUMP10のために使用され得る。以下でより詳細に説明されるように、本明細書の技法に従ったある実施形態は、2レベルのキャッシュを利用し得る。複数のレベルのキャッシュへの拡張は、当業者には容易に明らかである。

【００４９】

1つの非限定的な例では、最終レベルのミスがライトバック段階22においてルールキャッシュの中で発生し、これは次のように扱われる。(i)この目的のためだけに使用されるプロセッサレジスタの(新しい)集合に現在のオペコードおよびタグが保存され、(ii)制御がポリシーミスハンドラ(以下でより詳細に説明される)に移転され、(iii)ポリシーミスハンドラが、動作が許容されるかどうかを判断し、許容される場合には適切なルールを生成する。ミスハンドラがリターンするとき、ハードウェアが(iv)このルールをPUMP10のルールキャッシュにインストールし、(v)フォルトした命令を再び発行する。特権のあるミスハンドラと、システムソフトウェアおよびユーザコードの残りとの間を隔離するために、ミスハンドラの動作モードがプロセッサに追加され、これは、ルールキャッシュミスに際して設定されミスハンドラがリターンするときにリセットされる、プロセッサ状態のビットにより制御される。1つ1つのルールキャッシュミスにおいてレジスタを保存して復元する必要をなくすために、整数レジスタファイルが、ミスハンドラにのみ利用可能である16個の追加のレジスタを用いて拡張され得る。加えて、ルールの入力および出力は、ミスハンドラモード(たとえば、レジスタウィンドウ)にある間はレジスタとして出現し、ミスハンドラが(ただしそれだけが)タグを普通の値として操作することが可能になる。やはり、これらはすべて、ライトバック段階22の非限定的な例である。

【００５０】

ルールをPUMP10のルールキャッシュにインストールすることを終えてユーザコードに戻るために、新しいミスハンドラリターン命令が追加される。この特定の非限定的な例では、この命令は、ミスハンドラモードにあるときにのみ発行され得る。ミスハンドラモードにある間、ルールキャッシュは無視され、PUMP10が代わりに、単一のハードワイヤードルールを適用する。ミスハンドラによって扱われたすべての命令およびデータは、事前に定義されたMISSHANDLERタグを用いてタグ付けされなければならず、すべての命令の結果は同じタグを与えられる。このようにして、PUMP10のアーキテクチャは、ポリシーにより提供される保護をユーザコードが損ねるのを防ぐ。代わりに、PUMPは、ミスハンドラのアクセスについてフレキシブルなルールを実施するために使用され得る。タグは、ユーザコードによって分割可能ではなく、アドレス指定可能ではなく、または置換可能ではない。メタデータデータ構造およびミスハンドラコードを、ユーザコードが扱うことはできず、ユーザコードはルールをルールキャッシュに直接挿入することができない。

【００５１】

図19を参照すると、アルゴリズム1は、テイント追跡ポリシーのためのミスハンドラの動作を示す。別個のタグ(および、したがってルール)の数を最小にするために、ミスハンドラは、構築するあらゆる新しいデータ構造を「正規化する」ことによって、論理的に等価なメタデータに対して単一のタグを使用する。

【００５２】

ユーザに単一ポリシーを選ぶことを強いるのではなく、複数のポリシーが同時に実施され、新しいポリシーが後で追加される。これらの「無制限の」タグの例示的な利点は、任意の数のポリシーを同時に実施できるということである。これは、タグを、いくつかの構成要素ポリシーからのタグのタプルへのポインタにすることによって、実現され得る。たとえば、NXD+NWCポリシーをテイント追跡ポリシーと組み合わせるために、各タグはタプル(s,t)へのポインタであってよく、sはNXD+NWCタグ(DATAまたはCODEのいずれか)であり、tはテイントタグ(テイントの集合へのポインタ)である。ルールキャッシュのルックアップは同様であるが、ミスが発生するとき、両方の構成要素ポリシーが別々に評価される。動作は、両方のポリシーがそれを許容する場合にのみ許容され、得られるタグは2つの構成要素ポリシーからの結果のペアである。しかしながら、他の実施形態では、ポリシーがどのように組み合わされるか(すべての構成要素の間のANDのように単純ではない)を

表現することが可能であり得る。

【００５３】

図20を参照すると、アルゴリズム2は、任意のN個のポリシーのための合成ミスハンドラの一般的な挙動を示す。タプルの中のタグがどのように相関しているかに応じて、これは、タグの、およびしたがってルールの数の大幅な増大をもたらし得る。複数のポリシーを同時にサポートする能力を実証して、ワーキングセットのサイズに対するそのことの影響を測定するために、合成ポリシー(「合成」)が実験を通じて実装され、合成ポリシーは上で説明されたすべての4つのポリシーを備える。合成ポリシーは、以下でさらに詳細に説明される、サポートされるポリシー作業負荷の種類を表す。図4Aおよび図20において見られるように、合成ポリシーは、(i)空間的および時間的なメモリ安全性、(ii)テイント追跡、(iii)制御フロー整合性、および(iv)コードとデータの分離というポリシーを同時に実施する。

【００５４】

大半のポリシーは、適切な論理を選択するためにオペコード上にディスパッチする。NXD+NWCのような一部のポリシーは、動作が許容されるかどうかを確認するだけである。他のポリシーはデータ構造を調べることがある(たとえば、CFIポリシーは許容される間接的な呼出しおよびリターンidのグラフを調べる)。メモリ安全性は、アドレスの色(すなわち、ポインタの色)とメモリ領域の色との間で等しさを確認する。テイント追跡は、入力タグを組み合わせることによって未使用の結果タグを計算する(アルゴリズム1)。大きなデータ構造にアクセスしなければならないポリシー(CFI)、または大きな集団にわたって正規化しなければならないポリシー(テイント追跡、合成)は、オンチップキャッシュにおいてミスするであろう多数のメモリアクセスを行い、DRAMに向かうことがある。ベンチマークのすべてにわたって平均すると、NXD+NWCではミスをサービスするのに30サイクルを必要とし、メモリ安全性では60サイクルを必要とし、CFIでは85サイクルを必要とし、テイント追跡では500サイクルを必要とし、合成では800サイクルを必要とした。

【００５５】

動作が許容されないとポリシーミスハンドラが判定する場合、ポリシーミスハンドラは、適切なセキュリティフォルトハンドラを呼び出す。このフォルトハンドラが何をするかは、ランタイムシステムおよびポリシー次第である。通常は、フォルトハンドラは攻撃プロセスをシャットダウンするが、いくつかの場合には、代わりに適切な「安全値」を返すことがある。UNIX（登録商標）型のオペレーティングシステムを用いた増分的な展開のために、想定されるポリシーはプロセスごとに適用され、各プロセスがポリシーの異なる集合を得ることを可能にする。プロセスごとに適用されるものとして記述されることは限定的ではなく、むしろ例示的であり、当業者はこのことを認識する。このことはまた、プロセスのアドレス空間への、タグ、ルール、およびミスハンドリングのサポートを可能にし、OSレベルのコンテキスト切替えの必要をなくす。長期的には、OSを保護するためにもPUMPポリシーが使用できるかもしれない。

【００５６】

以下は、ランタイム、エネルギー、面積、および電力のオーバーヘッドを測定するための評価方法を詳述し、図1に示される128bのワード(64bのペイロードおよび64bのタグ)と修正されたパイプラインプロセッサ12とを使用して、PUMPハードウェアおよびソフトウェアの単純な実装形態に対してその評価方法を適用する。最適化された実装形態が、(ベースラインのプロセッサに対する相対的な)オーバーヘッドが最終的に所望されるバージョンであるとしても、簡単なPUMPの実装形態をまず説明して測定することが有用である。簡単な実装形態は、より洗練されたバージョンを得る前に重要な機構の基本的なバージョンを詳述するので、簡単な実装形態と最適化された実装形態の両方が説明される。

【００５７】

PUMPの物理的なリソースに対する影響を推定するために、メモリコストに主に注目し、それは、簡単なRISCプロセッサおよびPUMPハードウェア拡張においてはメモリが支配的な面積とエネルギーの消費者であるからである。32nmの低動作電力(LOP)プロセスがL1

メモリ(図1参照)のために考慮され、低スタンバイ電力(LSTP)がL2メモリのために考慮され、メインメモリおよびプロセッサのオンチップメモリの面積、アクセス時間、アクセス当たりのエネルギー、および静的(漏れ)電力をモデル化するためにCACTI 6.5を使用する。

【００５８】

ベースラインのプロセッサ(PUMPなし)はデータおよび命令のための別個の64KBのL1キャッシュと、統合された512KBのL2キャッシュとを有する。遅延が最適化されたL1キャッシュおよびエネルギーが最適化されたL2キャッシュが使用された。すべてのキャッシュがライトバックの規律を使用する。ベースラインのL1キャッシュは880ps前後のレイテンシを有し、1サイクルで結果を返すことができクロックを1nsに設定できると想定され、現代の埋め込みプロセッサおよび携帯電話プロセッサに匹敵する1GHzのサイクルの目標を与える。このプロセッサに対するパラメータが、図16の表3に提示される。

【００５９】

PUMPルールキャッシュ10のハードウェア実装形態の一実施形態は、段階14、16、20の中のすべてのアーキテクチャ状態をタグを用いて延ばすこと、およびPUMPルールキャッシュをプロセッサ12に追加することという、2つの部分を含み得る。オンチップメモリの中の各々の64bのワードを64bのタグを用いて延ばすことは、アクセス当たりのワードの面積およびエネルギーを増やし、アクセスのレイテンシを悪化させる。これはL2キャッシュでは許容可能である可能性があり、L2キャッシュはすでに多サイクルのアクセスレイテンシを有し、すべてのサイクルで使用されるとは限らない。しかし、L1キャッシュ(図1参照)にアクセスするためにレイテンシの余計なサイクルを追加することは、パイプラインにおけるストールにつながり得る。これを避けるために、この簡単な実装形態では、L1キャッシュの実質的な容量はベースラインの設計における容量の半分まで減らされ、次いでタグを追加する。このことは、同じ単一サイクルのアクセス権をL1キャッシュに与えるが、ミスの増大により性能を低下させ得る。

【００６０】

本明細書の技法に従ったある実施形態では、PUMPルールキャッシュ10は、従来のキャッシュアドレス鍵(アドレス幅より短い)と比較して長い一致キー(5個のポインタサイズのタグ+命令オペコード、すなわち328b)を利用し、128bの結果を返す。一実施形態では、完全に連想型のL1ルールキャッシュが使用され得るが、大きなエネルギーおよび遅延につながる(図16の表3参照)。代替として、本明細書の技法に従ったある実施形態は、図22に示されるように、4つのハッシュ関数から着想を得たマルチハッシュキャッシュ方式を利用することができる。L1ルールキャッシュは、単一のサイクルで結果を産生し、2番目のサイクルにおいて誤ヒットを確認するように設計されるが、L2ルールキャッシュは低エネルギーのために設計され、多サイクルのアクセスレイテンシを与える。やはり、図16の表3は、簡単な実装形態において使用される1024エントリーのL1ルールキャッシュおよび4096エントリーのL2ルールキャッシュのためのパラメータを示す。これらのキャッシュが容量に達するとき、簡単なfirst-in-first-out(FIFO)置換ポリシーが使用され、これは現在の作業負荷に対しては実践的によく機能するように見える(ここではFIFOはLRUの6%以内である)。

【００６１】

図2を参照すると、PUMPの性能への影響の推定は、ISA、PUMP、およびアドレストレースシミュレータの組合せを特定する。gem5シミュレータ24は、64ビットのAlphaベースラインISAでのSPEC CPU2006プログラム(gem5が失敗するxalancbmkおよびtontoを除く)のための命令トレースを生成する。各プログラムは、上で列挙された4つのポリシーおよび1Bの命令のウォームアップ期間のための合成ポリシーの各々についてシミュレートし、次いで、次の500Mの命令を評価する。gem5シミュレータ24では、各ベンチマークは、タグまたはポリシーなしでベースラインプロセッサ上で実行される。得られる命令トレース26が次いで、各命令のためのメタデータ計算を実行するPUMPシミュレータ28を通じて実行される。この「段階的な」シミュレーション戦略は、PUMPの結果によりプログラムの制御フローがそのベースライン実行から逸脱してはならない、フェイルストップ

ポリシーに対しては正確である。アドレストレースシミュレーションは、高度にパイプライン化されout-of-orderのプロセッサに対しては不正確であり得るが、簡単なin-orderの5段階および6段階のパイプラインに対しては極めて正確である。ベースライン構成では、gem5命令のシミュレーションおよびアドレストレースの生成30と、それに続く、アドレスシミュレータ32におけるカスタムのアドレストレースシミュレーションおよびアカウンティングは、gem5のサイクルアキュレートシミュレーションの1.2%の範囲内にあった。

【００６２】

PUMPシミュレータ28は、各ポリシーを実施するためのミスハンドラコード(Cで書かれている)を含み、メタデータタグは、ポリシーに応じて初期メモリ上に割り当てられる。PUMPシミュレータ28は、PUMP10ルールキャッシュにおけるアクセスパターンを捉え、関連するランタイムおよびエネルギーコストを推定して、L2ルールキャッシュにアクセスするのに必要なより長い待機サイクルを考慮することを可能にする。ミスハンドラコードを有するPUMPシミュレータ28もプロセッサ上で実行するので、gem5でのミスハンドラのための別個のシミュレーションがその動的な挙動を捉える。ミスハンドラコードはデータおよび命令キャッシュに影響する可能性があるので、ユーザコードとミスハンドラコードの両方からの適切にインターリーブされたメモリアクセスを含むマージされたアドレストレースが作成され、これは、メモリシステムの性能への影響を推定するための最終的なアドレストレースシミュレーションのために使用される。

【００６３】

以下の段落では、簡単なPUMPの実装形態の評価が、PUMPなしのベースラインと比較して与えられる。

【００６４】

評価の1つの点として、ベースラインプロセッサに加わるPUMP10の全体的な面積オーバーヘッドは190%であることに留意されたい(図16の表3参照)。この面積オーバーヘッドの主要な部分(110%)は、PUMP10のルールキャッシュに由来する。統合されたL2キャッシュが、残りの面積オーバーヘッドの大半に寄与する。L1 D/Iキャッシュは概ね同じままであり、それはそれらの実質的な容量が半分になるからである。この大きなメモリ面積オーバーヘッドは静的電力を概ね3倍にし、これはエネルギーオーバーヘッドの24%に相当する。

【００６５】

評価の別の点はランタイムオーバーヘッドに関する。大半のベンチマーク上でのすべての単一ポリシーに対して、この簡単な実装形態でも平均ランタイムオーバーヘッドは10%に過ぎず(図3Aおよび図3B参照;ボックスプロットを読むために、棒は中央値であり、ボックスは上位4分の1から下位4分の1(中央の50%のケース)をカバーし、点は各々の個々のデータ点を表し、ひげは外れ値(1.5×それぞれの4分位数を超える)を除く全範囲を示す)、追加のDRAMトラフィックに由来する支配的なオーバーヘッドが、プロセッサとの間でタグビットを転送するために必要とされる。メモリ安全性ポリシー(図3Aおよび図3B)について、大きなミスハンドラオーバーヘッドを示す少数のベンチマークがあり、新しく割り振られるメモリブロック上での強制的なミスが原因で全体のオーバーヘッドを40～50%に押し上げている。合成ポリシーランタイム(図では「CPI」または「CPIオーバーヘッド」と名付けられている)について、ベンチマークのうちの5つでミスハンドラにおけるオーバーヘッドが非常に大きく(図4A参照)、最悪の場合にはGemsFTDTにおいて780%に迫り、幾何平均が50%に達する。図4Bに示される合成ポリシーエネルギー(図では「EPI」または「EPIオーバーヘッド」と名付けられている)について、ベンチマークのうちの3つ(すなわち、GemsFTDT、astar、omnetpp)でミスハンドラにおけるオーバーヘッドが非常に大きく、最悪の場合には、GemsFTDTにおいて1600%、astarにおいて600%、omnetppにおいて520%に迫る。

【００６６】

(1)最終レベルのルールキャッシュミスを解決するために必要とされる大量のサイクル(1つの1つの構成要素のミスハンドラが調べられなければならないので)、および(2)ルールの数

の爆発という2つの要因がこのオーバーヘッドに寄与しており、ルールの数の爆発は、ワーキングセットのサイズを大きくし、ルールキャッシュミスレートを高める。最悪の場合、固有の合成タグの数は、各構成要素ポリシーにおける固有のタグの積であり得る。しかしながら、全体のルールは、最大の単一ポリシーであるメモリ安全性よりも、3倍から5倍の係数で増大する。

【００６７】

評価の別の点はエネルギーオーバーヘッドである。より幅広のワードが原因でより多数のビットを動かすことと、ミスハンドラコードが原因でより多数の命令を実行することの両方が、エネルギーオーバーヘッドに寄与し、単一ポリシーと合成ポリシーの両方に影響する(図3Bおよび図4B)。CFIおよびメモリ安全性ポリシーは、またしたがって合成ポリシーも大きなデータ構造にアクセスし、これはしばしばエネルギー的に高価なDRAMアクセスを必要とする。最悪の場合のエネルギーオーバーヘッドは単一ポリシーに対しては400%に迫り、合成ポリシーに対しては約1600%であり、幾何平均オーバーヘッドは220%前後である。

【００６８】

多くのプラットフォーム設計では、最悪の場合の電力、または等価的にはサイクル当たりのエネルギーが、制約になる。この電力上限は、プラットフォームが電池から引き出すことができる最大の電流、または、周囲の冷却とともにモバイルデバイスと有線デバイスのいずれかにおける最大の持続する動作温度により決まり得る。図4Cは、簡単な実装形態が最大電力上限を76%上げ、ベースライン実装形態と簡単なPUMP実装形態の両方においてlbmが最大の電力を駆動することを示す。この電力上限の上昇は最悪の場合のエネルギーオーバーヘッドより小さく、それは、一部のベンチマークがそれらの消費する余剰のエネルギーよりも大きく速度を下げることが一部には原因であり、また、エネルギーオーバーヘッドの大きいベンチマークがベースライン設計においてサイクル当たり最小の絶対的なエネルギーを消費するベンチマークであることが一部には原因であることに留意されたい。通常、これらのエネルギー効率の高いプログラムのデータワーキングセットはオンチップキャッシュに収まるので、それらのプログラムがDRAMアクセスのより高いコストを払うことはほとんどない。

【００６９】

上で説明された前述の実装形態を組み込む実施形態は、大半のベンチマークで適当な性能を達成し、一部のベンチマークでの合成ポリシーに対するランタイムオーバーヘッドおよびすべてのポリシーおよびベンチマークでのエネルギーと電力のオーバーヘッドは、許容不可能なほど高いよう見える。これらのオーバーヘッドに対処するために、本明細書の技法に従ったある実施形態に、一連の目標を定めたマイクロアーキテクチャ最適化が導入され、また組み込まれ得る。図17の表4では、これらの最適化が、PUMP構成要素と関連付けられるアーキテクチャパラメータの、全体的なコストに対する影響のために調べられる。PUMPルールキャッシュの実質的な容量を増やすために同一のルールを伴うオペコードのグループ化が使用され、DRAM転送の遅延およびエネルギーを減らすためにタグ圧縮が使用され、オンチップメモリにおける面積およびエネルギーを減らすために短いタグが使用され、ミスハンドラにおけるオーバーヘッドを減らすために統合構成要素ポリシー(UCP)および合成タグ(CTAG)キャッシュが使用される。

【００７０】

ここで説明されるのは、本明細書の技法に従ったある実施形態において使用され得るような「オペグループ」である。実践的なポリシーでは、いくつかのオペコードに対して類似するルールを定義するのは一般的である。たとえば、テイント追跡ポリシーでは、加算および減算の命令のためのルールは同一である(図19のアルゴリズム1参照)。しかしながら、簡単な実装形態では、これらのルールは、ルールキャッシュの中の別個のエントリーを占有する。この観察に基づいて、同じルールを伴う命令動作コード(「オペコード」)が「オペグループ」としてグループ化され、必要なルールの数を減らす。どのオペコードを一緒にグループ化できるかはポリシーに依存するので、ルールキャッシュのルックアップの

前にもオペコードをオペグループに変換するために、「don't-care」SRAMが実行段階18(図1)において拡張される。合成ポリシーに対して、300個を超えるAlphaオペコードが14個のオペグループに減らされ、ルールの総数は、1/1.1～1/6、平均で1/1.5に減る(図5AはすべてのSPECベンチマークにわたってこの効果を測定する)。このことは、シリコン面積における所与の投資に対するルールキャッシュの容量を実質的に増やす。オペグループはまた、強制的なミスの数を減らし、それは、グループの中の単一の命令でのミスが、グループの中の1つ1つの命令オペコードに適用されるルールをインストールするからである。図5Bは、オペグループ化がある場合とない場合の、合成ポリシーに対する様々なL1ルールキャッシュサイズのためのすべてのSPECベンチマークにわたるミスレートを要約している。図5Bは、ミスレートの範囲と平均の両方がオペグループ化により低減することを示している。具体的には、オペグループ最適化の後の1024エントリーのルールキャッシュは、オペグループ最適化なしの4096エントリーのルールキャッシュよりミスレートが低い。より低いミスレートは当然、ミスハンドラにおいて消費される時間およびエネルギーを減らし(図12Aおよび図12B参照)、より小さいルールキャッシュは面積およびエネルギーを直接減らす。

【００７１】

本明細書の技法に従ったある実施形態は、ここで説明されるメインメモリタグ圧縮を利用し得る。64bのワードに64bのタグを使用すると、オフチップメモリトラフィックが2倍になるので、関連するエネルギーがほぼ2倍になる。しかし、通常、タグは空間的な局所性を示し、多くの近くのワードは同じタグを有する。たとえば、図6Aは、合成ポリシーについての、gccベンチマークに対する各DRAM転送のための固有のタグの分布をプロットしたものであり、大半のワードが同じタグを有し、8ワードのキャッシュラインのDRAM転送当たり平均で約1.14個の固有のタグしかないことを示している。この空間的なタグの局所性は、オフチップメモリとの間で転送されなければならないタグビットを圧縮するために利用される。データはキャッシュラインにおいて転送されるので、キャッシュラインがこの圧縮のために基礎として使用される。アドレス指定を簡単なままにするために、キャッシュライン当たり128Bがメインメモリにおいて割り振られる。

【００７２】

しかしながら、図6Bに示されるように、128bのタグ付けされたワードを直接記憶するのではなく、8個の64bのワード(ペイロード)が記憶され、次いで8個の4bのインデックスが記憶され、次いで最大で8個の60bのタグが記憶される。インデックスは、60bのタグのいずれが関連するワードに付随するかを特定する。インデックスを収容するためにタグは60bに調整されるが、これはポインタとしてのタグの使用を損なうものではなく、バイトアドレス指定および16B(2つの64bのワード)の揃えられたメタデータ構造を仮定すると、64bのポインタの下側の4bは0として埋められ得る。結果として、インデックスのうちの4Bを転送した後で、残っていることは、キャッシュラインにおいて固有の7.5Bのタグを転送する必要だけである。たとえば、同じタグがキャッシュラインの中のすべてのワードにより使用される場合、1回目の読取りにおいて64B+4B=68Bの転送があり、そして2回目の読取りにおいて8Bの転送があり、全体で128Bではなく76Bである。4bのインデックスは、直接インデックスまたは特別値のいずれかであり得る。特別インデックス値はデフォルトタグを表すように定義されるので、この場合にはタグを何ら転送する必要がない。この方式でタグを圧縮することによって、DRAM転送当たりの平均エネルギーオーバーヘッドは110%から15%に減る。

【００７３】

上で提示された圧縮方式は、たとえば、簡潔性と、オフチップメモリエネルギーを減らすことに対する有効性との組合せにより、本明細書の技法に従った実施形態において利用され得る。マルチレベルタグページテーブル、可変粒度のTLB様構造、およびレンジキャッシュを含む、細粒度のメモリのタグ付けのための追加の代替的な賢い方式が存在し、これらも、本明細書の技法に従ったある実施形態においてDRAMのフットプリントを減らすために使用され得ることを、当業者は明確に認識する。

【００７４】
ここで説明されるのは、本明細書の技法に従ったある実施形態において、タグ変換がどのように実行され得るかである。図1を再び参照すると、各々のキャッシュされたルールは456bの幅であるので、簡単なPUMPルールキャッシュは大きい(110%の面積を加える)。PUMP10をサポートすることはまた、64bのタグを用いてベースラインオンチップメモリ(RFおよびL1/L2キャッシュ)を拡張することを必要とした。ここで各々の64bのワードに対して完全な64b(または60b)のタグを使用することは、面積およびエネルギーの重いオーバーヘッドを招く。しかしながら、64KBのL1-D$は8192ワードしか保持しないので、最大でも8192個の固有のタグしか保持しない。64KBのL1-I$とともに、L1メモリサブシステムにおいて、最大で16384個の固有のタグがあることがあり、これらはわずか14bのタグで表すことができ、システムにおける遅延、面積、エネルギー、および電力を減らす。キャッシュ(L1,L2)は時間的な局所性を利用するために存在し、この観察は、面積およびエネルギーを減らすために局所性を利用できることを示唆している。タグビットが14bに減らされる場合、PUMPルールキャッシュ一致キーは328bから78bに減らされる。
【００７５】
完全なポインタサイズのタグのフレキシビリティを失うことなく前述の節約の利点を得るために、幅の異なるタグが、異なるオンチップメモリサブシステムのために使用され、必要に応じてそれらのオンチップメモリサブシステムの間で変換し得る。たとえば、L1メモリにおいて12bのタグを使用し、L2メモリにおいて16bのタグを使用することがある。図7Aは、L1メモリサブシステムとL2メモリサブシステムとの間で実行され得るようなタグ変換を詳述する。L2キャッシュ34からL1キャッシュ36にワードを動かすことは、ワードの16bのタグを対応する12bタグに変換することを必要とし、必要であれば新しい関連付けを作成する。L2タグのためのL1マッピングがあるかどうかを示す余剰なビットを伴う、L2タグからL1タグへの変換のための簡単なSRAM38。図7Bは、L1タグをアドレスとして使用するSRAM39のルックアップとともに実行される、(ライトバックまたはL2ルックアップに際する)L1タグ40からL2タグ42への変換を詳述する。同様の変換は、60bのメインメモリタグと16bのL2タグとの間で発生する。
【００７６】
長いタグが長から短への変換テーブルの中にないとき、新しい短いタグが割り振られ、場合によっては、もはや使用されていない以前に割り振られた短いタグを回収する。ガベージコレクションおよびタグ使用カウントを含む、短いタグをいつ回収できるかを判定するための、探究すべきリッチな設計空間がある。簡潔にするために、短いタグが順番に割り振られ、短いタグ空間が使い果たされたときに所与のレベルを超えるすべてのキャッシュ(命令、データ、およびPUMP)をフラッシュし、特定の短いタグがいつ回収可能かを追跡する必要をなくす。キャッシュは、キャッシュのフラッシュを安価にする適切な技法を用いて設計され得る。たとえば、本明細書の技法に従ったある実施形態では、たとえば、参照によって本明細書に組み込まれる、K. Mai、R. Ho、E. Alon、D. Liu、Y. Kim、D. Patil、およびM. Horowitz、Architecture and Circuit Techniques for a 1.1GHz 16-kb Reconfigurable Memory in 0.18um-CMOS、IEEE J. Solid-State Circuits、40(1):261-275、2005年1月において説明され、当技術分野で知られているものなどの、軽量なgang clearを用いてすべてのキャッシュが設計され得る。
【００７７】
表3(図16に再現される)と比較すると、各L1ルールキャッシュのアクセスには51pJかかり、本明細書の技法は8bのL1タグでは10pJへの、または16bのL1タグでは18pJへの低減をもたらし、エネルギーのスケーリングはこれらの点の間のタグの長さに対して線形である。L1命令およびデータキャッシュに対するエネルギーの影響は小さい。同様に、16bのL2タグでは、L2 PUMPアクセスには、64bのタグの場合の173pJから減少して120pJかかる。L1タグを小さくすることは、L1キャッシュの容量を復元することも可能にする。12bのタグでは、全容量(76KB、実質的には64KB)のキャッシュが単一サイクルのタイミング要件を満たし、低減されたL1キャッシュの容量が招く簡単な実装形態の性能悪化を減

らす。結果として、L1タグ長の探究は12ビット以下に限定される。さらに短いタグはエネルギーを減らすが、フラッシュの頻度も増やす。

【００７８】

図8Aおよび図8Bは、L1タグ長の増大とともにフラッシュがどのように減少するか、ならびにL1ルールキャッシュミスレートに対する影響を示す。

【００７９】

ここで説明されるのは、ミスハンドラアクセラレーションに関連して使用され得る様々な技法である。本明細書の技法に従ったある実施形態は、4つのポリシーを単一の合成ポリシーへと合成し得る。図20を参照すると、アルゴリズム2において、Nポリシーのミスハンドラの各呼出しはタグのタプルを分解しなければならず、合成ポリシーに必要なルールはルールキャッシュミスレートを上げ、これらは図9Aにおいて特定される。テイント追跡およびCFIポリシーが個々に低いミスレートを有するとしても、メモリ安全性ポリシーからのより高いミスレートは、合成ポリシーのミスレートも高くする。個々のポリシーのより低いミスレートは、合成ルールがキャッシュ可能ではなくても、個々のポリシーの結果がキャッシュ可能であり得ることを示唆する。

【００８０】

図23に示されるものなどのPUMPマイクロアーキテクチャの様々な態様に関連して、合成ポリシーミスハンドリングを最適化するためにハードウェア構造が利用され得る。本明細書の技法に従ったある実施形態は、最新の構成要素ポリシーの結果がキャッシュされる、統合構成要素ポリシー(UCP;図21のアルゴリズム3参照)キャッシュ(UCP$)を利用し得る。そのような実施形態では、合成ポリシーのための一般的なミスハンドラは、構成要素ポリシーを解決しながらこのキャッシュにおいてルックアップを実行するように修正される(たとえば、図21のアルゴリズム3、行3などを参照)。構成要素ポリシーに対するこのキャッシュミスのとき、そのポリシー計算がソフトウェアにおいて実行される(かつ結果をこのキャッシュに挿入する)。

【００８１】

図24にも示されるように、UCPキャッシュは、追加のポリシー識別子フィールドとともに、普通のPUMPルールキャッシュと同じハードウェア組織を用いて実装され得る。FIFO置換ポリシーは、このキャッシュのために使用され得るが、構成要素ポリシーの再計算コストなどの尺度を使用して空間を優先順位付けることによってより良い結果を達成することが可能であり得る。適度な容量で、このキャッシュは大半のポリシー再計算を除去する(図9B;メモリ安全性に対する低ヒットレートは新しいメモリ割振りと関連付けられる強制的なミスが原因である)。結果として、ミスハンドラサイクルの平均の数は、大半の難しいベンチマークに対して1/5に減る(図9E)。必要な合成ルールは少数の構成要素ポリシールールの積であり得るので、L2 PUMPにおいてミスがあるとき、UCPキャッシュにおいて1つ1つのポリシーがヒットすることが可能である。GemsFDTDでは、3つ以上の構成要素ポリシーが、約96%の時間においてヒットした。

【００８２】

図23および図24にも含まれるように、キャッシュは、結果タグのタプルをその標準的な合成結果タグに変換するために追加され得る。前述のキャッシュは合成タグ(CTAG)キャッシュ(CTAG$)と呼ばれることがあり、いくつかの構成要素ポリシールールが結果タグの同じタプルを返すことは一般的であるので、CTAG$は効果的である(図9D)。たとえば、多くの場合、結果タグが異なっていても、Pc_{tag}は同じである。さらに、多くの異なるルール入力は同じ出力をもたらし得る。たとえば、テイント追跡では集合の和集合が実行され、多くの異なる和集合が同じ結果を有する。たとえば、(Blue,{A,B,C})が、{A}∪{B,C}と{A,B}∪{B,C}の両方の結果(テイント追跡)をBlueスロット(メモリ安全性)に書き込むための、合成の答えである。FIFO置換ポリシーがこのキャッシュに使用される。CTAGキャッシュは、平均のミスハンドラサイクルを1/2という別の割合に減らす(図9E参照)。

【００８３】

まとめると、2048エントリーのUCPキャッシュおよび512エントリーのCTAGキャッシュ

は、各L2ルールキャッシュミスで費やされる平均時間を800サイクルから80サイクルに減らす。

【００８４】

本明細書の技法に従ったある実施形態はまた、ルールを含むキャッシュのうちの1つまたは複数に記憶される1つまたは複数のルールをプリフェッチすることによって、性能を改善し得る。したがって、近い将来必要であり得る事前計算ルールを用いて強制ミスレートを下げることが追加で可能である。ある事例は、メモリ安全性ルールに対して高い値を有する。たとえば、新しいメモリタグが割り振られるとき、新しいルールがそのタグのために必要とされる(初期化(1)、オフセットをポインタに追加し移動する(3)、スカラーロード(1)、スカラー記憶(2))。その結果、これらのルールのすべてが、UCPキャッシュに一度に追加され得る。単一ポリシーのメモリ安全性の場合、ルールはルールキャッシュに直接追加され得る。これは、メモリ安全性ミスハンドラの呼出しの回数を1/2に減らす。

【００８５】

全体的な評価に関して、かつ図11Aを参照すると、アーキテクチャパラメータはある特定のコストに単調に影響し、エネルギー、遅延、および面積の間のトレードオフをもたらすが、単一のコスト基準の中での最小値を定義しない。タグビットが十分小さくなると、L1 D/Iキャッシュをベースラインの容量に復元できるので、ベースラインがL1タグ長を探究するための上限として採用されるが、その点を超えると、性能への影響が小さい状態でタグ長の減少がエネルギーを減らすという、閾値の効果がある。

【００８６】

図11Bは、タグ長を減らすことが大半のベンチマークプログラム(たとえば、leslie3d、mcf)に対する支配的なエネルギー効果であることを示しており、少数のプログラム(たとえば、GemsFDTD、gcc)がUCPキャッシュ容量を増やすことによる等しいまたはより大きい利益を示している。他のコストの問題を無視すると、エネルギーを減らすために、大きなミスハンドラキャッシュおよび少ないタグビットが選択される。ランタイムオーバーヘッド(図11A参照)も、より大きなミスハンドラキャッシュにより最小限にされるが、より少ないタグビットではなくより多くのタグビットから利益を得る(たとえば、GemsFDTD、gcc)。この利益の大きさは、ベンチマークおよびポリシーにより変化する。すべてのベンチマークにわたって、10bのL1タグを超える利益はSPEC CPU2006ベンチマークに対して小さいので、10bがエネルギーと遅延との間の妥協点として使用され、探究されるアーキテクチャパラメータの空間内の最小エネルギーレベルに近づけながら面積オーバーヘッドを減らすために、2048エントリーのUCPキャッシュおよび512エントリーのCTAGキャッシュを使用する。

【００８７】

図12Aおよび図12Bは、最適化を適用することのランタイムオーバーヘッドおよびエネルギーオーバーヘッドに対する全体的な影響を示す。一部のベンチマークに対しては1つ1つの最適化が支配的であり(たとえば、astarに対するオペグループ、lbmに対するDRAMタグ圧縮、h264refに対する短いタグ、GemsFDTDに対するミスハンドラアクセラレーション)、一部のベンチマークはすべての最適化からの利益を受け(たとえば、gcc)、各最適化は連続して、あるボトルネックを取り除くと次のボトルネックに曝される。ベンチマークからの異なる挙動は、以下で詳述されるベースライン特性に従う。

【００８８】

局所性の低いアプリケーションは、メインメモリのトラフィックが大きいことにより、DRAMにより決まるベースラインのエネルギーおよび性能を有する。そのようなベンチマーク(たとえば、lbm)におけるオーバーヘッドは、DRAMオーバーヘッドの傾向があるので、DRAMオーバーヘッドの低減がランタイムオーバーヘッドおよびエネルギーオーバーヘッドに直接影響する。より局所性の大きいアプリケーションは、ベースライン構成においてより高速であり、より少ないエネルギーを消費し、DRAMオーバーヘッドの影響をより受けず、結果として、これらのベンチマークは、低減されたL1容量ならびにL1 D/Iキャッシュおよびルールキャッシュにおけるタグエネルギーの影響をより大きく受ける。DRAM最

適化は、これらのアプリケーションに対する影響がより小さいが、短いタグを使用することが、エネルギーに対する大きな影響を有し、L1 D/Iキャッシュ容量の悪化をなくす(たとえば、h264ref)。

【００８９】

重い動的なメモリ割振りを伴うベンチマークは、新しく作成されるタグがキャッシュにインストールされなければならないので強制的なミスがあることにより、より高いL2ルールキャッシュミスレートを有する。これは、簡単な実装形態におけるいくつかのベンチマーク(GemsFDTD、omnetpp)に対するオーバーヘッドを高くする。本明細書で説明されるようなミスハンドラ最適化は、そのようなミスの一般的な場合のコストを減らし、オペグループ最適化は、容量ミスレートを下げる。簡単な実装形態では、GemsFDTDでは、200個の命令ごとにL2ルールキャッシュミスがあり、その780%のランタイムオーバーヘッドの大部分の原因である各ミスをサービスするために800サイクルを要する(図4A参照)。最適化により、GemsFDTDベンチマークは、400個の命令ごとにL2ルールキャッシュミスをサービスし、ミス当たり平均で140サイクルしかかからず、ランタイムオーバーヘッドを約85%に減らす(図10A参照)。

【００９０】

全体的に、これらの最適化は、メモリ割振りの大きいGemsFDTDおよびomnetppを除くすべてのベンチマークに対して、ランタイムオーバーヘッドを10%未満にする(図10A参照)。平均のエネルギーオーバーヘッドは60%に近く、4つのベンチマークだけが80%を超える(図10B参照)。

【００９１】

例示すると、PUMPの性能への影響は、4つの異なるポリシーの合成を使用して測定されることがあり(図14の表1参照)、これらのポリシーは、PUMPに様々な方法でストレスを与え、(1)メモリの中のデータとコードを区別するためにタグを使用し、簡単なコードインジェクション攻撃に対する保護を提供する、非実行可能データおよび非書込み可能コード(NXD+NWC)ポリシー、(2)ヒープに割り振られたメモリにおけるすべての空間的および時間的な侵害を検出し、実質的に無限の(260個の)数の色(「テイントマーク」)まで拡張する、メモリ安全性ポリシー、(3)間接的な制御の移転をプログラムの制御フローグラフの中の許容されるエッジのみに制約し、リターン指向プログラミング型の攻撃を防ぐ、制御フロー整合性(CFI)ポリシー(攻撃に対する脆弱性がある可能性がある粗粒度の近似ではなく、細粒度のCFIを実施する)、(4)各ワードが複数のソース(ライブラリおよびIOストリーム)により同時にテイントされる可能性があり得る、細粒度のテイント追跡ポリシー(一般化)という、一連のセキュリティ属性を示す。本明細書の他の箇所で述べられるように、これらは、その保護能力が文献において確立されているよく知られているポリシーであり、本明細書の説明は、PUMPを使用してポリシーを実施することの性能への影響を測定して減らすことに注目し得る。NXD+NWCを除き、これらのポリシーの各々は、基本的に無限の数の固有のアイテムを区別し、対照的に、メタデータビットの数が限られている解決法は、せいぜい、ひどく簡略化された近似しかサポートすることができない。上で述べられたように、PUMPの簡単な直接の実装形態は高価であり得る。たとえば、ポインタサイズ(64b)のタグを64bのワードに追加すると、システムにおけるすべてのメモリのサイズおよびエネルギー使用量が少なくとも2倍になり、これに加えて、ルールキャッシュが面積およびエネルギーを追加する。この簡単な実装形態では、測定される面積オーバーヘッドは約190%であり、幾何平均のエネルギーオーバーヘッドは約220%であり、その上、ランタイムオーバーヘッドは一部のアプリケーションでは期待外れである(300%を超える)。そのような大きなオーバーヘッドは、それが行うことのできる最良のことであっても、採用を妨げるであろう。

【００９２】

本明細書で説明されるものなどのマイクロアーキテクチャ最適化が、電力上限に対する影響を10%に減らすために、本明細書の技法に従ったある実施形態に含まれることがあり(図10C参照)、これは、最適化されたPUMPがプラットフォームの動作エンベロープにほとん

ど影響しないことを示唆する。DRAM圧縮はlbmに対するエネルギーオーバーヘッドを20%に減らす。速度も9%下げるので、電力要件は10%しか増えない。

【００９３】

最適化された設計の面積オーバーヘッドは、簡単な設計の190%(たとえば、図16の表3参照)と比較して、110%前後である(たとえば、図18の表5参照)。短いタグはL1キャッシュおよびL2キャッシュの面積を大きく減らし(今やベースラインに5%追加するだけである)、ルールキャッシュの面積を大きく減らす(26%しか追加しない)。反対に、最適化された設計は、ランタイムオーバーヘッドおよびエネルギーオーバーヘッドを減らすためにいくらかの面積を費やす。UCPキャッシュおよびCTAGキャッシュは33%の面積オーバーヘッドを追加するが、短いタグのための変換メモリ(L1とL2の両方)は別の46%を追加する。これらの追加のハードウェア構造は面積を追加するが、ネットでのエネルギー低減をもたらし、それは、それらが頻繁にはアクセスされず、UCPキャッシュおよびCTAGキャッシュもミスハンドラサイクルをかなり減らすからである。

【００９４】

本明細書で説明されるようなモデルおよび最適化の1つの目標は、同時に実施される追加のポリシーをある実施形態が追加することを、比較的簡単にすることである。簡単なPUMP設計での合成ポリシーは、ミスハンドラランタイムの大きな増大により、いくつかのベンチマークでは付加的なものよりも大きなコストを招いたが、これらはミスハンドラ最適化により減らされる。

【００９５】

図13A(CPIオーバーヘッドのための)および図13B(EPIオーバーヘッドのための)は、最初に各単一ポリシーのオーバーヘッドを示し、次いで、最も複雑な単一ポリシーであるメモリ安全性にポリシーを追加する合成を示すことによって、ポリシーの付加的な追加がランタイムオーバーヘッドにどのように影響するかを示す。この進行は、どのオーバーヘッドが、より大きなオーバーヘッドのポリシーを追加することではなく単に任意のポリシーを追加することにより生じるかを、より明らかにする。ここで4つのポリシーを超えるスケーリングの感覚を得るために、CFIポリシー(リターンおよび計算されたジャンプ/呼出しを行う)とテイント追跡ポリシー(テインティングおよびI/Oテインティングをコーディングする)が2つの部分に各々分解される。追加のポリシー追跡のランタイムオーバーヘッドは徐々に第1の複雑なポリシー(メモリ安全性)を上回り、非外れ値に対する顕著なランタイムの影響はなく(最悪の場合の非外れ値は9%から10%オーバーヘッドを増やす)、大部分がミスハンドラ解決の複雑さの増大により、各々の新しい種類のポリシーが追加されるにつれて2つの外れ値においてより大きな増大(20～40%)が見られる。エネルギーは同様の傾向に従って、非外れ値のポリシーに対して穏やかな影響(幾何平均が60%から70%増大する)を伴い、これはGemsFDTDを除くすべてのものに当てはまる。

【００９６】

関連する成果の簡単な概要が図15で再現される表2において特定される。

【００９７】

本明細書の技法に従ったポリシープログラミングモデルによれば、PUMPポリシーは、タグ値の集合を、何らかの所望のタグ伝播および実施機構を実装するためにこれらのタグを操作するルールの集合とともに含む。ルールには、システムのソフトウェア層(シンボリックルール)またはハードウェア層(コンクリートルール)という2つの形式がある。

【００９８】

たとえば、PUMPの動作を例示するために、プログラム実行の間にリターンポイントを制約するための簡単な例示的なポリシーを考える。このポリシーの動機は、リターン指向プログラミング(ROP)として知られているあるクラスの攻撃に由来し、ROPでは、攻撃者は、攻撃されているプログラムのバイナリ実行可能ファイルにおいて「ガジェット」の集合を特定し、これらを使用して、各々が何らかのガジェットを指すリターンアドレスを格納するスタックフレームの適切なシーケンスを構築することにより、複雑な悪意のある挙動を組み立てる。次いで、バッファオーバーフローまたは他の脆弱性が、所望のシーケンス

を用いてスタックの上部を上書きするために悪用され、断片が順番に実行されるようにする。ROP攻撃を制限する1つの簡単な方法は、リターン命令のターゲットをよく定義されたリターンポイントに制約することである。これは、メタデータタグターゲットを用いて、有効なリターンポイントである命令をタグ付けすることによって、PUMPを使用することにより達成される。リターン命令が実行されるたびに、リターンが発生したばかりであることを示すために、PCのメタデータタグがcheckに設定される。次の命令では、PCタグはcheckであり、現在の命令のタグがtargetであることを検証し、そうでない場合にはセキュリティ侵害をシグナリングする。メタデータをよりリッチにすることによって、どのリターン命令がどのリターンポイントに戻れるかを正確に制御することが可能である。メタデータをさらにリッチにすることによって、完全なCFI確認を実施することができる。

【００９９】

ポリシー設計者およびPUMP10のソフトウェア部分の観点からは、ポリシーは、小さな領域固有の言語で書かれたシンボリックルールを使用してコンパクトに記述され得る。例示的なシンボリックルールおよびそのプログラム言語は、たとえば、「PROGRAMMING THE PUMP, Hardware-Assisted Micro-Policies for Security」という表題のセクションにおいて説明される。

【０１００】

シンボリックルールは、多種多様なメタデータ追跡機構をコンパクトに符号化し得る。しかしながら、ハードウェアレベルでは、主要な計算を遅くするのを避けるために、効率的な解釈のために調整された表現に対してルールが必要である。この目的で、コンクリートルールと呼ばれる、より低水準のルールのフォーマットが導入され得る。直観的には、所与のポリシーに対する各シンボリックルールは、コンクリートルールの等価な集合へと拡張され得る。しかしながら、単一のシンボリックルールは一般に、無限の数のコンクリートルールを生成し得るので、この精緻化は遅延して実行され、システムが実行される間に必要に応じてコンクリートルールを生成する。

【０１０１】

メタデータタグを伴う(たとえば、ROPよりリッチな)ポリシーに対して、シンボリックルールからコンクリートルールへの変換は同じ全般的な道筋に従うが、詳細は少しだけより複雑である。たとえば、テイント追跡ポリシーは、各々が任意のサイズのテイントの集合を記述する(所与のデータに寄与した可能性のあるデータソースまたはシステム構成要素を表す)、メモリデータ構造へのポインタとしてタグを解釈する。loadオペグループのためのシンボリックルールは、ロードされた値のテイントが、命令自体、ロードのためのターゲットアドレス、およびそのアドレスにおけるメモリでのテイントの和集合であるはずであることを述べている。シンボリックルールおよびそのプログラム言語は、前に特定された「PROGRAMMING THE PUMP, Hardware-Assisted Micro-Policies for Security」という表題の論文の縦覧から、参照によって組み込まれ、入手可能である。

【０１０２】

別個のタグの数(および、したがってルールキャッシュに対する圧力)を減らすために、メタデータ構造は内部的に標準的な形式で記憶されることがあり、タグは不変であるので、共有が完全に利用される(たとえば、集合要素は、集合が共通のプレフィックス部分集合を共有してコンパクトに表現され得るように、標準的な順序で与えられる)。これらの構造は、もはや必要とされないとき、(たとえば、ガベージコレクションによって)回収され得る。

【０１０３】

ある実施形態は合成ポリシーを利用し得る。複数の直交するポリシーが、タグをいくつかの構成要素ポリシーからのタグのタプルへのポインタにすることにより、同時に実施され得る(一般に、複数のポリシーは直交していないことがある)。たとえば、テイント追跡ポリシーを用いて第1のリターンオペグループ(ROP)ポリシーを作るために、各タグをタプル(r,t)の表現へのポインタとし、rはROPタグ(コード位置識別子)であり、tはテイントタグ(テイントの集合へのポインタ)である。キャッシュルックアッププロセスは厳密に同じであるが、ミスが発生すると、ミスハンドラがタプルの構成要素を抽出し、シンボリックルー

ルの両方の集合を評価するルーチンにディスパッチする。適用されるルールを両方のポリシーが有する場合にのみ、動作は許容される。この場合、得られるタグは、2つのサブポリシーからの結果を格納するペアへのポインタである。

【０１０４】

ポリシーシステムおよび保護に関連して、ポリシーシステムは、各ユーザプロセス内のメモリの別個の領域として存在する。ポリシーシステムは、たとえば、ミスハンドラのためのコード、ポリシールール、およびポリシーのメタデータタグを表すデータ構造を含み得る。プロセスの中にポリシーシステムを置くことは、既存のUnix（登録商標）プロセスモデルに関して最小限に侵害的であり、ポリシーシステムとユーザコードとの間の軽量な切替えを容易にする。ポリシーシステムは、次に説明される機構を使用してユーザコードから隔離される。

【０１０５】

明らかに、攻撃者がメタデータタグを書き直せる場合、またはそれらの解釈を変更できる場合、PUMPにより提供される保護は無意味である。本明細書で説明される技法は、そのような攻撃を防ぐように設計される。カーネル、ローダ、および(いくつかのポリシーのための)コンパイラが信頼される。具体的には、コンパイラに頼って、初期タグをワードに割り当て、必要な場合に、ルールをポリシーシステムに伝える。ローダは、コンパイラにより提供されるタグを保存し、コンパイラからロードへの経路は、たとえば暗号学的な署名を使用して、改竄から保護される。

【０１０６】

本明細書の技法に従ったある実施形態は、各プロセスのための初期メモリイメージをセットアップする、標準的なUnix（登録商標）型のカーネルを使用し得る(マイクロポリシーを使用してこれらの仮定の一部をなくし、さらにTCBのサイズを減らすことが可能であり得る)。そのような実施形態では、ルールキャッシュミスハンドリングソフトウェアが正しく実装されることがさらに仮定される。ルールキャッシュミスハンドリングソフトウェアは小さいので、正式な検証の良好なターゲットである。1つの重要なことは、プロセスにおいて実行されるユーザコードが、プロセスのポリシーにより提供される保護を損ねるのを防ぐことである。ユーザコードは、(i)タグを直接操作することが可能であるべきではなく、すなわちすべてのタグの変更が現在有効なポリシールール/複数のポリシールールに従って実行されるべきであり、(ii)ミスハンドラによって使用されるデータ構造およびコードを操作することが可能であるべきではなく、(iii)ハードウェアルールキャッシュにルールを直接挿入することが可能であるべきではない。

【０１０７】

アドレス指定に関連して、ユーザコードによるタグの直接の操作を防ぐために、1つ1つの64bのワードに添付されるタグは、それら自体が別々にアドレス指定可能ではない。具体的には、タグを読み取り、または書き込むために、タグまたはタグの一部分にのみ相当するアドレスを指定することが可能ではない。すべてのユーザアクセス可能命令は、アトミックな単位としての(データ,タグ)のペアに対して動作し、標準的なALUが値部分に対して動作しPUMPがタグ部分に対して動作する。

【０１０８】

本明細書の技法に従ったある実施形態におけるミスハンドラアーキテクチャに関連して、ポリシーシステムは、PUMPキャッシュに対するミスに際してのみアクティブ化され得る。ポリシーシステムとユーザコードを隔離するために、ミスハンドラ動作モードがプロセッサに追加される。整数レジスタファイルは、レジスタを保存して復元することを避けるために、ミスハンドラのみが利用可能な16個の追加のレジスタを用いて拡張される。16個の追加のレジスタの使用は例示的であり、実際には、整数レジスタファイルをより少数/多数のレジスタに拡張する必要があり得ることに留意されたい。フォルトした命令のPC、ルール入力(オペグループおよびタグ)、およびルール出力は、ミスハンドラモードにある間はレジスタとして出現する。ミスハンドラリターン命令が追加され、これで、コンクリートルールをキャッシュにインストールすることが終了し、ユーザコードに戻る。

【０１０９】
本明細書の技法に従ったある実施形態では、プロセッサ12がミスハンドラモードにある間、PUMP10の普通の挙動は不関与である。代わりに、単一のハードワイヤードルールが適用される。ミスハンドラにより扱われるすべての命令およびデータが、いずれのポリシーによって使用されるタグとも異なる事前に定義されたミスハンドラタグでタグ付けされなければならない。このことは、同じアドレス空間におけるミスハンドラコードとデータとユーザコードとの隔離を確実にする。ユーザコードはポリシーシステムデータまたはコードを扱うことができず、またはそれらを実行することができず、ミスハンドラは偶発的にユーザデータおよびコードを扱うことができない。ミスハンドラリターン命令は、ミスハンドラモードのみにおいて発行することができ、ユーザコードがルールをPUMPに挿入するのを防ぐ。
【０１１０】
これまでの研究は、安全性およびセキュリティポリシーをコンパクトに表現または近似するために賢い方式を使用しているが、これはしばしば意図されるポリシーに対する妥協であり、コンパクトさと引き換えに複雑さを犠牲にすることがある。本明細書で説明されるように、ランタイムオーバーヘッドをほとんどまたはまったく追加することなく、より完全かつより自然に、セキュリティポリシーの必要性を捉えるよりリッチなメタデータを含めることが可能である。メタデータ表現およびポリシーの複雑さに対して不変の制限を課すのではなく、PUMP10は性能のグレイスフルデグラデーションをもたらす。このことは、一般的な場合の性能およびサイズに影響を与えることなく、必要な場合にポリシーがより多くのデータを使用することを可能にする。このことはさらに、さらに複雑なポリシーを簡単に表現して実行できるので、ポリシーの付加的な改良および性能調整を可能にする。
【０１１１】
メタデータベースのポリシー実施の価値を証明するものが続々と現れる中で、本開示は、ソフトウェアで定義されたメタデータ処理のためのアーキテクチャを定義し、ランタイムオーバーヘッドの大半をなくすためのアクセラレータを特定する。専用のハードウェアメタデータ伝播方法に匹敵する性能を達成する4つのマイクロアーキテクチャ最適化(オペグループ、タグ圧縮、タグ変換、およびミスハンドラアクセラレーション)とともに同時にサポートされる、メタデータビットの数またはポリシーの数に対する制約のない(すなわち、あらゆる制約から自由である)、アーキテクチャが本明細書において紹介され説明される。ソフトウェアで定義されるメタデータポリシーモデルおよびそのアクセラレーションは、信頼できる情報フローの制御、細粒度のアクセス制御、整合性、同期、レース検出、デバッグ、アプリケーション固有のポリシー、ならびに動的コードの制御された生成および実行を含む、ここで示されるもの以外の広範囲のポリシーに適用可能である。
【０１１２】
本明細書で説明される様々な態様および実施形態のいくつかの非限定的な利点は、(i)このアーキテクチャによりサポートされるポリシーをコンパクトにかつ正確に記述するためのプログラミングモデルおよびサポートインターフェースモデル、(ii)よく研究されているポリシーの4つの異なるクラスを使用したポリシーの符号化および合成の詳細な例、ならびに(iii)これらのポリシーに対する要件、複雑さ、および性能の定量化を提供する。
【０１１３】
本明細書で説明されるような実施形態のプログラミングモデルは、他のポリシーのホストを符号化し得る。情報フロー制御は、ここでは簡単なテイント追跡モデルよりリッチであるが、暗黙的なフローを追跡することは、RIFLE型のバイナリ変換により、または、コンパイラからのある程度のサポートとともにPC tagを使用することにより、サポートされ得る。マイクロポリシーは、軽量なアクセス制御およびコンパートメント化をサポートすることができる。偽造不可能なリソースを区別するためにタグが使用され得る。固有の生成されたトークンが、データを検印して保証するための鍵として機能することができ、これを次いで強力な抽象化のために使用することができ、データが認証されたコード構成要素のみによって作成され破壊されることを保証する。マイクロポリシールールは、不変性お

よび線形性などのデータ不変条件を強制することができる。マイクロポリシーは、データまたは特徴のためのフル/エンプティビットなどの同期プリミティブに対するアウトオブバンドメタデータとして、または、ロック状態でレース条件を検出するための状態として、並列処理をサポートすることができる。システムアーキテクトは、1つ1つの行を検査することなく、または書き直すことなく、既存のコードに固有のマイクロポリシーを適用することができる。

【０１１４】

本明細書で説明されるPUMP10の設計は、フレキシビリティと性能の魅力的な組合せを提供し、低水準で細粒度のセキュリティポリシーの多様な集合を、多くの場合に専用の機構に匹敵する単一ポリシー性能とともにサポートしながら、ルールの複雑さが増すにつれて大抵の場合に性能のグレイスフルデグラデーションを伴うよりリッチな合成ポリシーをサポートする。さらに、PUMPによって提供される機構は、固有のソフトウェア構造を保護するために使用され得る。本明細書の技法に従ったある実施形態は、PUMP10を使用して「コンパートメント化」マイクロポリシーを実施することによって、かつ、これを使用してミスハンドラコードを保護することによって、特別なミスハンドラ動作モードを置き換えることができる。最後に、本明細書で説明されるように、ポリシーの直交する集合が組み合わされることがあり、ここで、各々の集合により提供される保護は、他方の集合とは完全に独立である。しかし、ポリシーはしばしば相互作用する。たとえば、情報フローポリシーは、メモリ安全性ポリシーにより割り振られている未使用の領域にタグを置くことが必要であり得る。ポリシー合成は、表現と、効率的なハードウェアサポートの両方に関連して、分析を必要とする。

【０１１５】

ここで説明されるのは、ヒープに割り振られたメモリにおけるすべての時間的および空間的な侵害を特定する、本明細書の技法に従ったある実施形態における、メモリ安全性ポリシーの実装形態を示すさらなる例である。少なくとも1つの実施形態において、各々の新しい割振りに対して、未使用の色idであるcを作り、新しく作られたメモリブロックの中の各メモリ位置のタグとしてcを書き込む(たとえば、memsetなどを介して)ために、処理が実行され得る。新しいブロックへのポインタもcとタグ付けされる。その後、処理がポインタをデリファレンスするために実行されるとき、処理は、ポインタのタグが、ポインタが参照するまたは指すメモリセルのタグと同じであることを確認することを含み得る。ブロックが解放されるとき、ブロックのすべてのセルのタグは、自由なメモリを表す定数Fに変更され得る。ヒープは最初はFとタグ付けされ得る。特別なタグ⊥が非ポインタのために使用され得る。したがって、一般に、ある実施形態は、色cまたは⊥のいずれかであるメモリ位置に対してタグtを書き込むことができる。

【０１１６】

メモリセルはポインタを格納し得るので、一般に、メモリの中の各ワードは2つのタグと関連付けられ得る。そのような実施形態では、各メモリセルのタグはペア(c,t)へのポインタであり、ここでcは、このセルが割り振られたメモリブロックのidであり、tは、セルに記憶されているワードのタグである。ある実施形態は、シンボリックルールに関するポリシーを指定するための、本明細書の他の箇所で説明されるルール関数に基づいて領域固有の言語を使用することができる。loadおよびstoreのためのルールは、これらのペアをパックおよびアンパックすることとともに、各メモリアクセスが有効であることを確認することを担当する(すなわち、アクセスされるセルはこのポインタによって指されるブロック内にある)。

$load:(-,-,c_1,-,(c_2,t_2))$

$\rightarrow(-,t_2)$ if $c_1=c_2$

$store:(-,-,t_1,c_2,(c_3,t_3))$

$\rightarrow(-,(c_3,t_1))$ if $c_2=c_3$

【０１１７】

前述のルールおよび他のルールにおいて実行される確認は、シンボリックルールが有効で

ある条件(たとえば、上のstoreルールでは$c_2=c_3$)として現れる。「-」記号は、ルールにおけるdon't careフィールドを示す。
アドレス算術演算がポインタタグを保存する:
add: (-,-,c,⊥,-)→(-,c)
不変条件を保つために、ポインタ上のそのタグは割振りから発生することだけが可能であり、最初からデータを作成する動作(たとえば、定数のロード)はそのタグを⊥に設定する。

【０１１８】

メモリ安全性ポリシーを実施するある実施形態では、それに従って、たとえば、タグ付けされた命令および一時的なルール(たとえば、使用されるとキャッシュから削除され得る)を使用してメモリ領域をタグ付けするために、mallocおよびfreeなどの動作が変更され得る。mallocに関連して、処理は、一時的なルールを介して、新しい領域へのポインタのために未使用のタグを生成し得る。たとえば、moveのためのルールは、
move: $(-,t_{malloc},t,-,-)\rightarrow^{1}(-,t_{newtag})$
などの一時的なルールであり得る。1の上付き文字を伴う矢印(たとえば、$\rightarrow^{1}$)は、一時的なルールを示し得る。次いで、タグ付けされたポインタを返す前に、特別なstoreルール
store: $(-,t_{mallocinit},t_1,c_2,F)\rightarrow(-,(c_2,t_1))$
を使用して、割り振られる領域の中の1つ1つの作業に0を書き込むために、新しくタグ付けされたポインタが使用され得る。ある後の時点で、freeは、領域をfreeリストに返す前に、割り振られていないものとして領域を再タグ付けするために、変更されたstore命令を使用し得る。
store: $(-,t_{freeinit},t_1,c_2,(c_3,t_4))\rightarrow(-,F)$

【０１１９】

メモリ安全性ポリシーを使用するそのような実施形態では、オペグループが次のようにルール集合を記述するために使用され得る。
(1) nop,cbranch,ubranch,ijump,return: (-,-,-,-,-)→(-,-)
(2) ar2sld: (-,-,⊥,⊥,-)→(-,⊥)
(3) ar2sld: (-,-,c,⊥,-)→(-,c)
(4) ar2sld: (-,-,⊥,c,-)→(-,c)
(5) ar2sld: (-,-,c,c,-)→(-,⊥)
(6) ar1sld: (-,-,t,-,-)→(-,t)
(7) ar1ld,dcall,icall,flags: (-,-,-,-,-)→(-,⊥)
(8) load: $(-,-,c_1,-,(c_2,t_2))\rightarrow(-,t_2)$ if $c_1=c_2$

【０１２０】

【数１】

$$(9)\quad \text{store:}\quad (-,-,t_1,c_2,(c_3,t_3))\rightarrow(-,(c_3,t_1))\ \text{if}\ c_2=c_3\wedge c_i\notin\{t_{mallocinit},t_{freeinit}\}$$

【０１２１】

(10) store: $(-,t_{mallocinit},t_1,c_2,F)\rightarrow(\rightarrow,(c_2,t_1))$
(11) store: $(-,t_{freeinit},t_1,c_2,(c_3,t_4))\rightarrow(\rightarrow,F)$
(12) move: $(-,t_{malloc},t,-,-)\rightarrow^{1}(-,t_{newtag})$

【０１２２】

【数２】

$$(13)\quad \text{move:}\quad (-,\overline{t_{malloc}},t,-,-)\rightarrow(-,t)$$

【０１２３】

ポリシー指定のために上で使用されたシンボリックルールは変数を使用して書かれること

があり、少数のシンボリックルールが無限の領域の別個の値にわたってポリシーを記述することを可能にする。しかしながら、ルールキャッシュに記憶されるコンクリートルールは、特定の具体的なタグ値を指す。たとえば、23および24が有効なメモリブロック色である場合、ある実施形態は、c=23およびc=24に対してPUMPルールキャッシュにおいて上のシンボリックルール(3)という具体的な事例を伴うコンクリートルールを使用し得る。たとえば、ある実施形態が⊥を0として符号化し、don't careフィールドを0としてマークすると仮定すると、上のシンボリックルール(3)のためのコンクリートルールは次の通りである。

ar2sld:(0,0,23,0,0,)→(0,23)

ar2sld:(0,0,24,0,0,)→(0,24)

本明細書の他の箇所の議論と一貫して、少なくとも1つの実施形態では、ミスハンドラは、具体的な入力タグを取得し、ルールをPUMPルールキャッシュへと挿入するためにシンボリックルールからコンパイルされたコードを実行して関連する具体的な出力タグを産生することができる。シンボリックルールが侵害を特定するとき、制御はエラーハンドラに移転し、新しいコンクリートルールはPUMPルールキャッシュに挿入されない。

【０１２４】

ここで説明されるのは、本明細書の議論と一貫したソフトウェアで定義されるメタデータ処理(SDMP:software defined metadata processing)をサポートするためにメタデータタグおよびPUMPを用いてさらに拡張されるRISC-Vアーキテクチャに基づく、本明細書の技法に従ったある実施形態である。RISC-Vは、縮小命令セットコンピューティング(RISC)命令セットアーキテクチャ(ISA)のオープンソース実装形態として特徴付けられ得る。そのような実施形態では、メタデータタグは、各ワードのための命令とデータの両方に付けられる。RISC-Vアーキテクチャでは、ワードは64ビットである。RISC-Vアーキテクチャは、ワードサイズが64ビットであるRV64およびワードサイズが32ビットであるRV32という、ワードサイズが異なる変形を提供する。レジスタおよびユーザアドレス空間の幅およびサイズは、ワードサイズとともに変化し得る。タグのサイズおよび幅は、ワードのサイズまたは幅とは無関係であり得るが、より典型的には、ある実施形態では同じであり得る。当技術分野において知られているように、RISC-Vアーキテクチャは32ビットの命令を有するので、64ビットのワードサイズをサポートしそれを使用して動作するある実施形態は、単一のタグ付けされたワードに2つの命令を記憶し得る。RISC-Vアーキテクチャの前述の態様および他の態様は、メタデータタグ、PUMP、およびSDMPとともに使用するためにRISC-Vアーキテクチャを拡張することに関連する様々な技法および特徴の使用に関連して、本明細書の他の箇所で論じられる。

【０１２５】

RISC-Vアーキテクチャは、たとえば、RISCV.ORGのウェブサイトにおいて、およびカリフォルニア大学バークレー校を通じてTechnical Report UCB/EECS-2014-54としてたとえば公に入手可能である、本明細書に参照によって組み込まれる、「The RISC-V Instruction Set Manual Vol. I, User-Level ISA, Version 2.0」、2014年5月6日、Waterman、Andrew他(「RISC-Vユーザレベル ISA」とも呼ばれる)において説明されるような、ユーザレベル命令を含む。RISC-Vアーキテクチャはまた、たとえば、RISCV.ORGのウェブサイトにおいて、およびカリフォルニア大学バークレー校を通じてTechnical Report UCB/EECS-2015-49としてたとえば公に入手可能である、本明細書に参照によって組み込まれる、「The RISC-V Instruction Set Manual Volume II: Privileged Architecture, Version 1.7」、2015年5月9日(「RISC-V特権ISA」とも呼ばれる)において説明されるような、オペレーティングシステム、取り付けられた外部デバイスなどを実行するのに必要な、特権命令および追加の機能を含む、特権アーキテクチャを組み込む。

【０１２６】

RISC-Vアーキテクチャのある実施形態は、ユーザ/アプリケーション(U)特権レベルに対するレベル0、スーパーバイザー(S)特権レベルに対するレベル1、ハイパーバイザー(H)特権レベルに対するレベル2、およびマシン(M)特権レベルに対するレベル3という、4つのRIS

C-V特権レベルを有し得る。上記では、RISC-V特権レベルは0から3まで上から下にランク付けされることがあり、レベル0は最高または最大の特権レベルを示し、レベル3は最低または最小の特権レベルを示す。そのような特権レベルは、異なる構成要素の間での保護と、背後の実行環境へのトラップなどの例外を提起させるであろう現在の特権レベルまたはモードにより許可されない動作を実行するコードを実行する試みとを、提供するために使用され得る。マシンレベルは最高の特権を有し、RISC-Vハードウェアプラットフォームに対する唯一の必須の特権レベルである。マシンモード(Mモード)におけるコードの実行は、マシン実装への低水準のアクセス権を有するので、本質的に信頼される。ユーザモード(Uモード)およびスーパーバイザーモード(Sモード)はそれぞれ、従来のアプリケーションおよびオペレーティングシステムの使用に対して意図されたものであり、一方でハイパーバイザーモード(Hモード)は仮想マシンモニタをサポートすることが意図されている。各特権レベルは、任意選択の拡張および変形を伴う特権ISA拡張のコアセットを有する。RISC-Vアーキテクチャの実装形態は少なくともMモードをサポートしなければならず、大半の実装形態は少なくともUモードおよびMモードをサポートすることに留意されたい。スーパーバイザーレベルのオペレーティングシステムのコードと、Mモードで実行する他のより特権的なコードとをさらに隔離するために、Sモードが追加され得る。ユーザコードまたはアプリケーションコードは通常、サポートされるより高い特権モードまたはレベル(たとえば、H、S、またはMモード)のうちの1つで実行するトラップハンドラへの制御の移転を強制する、トラップ(たとえば、スーパーバイザー呼出し、ページフォルト)または中断が発生するまで、Uモードで実行し得る。次いで、トラップハンドラのコードが実行され、次いで制御がトラップを引き起こした元のユーザコードまたはアプリケーションに返され得る。ユーザコードまたはアプリケーションのそのような実行は、トラップハンドラ呼出しをトリガしたUモードにおける元のトラップされた命令において、またはその後で再開し得る。RISC-V実装形態におけるサポートされるモードの様々な組合せは、単一のMモード、MおよびUの2つのモード、M、S、およびUの3つのモード、またはすべての4つのモードM、H、S、Uのみを含み得る。本明細書で説明される少なくとも1つの実施形態では、前述の特権レベルの4つすべてがサポートされ得る。最低でも、本明細書の技法に従ったある実施形態は、MモードおよびUモードをサポートし得る。

【０１２７】

RISC-Vアーキテクチャは、1つまたは複数の関連する特権レベルによってアトミックに読み取られ修正され得る制御ステータスレジスタ(CSR)を有する。一般に、CSRは、4つの特権レベルのうちの第1のレベル、および第1のレベルより高い4つの特権レベルのうちの任意の他のレベルにおいてアクセス可能であり得る。たとえば、プログラムがUモード(レベル3)において実行しており、ルールキャッシュミスなどのトラップが発生して、それにより制御がルールキャッシュミスハンドラコードなどのトラップハンドラに移転し、より高い特権またはモード(レベル0～2のいずれか)で実行すると仮定する。トラップが発生すると、Mモードで実行しているトラップハンドラがアクセス可能な、たとえば、そうでなければより低い特権レベルで実行しているいずれの他のコードもアクセス可能ではない(たとえば、H、S、またはUモードのコードがアクセス可能ではない)CSRに、情報が置かれ得る。少なくとも1つの実施形態では、ルールキャッシュミスハンドラは、PUMP保護のレベルを超える特権レベルで実行し得る(たとえば、Hモード、Sモード、またはMモードで実行し得る)。そのような実施形態では、本明細書の他の箇所で説明されるように、タグの定義およびポリシーは、ルールキャッシュミスハンドラレベルではオペレーティングシステムにわたって(たとえば、仮想マシンごとに)グローバルであることがあり、これにより、同じタグの定義およびポリシーがすべての実行中のコードにわたって適用されることがある。少なくとも1つの実施形態では、アプリケーションまたはプロセスごとにポリシーがサポートされることがあり、ここで、そのようなポリシーはグローバルにインストールされ、PC(現在の命令を特定するプログラムカウンタ)および/またはコードは、プロセスまたはアプリケーション固有のルールを区別するためにタグ付けされ得る。仮想マシン(VM)がメモリを共有しないある実施形態では、ポリシーはVMごとに定義され得る。

【０１２８】
本明細書の他の箇所の議論と一貫して、PUMPはSDMPのためのルールキャッシュとして特徴付けられ得る。命令のタグの集合と、命令入力と、動作の結果に対するタグとの間にマッピングがあり得る。タグ処理は、命令の普通の動作とは独立であり並列である。少なくとも1つの実施形態では、PUMPは普通のRISC-V動作と並列に実行し、動作の結果に対するタグを供給する。PUMPはキャッシュであるので、ルールキャッシュミスは、PUMPが特定の命令、したがってPUMP入力のある特定の対応する集合を受信する最初のときに(たとえば、強制的)、または、PUMPがキャッシュにルールを保持することが不可能であったとき(たとえば、キャッシュの容量を超えたので、ルールがルールキャッシュから排除された、または場合によっては競合する)に、発生する。ルールキャッシュミスは、ミスハンドラシステム(たとえば、ルールキャッシュミスハンドラ)のコードにより次いで扱われるミストラップを引き起こす。入力はPUMP CSRを通じてミスハンドラに伝えられることがあり、ルール挿入はやはりCSRを通じてPUMPに戻されることがある。これは以下でより詳細に論じられる。5つのPUMP入力タグがある第1の実施形態が本明細書の他の箇所で論じられる。変形として、ある実施形態は、異なる数のタグおよび他のPUMP入力を含み得る。PUMPタグ入力の具体的な数は、命令セットおよびオペランドとともに変化し得る。たとえば、RISC-Vアーキテクチャに基づく一実施形態では、以下がPUMP入力として含まれ得る。
1．Opgrp - 特定のオペグループが現在の命令を含むことを示す。一般に、オペグループは命令のグループの抽象化であり、本明細書の他の箇所で論じられる。
2．PCtag - PCのタグ
3．CItag - 命令のタグ
4．OP1tag - 命令へのRS1入力のタグ
5．OP2tag - 命令へのRS2入力のタグ(またはCSR命令のときはCSRのタグ)
6．OP3tag - 命令へのRS3入力のタグ
7．Mtag - 命令へのメモリ入力または命令のメモリターゲットのタグ
8．funct12(funct7) - 本明細書の他の箇所で説明されるようないくつかの命令で発生する拡張されたオペコードビット
9．subinstr - あるワードに詰め込まれた複数の命令があるとき、この入力は、そのワードの中のどの命令がPUMPにより動作している現在の命令であるかを特定する。
【０１２９】
以下は、RISC-Vアーキテクチャに基づく一実施形態におけるPUMP出力として含まれ得る。
1．Rtag - 結果のタグ:宛先レジスタ、メモリ、またはCSR
2．newPCtag - この動作の後のPCのタグ(たとえば、場合によってはPCnew tagと本明細書では呼ばれる)
【０１３０】
たとえば、トラップの発生の時間においてUモードで実行しているユーザコードから、CSRを介してMモードで実行しているルールキャッシュミスハンドラなどのトラップハンドラに、情報が伝えられ得る。同様の方式で、CSRを介してUモードでプログラム実行を再開するときにMモードにあるトラップハンドラの間で情報が伝えられることがあり、ここで、CSRの中の情報はUモードでアクセス可能な対応するレジスタに置かれることがある。このようにして、1つの特権レベルにおけるCSRと他の特権レベルにおけるレジスタとの間にはマッピングがあり得る。たとえば、本明細書の技法に従ったある実施形態では、MモードハンドラおよびPUMPがアクセス可能であるCSRが定義されることがあり、ここで、特定の命令オペランドタグが、入力としてタグをPUMPおよびルールキャッシュミスハンドラに伝えるために、トラップが発生するとCSRに書き込まれる。同様の方式で、CSRは、ルールキャッシュミスの後でプログラム実行を再開するときなどに(たとえば、一致ルールが現在の命令のためのPUMPルールキャッシュにおいて見つからないときにルールキャッシュミスが発生する)、トラップハンドラおよび/またはPUMP(Uモードより高い特権レ

ベルで動作する)からUモードで実行している他のコードに情報を伝えるために使用され得る。たとえば、CSRはPCnewおよびRDのためのPUMP出力タグを出力または伝播するために使用され得る。加えて、CSRは、異なる活動が特定のCSRへの書込みに応答して発生し得るように定義され得る。たとえば、ルールキャッシュミスハンドラコードは、特定のCSRに書き込むことによって、新しいルールをPUMPのルールキャッシュへと書き込む/挿入することができる。定義される特定のCSRは実施形態により変化し得る。

【０１３１】

図25を参照すると、本明細書の技法に従った一実施形態において定義され使用され得る、CSRの例が示されている。表900は、16進数のCSRアドレスを伴う第1の列902と、特権の第2の列904と、CSR名を示す第3の列906と、CSRの説明を伴う第4の列908とを含む。表900の各行は、異なる定義されたCSRに対する情報を特定し得る。900の中の様々なCSRは、実施形態に含まれ得る追加の特徴に関連して、本明細書の他の箇所でもより詳細に説明され得る。

【０１３２】

行901a～cは、PUMPによりコードおよび/または命令をタグ付けするために使用される特定のタグ値を有するCSRを特定する。少なくとも1つの実施形態では、エントリー901aにより定義されるsboottag CSRは、システムにおいて使用される第1の初期のまたは最初のタグ値を含み得る。前述の最初のタグ値はブートストラップタグ値と呼ばれ得る。一態様では、ブートストラップタグ値は、すべての他のタグがそこから導出され得る、またはそれに基づき得る、「シード」として特徴付けられ得る。したがって、一実施形態では、すべての他のタグを生成するための始点として、ブートストラップタグが使用され得る。オペレーティングシステムにおけるブートストラップコードの開始位置の初期ロードと同様の方式で、ハードウェアは、ブートストラップタグとして使用される特定の事前に定義されたタグ値にCSR901aを初期化するために使用され得る。ブートストラップタグが本明細書の技法に従ってシステムをブートすることの一部として読み取られると、sboottag CSRはクリアされ得る。たとえば、オペレーティングシステムコードの特権部分は、ブートストラップタグ値を使用して初期タグの伝播を実行するルールを呼び出す命令を含み得る。ブートストラップタグの使用ならびにタグの生成および伝播はさらに、本明細書の他の箇所で説明される。行901bは、本明細書の他の箇所で説明されるような公に信頼されていないソースからのデータをタグ付けするために使用されるタグ値を格納するCSRを特定する。行901cは、データおよび/または命令をタグ付けするときにデフォルトのタグ値として使用され得る、デフォルトタグ値を格納するCSRを特定する。

【０１３３】

行901dおよびeはそれぞれ、オペグループ/ケアテーブル(たとえば、オペコードのためのオペグループおよびcare/don't careビットを含むマッピングテーブルまたは変換テーブルとも本明細書の他の箇所では呼ばれる)に書き込むためのアドレスおよびデータを示す。行901eにより示されるCSRへの書込みは、オペグループ/ケアテーブルへの書込みをトリガする。行901fは、PUMPルールキャッシュをフラッシュするために書き込まれ得るCSRを特定する。行901g～901mは、PUMPおよびルールキャッシュミスハンドラに現在の命令のためのタグ入力を提供するCSRを特定する。行901j～mは各々、ルールキャッシュミスを引き起こす、処理されている現在の命令のオペランドのための様々なオペランドタグを示し、これにより、命令は最大で4つのオペランドを含み得る(4つのオペランドのうちの3つはレジスタ(CSR901j～l)であり、4番目のオペランドは行901mにより示されるCSRに記憶されたタグを伴うメモリ位置である)。行901nは、本明細書の他の箇所で説明されるように、現在の命令のオペコードが拡張されたfunc12フィールドを使用するときに拡張されたオペコードビットを保持するCSRを特定する。行901oは、ワードの中のどのサブ命令が参照されている現在の命令であるかを示すCSRを特定する。本明細書の他の箇所で論じられるように、単一のタグ付けされたワードは64ビットであることがあり、各命令は32ビットであることがあり、これにより、2つの命令は単一のタグ付けされたワードに含まれることがある。行901oにより示されるCSRは、2つの命令のいずれがPUMPによって

処理されているかを特定する。行901p～qは、それぞれ新しいPC(次の命令のための新しいPC tag)およびRD(現在の命令の結果のための宛先レジスタ、アドレス)の、PUMP出力タグを含むCSRを特定する。901qにより示されるCSRへの書込みは、PUMPルールキャッシュへのルール(たとえば、PUMPルールキャッシュミスをトリガした現在の命令と一致する)の書込みを引き起こす。行901rはPUMP動作のためのtagmodeを特定する。tagmodeは本明細書の他の箇所でより詳細に説明される。

【０１３４】

少なくとも1つの実施形態では、オペグループおよびcare/don't careビットを記憶するために使用される1つまたは複数のテーブル(たとえば、オペグループ/ケアテーブル)は、901eにより示されるsopgrp値をCSRに書き込むことによって埋められることがあり、ここで、前述のCSR901eの内容は、901dにより示されるsopgrpaddr CSRに記憶されているアドレスに書き込まれる。ルールは、エントリー901qにより定義されるstrag CSRに書き込んだことに応答して、PUMPルールキャッシュに書き込まれ、またはインストールされ得る。書き込まれるルールは、オペコード(またはより具体的にはオペコードのためのオペグループ)と一致するタグ値と、(たとえば、PUMP CSR入力901g～oに基づく)PUMP CSRを介したPUMPへの入力としての現在の命令のためのタグ値とを指定する、ルールである。

【０１３５】

CSR動作に対するタグ付けおよびタグ保護を許容するために、データフローは、CSRタグがPUMPへの入力またはPUMPからの出力であることを許容する。RISC-Vアーキテクチャによれば、それぞれCSRから読み取るべき、またはCSRに書き込むべき、読取りおよび書込み命令がある。PUMPについてのCSR命令に関連して、PUMPへのR2tag入力は現在のCSRタグである。CSR読取り/書込み命令(たとえば、csrrc、csrrci、csrrs、csrrsi、csrrw、csrrwi)は、(1)RD、および(2)命令により参照されるCSRという2つの入力を書き込む。この場合、PUMP出力のR tag(または宛先のRD tag)は、PUMPによって出力されるCSR tagを指定し、CSRtagを直接レジスタ宛先タグにコピーする。

・ RDtag←CSRtag
・ CSRtag←Rtag

【０１３６】

列904により示される特権に関連して、行901rにより定義されるCSR mtagmodeは、マシンレベルまたはMモードレベルで実行しているコードによる読取り/書込みのためにアクセス可能である。行901a～qにより定義される残りのCSRは、少なくともスーパーバイザーレベルまたはSモードレベルで実行しているコードによる読取り/書込みのためにアクセス可能である。したがって、様々なCSRのために列904において示される特権は、コードが特定のCSRにアクセスするために、実行中のコードの最小のRISC-V特権レベルを示す。ある実施形態は、例900において示されるものとは異なるRISC-V特権レベルを、ある実施形態において使用されるCSRに割り当て得る。

【０１３７】

本明細書の技法によるある実施形態は、PUMPにより実行されるタグ伝播に影響する複数のタグモードを定義し得る。現在のタグモードは、行901rにより定義されるようなCSR mtagmodeに記憶されている現時点での値によって特定される。少なくとも1つの実施形態では、タグモードは、PUMPに関連して使用されるCSR保護モデルを定義するために、RISC-Vで定義される特権(たとえば、上で説明されたM、H、S,およびUモード)と組み合わせて使用され得る。

【０１３８】

ルールキャッシュミスハンドラの配置を構成可能にするために、RISC-V特権をさらに拡張する保護モデルが利用され得る。特権レベルによってPUMP CSRアクセスを完全に定義するのではなく、CSRアクセスはさらに、RISC-V特権レベルと組み合わせて現在のタグモードに対して相対的に定義され得る。したがって、本明細書の技法による少なくとも1つの実施形態では、実行中のコードがCSRにアクセスすることが許容されるかどうかは、CSR

の最低のRISC-V特権レベル、現在のタグモード、および実行中のコードの現在のRISC-V特権レベルに依存し得る。タグモードは以下でより詳細に論じられる。

【０１３９】

図26を参照すると、本明細書の技法によるある実施形態において使用され得るタグモードの例が示されている。表910は、mtagmodeビット符号化(912)、動作(914)、およびタグ結果(916)という列を含む。表910の各行は、様々な可能なタグモードに対する情報を示す。タグモードが911aにより示されるように000であるとき、PUMPはオフであり、使用されておらず、タグ結果を生成しない。タグモードが010であるとき、PUMPは、すべての結果にデフォルトタグ(たとえば、宛先または結果のレジスタもしくはメモリ位置に対してはRtag)を書き込む。

【０１４０】

行911c～fに関連して、異なるRISC-V特権レベルで実行しているコードのためのPUMPを関与させるために、または関与させないために指定され得る、様々なタグモードが示されている。PUMPが関与しているとき、PUMPはアクティブであると特徴付けられ、有効にされ、コードが実行されるときに保護を提供することがあり、これにより、ポリシーのルールがコード実行の間に実施される。対照的に、PUMPが不関与であるとき、PUMPは非アクティブであると特徴付けられ、無効にされ、コードが実行されるときに保護を提供しないことがあり、これにより、ポリシーのルールがコード実行の間に実施されない。PUMPが不関与であるとき、タグは、現在の命令のタグ値と一致するタグ値を伴うルールの評価に基づいてタグを伝播させるのではなく、1つまたは複数のデフォルトのタグ伝播ルールを使用して伝播され得る。PUMPが関与するか不関与であるかは、異なるRISC-V特権レベルで実行するコードに起因する具体的な仮定される信頼のレベルおよび望まれる保護のレベルとともに変化し得る。

【０１４１】

タグモード911c～fに関連して、例900のすべてのPUMP CSRは、901rにより示されるmtagmode CSRを除き、PUMPが不関与であるときにのみアクセス可能であり得る。すなわち、例900のPUMP CSRは、901rにより示されるmtagmode CSRを除き、現在のRISC-Vの動作特権で実行しているコード、または、タグモードにより示される最高ランクのPUMP特権よりも優先されるモード(たとえば、911cにより示される最高ランクの特権はUモードであり、911dにより示される最高ランクの特権はSモードであり、911eにより示される最高ランクの特権はHモードであり、911fにより示される最高ランクの特権はMモードである)にのみアクセス可能である。

【０１４２】

タグモードが911cにより示されるように100であるとき、PUMPは、RISC-V特権レベルがUモードよりも高い、または高められた特権レベルを示すときに、不関与であり動作しない。したがって、タグモード911cは、コードがUモードで実行しており、それにより、Uモードより高い特権レベルで(たとえば、S、M、またはHモードで)実行しているコードが信頼されることを示すときにのみ、保護を提供するPUMPおよびそのルールが関与し実施されることを示す。タグモードが911cにより示されるように100であり、実行中のコードのRISC-V保護レベルがS、M、またはHモードであるとき、PUMPは不関与であり、そのCSRは、S、M、またはHモードだけで実行するコードがアクセス可能である(たとえば、CSRはUモードで実行しているコードはアクセス可能ではない)。

【０１４３】

タグモードが911dにより示されるように101であるとき、RISC-V特権レベルがSモードよりも高い、またはより高められた特権レベルを示すときに、PUMPは不関与であり動作しない。したがって、タグモード911dは、コードがSモードおよびUモードで実行しており、それにより、Sモードより高い特権レベルで(たとえば、MまたはHモードで)実行しているコードが信頼されることを示すときにのみ、保護を提供するPUMPおよびそのルールが関与し実施されることを示す。タグモードが911dにより示されるように101であり、実行中のコードのRISC-V保護レベルがMまたはHモードであるとき、PUMPは不関与であ

り、そのCSRは、MまたはHモードで実行するコードだけがアクセス可能である(たとえば、CSRはSまたはUモードで実行しているコードがアクセス可能ではない)。

【０１４４】

タグモードが911eにより示されるように110であるとき、RISC-V特権レベルがHモードよりも高い、またはより高められた特権レベルを示すときに、PUMPは不関与であり動作しない。したがって、タグモード911eは、コードがHモード、Sモード、およびUモードで実行しており、それにより、Hモードより高い特権レベルで(たとえば、Mモードで)実行しているコードが信頼されることを示すときにのみ、保護を提供するPUMPおよびそのルールが関与し実施されることを示す。タグモードが911eにより示されるように110であり、実行中のコードのRISC-V保護レベルがMモードであるとき、PUMPは不関与であり、そのCSRは、Mモードで実行するコードだけがアクセス可能である(たとえば、CSRはSまたはU、H、またはSモードで実行しているコードがアクセス可能ではない)。

【０１４５】

タグモードが911fにより示されるように111であるとき、M、H、S、およびUのすべてのRISC-V特権レベルに対して、PUMPは常に関与し動作する。したがって、タグモード911fは、コードがMモード、Hモード、Sモード、およびUモードのいずれかで実行しており、それにより、コードが本質的に信頼されないことを示すとき、保護を提供するPUMPおよびそのルールが関与し実施されることを示す。911fにより示されるようなタグモード=111では、PUMPは決して不関与にならず、そのCSRはいずれの実行中のコードもアクセス可能ではない。

【０１４６】

行911c～fにより示されるタグモードに関連して、実行中のコードの現在のRISC-V特権レベルが、タグモードにより示される最高の関与するPUMPレベルより高いとき、PUMPは不関与であることがあり、タグは1つまたは複数のデフォルトのタグ伝播ルールを使用して伝播されることがある。

【０１４７】

タグモードが行911aにより示されるように000という符号化を有するとき(PUMPがオフであることを示す)、またはタグモードが行911bにより示されるように010という符号化を有するとき(書込みデフォルトモードを示す)、表900のすべてのCSRは、Mモードで実行するコードのみによってアクセス可能であり得る。

【０１４８】

したがって、本明細書の技法に従った少なくとも1つの実施形態では、実行中のコードがCSRにアクセスすることを許容されるかどうかは、CSRの最低のRISC-V特権レベル(表900の列904において指定されるものなど)、現在のタグモード、および実行中のコードの現在のRISC-V特権レベルに依存し得る。たとえば、タグモードを考慮しないRISC-Vアーキテクチャでは、Uモードで実行しているコードは、900において定義されたCSRのいずれにも、すべてのそのようなCSRに対して904により示される最低の特権レベルが原因で、アクセスすることが許容されない。しかしながら、タグモードを考慮することなく、少なくともHモードの特権で実行しているコードは、901rを除き900のすべてのCSRにアクセスすることが許容され、Mモードで実行しているコードは、900のすべてのCSRにアクセスすることが許容される。ここで、904の最低のRISC-V特権およびタグモードに従って、900のCSRのためのCSRアクセス権を判定することを考える。たとえば、Hレベルで実行しているコード部分Aを考える。コード部分Aは、タグモードが911cにより示されるように100であるとき、またはタグモードが911dにより示されるように101であるときに、(表900の)CSR901a～qにアクセスすることが許容される。しかしながら、Sモードで実行しているコード部分BはCSR901a～qにアクセスすることが許容されないことがあり、それは、コード部分Bが、そのようなCSRのために904において定義されるCSR特権レベルにより指定される最低の特権レベルを有しないからである。したがって、たとえば、コード部分Aは、表900において定義されるようなCSRを使用してHレベルで実行する一実施形態におけるキャッシュミスハンドラであり得る。第2の例では、CSR901a～qに対して定義さ

れる最低のRISC-V特権がSRWであると仮定する(そのようなCSRにアクセスするための最低の特権レベルとしてSモードを示す)。Hモードで実行しているコード部分Aは、タグモードが911cのように100であるとき、およびタグモードが911dのように101であるときにCSR901a～qにアクセスすることが許容され、Sモードで実行しているコード部分Bは、タグモードが911cのように100であるときにCSR901a～qにアクセスすることが許容される。したがって、コード部分AまたはBはキャッシュミスハンドラのコードであり得る。

【０１４９】

少なくとも1つの実施形態では、911aのoffタグモードは、ブートアッププロセスの適切な部分の間などの、PUMPがオフであるであるときの現在のタグモードであり得る。911bのdefault tagタグモードは、同じデフォルトタグ(たとえば、CSR901cにより示されるような)を有するようにメモリ位置を初期化するときの現在のタグモードであり得る。一般に、4つの特権モードがRISC-Vアーキテクチャにおいて指定されるが、ある実施形態は、第1の特権レベルがユーザモードまたは非特権モードを示し、第2の特権レベルが実行の高められたモードまたは特権モード(たとえば、UNIX（登録商標）ベースのオペレーティングシステムにおけるカーネルモードと同様)を示す、異なる数の特権モードを代わりに使用し得る。そのような実施形態では、PUMPは、ユーザモードまたは非特権モードでコードを実行するときに、関与しておりポリシールールを実施していることがあり、PUMPは、第2の高められた特権モードでコードを実行するときに、不関与である(たとえば、PUMP保護オフまたはルールを実施しない)ことがある。このようにして、ある実施形態は、PUMPルールキャッシュに新しいルールを記憶するために、ミスハンドラなどの信頼される特権コードまたは高められた特権コードを実行するときに、を関与させないことがある。

【０１５０】

上で述べられたように、ある実施形態は、たとえば、PUMPが不関与であるとき、かつ/または、PUMPに対するdon't careが新しいPC tagおよびR tagを出力することをルールが指定するとき(たとえば、そのようなdon't care値は現在の命令の特定のオペコードのためのケアベクトルにより示され得る)、デフォルトの伝播ルールを使用して、PUMPが新しいPC tagおよびR tagを出力することを判定し得る。一実施形態では、以下が、使用されるデフォルトの伝播ルールにおいて具現化される論理を示し得る。

・ newPCtagはデフォルトの伝播のためのPCtagである

・ RtagはCSRの読取りおよび書込みの動作のためのRS1tagである;RDtagはRS2tag(CSRtag)を割り当てられる

- タグがデータ値とともにスワップすることを可能にする
- RDtag←RS2tag←元のCSRtag
- CSRtag←Rtag←元のRS1tag

・ RtagはCSRR?I、CSRRS、CSRRCのためのRS2tag(CSRtag)である

- CSRtagを不変に保つ
- RDtag←RS2tag←元のCSRtag
- CSRtag←Rtag←元のRS2tag←元のCSRtag

・ RtagはJALおよびJALR命令のためのPCtagである(これはリターンアドレスのためのものである)

・ RtagはAUIPC命令のためのPCtagである。RISC-Vでは、AUIPC(add upper immediate to PC)命令が、PC関連のアドレスを構築するために使用され、Uタイプのフォーマットを使用する。AUIPCは、20ビットのU-immediateから32ビットのオフセットを形成し、下位12ビットを0で埋め、このオフセットをPCに追加し、次いで結果をレジスタrdに置く。

・ RtagはLUI命令のためのCItagである。RISC-Vでは、LUI(load upper immediate)命令は、32ビットの定数を構築するために使用され、Uタイプのフォーマットを使用する。LUIは、宛先レジスタRDの上位20ビットにU-immediate値を置き、下位12ビットを0で埋める。

・ Rtagは、非メモリ、非CSR、および非JAL(R)/AUIPC/LUI動作のためのRS1tagである

・ Rtagはメモリ書込み動作のためのRS2tagである
・ Rtagはメモリロード動作のためのMtagである
【０１５１】
RISC-Vアーキテクチャに基づく本明細書の技法の少なくとも1つの実施形態では、新しいPUMPミストラップがルールキャッシュミスの発生に対して定義され得る。PUMPミストラップは、仮想メモリのフォルトまたは不当な命令よりも優先度が低いことがある。
【０１５２】
RISC-Vアーキテクチャを使用した本明細書の技法に従った少なくとも1つの実施形態では、データとメタデータとの間の厳密な分離および隔離が維持されることがあり、ここで、タグメタデータ処理と普通の命令処理との間には分離および隔離がある。したがって、メタデータルール処理と、普通のまたは典型的なプログラム命令実行との間で、別々の実行領域が維持され得る。実行中のコードの命令およびデータと関連付けられるタグのためのPUMPを使用して実行されるメタデータ処理が実行され得る。PUMPルールキャッシュミスは、現在の命令と一致するルールを生成または取得してそのルールをPUMPルールキャッシュに記憶するルールキャッシュミスハンドラへの制御の移転を引き起こす、トラップをもたらす。CSRを使用して、上で述べられた実行領域の間で情報が伝えられ得る。実行中のプログラムの命令実行領域からメタデータルール処理領域に切り替えるとき(ルールキャッシュミスハンドラがルールキャッシュミストラップを介してトリガされるときなど)、(トラップを引き起こす)命令に関連するタグおよび他の情報が、CSRを使用して、PUMPに、またミスハンドラにも、入力として与えられ得る。同様の方式で、メタデータルール処理領域から実行中のプログラムの命令実行領域に制御を移転するとき(ルールキャッシュミストラップを扱った後でルールキャッシュミスハンドラから戻るときなど)、PUMP出力はCSRを使用して伝えられることがあり、ここで、CSRの内容が次いで、命令実行領域において対応するマッピングされたレジスタに記憶される。本明細書の議論と一貫して、ルールにマッピングしない命令(たとえば、その命令に対する一致するルールがキャッシュの中に位置せず、そのような一致するルールが現在の命令に対して存在しないとキャッシュミスハンドラが判定する)は、そのルールの実行が許容されないことを示し、これによりトラップまたは他のイベントがトリガされる。たとえば、プロセッサは、現在のプログラムコードの実行を停止し得る。
【０１５３】
このようにして、同じRISC-Vプロセッサおよびメモリが両方の領域において使用され得るとしても、前述の領域と関連するデータパスとの間に厳密な分離があり得る。本明細書の技法を使用すると、実行中のコードの命令は、メタデータタグもしくはルールを読み取ることまたは書き込むことが許容されない。命令およびデータのタグ付けを含むすべてのメタデータの変換は、PUMPを通じて行われ得る。同様に、PUMPキャッシュへのルールの挿入は、メタデータサブシステムのルールキャッシュミスハンドラだけによって実行され得る。メタデータサブシステムまたは処理システムによって実行される処理に関連して、実行中のコードのメタデータタグは、PUMP CSRに置かれ、メタデータシステムに入力されメタデータシステムにより操作される「データ」になる(たとえば、ポインタはメタデータメモリ空間の中へのものである)。メタデータサブシステムは、ルールに従った処理のために、PUMP入力CSRを介してPUMP入力を読み取る。命令がルールを介して進行することが許容される場合、PUMPはタグ結果(たとえば、Pc new tagおよびR tagなどのための)を定義されたPUMP出力CSRに書き込む。ルールキャッシュへのルールの挿入は、特定のCSR(たとえば、901qにおけるsrtag CSRなど)に書き込んだことに応答してトリガされ得る。このようにして、すべてのタグの更新は、PUMPの中のルールを通じて行われ、メタデータサブシステムによって制御される。メタデータサブシステムは、ルールキャッシュミスが発生すると呼び出されるキャッシュミスハンドラを介してのみ、PUMPキャッシュにルールを挿入することができる。加えて、RISC-Vアーキテクチャを使用して本明細書で説明されるような少なくとも1つの実施形態では、メタデータ処理と普通の命令処理との間の前述の分離は、「RISC-Vユーザレベル ISA」および「RISC-V特権ISA」における命

令以外の新しい命令を追加することなく維持され得る。本明細書の他の箇所での議論と一貫して、本明細書の技法に従ったある実施形態は、データとメタデータとの間の厳密な分離および隔離を維持することができ、これにより、タグに基づくメタデータ処理と普通の命令処理との間には分離がある。少なくとも1つの実施形態では、そのような分離は、別個のプロセッサおよび別個のメモリを伴う別個の物理的なメタデータ処理サブシステムを有することにより維持され得る。したがって、第1のプロセッサおよび第1のメモリは、プログラムを実行する命令を処理するときに使用されることがあり、第2のプロセッサおよび第2のメモリは、ルールキャッシュミスハンドラのコードを実行するときなどにメタデータ処理を実行することとともに使用するメタデータ処理サブシステムに含まれることがある。

【０１５４】

図27を参照すると、本明細書の技法によるある実施形態に含まれ得る構成要素の例1000が示されている。例1000は、実行中のプログラムのための普通の処理に関連して使用される第1のサブシステムまたはプロセッサ1002、およびメタデータ処理サブシステムまたはプロセッサ1004を含む。第1のサブシステム1002は、普通のプログラム実行に関連して使用されるプログラム実行サブシステムとして特徴付けられ得る。サブシステム1002は、プログラムコードを実行してデータを使用することに関連して使用される構成要素を含むプロセッサであり、ここで、そのようなコードおよびデータは、メタデータ処理サブシステム1004とともに使用するために本明細書の他の箇所で説明されるようなタグを含む。サブシステム1002は、メモリ1008aと、命令またはI-store1008bと、ALU(算術論理装置)1008dと、プログラムカウンタ(PC)1008eとを含む。PUMP1003は、サブシステム1002におけるコードの実行に関連して使用され得るが、メタデータ処理サブシステム1004の一部であると見なされ得ることに留意されたい。サブシステム1002の中のすべてのコードおよびデータは、データ1002bと関連付けられるタグ1002aにより全般に示されるようにタグ付けされることがあり、ここで、1002aおよび1002bはメモリ1008aに記憶され得る。同様に、要素1001aはPC1008eの命令のタグを示し、1001bは命令1008bのタグを示し、1001cはメモリ位置1008aのタグを示し、1001dはレジスタ1008cのタグを示す。

【０１５５】

メタデータ処理サブシステム1004は、PUMP1003に入力として与えられる現在の命令および関連するデータのタグを使用したメタデータルール処理に関連して使用される構成要素を含む、プロセッサ(メタデータプロセッサとも呼ばれる)である。PUMP1003は、本明細書の他の箇所で説明されるようなものであることがあり、ルールキャッシュを含む。たとえば、少なくとも1つの実施形態では、PUMP1003は図22に示される構成要素を含み得る。本明細書の技法に従った少なくとも1つの実施形態に含まれ得る、PUMP1003の構成要素、PUMP入力および出力のために使用される関連するPUMP CSR、ならびに関連する論理のより詳細な説明および例が、以下および本明細書の他の箇所でより詳細に説明される。サブシステム1004は、メタデータ処理のために使用される別個のプロセッサであり、サブシステム1002の構成要素と同様の構成要素を含む。サブシステム1004は、メモリ1006aと、I-store1006bと、レジスタファイル1006bと、ALU1006dとを含む。メモリ1006aは、メタデータルール処理に関連して使用されるメタデータ構造を含み得る。たとえば、メモリ1006aは、ポインタであるタグによって指される構造またはデータを含み得る。ポインタタグおよびポインタタグにより指される構造/データの例は、CFIポリシーなどに関連して、本明細書の他の箇所で説明される。I-store1006bおよびメモリ1006aは、メタデータ処理を実行するミスハンドラなどの命令またはコードを含み得る。メタデータプロセッサ1004は、プログラム実行に関連して使用されるデータメモリ1008aなどの、1002の他の構成要素にアクセスする必要がなく、それは、メタデータプロセッサ1004が(たとえば、タグおよびルールに基づいて)メタデータ処理しか実行しないからである。サブシステム1004は、別個のメモリ1006aなどの、固有の構成要素を含み、メタデータ処理コードおよびデータをサブシステム1002に記憶する必要がない。むしろ、PUMP1

003によって使用され得る現在の命令のタグなどの任意の情報が、メタデータ処理サブシステム1004に入力(たとえば、PUMP入力1007)として与えられる。

【０１５６】

例1000は、本明細書の他の箇所で説明されるように、普通のプログラム実行のために使用されるのと同じサブシステムでメタデータ処理を実行するのではなく、別個のメタデータ処理サブシステム1004を有する代替的な実施形態を示す。たとえば、別個のメタデータプロセッサまたはサブシステム1004を有する代わりに、ある実施形態は、PUMP1003およびサブシステム1002のみを含み得る。単一のプロセッサを伴うそのような実施形態では、CSRは、メタデータ処理と、ユーザプログラムを実行する普通の処理モードとの間で情報を伝え、それにより分離および隔離を行うために、本明細書で説明されるように使用され得る。別個のメタデータプロセッサではなく単一のプロセッサを伴うそのような実施形態では、ミスハンドラのコードは、それが保護されるような方式で単一のメモリに記憶され得る。たとえば、別個のメタデータプロセッサまたはサブシステムなしで、ミスハンドラのコードは、アクセスを制限するための本明細書の他の箇所で説明されるようなタグを使用して保護されることがあり、ユーザコードによりアドレス指定可能ではないメモリの一部分などにマッピングされることがある。

【０１５７】

ここで説明されるのは、PUMP I/O(入力/出力)に関するさらなる詳細である。以下で説明されるPUMP I/Oは、普通のコード実行と同じプロセッサまたはサブシステムを使用し得るPUMP、ならびに、例1000などの別個のプロセッサまたはサブシステムを使用し得るPUMPの実施形態に適用されることに留意されたい。さらに、以下で説明されるPUMP I/Oは、RISC-Vアーキテクチャに基づく実施形態とともに使用されることがあり、他のプロセッサアーキテクチャとともに使用するために一般化されることがある。

【０１５８】

図28を参照すると、本明細書の技法によるある実施形態においてPUMP I/Oを要約する例1010が示されている。図1および図24などに関連して本明細書の他の箇所で説明されるように、PUMPは段階5および6で動作する。PUMP入力は、現在の命令のためのPUMPのルールキャッシュの中に一致するルールがもしあればそれを見つけるために、普通のPUMP検証に関連して使用される(たとえば、現在の命令がポリシールールを使用して許容されるかどうかを検証する)。普通のPUMP検証は、6段階のパイプラインについて本明細書の他の箇所で説明されるような段階5の一部などの、1つ1つの命令に対して行われ得る。加えて、PUMP入力は、6段階のパイプラインの段階6において行われ得るものなどの、ルールキャッシュへのルールの挿入を制御することに関連して使用され得る。普通のPUMP検証と関連付けられるPUMP I/Oは、上(入力1012)から下(出力1014)までの垂直な方向の入力および出力によって例1010において示される。PUMPルールキャッシュへのルールの挿入を制御することと関連付けられるPUMP I/Oは、左(入力1016)から右(出力1018)までの水平な方向の入力および出力によって例1010において示される。加えて、要素1012は、他の箇所でより詳細に説明されるように、ルール挿入に関連しても使用される追加の入力を示す。

【０１５９】

まず、普通のPUMP検証処理と関連付けられるPUMP I/Oを考える。PUMP入力1012は、PC tag、CI tag、命令オペランドタグ(たとえば、OP1 tag、OP2 tag、またはCSR tag(RISC-VにおけるCSRベースの命令のための))、OP3 tag、M Tag(メモリ命令に対するメモリ位置タグのための。Mtagはメモリ命令に対してはMR tagとも本明細書で呼ばれ得ることに留意されたい)、オペコード情報(たとえば、Opgrp入力により示されるオペグループ、拡張されたオペコードに対するRISC-Vのためのfunct12(funct7)入力、どの命令が例200および220などにおける複数の命令を含む命令ワードの中の現在の命令であるかのインジケータを提供するsubinstr入力)、およびケア入力ビットなどの、タグを含み得る。Opgrpは現在の命令のためのオペグループであることがあり、ここでOpgrpは本明細書の他の箇所で説明されるような前の段階(たとえば、段階3または段階4)の出力であり得

る。Funct12(funct7)PUMP入力は、命令ワードの追加のビットを使用するRISC-Vオペコードのための追加のオペコードビットがもしあれば、その追加のオペコードビットであり得る(たとえば、例400)。PUMP出力1014は、Rtag(たとえば、命令結果レジスタおよび宛先メモリ位置のためのタグ)、PC new tag(次の命令のために使用されるPCに付けられる伝播されたタグを示す)、および、段階6のミスハンドラへのトラップをもたらすPUMPルールキャッシュミスがあったかどうかを示すインジケータ1014aを含み得る。

【０１６０】

ケアビット1012aは、どのPUMP入力1012およびどのPUMP出力1014が特定の命令に対してケアされる/ケアされない(たとえば、無視される)かを示し得る。PUMP入力に関するケアビットは、funct12のためのケアビットおよびfunct7のための第2のケアビットを含み得る。本明細書の他の箇所で説明されるように、前述のケアビットの両方が、現在の命令の特定のオペコードがRISC-V命令のための拡張された12オペコードビット部分(たとえば、例400の404a)のための任意のビットを含むかどうかを示す。funct12およびfunct7の両方のケアビットが「don't care」である場合、拡張された12オペコードビット部分の12ビットすべてがマスクされる。funct7が「care」を示す場合、拡張された12オペコードビット部分のすべての下位5ビットがマスクされる。funct12が「care」を示す場合、拡張された12オペコードビット部分のマスキングはない。

【０１６１】

ここで、PUMPルールキャッシュへのルールの挿入を制御することと関連付けられるPUMP I/Oを考える。PUMP入力1016は、PUMPキャッシュルールの挿入に関連して入力1012と組み合わせて使用され得る。PUMP入力1016は、Op1data(メタデータプロセッサまたはサブシステムからの出力)、命令(段階6からの)、タグモード(メタデータプロセッサまたはサブシステムからの出力)、および特権(RISC-V特権を示すpriv)を含み得る。1016のタグモードおよびpriv入力は、メタデータプロセッサにおいて実行中のミスハンドラまたは他のコードなどのコードが、以下および本明細書の他の箇所で説明され様々な入力(たとえば、入力1012など)をメタデータプロセッサに提供するCSRにアクセスするために十分な特権を有するかどうかを判定するために、メタデータプロセッサまたはサブシステムによって使用される。Rdata1018は、段階6において使用するためのメタデータプロセッサまたはサブシステムへの入力(たとえば、キャッシュミスハンドラ処理入力)である。Op1data、R data、および例1010の他の項目は、以下の段落および図面おいて詳細に説明されることに留意されたい。

【０１６２】

したがって一般に、例1010において、要素1012はユーザコードを実行するプロセッサ(たとえば、1002などの非メタデータプロセッサまたはサブシステム)からPUMPおよびメタデータプロセッサへの入力を示し、要素1014はメタデータプロセッサによって生成される出力を示し、要素1016はPUMPへのメタデータプロセッサ入力によって生成される出力を示し、要素1018はメタデータプロセッサへの入力を示す。

【０１６３】

図29を参照すると、本明細書の技法に従ったある実施形態における、オペグループ/ケアテーブル(たとえば、例420の要素422)に関連したI/Oを要約する例1020が示されている。本明細書の他の箇所で説明されるように、オペグループ/ケアテーブルは、現在の命令のオペコードのためのオペグループおよびケアビットをルックアップして出力するために、各命令に対して使用され得る。I/Oのこの第1のフローは、上(入力1022)から下(出力1024)への垂直な方向の入力および出力によって1020において示される。本明細書の他の箇所で説明されるように、入力1022は、オペコード/ケアテーブルへのインデックスとして使用される、オペコードまたはその一部分(たとえば、例420のオペコード部分の例に関連して説明されるものなど)であり得る。入力1022は段階3からのものであり得る。出力1024は、特定のオペコードのためのオペグループ(opgrp)およびケアビットであり得る。出力1024は段階5への入力である(たとえば、1012に含まれるようなPUMP入力opgrpおよびcareの2つ)。

【０１６４】
I/Oの第2のフローは、左(入力1026)から右(出力1028)への水平な方向の入力および出力によって1020において示されている。1020におけるI/Oの第2のフローは、メタデータプロセッサおよび段階6への入力であるPUMP出力Rdata1028の選択を制御することに関連して実行される処理を説明するものである。入力1026は1016に関連して上で説明されたようなものである。出力1028は1018に関連して上で説明されたようなものである。
【０１６５】
図30を参照すると、本明細書の技法に従ったある実施形態においてPUMPにより実行される処理を抽象的に表す例1030が示されている。例1030は、例1010の水平なPUMP I/Oフロー(たとえば、要素1012、1016、および1018)に関連して上で説明されたルール挿入のためのPUMP制御に対応するPUMP制御1031を含む。例1030は、例1010の垂直なPUMP I/Oフロー(たとえば、要素1012および1014)に関連して上で説明されたような各命令のために実行される普通のPUMP検証パスI/Oフローに対応する、マスキング1032、ハッシュ1034、ルールキャッシュルックアップ1036、および出力タグ選択1038を含む。マスキング1032は、1012のケアビットを適用して1012の使用されていないPUMP入力をマスクすることを示す。ハッシュ1034は、1036により示されるルールキャッシュルックアップの間に使用されるハッシュの計算結果を示す。一実施形態において1032、1034、および1036により示される論理を実装するために使用され得る構成要素が、図22に関連して示され説明される。出力タグ選択1038は、ケアベクトルビット(入力1012に含まれるケア)およびhtagmode CSR(現在のタグモードを示す)に基づいて、1014に含まれるようなPUMP出力RtagおよびPC new tagの選択を示す。
【０１６６】
図31を参照すると、本明細書の技法に従ったある実施形態におけるPUMPの出力タグ選択1038の論理を実装するために使用され得る構成要素を示す例1040が示されている。例1040はマルチプレクサ(MUX)1043a～bを含む。一般に、MUX1043aは、PUMPにより出力されるようなPC new tag 1043(たとえば、1014のPC new tag)の最終的なタグ値を選択するために使用されることがあり、MUX1043bは、PUMPにより出力されるようなR tag 1047(たとえば、1014のR tag)の最終的なタグ値を選択するために使用されることがある。要素1042は、MUX1043aのためのセレクタとして使用される入力を示す。入力1042は、PC new tag 1043として1041aまたは1041bのいずれかを選択するために使用される。入力1042は、engaged(PUMPが関与しているかどうかを示すブーリアン)と論理積(&&)をとられたPCnew tagケアビット(たとえば、1012のケアビットからの)を含み得る。要素1043は、MUX1043bのためのセレクタとして使用される入力を示す。入力1043は、R tag 1047として1045a～1045bにより示される入力のうちの1つを選択するために使用される。入力1043は、engagedと論理積(&&)をとられたRtagケアビット(たとえば、1012のケアビットからの)を含み得る。したがって、一般的に、PUMP入力1012に含まれるケアビットは、どのPUMP入力がdon't careであるか(マスクされるか)、およびどのPUMP出力(RtagおよびPCnew tag)がdon't careであるか(マスクされるか)を特定する。また、出力1043および1047は、PUMPが不関与であるときには「don't care」値として扱われ、それは、PUMP動作の閾値として現在のtagmodeが指定するものよりも高い特権レベルで、プロセッサが実行しているからである。
【０１６７】
要素1049は、現在のRISC-V特権および現在のtagmodeの関数としてブーリアンengagedがどのように判定されるかを示す。要素1049は、当技術分野で知られている標準的な表記を使用した論理表現を含み、「A==B」はAとBの間の等しさについての論理テストを示し、「A&&B」はAとBの論理積演算を示し、「A｜｜B」はAとBの排他的論理和演算を示す。
【０１６８】
要素1041aおよび1045aは、ルールキャッシュルックアップ1036からの出力である1043aへの入力を示す。PC tag 1041bはPUMP入力1012に含まれるPC tagである。他の入力1041bは一般に、PUMPにより出力される最終的なR tag 1047として選択される可能

性があり得る複数の他の入力を示す。たとえば、一実施形態では、他の入力1041bは、M tag、PC tag、CI tag、OP1 tag、OP2 tag、OP3 tag、および、場合によっては命令に応じて他のものを含み得る。特定のR tag出力1047は、具体的なRISC-V命令/オペコードとともに変化し得る。

【０１６９】

以下は、一実施形態におけるPUMP出力値として生成されるR tag 1047およびPC new tag 1043の特定の値を要約し得る。以下は、異なるRISC-V命令に対する特定のR tag出力値を示すことに留意されたい。したがって、最終的なPUMP R tag値として出力される特定のR tag値は、後続のメタデータ処理に関連してそのようなPUMP出力を利用する命令とともに変化し得る。

1．PCtagは、出力ケアビットがPC new tagに対してオフであるときに変化しない
2．Rtagは、CSRRW動作のためのOp1tagである
3．Rtagは、CSRR?I、CSRRS、CSRRC動作のためのOp2tag(CSRtag)である
4．Rtagは、JALおよびJALR命令のためのPCtagである
5．Rtagは、AUIPC命令のためのPCtagである
6．Rtagは、LUI命令のためのCItagである
7．Rtagは、出力ケアビットがオフであるとき(Rtagに対するケアを示す)、非メモリ、非CSR、非JAL(R)/AUIPC/LUI動作のためのOp1tagである
8．Rtagは、出力ケアビットがオフであるとき、メモリ書込み動作のためのOp2tagである
9．Rtagは、出力ケアビットがオフであるとき、メモリロード動作のためのMtagである

【０１７０】

図32を参照すると、本明細書の技法に従ったある実施形態におけるPUMP I/Oを制御するために使用され得る構成要素の例1050が示されている。一般に、例1030に戻って参照すると、1050の構成要素は、1032に論理的に加わる(たとえば、図22の構成要素とインターフェースする)別の層を備え得る。要素M1～M14は、それらへの様々な入力の選択のために使用されるマルチプレクサを示す。要素1052は一般に、現在の命令のための1012からの入力であるopcode、PC tag、CI tag、Op1 tag、Op2 tag、Op3 tag、およびM tagを示す。要素1056は一般に、マルチプレクサM1～M7の選択された出力を記憶するために使用されるレジスタの行を指す。RISC-Vアーキテクチャに基づく一実施形態では、行1056の中のボックスの各々はレジスタであることがあり、具体的には、本明細書の他の箇所で説明されるような特定の値を格納するCSRであることがある(たとえば、例900は一実施形態において使用され得るCSRを示す)。

【０１７１】

例1050の要素1052は、1012の入力のすべては含まないことに留意されたい。たとえば、1012のfunct12(funct7)およびsubinstr入力は、簡潔にするために例1050には示されていない。しかしながら、1012からの入力funct12(funct7)およびsubinstrも1052に含まれ得ることを当業者は理解する。より一般的には、入力1052は、ある実施形態において使用され得るメタデータルール処理への特定の入力のために適合され得る。

【０１７２】

PUMPが普通のPUMP検証のための命令のための処理を実行しているとき(たとえば、現在の命令が許容されるかどうかをポリシールールを使用して検証する)、入力1052は単に出力1054として通過し得る。この場合の出力1054は、メタデータ処理のために、図22の構成要素への入力などの入力としてPUMPへと流れる(またはより一般的には、メタデータプロセッサまたはサブシステムへと流れる)。普通のPUMP検証では、PUMPは次いで出力1014(たとえば、現在の命令のための一致ルールがルールキャッシュにおいて見出される場合、または別様にキャッシュミス1014aを生成する場合、RtagおよびPC new tag)を産生し得る。

【０１７３】

ルールキャッシュミスが発生すると、第1のステップとして、現在の命令のための1052からの現在の値が1056のレジスタG1～G7にロードされる。したがって、G1～G7は、ルー

ルキャッシュミスを引き起こした現在の命令のためのオペコードおよびタグ値のスナップショットを含み、そのような値はここで、そのような処理に必要とされるようなG1～G7の1つまたは複数の所望の値を読み出すキャッシュミスハンドラにより、後続の処理に関連して使用され得る。

【０１７４】

したがって、第2のステップにおいて、キャッシュミスハンドラが実行し、G1～G7からの値を入力として読み取り、現在の命令のための新しいルールを生成する。マルチプレクサM16は、G1～G7からの様々な可能な入力の選択を制御するために使用されることがあり、ここで、M10からの選択された出力は、キャッシュミスハンドラによる処理のためにR data 1053としてとして示される(たとえば、キャッシュミスハンドラは、プログラムコードを実行するときと同じプロセッサ上で実行することがあり、またはそうでなければ、例1000などのように別個のメタデータプロセッサ上で実行することがある)。現在の命令のための入力G1～G7がルールキャッシュミスを引き起こすと仮定すると、キャッシュミスハンドラは、キャッシュに挿入されるべき新しいルールを判定するために処理を実行する。キャッシュミスハンドラは、判定されたばかりの新しいルールのための出力R tagおよびPC new tagを生成し、RtagをRtag CSR G8に書き込み、PC new tagをPC new CSR G9に書き込む。例1050において、Op1data1051は、新しいルールのための出力RtagおよびPC new tagなどのメタデータプロセッサによって生成される出力を示し、ここで、そのような出力が次いで、説明されたようにCSR G8およびG9に記憶される。

【０１７５】

この時点では、CSR G1～G9の値は、キャッシュミスハンドラによって生成されたばかりの新しいルールのためのタグ値であり、第3のステップにおいて新しいルールとしてルールキャッシュに挿入される/書き込まれることがある。RISC-Vアーキテクチャについての本明細書の技法を使用する少なくとも1つの実施形態では、G8により示されるR tag CSRへの書込みは、ルールキャッシュへの新しいルール(たとえば、CSR G1～G9の内容)の書込みをトリガする。挿入に関連して、CSR G1～G7は出力1052として与えられ、CSR G8およびG9はルールキャッシュへの記憶のためにPUMPへの出力1055として与えられる。より具体的には、一実施形態では、出力1052および1055は、ルール挿入のために図22の構成要素に与えられ得る。

【０１７６】

簡単なケースでは、ある実施形態は、すぐ上で説明されたように(たとえば、出力1052および1055を介して)新しいルールのためのCSR G1～G9の内容をPUMPルールキャッシュに書き込むことによって、現在のルールミスを満たすように1つの新しいルールを挿入することができる。そのような実施形態では、マルチプレクサM1～M7は必要ではなく、それは、キャッシュミスハンドラを実行するメタデータルールプロセッサにより出力されるOp1data1051のみが、新しいルールのためのR tagおよびPC new tagを生成するからである。しかしながら、ある実施形態はまた、複数のルールをプリフェッチすること、またはルールキャッシュへと挿入することを許容し得る。たとえば、ルールキャッシュミスが発生すると、キャッシュミスハンドラは、現在の命令のための単一の新しいルールだけではなく、ルールキャッシュに書き込まれる/挿入されるべき複数のルールを判定し得る。この場合、Op1data1051は、opcode、PCtag、CItag、Op1tag、Op2tag、Op3tag、およびMtagの追加の新しい値(CSR G1～G7に書き込まれる)、ならびに、RtagおよびPC new tagの新しい値(CSR G8およびG9に書き込まれるような)を含み得る。そのような場合、マルチプレクサM1～M7は、それぞれCSR G1～G7のための入力として、Op1data1051からの前述の新しい値を選択するために使用され得る。

【０１７７】

一般に、Op1data1051はメタデータプロセッサからPUMPへの出力を示し、R data 1053はPUMPからメタデータプロセッサへの出力を示す。また、要素1052はユーザコードを実行するプロセッサからPUMPへの入力を示し(たとえば、普通の命令処理の一部として)、ここで、要素1054の値は普通のPUMP検証を実行するときの1052の値と等しい(たと

えば、現在の命令が許容されるかどうかをポリシールールを使用して検証する)。

【０１７８】

図33を参照すると、分岐予測を伴うRISC-Vアーキテクチャについて、本明細書の技法に従った一実施形態における6段階のプロセッサパイプラインと組み合わせてPUMP処理段階を示す例1060が示されている。例1060は、実行されるべき次の命令をフェッチすること(たとえば、Iキャッシュ1063aにフェッチされた命令を記憶すること)および分岐予測を含む段階1、復号命令段階を示す段階2、レジスタから値を取得すること(たとえば、レジスタ読取り)および現在の命令のための分岐解決を含む段階3、命令実行を含む段階4(たとえば、高速ALU演算を実行し、浮動小数点(FP)、整数の乗算および除算などの、多段階の演算を始める)、多段階の演算に対する応答を受信してメモリオペランドを要求することを含む段階5、ならびに、命令をコミットすること(たとえば、宛先へ、および1069により示されるようなデータキャッシュ1063bに結果を記憶すること)と、例外、トラップ、および中断を扱うこととを含む段階6を伴う、6段階のパイプラインを示す。PUMP処理段階も例1060に示されている。要素1062は、opgrp/ケアテーブルのルックアップが段階3において実行されることがあり、出力1062aがPUMPハッシュ1064へ段階4において入力として与えられることを示す。PUMPハッシュ1064への他の入力は、Mtag1061(たとえば、現在の命令のためのオペランドであるメモリ位置のタグ)および他のタグ値1062bを含み、これにより、入力1061および1062a～bは、PUMPルールキャッシュ1066におけるキャッシュアドレスまたは位置を示す出力1064aを判定するために使用される。命令オペランド、PC、現在の命令などの他のタグ値1062bの例は、本明細書の他の箇所で説明され、現在の命令のためのルールキャッシュ1066における位置を判定することに関連して使用されることがある(たとえば、図22)。要素1068は、段階5からのPUMP処理の出力1066aに基づくキャッシュルールミス検出を示す。出力1066aは、現在の命令のためのルールキャッシュミスがあったかどうかについてのインジケータを含み得る。1066aが潜在的なヒットを報告する場合、1068は、そのヒットが真のヒットであるか偽のヒットであるかを判定し、偽のヒットをミスに変える。要素1066bは、ルールキャッシュミスがなく、現在の命令と一致するルールがキャッシュの中にある場合、段階6へのPUMP出力を示す。出力1066bは、PC new tagおよびR tagを含み得る。例1060のPUMP段階は変化し得ることに留意されたい。たとえば、オペグループ/ケアルックアップ1062は、段階3ではなく段階4において実行されることがあり、PUMPルールキャッシュ位置の判定およびルックアップは、(たとえば、具体的なPUMPルールキャッシュの実装形態に応じて)両方とも段階5において行われる。

【０１７９】

非メモリ動作に関連して、MtagはPUMP段階への入力として必要ではなく、PUMPはMtagを伴わず処理の実行を続けることができる。メモリ動作命令の場合、PUMPはMtagがメモリから取り出されるまでストールする。代わりに、ある実施形態は、本明細書の他の箇所で説明されるようにMtag予測を実行し得る。本明細書の他の箇所の議論と一貫して、PC new tagは、図1に関連して示され説明されるように、段階1に戻って提供される必要がある。命令がコミットする限り、PC new tagは次の命令に適切なPC tagである。現在の命令がコミットしない(たとえば、ルールキャッシュのヒットがない)場合、PC new tag(段階1に戻されるような)はルールキャッシュミスハンドラによって判定される。トラップハンドラが開始するとき、またはコンテキスト切替えが実行されるとき(たとえば、PC復元)、タグは保存されたPCから来る。

【０１８０】

本明細書で説明されるように、ある実施形態は単一のタグを各ワードと関連付け得る。少なくとも1つの実施形態では、各タグと関連付けられるワードサイズは64ビットであり得る。タグ付けされたワードの内容は、たとえば、命令またはデータを含み得る。そのような実施形態では、単一の命令のサイズ。しかしながら、ある実施形態は、64ビット以外の異なるサイズである命令もサポートし得る。たとえば、ある実施形態は、本明細書の他の箇所でより詳細に説明されるような、確立された縮小命令セットコンピューティング(RIS

C)の原理に基づくオープンソース命令セットアーキテクチャ(ISA)である、RISC-Vアーキテクチャに基づき得る。RISC-Vアーキテクチャを使用する実施形態は、たとえば、32ビット命令ならびに64ビット命令などの複数の異なるサイズの命令を含み得る。そのような場合、本明細書の技法に従ったある実施形態は、単一のタグを単一の64ビットワードと関連付けることができ、ここで、それゆえ、その単一のワードは1つの64ビット命令または2つの32ビット命令を含み得る。

【０１８１】

図34を参照すると、本明細書の技法に従ったある実施形態における命令と関連付けられ得るタグの例200が示されている。要素201は、単一のタグ202aが単一の命令204aと関連付けられる、上で述べられたケースを示す。少なくとも1つの実施形態では、202aおよび204aの各々のサイズは64ビットワードであり得る。要素203は、単一のタグ202bが2つの命令204bおよび204cと関連付けられる、上でやはり述べられた代替形態を示す。少なくとも1つの実施形態では、202bのサイズは64ビットワードであることがあり、命令204bおよび204cは各々、タグ202bと関連付けられる同じ64ビット命令ワード205に含まれる32ビット命令であり得る。より一般的には、実施形態において使用される命令サイズに応じて、単一のタグ付けされた命令ワードの中に2つより多くの命令があり得ることに留意されたい。要素203により示されるように、タグ付けの粒度が命令の粒度と一致しない場合、複数の命令が単一のタグと関連付けられる。いくつかの事例では、同じタグ202bが命令204b、204cの各々のために使用され得る。しかしながら、いくつかの事例では、同じタグ202bが、命令204b、204cの各々のために使用されないことがある。以下の段落では、単一のタグ付けされたワードと関連付けられる単一の命令ワードに含まれる204bおよび204cなどの複数の命令の各々は、サブ命令とも呼ばれ得る。

【０１８２】

したがって、ここで説明されるのは、同じ命令ワードの中の複数のサブ命令に関連してある実施形態において使用され得る技法であり、これにより、異なるタグが複数のサブ命令の各々に関連して使用され得る。

【０１８３】

図35を参照すると、本明細書の技法に従ったある実施形態において使用され得る命令およびタグを示す例が示されている。例220は、2つの32ビットサブ命令204bおよび204cを含む単一の64ビット命令ワード205を含む。タグ202bは、例200において上で説明されたような命令ワード205のタグであり得る。本明細書の技法に従った少なくとも1つの実施形態では、命令ワード205のタグ202bは、タグのペアを含む別のメモリ位置222へのポインタ221であることがあり、ここで、このペアは命令ワード205のサブ命令204b～cの各々のタグを含む。この例220では、タグのペア222は、サブ命令1 204bのための第1のタグであるtag1を示す222aを含み、サブ命令2 204cのための第2のタグであるtag2を示す222bも含む。少なくとも1つの実施形態では、ペア222の各タグ222a～222bは、非ポインタタグ(たとえば、スカラー)であることがあり、本明細書で説明されるような処理のためにPUMPによって使用される情報を含むさらに別のメモリ位置へのポインタタグであることがあり、またはそうでなければ、1つまたは複数のポインタフィールドおよび/もしくは1つまたは複数の非ポインタフィールドを含むより複雑な構造であることがある。たとえば、tag 1 222aはサブ命令1 204aのためのポインタタグであることがあり、tag2 222bはサブ命令2 204bのためのポインタタグであることがある。220に示されるように、要素223aは、サブ命令1 204bを処理するためにPUMPにより使用される情報を含む別のメモリ位置224aを指す、またはそれを特定するtag1 222aを示し、要素223bは、サブ命令2 204cを処理するためにPUMPにより使用される情報を含む別のメモリ位置224bを指す、またはそれを特定するtag2 222bを示す。実施形態およびサブ命令に応じて、224aおよび224bの各々は非ポインタであることがあり、メモリ位置へのさらに別のポインタであることがあり、または、1つまたは複数のポインタと1つまたは複数の非ポインタの何らかの組合せを含む複雑な構造であることがあることに留意されたい。

【０１８４】

同じ命令ワード205内に複数のサブ命令を有するある実施形態では、命令ワード205に含まれるサブ命令のうちのいずれがある時点で実行されているかを示す追加の入力が、PUMPに与えられ得る。たとえば、命令ワード205の中に2つのサブ命令204b～cがある場合、PUMPへの追加の入力は0または1であることがあり、これらはそれぞれ、サブ命令1 204bまたはサブ命令2 204cがある特定の時点で実行されているかどうかを示す。RISC-Vアーキテクチャに基づく本明細書の他の箇所の議論と一貫する少なくとも1つの実施形態では、PUMPへの追加の入力(どのサブ命令が実行されているかを示す)を記録または記憶する、CSR(本明細書の他の箇所で説明されるssubinstr CSRなど)が定義され得る。少なくとも1つの実施形態では、PUMPは普通は、CSRを使用せずにデータパスから(たとえば、コード実行領域から)前述の追加の入力を受信し得る。しかしながら、ルールミスに際して、前述の追加の入力はCSRに記録され得るので、ルールミスハンドラが実行しているメタデータ処理領域は、前述の追加の入力を取得することができる(たとえば、前述の追加の入力のためのCSR値はルール挿入に際してPUMPに与えられる)。

【０１８５】

さらに例示すると、ある実施形態は、プログラムの中の2つの位置の間での制御の移転を行うサブ命令を含み得る。そのようなサブ命令の例は、コードの中のソース位置からコードの中のターゲット(たとえば、シンクまたは宛先)位置へと制御をジャンプする、分岐する、返す、またはより一般的には移転することを行う、サブ命令であり得る。本明細書の他の箇所で説明されるCFIすなわち制御フロー整合性に関連して、CFIポリシーのPUMP実装ルールに、プログラムによりサポートされる位置だけに位置間の移転を限定または制御させることが望ましいことがある。たとえば、タグT1を有するコード中のソース位置からタグT2を有するコード中のターゲット位置への制御の移転が行われる場合を考える。CFIポリシーを実施する際にPUMPにより使用される情報は、制御をT2に移転することが許容される有効なソース位置のリストであり得る。CFIポリシーのある実施形態では、ソース位置からターゲット位置に制御を移転するときに2つの命令またはオペコードの2回の確認を行うために、2つのルールが使用され得る。図36の例230に示されるような、移転または呼出しの疑似コード表現を考える。例230では、fooルーチン231の中のソース位置からルーチンbar233の中のターゲット位置へと制御を移転する(231a)、呼出しが行われ得る。具体的には、タグT1を有するソース位置X1 232からタグT2を有するターゲット位置X2 234へと制御が移転され得る(231a)。ターゲット位置X2は、ルーチンbarのコード233aの本体の中の第1の命令であり得る。CFIポリシーのルールは、232から234への移転が許容される、または有効であるかどうかを確認するために使用され得る。少なくとも1つの実施形態では、232から234への制御の移転が有効であることを確実にするための確認を各々が実行する、CFIポリシーの2つのルールが使用され得る。ソース位置X1における命令は、制御がそこからターゲットに移転される、分岐点またはソース点である。ソースにおいて(たとえば、ソース位置X1 232において命令を実行する前に)、PCのタグをマークまたは設定してソース位置を示すために、第1のルールが使用され得る。たとえば、第1のルールは、PCのタグをアドレスX1としてマークまたは設定して、ソース位置を示し得る。その後、ターゲット位置X2 234において命令を実行する前に、ソース位置X1が、制御がそこからターゲット位置X2へ移転されることが許容される有効なソース位置であるかどうかを確認するために、第2のルールが使用され得る。

【０１８６】

少なくとも1つの実施形態では、第2のルールの確認は、ソース位置232(たとえば、ソース位置アドレスX1を示す)を特定するPCのマークされたタグ(第1のルールにより設定されるような)が、制御がそこからターゲット位置234に移転され得る有効なソース位置を特定するかどうかを判定することによって、実行され得る。そのような実施形態では、第2のルールは、ターゲット位置234に制御を移転することが許容されるすべての有効なソース位置を示す定義されたリストを供給され得る。少なくとも1つの実施形態では、定義されたリストは、有効なソース位置を、たとえば上で述べられたX1などのそれらのアドレスによって、特定し得る。

【０１８７】
図37を参照すると、本明細書の技法に従ったある実施形態において、ソース位置およびターゲット位置のサブ命令に関連して使用され得るタグを示す例240が示されている。例240は、上で説明されたような単一の命令ワードの2つのサブ命令204b～cのために指定される単一のタグ202bを示す要素203を含む。命令ワードのタグ202bは、2つのサブ命令204b～cのための2つのタグ242a～bをそれぞれ示すタグのペア242を指し得る。2つのタグ242a～bの各々は一般に、2つのタグ242a～bの各々と関連付けられる具体的なサブ命令に応じて、ソース位置またはターゲット位置に関連してCFI検証のためにPUMPルールにより使用される情報へのポインタであり得る。
【０１８８】
例240は、2つのサブ命令204b～cがターゲット位置である一実施形態における構造を示す。サブ命令タグ242aは、ソースidフィールド245aおよび許容ソース集合フィールド245bを含む、構造245の位置を指す(243a)。ソースidフィールド245aは、サブ命令204bがターゲット位置である場合などの、サブ命令204bがソース位置ではない場合にはヌルであり得る。ソース集合フィールド245bは、サブ命令204bを含む特定のターゲット位置に制御を移転することが許容される1つまたは複数の有効なソース位置を特定するリスト構造247を含む位置へのポインタであり得る。少なくとも1つの実施形態では、リスト構造247は、有効なソース位置のサイズまたは数を示す第1の要素を含み得る。したがって、「n」(nは0より大きい整数である)というサイズ247aは、リスト247の中の要素247b～nにより示されるソース位置の数を示す。要素247b～nの各々は、サブ命令204bを含むターゲット位置に制御を移転できる異なる有効なソース位置を特定し得る。少なくとも1つの実施形態では、許容されるソース247b～nの各々は、たとえば有効なソース位置のうちの1つのアドレスである、スカラーまたは非ポインタであり得る。
【０１８９】
例240において、サブ命令2 204cとともに使用される要素243b、246、および248はそれぞれ、サブ命令1 204bとともに使用される要素243a、245、および247と同様である。一般に、240の構造を使用するそのような実施形態では、存在しないあらゆる項目がヌルまたは0の値を割り当てられ得る。命令ワード205が、ソース位置でも宛先位置でもないサブ命令のペア204b～cを含む場合、タグ202bはヌルであり得る(たとえば、または別様に、構造242を指さない非ポインタまたは他のポインタを特定し得る)。サブ命令204b～cのうちの1つが移転のソース位置でもターゲット位置でもない場合、242において関連するタグはヌルである。たとえば、サブ命令204bがソース位置でもターゲット位置でもないが、サブ命令204がターゲットである場合、242aはヌルであることがあり、242bは例240において示されるようなものであることがある。サブ命令204b～cがソース位置ではない場合、そのソースidはヌルである(たとえば、例240の204b～204cがターゲット位置であり、245aと236aの両方がヌルであるので)。サブ命令204b～cがターゲット位置ではない場合、その許容ソース集合フィールドポインタはヌルである。たとえば、サブ命令204bがターゲット位置ではなくソース位置を特定する場合、ソースid245aはソース位置命令のアドレスを特定し、245bはヌルである。
【０１９０】
さらに例示すると、例240に説明されるものなどの構造を使用した別の例250への参照が図38で行われ、違いは、サブ命令のうちの第1である251aがソース位置でもターゲット位置でもなく、サブ命令のうちの第2である251bが3つの有効なソース位置のいずれかから制御が移転され得るターゲット位置であるということである。要素251a～bは、タグ251を有する単一のタグ付けされたワードに含まれる2つの32ビットサブ命令を示し得る。タグ251は、サブ命令251a～bのためのタグのペアを伴う構造252を含むメモリの中の位置を特定するポインタ1228であり得る。要素252は例240の242と同様であり得る。要素252aはサブ命令251aのためのタグポインタであることがあり、要素252bはサブ命令251bのためのタグポインタであることがある。サブ命令251aはソース位置でもターゲット位置でもないので、252aは0により示されるようにヌルである。サブ命令251bはターゲッ

ト位置であるので、252bは構造254へのポインタ1238である。要素254は例240の246と同様であり得る。要素254aはソースidフィールド(246aのような)であり、要素254bは許容ソース集合構造256(例240の248と同様)へのポインタ(アドレス1248)を含む許容ソース集合フィールド(246bのような)である。サブ命令251bはターゲット位置であるだけでソースではないので、254aのソースidはヌルである。要素256は例240の248と同様であり得る。要素256aは、有効なソース位置の数を示すサイズフィールド(248aのような)であり得る。要素256b～dは、たとえば有効なソース位置命令のアドレスであり得る、有効なソースidを示し得る。この例では、256aは、エントリー256b～dにそれぞれ記憶されているアドレス50bc、5078、5100を有する3つの有効なソース位置があることを示す。上記に関連して、一般に命令はターゲットとソースの両方であり得るので、ターゲットであることが、ソースidが常にヌルであることを意味しないことに留意されたい。たとえば、命令がターゲットとソースの両方である場合、ソースidはヌルではなく、命令のタグは許容可能な/許容されるソースのリストを含む。

【０１９１】

エントリー256b～dの中、およびより一般的には、許容ソース集合の任意の許容されるソース(たとえば、例240の248の248b～nのいずれか)の中などの、ソース位置のアドレスは、バイトレベルのアドレス粒度であり得ることに留意されたい。

【０１９２】

単一のタグ付けされたワードに含まれる複数の命令(サブ命令とも呼ばれる)についてすぐ上で説明されたのと同様の方式で、実施形態は、データの単一のタグ付けされたワードよりも少ないデータ部分へのアクセスを許容し得る。たとえば、ある実施形態は、バイトレベルでデータにアクセスする命令を含むことがあり、単一のタグ付けされたワードに含まれる複数のサブ命令の各々のために異なるタグを提供するのと同様の方式で各バイトが固有の関連するタグを有し得るように、バイトレベルのタグ付けを行うことが望ましいことがある。以下の例では、64ビットワードに含まれる8バイトの各々が固有の関連するタグを有し得る、バイトレベルのタグ付けを行うことが言及される。しかしながら、より一般的には、本明細書の技法は、単一のタグ付けされたワードに含まれる任意の数の複数のデータアイテムをタグ付けするサブワードを提供するために使用され得る。そのような場合、タグ付けされたデータワードと関連付けられるタグは、タグ付けされたデータワードのバイトのためのバイトレベルタグを特定する構造へのポインタであり得る。

【０１９３】

図39を参照すると、本明細書の技法に従ったある実施形態において使用され得るバイトレベルタグ付けの例260が示されている。要素262はタグ付けされた64ビットワード265と関連付けられるタグ262aを示し、ここでワード265はB1～B8として示される8バイトを含む。タグ262aは、データワード265のバイトB1～B8の各々のためのタグを含む構造266のメモリ位置を指す(261)ポインタであり得る。構造266は、構造の中の残りのエントリーの数を示すサイズフィールドである第1のフィールド265aを含み得る。構造の中の各々の後続のエントリーは、タグ値を含むことがあり、その特定のタグ値を有するワード265の1つまたは複数のバイトを示すことがある。この例では、サイズ265aは8であり、265のバイトB1～B8の各々が異なるタグ値を有する。要素266a～hはそれぞれ、ワード265のバイトB1～B8のタグ値を示す。

【０１９４】

図40を参照すると、本明細書の技法に従ったある実施形態において使用され得るバイトレベルタグ付けの第2の例267が示されている。要素262はタグ付けされた64ビットワード265と関連付けられるタグ262aを示し、ワード265はB1～B8として示される8バイトを含む。タグ262aは、データワード265のバイトB1～B8の各々のタグを含む構造268bのメモリ位置を指す(268a)ポインタであり得る。構造268bは、構造の中の残りのエントリーの数を示すサイズフィールドである第1のフィールド265bを含み得る。したがって、265bは図39の265aと同様である。構造268bの中の各々の後続のエントリーは、タグ値を含み、その特定のタグ値を有するワード265の1つまたは複数のバイトを示し得る。この

例では、サイズ265bは、7つの後続のエントリー266a～266fおよび268cを示す7である。要素266a～fは、図39の例260に関連して説明されるようなものである。要素268cは、タグ7がバイトB7とB8の両方のタグであることを示す。したがって、構造268bは、図39の構造266より1つ少ないエントリーを含み、それは、例267において、バイトB7とB8の両方がタグ7の同じタグ値を有するからである。このようにして、データワードのタグ(たとえば、262a)により指される構造(たとえば、268b)は、特定のバイトレベルタグに応じて必要とされるような可変の数のエントリーを有し得る。

【０１９５】

データアクセスの粒度の具体的なレベルは、実施形態における具体的なアーキテクチャおよび命令セットとともに変化し得ることに留意されたい。上記のことが、バイトレベルのデータアクセスを許容するある実施形態においてバイトレベルタグ付けを行うために使用され得る。変形として、ある実施形態は異なるレベルの粒度でデータアクセスをサポートすることができ、本明細書の技法はあらゆるサブワードタグ付けのレベルの粒度に容易に拡張され得る。

【０１９６】

同様に、例260および267は、バイトレベルデータタグ付けまたは他のサブワードデータタグ付けを保つために使用され得るデータ構造の一例を示す。変形として、ある実施形態は、ツリー構造または他の階層構造を使用して、単一のタグ付けされたデータワードのバイトのためのバイトレベルタグを指定することができる。バイトレベルタグを表すツリー構造または他の階層的構造は、たとえば、本明細書の他の箇所で説明される図78～図81のそれぞれ要素100、120、130、および140に関連して、ワードレベルタグを記憶するための本明細書で説明される階層的構造と同様であり得る。

【０１９７】

さらに例示すると、ある実施形態は、ツリー構造を使用して、図41の例270のようにバイトレベルタグを表すことができる。例270では、要素262は、バイトB1～B8を含むタグ付けされたワード265と関連付けられるタグ262aを示し得る。タグ262aは、B1～B8 265のバイトレベルタグを表すツリー構造に対するポインタまたはアドレスであり得る。たとえば、タグ262aは、ツリー構造のルートノード272の位置を指し得る。この例のツリー構造は、レベル1におけるルートノード272、レベル2におけるノード274a～b、レベル3におけるノード276a～d、およびレベル4におけるノード278a～hを含み得る。ツリーの各ノードは、1つまたは複数のバイトのバイト範囲と関連付けられ得る。ツリーのリーフは、バイトB1～B8のバイトレベルタグを示し得る。したがって、ツリーの非リーフノードは、タグ値を指定せず、むしろ、1つまたは複数のより低いレベルにある1つまたは複数の子孫ノードが、非リーフノードと関連付けられるバイト範囲のためのバイトレベルタグを判定するために調べられる必要があることを、示す。リーフノードは、265の複数のバイトの範囲に対して、同種のまたは同じタグ値を示し得る。各非リーフノードは、非リーフノードの左側子ノードに対する左側ポインタと、非リーフノードの右側子ノードに対する右側ポインタとを含み得る。親ノードの子ノードの各々は、親ノードと関連付けられるバイト範囲の区分を表し得る。

【０１９８】

例270は、同種のバイトレベルタグがなく、265のバイトB1～B8の各々が異なるタグ値を有する、ツリー構造を示す。(たとえば、図78～図81のそれぞれ要素100、120、130、および140についての)本明細書の他の箇所の議論と一貫して、ある実施形態は、あるサブツリーが第1のノードをそのルートとして有し、第1のノードと関連付けられるバイト範囲に対して同種のタグ値を示す場合、そのサブツリーからの子孫ノードを省略することができる。たとえば、さらに例示すると、図42が参照される。例280において、要素262は、上で説明されたようなバイトB1～B8を含むタグ付けされたワード265と関連付けられるタグ262aを示し得る。タグ262aは、B1～B8 265のバイトレベルタグを表すツリー構造に対するポインタまたはアドレスであり得る。この例280では、バイトB1～B8の各々は同じタグT1を有するので、ツリー構造はルートノード281しか含まなくてよい。バイト

B1～B8のバイトレベルタグは、時間とともに修正または変更されることがあり、タグ262aにより指されるツリー構造または他の構造はそれに従って、そのようなバイトレベルタグの修正を反映するように更新されることがある。

【０１９９】

同じデータワード265内で、バイトレベルタグ付け、またはより一般にはサブワードタグ付けを提供するある実施形態では、どの1つまたは複数のバイトレベルタグが参照されているか(バイトのうちのどの1つまたは複数がワード265に含まれているかに対応する)を示す追加の入力が、PUMPに与えられ得る。たとえば、単一のタグ付けされたデータワード265の中に8バイトB1～B8があるバイトレベルタグ付けでは、PUMPへの追加の入力は8ビットのビットマスクであることがあり、ここで、8ビットの各々は、バイトB1～B8のうちの異なる1つと関連付けられ、ワード265の特定のバイトのためにバイトレベルタグを使用するかどうかを示す。変形として、ある実施形態は、開始バイトおよび長さまたはサイズなどの、バイト範囲を指定することによって1つまたは複数のバイトを示し得る(たとえば、開始バイトB4を指定し、5というサイズまたは長さを示すことによって、バイトB4～B8を示す)。RISC-Vアーキテクチャに基づく本明細書の他の箇所の議論と一貫した少なくとも1つの実施形態では、1つまたは複数のバイトB1～B8のどの1つまたは複数のバイトレベルタグがPUMPによって使用されるべきかを示す追加の入力を記録または記憶する、CSRが定義され得る。追加の入力は、たとえば、PUMPにより使用される特定のバイトレベルタグを特定するビットマスクまたは他の適切な表現であり得る。少なくとも1つの実施形態では、PUMPは普通は、CSRを使用することなく、どの1つまたは複数のバイトがデータパスから(たとえば、コード実行領域から)の入力として使用されるべきかを示す、前述の追加の入力を受信し得る。しかしながら、ルールミスに際して、前述の追加の入力がCSRに記録され得るので、ルールミスハンドラが実行しているメタデータ処理領域は、前述の追加の入力を取得することができる(たとえば、前述の追加の入力のCSR値がルール挿入に際してPUMPに提供される)。

【０２００】

本明細書の他の箇所で論じられるように、ポリシーレベルにおいて、多くの命令が同様の方式で扱われ得る。たとえば、加算および減算の命令のオペレーションコードすなわちオペコードは通常、それらのメタデータを同じように扱うことがあり、それにより、両方のオペコードが、PUMPへの同じタグ入力およびPUMPにより伝播される同じタグ出力を考慮することによって、特定のポリシーに対してルールレベルで同様に振る舞い得る。そのような場合、加算および減算のオペコードは、単一のオペレーショングループすなわち「オペグループ」へと一緒にグループ化され得るので、ルールの同じ集合がその特定のオペグループの中のすべてのオペコードのために使用され得る。オペコードがどのように一緒にグループ化されるかは、ポリシーに依存するので、ポリシーとともに変化し得る。一実施形態では、ある特定のオペコードをポリシーレベルごとに関連するオペグループにマッピングする、変換またはマッピングテーブルが使用され得る。言い換えると、マッピングはポリシーごとに変化し得るので、異なるマッピングテーブルが各ポリシーに対して作成され得る(または複数のポリシーの指定されたグループが同じオペコードからオペグループへのマッピングを有する)。

【０２０１】

ある特定のオペコードに対して、変換またはマッピングテーブルは、上で述べられたようなオペグループを判定することができ、その特定のオペコードのための追加の情報も判定することができる。そのような追加の情報は、どのPUMP入力およびPUMP出力(たとえば、入力タグおよび伝播された出力タグ)がそれぞれ、ルール処理のための入力として実際に使用され特定のオペコードのためのルール処理の関連する出力として伝播されるかを示し得る、本明細書の他の箇所でも論じられるようなcare/don't careビットベクトルを含み得る。don't careビットベクトルは、ある実施形態では任意のPUMP入力および出力に関して判定され得る。一実施形態では、don't careビットベクトルは、どの入力タグおよび出力タグが関連するかを示すことができ、どの特定のオペコードビットが特定のオペコー

ドのために実際に使用されるかも示すことができる。これは、RISC-Vアーキテクチャおよび命令フォーマットに関してより詳細に以下で説明されるが、異なるアーキテクチャの他の適切な命令フォーマットに関連してもより一般的に使用され得る。特定のオペコードのためのオペグループおよびcare/don't careビット(たとえば、以下で論じられる例420の要素422)を含む前述の変換またはマッピングテーブルは、本明細書の他の箇所ではオペグループ/ケアテーブルとも呼ばれ得る。

【０２０２】

RISC-Vは、オペコードのための命令ビットの異なる集合を各々使用する複数の異なる命令フォーマットを有する。図43の例400を参照すると、RISC-Vアーキテクチャの命令を使用するある実施形態において異なるオペコードのための異なるビット符号化に含まれ得る命令のビットが示されている。一般に、RISC-Vアーキテクチャは、命令の異なるビットがオペコード符号化の一部として使用され得る、複数の命令フォーマットを含む。32ビット命令では、全体で最高22ビットがオペコードの符号化を表すために使用され得る。要素404は、命令フォーマットに応じて、特定のオペコードのためのビット符号化を表すために使用され得る、RISC-Vアーキテクチャにおける命令の部分を表す。要素404は、特定のオペコードを符号化する際に使用され得るビットの3つのフィールド、404a～404cを含む。要素404aは、7ビットの第1のオペコードフィールドであるopcode Aを示す。要素404bは、3ビットの第2のオペコードフィールドであるfunct3を示す。要素404aは、12ビットの第3のオペコードフィールドであるfunct12を示す。命令(たとえば、システム呼出しなど)に応じて、オペコード符号化は、404a～cにより示されるビットのうち最大で22個すべてを含み得る。より具体的には、RISC-Vでは、オペコードは、404cの7ビットだけを使用して、404bおよび404cだけの10ビットを使用して(404aを除く)、または404a～cの22ビットすべてを使用して、符号化され得る。さらなる変形として、RISC-Vアーキテクチャの命令は、402により示されるフィールドを使用したオペコード符号化を有し得る。要素402は、上で説明されたようなビット404bおよび404cという2つのフィールドを含む。加えて、オペコード符号化においてfunct12 404aの12ビットすべてを使用するのではなく、命令はfunct7 402aにより示されるような12ビットのうちの7ビットだけを使用し得る。したがって、さらに別の可能性として、オペコードは、要素402によって示されるようなフィールド402、404b、および404cを使用した符号化を有し得る。

【０２０３】

本明細書の技法に従ったある実施形態において使用され得るマッピングまたは変換テーブルを示す例420が図44に示されている。上で論じられたように、オペコード421は、オペコード421のためのマッピングされた出力424をルックアップまたは判定するための、オペコードマッピングテーブル422への入力またはインデックスとして与えられ得る。マッピングされた出力424は、特定のオペコード421のためのPUMP入力および出力に対するオペグループおよびcare/don't careビットベクトルを含み得る。RISCアーキテクチャおよび命令フォーマットに基づくある実施形態では、オペコードは最大で22ビットの符号化を有する可能性があり得る。しかしながら、テーブルへのインデックスとしてそのような大きな22ビットのオペコードを使用することは、22ビットのオペコードを収容するのに必要な多数のエントリーが原因で、不適当である(たとえば、テーブルは、関連するオペグループおよびcare/don't careビットベクトル情報を示す、各オペコードのためのエントリーを含むことがあり、これは22ビットのオペコードに対して数百万個のエントリーをもたらす)。そのような実施形態においてテーブル422のサイズを減らすために、テーブル422は22ビットのオペコードフィールドの一部分のみを使用してインデクシングされ得る。たとえば、少なくとも1つの実施形態では、オペコード421の入力は、例400において要素404bおよび404cにより示されるようなオペコードの10ビットであり得る。したがって、テーブル422は、オペコードのオペグループおよび関連するcare/don't careビットベクトルを判定するために、オペコードの404bおよび404cのオペコードビットを使用してインデクシングされ得る。

【０２０４】

そのような実施形態では、命令のfunct 12 404aの残りの12個のオペコードビットはPUMPへの入力として与えられることがあり、ここで404aの適切な部分が特定のオペコードのためにマスクされる。funct12 404aのどの特定のビットが特定のオペコードのためにマスクされるべき/されるべきではないかに関する情報は、オペコードのためのマッピングテーブル422のルックアップから出力されるcare/don't careビットベクトル情報に含まれ得る。RISC-Vアーキテクチャに基づく少なくとも1つの実施形態では、care/don't careビットベクトル情報は、オペコードのためのfunct12 404aの12個のオペコードビットに関して以下のうちの1つを示し得る。

1．404aのビットは使用されないので12ビットすべてがマスクされてもよい;

2．402aにより示されるような12ビットのうちの7ビットが使用され、404aの下位5ビット(たとえば、ビット20～25)はマスクされる;または

3．404aの12ビットすべてが使用されるので、404aのビットのマスキングはない

【０２０５】

また、そのような実施形態では、funct12 404aの12個のオペコードビットは、PUMPへのルール挿入を実行することに関連してPUMP入力として与えられる、本明細書の他の箇所で説明されるsfunct12 CSRなどのCSRに記録または記憶され得る。少なくとも1つの実施形態では、PUMPは普通は、CSRを使用せずにデータパスから(たとえば、コード実行領域から)前述のオペコードビットを受信し得る。しかしながら、ルールミスに際して、前述のものがCSRに記憶され得るので、ルールミスハンドラが実行しているメタデータ処理領域は、前述のものを入力として得ることができる(たとえば、CSR値がルール挿入に際してPUMPへの入力として与えられる)。

【０２０６】

本明細書の技法に従った少なくとも1つの実施形態では、複数のユーザプロセスが仮想メモリ環境を使用して実行することができ、仮想メモリ環境において、物理ページがユーザプロセスアドレス空間にマッピングされる。本明細書の技法は、複数のユーザプロセスの間でのメモリの物理ページの共有を可能にするために利用されることがあり、ここで、1つまたは複数の物理ページの同じ集合が、複数のユーザプロセスアドレス空間へと同時にマッピングされ得る。少なくとも1つの実施形態では、共有が許容可能であるようなプロセスにより使用されるタグは、ユーザプロセスアドレス空間にわたって同じ値および意味または解釈をグローバルに有するものとして特徴付けられ得る。

【０２０７】

図45を参照すると、本明細書の技法に従ったある実施形態におけるプロセス間での物理ページの共有を示す例430が示されている。例430は、アドレス空間434を有するプロセスP1とアドレス空間436を有するプロセスP2とを含む。要素434は、0からMAXまでの仮想メモリプロセスアドレス空間または範囲を示すことがあり、MAXはP1により使用される最大の仮想メモリアドレスを示し、0はP1により使用される最小の仮想アドレスを示す。当技術分野において知られているように、メモリ432の物理ページは434などの仮想アドレス空間へとマッピングされることがあり、ここで、マッピングされる物理ページの内容は、そのようなマッピングされる物理ページのマッピングされた仮想アドレスを使用してP1によりアクセスされ得る。たとえば、物理ページA432aは、P1の仮想アドレス空間の副範囲X1へとマッピングされ得る。プロセスP1は、たとえば、副範囲X1の中の特定の仮想アドレスを参照することによって、ページA432aの中の位置からデータアイテムまたは命令を読み取り得る。

【０２０８】

同様に、メモリ432の物理ページは仮想アドレス空間436へとマッピングされることがあり、ここで、マッピングされる物理ページの内容は、そのようなマッピングされる物理ページのマッピングされる仮想アドレスを使用してP2によりアクセスされ得る。たとえば、物理ページA432aは、P2の仮想アドレス空間の副範囲X2へとマッピングされ得る。プロセスP2は、たとえば、副範囲X2の中の特定の仮想アドレスを参照することによって、ページA432aの中の位置からデータアイテムまたは命令を読み取り得る。

【０２０９】
タグ431はページA432のメモリ位置のタグを示すことがあり、ここでそのようなタグは、本明細書で説明されるようなルール処理に関連してPUMPにより使用されることがある。ページA432は示されるようにマッピングを介してP1とP2の両方によって共有されるので、タグ431の同じ集合はまた、P1とP2の両方の命令を実行することに関連してPUMPによって使用される。そのような実施形態では、タグ431はP1とP2の両方により共有されるグローバルタグとして特徴付けられ得る。加えて、少なくとも1つの実施形態では、複数のプロセスP1およびP2により共有されるグローバルタグ431は、同じルールおよびポリシーを使用するなどして、同様の方式で解釈される。たとえば、100という値を有する第1のタグは、432aにおいて第1のメモリ位置と関連付けられ得る。第1のタグは、特定の実行中の命令が第1のメモリ位置またはその内容を参照する動作を実行することが許容可能であるかどうかを判定する、ポリシーのルールに関連して使用される第1のメモリ位置の色付けを表す値を示し得る。第1のタグは、P1とP2の両方の命令実行に関連してルールによって同じ色として解釈され得る。たとえば、100というタグ値は、P1とP2の両方に関連してルールによって同じ色として解釈される必要がある。さらに、ポリシーおよびルールの同じ集合またはインスタンスが、P1とP2の両方のためにPUMPによって使用され得る。

【０２１０】
上で説明されたように共有されるメモリ上でグローバルタグを使用するような実施形態では、プロセスごとに異なるアクセス、権限、もしくは動作をさらに区別または許容することも認めるのが望ましいことがある。たとえば、ページA432aがP1とP2の両方により共有されるデータを含むと仮定する。しかしながら、共有されるページA432aをタグ付けするためにグローバルタグが使用されるとしても、プロセスごとに432aの共有されるデータに関して異なる動作またはアクセスを許容するのが望ましいことがある。たとえば、プロセスP1はページ432aへの書込みアクセス権を有することがあり、プロセスP2はページ432aへの読取り専用アクセス権を有することがある。しかしながら、432aは、グローバルタグでタグ付けされた共有されたメモリページであり得る。共有されたページ上のグローバルタグを伴うような実施形態では、同じポリシーおよびルールの集合がP1およびP2に関連して使用されることがあり、ここで、各プロセスのための異なる読取りおよび書込みアクセス能力は、PCの異なるタグ値を使用して区別されることがある。たとえば、プロセスP1は432aにおいてメモリ位置への書込みを実行する第1の命令を含むことがあり、現在のPC tagはXという値を有する。アクセスポリシーのルールは以下の論理を実行し得る。
PCtag=Xの場合、書込みを許容する
PCtag=Yの場合、読取りのみを許容する
そのような場合、PC tagは、プロセスP1に対する書込みアクセスを許容するようにルールにより解釈されるXという値を有するので、P1は第1の命令を実行することが許容される。プロセスP2は432aにおいてメモリ位置への書込みを実行する第2の命令を実行していることがあり、現在のPC tagはYという値を有する。そのような場合、PC tagは、プロセスP2のために書込みアクセスを許容せず、むしろ読取り専用のアクセスを許容するようにルールによって解釈されるYという値を有するので、P2は第2の命令を実行することが許容されない。

【０２１１】
したがって、少なくとも1つの実施形態では、PC tagは、プロセスごとに異なり得る特権、アクセス権、または権限を符号化するために使用されることがあり、これにより、具体的な許容される特権、アクセス権、または権限は、異なるPC tag値によって表されることがある。

【０２１２】
ある実施形態は、任意の適切な方式で、各プロセスのために使用されるべき具体的なPC tag値を指定し得る。たとえば、特権コードは、特定のプロセスのために使用されるべきPC tag値を最初に指定するオペレーティングシステムの起動または初期化の一部として、実

行することができる。変形として、ある実施形態は、共有されたページA432aをプロセスアドレス空間へとマッピングすることの一部として、マッピング動作を実行することができる。マッピングを実行するときにオペレーティングシステムによって適用されるルールは、具体的なプロセスに基づく所望のアクセス権、特権、または権限を示す出力として、特定のPC tagを伝播または産生することができる。

【０２１３】

このようにして、ルールの同じ集合を、グローバルタグを有する共有されたページとともに使用することができ、ここで、ルールはPC tagに基づくアクセス権、権限、または特権の違いについての論理を符号化する。PC tagはメモリ位置へのポインタであることもあり、それにより、ポインタタグは、他のタグに関連して本明細書で説明されるような方式で異なるポリシーのための異なるタグ値を含む構造を指すことに留意されたい。このようにして、PC tag値の同じ集合は、ポリシーとともに変化し得るプロセスの異なる能力を示すために使用され得る。たとえば、P1について上で説明されたようなXというPC tag値は、共有された領域のためのメモリ安全性ポリシーまたはデータアクセスポリシーについて、上で説明されたような第1の用途を有し得る。Xという同じPCタグ値は、制御フロー整合性(CFI)などの第2の異なるポリシーのルールによって伝えられる第2の用途および意味を有し得る。

【０２１４】

許容可能な呼出し、ジャンプ、リターンポイントなどの静的な定義に基づいて制御の移転を制約することに関連して使用され得る、CFIポリシーの態様が本明細書で説明される。しかしながら、CFIポリシーに含まれ得る追加の態様または次元は、動的なまたはランタイムの呼出し情報の実施に関するので、リターンである制御の移転が行われ得る条件をさらに精緻にする。さらに例示すると、ルーチンfoo502、bar504、およびbaz506を含む、図46の例500への参照が行われる。ルーチンFoo502は、bar504への制御のランタイム移転501aをもたらすルーチンbarを呼び出す、アドレスX1における呼出し命令を含み得る。ルーチンbar504は次いで、制御をルーチンbarからアドレスX2に返す(501b)リターン命令を含む。したがって、X2は、X1におけるルーチンbarの呼出しに続く、ルーチンfooにおける命令のリターンポイントアドレスまたは位置を示す。ルーチンFoo502は、baz506への制御のランタイム移転501cをもたらすルーチンbaz506を呼び出す、アドレスY1における第2の呼出し命令を含み得る。ルーチンbaz506は次いで、制御をルーチンbarからアドレスY2に返す(501d)リターン命令を含む。したがって、Y2は、Y1におけるルーチンbazの呼出しに続く、ルーチンfooにおける命令のリターンポイントアドレスまたは位置を示す。

【０２１５】

静的なCFIポリシーは、たとえば、動的なランタイム制御フローの態様を反映する現在のランタイムスタックまたは呼出しチェーンに基づいて、制御のフローまたは移転をさらに制約することなく、任意の2つの移転ポイント間でのすべてのあり得る制御フローを許容し得る。たとえば、foo502が500において示されるようにbar504を呼び出すことができる場合、fooの中のX1におけるbarの呼出しの後で、barから命令のアドレスX2へとリターンするような、静的に許容される制御フローがある。しかしながら、fooが呼び出されていない場合、または、fooがこれまでに、bar呼出しの前にリターンするであろう何かへの別の呼出しを呼び出しただけである場合、X2にリターンするためのリターンリンクを用いることは可能であるべきではない。例500に示されるようなランタイム実行の別の例として、Bar504から501aの呼出しにより、501dを通じてY2においてFoo502にリターンすることが可能であるべきではない。

【０２１６】

ここで説明されるのは、リターンフローパス制御を制御する動的CFIリターンポリシーを実施するために、CFIポリシーのルールを拡張することに関連して使用され得る技法である。X2などの特定のリターン位置へのリターンが、X1におけるbarの呼出しなどの特定の呼出し(call)または呼出し(invocation)に続いて行われるときにのみ有効であることを動

的なCFIリターンポリシーが確実にするために、動的なCFIリターンポリシーは、無効なリターンを除外するために呼出しが行われるとき、1つまたは複数のタグなどに情報を記憶することができる。当技術分野において知られているように、RISC-V命令セットのJAL(ジャンプアンドリンク)命令などを使用して呼出しが行われるとき、リターンアドレスはリターンアドレスレジスタRAに保存される。RISC-V命令セットはまた、リターン命令の例であるJALR(ジャンプアンドリンクレジスタ)命令を含む。一態様では、JALからRAレジスタに保存されるリターンアドレスは、その点にリターンする「能力」として特徴付けられ得る。少なくとも1つの実施形態では、JAL命令は、適切なタグ能力を得られたリターンアドレスへとルールにプッシュさせるタグを用いてタグ付けされ得る。たとえば、リターンアドレスレジスタとしてのRAについて、ルールは、RAレジスタが有効なまたは適切なリターンアドレスを含むことを示すタグをRAレジスタに付けることができ、後の時点で、RAレジスタのアドレスが、制御が移転され得るリターンポイントとして使用され得る。言い換えると、RAレジスタのタグは、制御のリターン移転を実行するためにPCへとロードされるリターンアドレスとして使用されるべき、RAにおけるアドレスに対するパーミッションを与える。RAのアドレスを用いてPCをロードするとき、RAタグはまた、CFIポリシーのルールによってPCタグとして記憶され得る。

【０２１７】

リターン時に制御フローを制限するために使用され得る技法をさらに例示すると、ある実施形態は、expect-Aなどのdynamic-CFI-tagで各リターンポイント(たとえば、X2、Y2)をコードタグ付けし得る。また、各JAL命令(または呼出し命令)をコードタグ付けすると、JAL命令のために評価されるルールが、適切なdynamic-CFI-return-to-AタグでRAレジスタの中のリターンアドレスをタグ付けするようになる(ここでリターンアドレスはJALによって計算される)。dynamic-CFI-return-to-Aタグでタグ付けされたレジスタを使用する各JALR命令などの各リターンに対して、PUMPルールは、他の静的なCFIポリシールールに関連して実行され得るようなPCへとタグ(dynamic-CFI-return-to-Aタグ)を伝播する。CFIポリシーのルールは、リターン命令のために使用されるRAレジスタを確認する論理を具現化し得る。リターンのために使用されるRAレジスタがdynamic-CFI-return-to-Aタグでタグ付けされない場合、RAレジスタはJALR命令とともに使用するのが許容される有効なリターンアドレスを含まないことが知られる。リターンポイント(たとえば、X2およびY2)において、ルールは、(たとえば、X2における命令のタグとして)expect-Aコードタグに遭遇すると、dynamic-CFI-return-to-AでPCがタグ付けされることを確認し、PCからCFI-return-to-Aタグをクリアするように、論理を具現化することができる。

【０２１８】

上の結果として、コードはいずれのリターンアドレスにも戻ることが防がれる。さらに、リターンアドレスが別のレジスタなどの別の位置にコピーされる場合、ルールは、コピーされた値がリターン認可能力を保持するのを防ぐことができ、このことは、同じ呼出しのために複数のリターンを実行するために使用され得るレジスタにおいてリターンアドレスのコピーをコードが行うのを防ぐ。上の別の結果として、スタック上の有効なリターンアドレス(適切にタグ付けされた)が新しいアドレス(適切にタグ付けされていない)により上書きされ、新しいアドレスへのリターンが試みられる場合、リターンは防がれる。

【０２１９】

ある実施形態はまた、dynamic-CFI-return-to-Aタグを2回以上使用する能力を防ぎ、またはさらに制限するためのルールを含み得る。第1の実装形態として、ある実施形態は、リターンアドレス(dynamic-CFI-return-to-Aタグでタグ付けされたRAレジスタに記憶されるような)がどこに書き込まれ得るか、またはコピーされ得るかを制約するルールを使用し得る。たとえば、ある実施形態は、適切にコードタグ付けされた関数コードにおけるスタックにリターンアドレスを書き込むことを、適切にタグ付けされたRAレジスタのリターンアドレスにのみ許容するルールを使用し得る。第2の代替的な実装形態として、ある実施形態は、PC状態(たとえば、PC tag)およびアトミックメモリ動作を使用してリターンアドレスを線形にする(たとえば、呼出しに後続する、または呼出しの後に発生するようにす

る)ルールを含み得る。たとえば、呼出しを実行すると、valid-return-addressを示すようにPC tagが設定される。PC tagがvalid-return-addressに設定される場合にのみ、ルールはリターンを許容し得る。リターンアドレスをメモリに書き込むときに、PC tagをno-return-addressに設定する追加のルールが使用され得る。リターンアドレスをターゲットレジスタにコピーするとき、PC tagをno-return-addressに設定し得るルールが使用されることがあり、ターゲットレジスタはvalid-return addressとしてタグ付けされない。RAレジスタからのリターンアドレスを使用して算術演算が実行されるときに、結果がvalid return addressとしてタグ付けされないという、ルールが使用され得る。non-return-addressとのアトミックなスワップ動作を用いてメモリからリターンアドレスを復元することのみを許容する(たとえば、PC tagがvalid-return-addressに設定されている場合)、ルールが使用され得る。

【０２２０】

ある実施形態はさらに、スタック保護ポリシーを提供するためのルールをさらに定義し得る。一態様では、スタック保護ポリシーは一部が、ルールがポリシー実施のために命令とデータの両方のタグを使用し得るメモリ安全性などの、1つまたは複数の他のポリシーの拡張であると見られ得る。以下の議論および本明細書の他の箇所では、ルーチンおよびプロシージャなどの用語は、交換可能に使用されることがあり、より一般的には、呼び出されるとコールスタック上への新しいスタックフレームの作成をもたらすコードの呼出し可能ユニットを指すことに、留意されたい。コードの呼出し可能ユニットのためにやはり使用され得る他の名称は、関数、サブルーチン、サブプログラム、メソッドなどを含み得る。

【０２２１】

図47を参照すると、本明細書の技法に従ったある実施形態におけるランタイム呼出しのためのフレームのコールスタックを示す例520が示されている。520では、ルーチンfoo502がG1への呼出しを実行し、G1が次いでG2を呼び出すと仮定する。したがって、実行の時点で、ルーチンfooは実行中であり、ルーチンG1の第1の呼出しをすでに行っており、G1はルーチンG2の呼出しをすでに行っている。要素522は、ルーチンfooのための第1のコールスタックフレームを表し得る。要素524は、ルーチンG1のための第2のコールスタックフレームを表し得る。要素526は、ルーチンG2のための第3のコールスタックフレームを表し得る。

【０２２２】

ランタイムの呼出しインスタンス(call instance)または呼出し(invocation)のためにスタックフレーム(522、524、526など)に記憶される情報は、たとえば、リターンアドレス、レジスタのためにその呼出しインスタンスにより使用されるデータ、変数またはデータアイテムなどを含み得る。要素522aおよび524aは、それぞれ、fooのためのフレーム522およびG1のためのフレーム524に含まれるリターンアドレスを示し得る。悪意のあるコードによって実行され得るものなどの、1つの一般的な攻撃は、スタック520に記憶されている522aおよび524aなどのリターンアドレスを変更することであり得る。動的なCFIリターンポリシーのために本明細書の他の箇所で説明される(たとえば、例500における図46に関連して説明される)技法などを使用することは、不適切に変更されたスタック位置からリターンアドレスを使用することなどの、不適切なまたは無効なリターンを防ぎ得る。しかしながら、スタック保護を提供し、リターンアドレスなどのスタック記憶位置の不適切な変更を防ぐ、追加のルールを実施することもさらに望ましいことがある。したがって、スタックフレーム保護ポリシーのためのそのような追加のルールは、522aの不適切な変更を許容し次いで不適切に変更されたリターンアドレスを使用するリターンを止めるのではなく、522aまたは524aの変更を防ぐことができる。

【０２２３】

以下でより詳細に説明されるように、様々なレベルのスタック保護が提供され得る。一態様では、スタック保護は、静的なプロシージャに基づいて判定されることがあり(本明細書の他の箇所で説明される静的権限保護モデルとも呼ばれる)、または、プロシージャとその特定のプロシージャの呼出しインスタンスの両方に基づいて判定されることがある(本明細

書の他の箇所で説明されるインスタンス権限保護モデルとも呼ばれる)。静的権限保護モデルでは、スタック保護ポリシーのルールは、フレームを作成する特定のプロシージャまたはルーチンに基づいて、スタック保護を提供し得る。たとえば、520のようにスタックがfooの単一のインスタンスに対して単一のフレームのみを含むのではなく、ある時点において現在の呼出しチェーンにおいて含むfooの複数の呼出しインスタンスが、したがって、ルーチンfooのためのスタックにおける複数の呼出しスタックフレーム(たとえば、fooの再帰的な呼出しなどに基づくものなど)が存在し得る。静的なルーチンまたはプロシージャに基づいて、fooの任意のインスタンスが、fooのあるインスタンスのための任意のコールスタックフレームの中の情報を変更し、またはそれにアクセスすることが可能であり得る。たとえば、fooインスタンス1はコールスタックフレーム1を有することがあり、fooインスタンス2はコールスタックフレーム2を有することがある。スタック保護の静的なルーチンまたはプロシージャに基づいて、fooインスタンス1のコードはスタックフレーム1および2にアクセスすることが可能であることがあり、fooインスタンス2のコードもスタックフレーム1および2にアクセスすることが可能であることがある。そのような実施形態では、同じプロシージャまたはルーチンfooのすべてのインスタンスのためのコールスタックフレームは、同じタグで色付けされ得る。たとえば、fooインスタンス1のためのフレーム1およびfooインスタンス2のためのフレーム2はともに、タグT1で色付けされ得るので、メモリ安全性ポリシーのルールは、同じルーチンまたはプロシージャの異なるインスタンスにわたって上で述べられたスタックフレームアクセスを許容する。

【０２２４】

さらに細粒度のスタック保護として、ある実施形態は、静的なルーチンまたはプロシージャならびにルーチンまたはプロシージャの特定のランタイムのインスタンス(たとえば、インスタンス権限保護モデル)に基づいてスタックのアクセスをさらに制限する、スタック保護ポリシーのルールを使用し得る。たとえば、上で述べられたように、fooインスタンス1はコールスタックフレーム1を有することがあり、fooインスタンス2はコールスタックフレーム2を有することがある。静的なルーチンまたはプロシージャ、およびスタック保護のための呼出しインスタンスにも基づいて、fooインスタンス1のコードはスタックフレーム1にアクセスできるがスタックフレーム2にアクセスできないことがあり、fooインスタンス2のコードはスタックフレーム2にアクセスできるがスタックフレーム1にアクセスできないことがある。そのような実施形態では、プロシージャまたはルーチンの各呼出しインスタンスのためのコールスタックフレームは、異なるタグで色付けされ得る。たとえば、fooインスタンス1のためのフレーム1はタグT1で色付けされることがあり、fooインスタンス2のためのフレーム2はタグT2で色付けされることがあるので、メモリ安全性ポリシーのルールは、各々の特定の呼出しおよびルーチンまたはプロシージャに基づいて、上で述べられたスタックフレームアクセスを許容する。

【０２２５】

ある実施形態はさらに、各々異なる色でスタックフレームの中の異なるオブジェクトまたはデータアイテムを色付けすることなどによって(本明細書の他の箇所で説明されるオブジェクト保護モデルとも呼ばれる)、単一プロシージャの呼出しインスタンスのためのスタックの異なる領域または部分に対して、より細かいレベルの粒度を提供することができる。本明細書の他の箇所で説明されるように、スタックフレームは、ルーチンまたはプロシージャの特定の呼出しにおいて使用されるデータアイテムまたはオブジェクトのためのストレージを含むことがあり、ここで、各々のそのようなデータアイテムまたはオブジェクトは異なる色でタグ付けされることがある。たとえば、図48を参照すると、ルーチンまたはプロシージャfooおよび関連するタグ付けされたメモリによって割り振られるストレージをスタックフレーム531の中に有するデータアイテム540を示す例530が示されている。要素540はルーチンfooにおいて割り振られるストレージを有する変数540a～cを示し、要素531はコールスタックにおけるルーチンfooのこの特定の呼出しインスタンスのためのコールスタックフレームを表す。要素531は、変数array540aのためのメモリ領域532と、変数line540bのためのメモリ領域534と、変数password540cのためのメモリ領域

536とを含む。加えて、フレーム531は記憶されたリターンアドレスのためのメモリ領域538を含む。異なる領域532、534、536、および538の各々は、533により示される異なるタグでタグ付けまたは色付けされ得る。領域532の中の各ワードはRed1でタグ付けされ得る。領域534の中の各ワードはRed2でタグ付けされ得る。領域536の中の各ワードはRed3でタグ付けされ得る。領域538の中の各ワードはRed4でタグ付けされ得る。

【０２２６】

またさらなる変形として、ある実施形態は、コードの集合(たとえば、ルーチン、プロシージャなど)に対して異なる信頼領域または境界を定義し、異なるレベルの保護を提供し得る。たとえば、呼び出されたすべてのルーチンが、同じレベルの信用を有するとは限らない。たとえば、開発者は、自分が書いたルーチンの第1の集合を持っていることがあり、第1の集合のコードにより実行される動作が悪意のあるコードを含まないという高いレベルの信用を持っていることがある。しかしながら、ルーチンの第1の集合は、第三者により提供された、またはインターネットから取得されたライブラリに呼出しを行うことがある。このライブラリは信頼されないことがある。したがって、ある実施形態は、コードの異なる本体および各々により使用される具体的なデータアイテムに基づいて、保護のレベルを変化させ得る。たとえば、図49の例550を参照すると、ライブラリの中の信頼されるユーザコード呼出しルーチンevilにおけるルーチンfooを仮定し、パラメータとして領域534へのポインタ(データアイテムline540bへのポインタ)をevilに渡す。そのような場合、異なる色で531の各領域を色付けまたはタグ付けするのではなく、領域532、536、および538はすべてRed5などの同じ色で色付けされることがあり、領域534はRed6などの異なる色でタグ付けされることがある。これは、ルーチンevilが信頼されないコードであると見なされるので、ルーチンevilによりアクセスされるメモリ領域534が、メモリ安全性のレベルとして531の他の領域と異なる色でタグ付けされることをさらに確実にするために使用され得る。加えて、evilに渡される領域534へのポインタは、領域534と同じ色Red6で色付けまたはタグ付けされ得る。このようにして、メモリ安全性ポリシールールは、evilにより使用されるメモリへのアクセスを、Red6でタグ付けされたものに限定することができる。

【０２２７】

特定のルーチン、ライブラリ、または本体またはコードが、特定のレベルの信用を有するかどうかは、1つまたは複数の基準および入力を使用した分析に基づいて判定され得る。たとえば、ランタイム分析およびライブラリのコードの使用に基づいて、信用のレベルが判定され得る。たとえば、他のまだ知られていないまたは信頼されていない、外部のまたはサードパーティのライブラリへの呼出しをライブラリが行う場合、信用のレベルは比較的低いことがある。コードの本体の信用のレベルは、コードの取得元のソースまたは位置上で使用され得る。たとえば、インターネットから取得されたライブラリからのコードの使用は信頼されないものと見なされ得る。対照的に、信頼されないコードを呼び出さない特定の開発者により開発されたコードは、高いレベルの信用を有し得る。

【０２２８】

スタックフレームおよびスタック保護の前述の態様および他の態様は、以下でより詳細に説明される。

【０２２９】

スタックフレームに関連して、かつ例530を再び参照すると、コンパイラが、整数(フレームのサイズ)を既存のスタックポインタに追加することによって、新しいスタックポインタを作成し得る。古いスタックポインタは、スタック上に(フレームへと)にプッシュされ、次いでスタックから読み戻すことによって復元され得る。スタックポインタへの追加は、データアイテム540a～cについて531において上で説明されたような多数の独立のオブジェクトを含む、フレームの全体サイズを表し得る。スタックはこれらの3つのデータアイテム540a～cのための空間を必要とし、コンパイラはデータアイテム540a～cに必要な総空間を判定することが可能である。標準的な使用法では、コンパイラは、これらのデータアイテム540a～cのためのストレージ532、534、および536にそれぞれ、スタックポイ

ンタ(またはスタックポインタから作成されるフレームポインタ)からアドレスを計算することによってアクセスする。したがって、ある実施形態におけるコンパイラ、ランタイム、および呼出規約は、簡単なポインタ計算を行うことによって、スタック呼出しフレームの異なる領域へのポインタを作成して使用し得る。

【０２３０】

静的権限保護モデルは、フレームを作成するルーチンまたは手順などの、静的なコードブロックに属するオブジェクトに対する権限を示す。したがって、本明細書の他の箇所で論じられるように、フレームを作成するプロシージャfooは、そのフレームの中のものへのポインタを作成する権限を有する。最も簡単な場合には、fooが作成するフレームがスタック上でより早くてもまたは遅くても、fooがそれらのフレームのいずれにもアクセスすることを同じ権限が許容する。静的権限とは、タグ(たとえば、メモリセルの色、色付けされたポインタ、色付けされたポインタを作成するコードタグ(たとえば、命令タグまたは命令のタグとも呼ばれる))が、ロードのときに事前に割り振られ得ることを意味する。インスタンス権限保護は、スタック上での関数呼出しの深さに基づく権限を提供する。オブジェクト保護は、スタックフレームだけではなくスタックに割り振られるオブジェクトのレベルでの保護を示す。したがって、オブジェクト保護は、フレーム内のあるオブジェクト(たとえば、アレイ、バッファ)から同じフレーム上の別のオブジェクトへのオーバーフローの検出および防止を可能にし、これは、静的権限保護モデルまたはインスタンス保護モデルを用いた簡単なスタックフレーム粒度のPUMPルールを使用すると達成されないものである。オブジェクト保護は、静的権限保護モデルとインスタンス保護モデルの両方に適用され得る。オブジェクト保護の変形として、ある実施形態はまた、整数などの複数の異なるデータアイテムのサブオブジェクトとアレイとを含む構造などの、階層的なオブジェクトのために階層的なオブジェクト保護を利用し得る。第1のオブジェクトが1つまたは複数のサブオブジェクトの1つまたは複数のレベルを含む階層的なオブジェクトを伴う少なくとも1つの実施形態では、第1のタグが第1のオブジェクトのために生成されることがあり、次いで追加のサブオブジェクトタグが第1のタグに基づいて生成されることがある。各サブオブジェクトタグは、異なるサブオブジェクトをタグ付けするために使用され得る。サブオブジェクトタグは、階層におけるサブオブジェクトの具体的な場所を示す値であり得る。たとえば、タグT1は、サブオブジェクト2および3として2つのアレイを含む構造とともに使用するために生成され得る。2つのアレイ各々のための異なるサブオブジェクトタグは、T1から生成され、2つのアレイサブオブジェクトをタグ付けするために使用され得る。

【０２３１】

ここで説明されるのは、本明細書の技法に従ったある実施形態における異なるスタック動作のためのスタックメモリに関連して実行され得る処理である。開始時に、スタックメモリは、フリースタックフレームタグを使用してすべてのメモリセルがマークまたはタグ付けされるようにし得る。本明細書の他の議論および技法と一貫して、そのようなタグ付けはPUMPルールを呼び出すことによって実行され得る。フリースタックフレームタグへのスタックメモリセルの初期のタグ付けは、一度にスタック全体に対して実行されないことがあり、むしろ、スタックを拡張するカーネルページフォルトハンドラにおいて付加的に実行され得ることに留意されたい。

【０２３２】

コンパイラなどによって新しいスタックフレームを割り振ることに関連して、新しく割り振られたフレームのために新しいフレームタグが作成され得る。新しいフレームへのポインタは、新しいフレームタグでタグ付けされ得る。たとえば、ある実施形態は、新しいフレームポインタを作成する命令(たとえば、(スタックポインタに追加することによる)ポインタ計算を実行する加算命令など)をタグ付けすることができ、ここで、その命令のタグは、ポリシールールが新しいフレームタグを作成することをトリガする。ルールおよびタグ伝播を使用して、スタックポインタにタグ付けするために、特別なタグが作成され使用され得る。続いて、各フレームポインタに対して、固有のフレームポインタタグがスタック

ポインタ特別タグから導出されることがあり、フレームポインタは固有のフレームポインタタグでタグ付けされることがある。そのような実施形態では、フレームポインタタグは、スタックポインタのタグ付けされたコピー(たとえば、addまたは0)から作成され得る。

【０２３３】

ルーチンまたはプロシージャの新しい呼出しなどのために新しいスタックフレームが割り振られるとき、新しく割り振られたスタックフレームのメモリセルは、たとえば、厳密なオブジェクト初期化と呼ばれる第1の技法、または遅延オブジェクト色付けと呼ばれ得る第2の技法を使用して、タグ付けまたは色付けされ得る。

【０２３４】

厳密なオブジェクト初期化の第1の技法では、新しく割り振られたフレームのフリースタックフレームセルは最初、すべてが、フレームの静的オブジェクトに基づくなどして、意図される1つまたは複数の色に色付けされ、またはタグ付けされる。そのような初期の色付けは、たとえば関連する呼出しのための情報を記憶するためにフレームを後で使用する前に、新しく割り振られたフレームの初期の処理の一部として実行され得る。ある実施形態は、フレームの静的オブジェクトに基づくなどして、意図される1つまたは複数の色へのフリースタックフレームセルの色付けまたはタグ付けをルールが実行することをトリガするコードを追加し得る。命令のコードタグは、関連するメモリセルの色付けを認可および定義するために使用され得る。フレームの色付けされたメモリセルの後続の記憶または読取りは、メモリ安全性ポリシールールに従うなどして、フレームメモリセルの色に基づいて許容され、またはされないことがある(たとえば、色C1でタグ付けされたメモリセルに対して、ルールは、メモリ動作が、同じ色C1であるタグを有するポインタを使用して色付けされたメモリセルの内容にアクセスすることを許容するが、ポインタが異なる色C2である場合にはメモリ動作を許容しないことがある)。加えて、命令のコードタグは、プロシージャ内でメモリ動作を実行するための権限を与えることがある。

【０２３５】

遅延オブジェクト色付けという第2の技法では、厳密なオブジェクト初期化技法のようなすべてのスタックオブジェクトの初期の色付けがない。むしろ、遅延オブジェクト色付けでは、フリースタックフレームとしてタグ付けされたスタックメモリ位置への記憶は、書込み者に基づくメモリ位置の色の記憶、また変更を許容する、ルールのトリガをもたらす。フリースタックフレームとしてタグ付けされたスタックメモリ位置の読取りは、初期化されていないメモリの読取りであり、ポリシーが初期化されていないメモリの読取りを許容するか/しないかに応じて、許容される/許容されないことがある。遅延オブジェクト色付けでは、作成に際してフレームのすべてのメモリセルを完全に最初にタグ付けするためのルールを呼び出す、コードの初期ブロックが実行されない。むしろ、記憶動作に関連して呼び出されるルールによって、メモリセルがタグ付けされる。

【０２３６】

少なくとも1つの実施形態では、厳密なオブジェクト初期化を使用するか、または遅延オブジェクト色付けを使用するかは、保護の所望のレベルおよび受け入れがたい脆弱性の発生に依存し得る。

【０２３７】

スタック/フレームポインタからデータに直接アクセスするルーチンまたはプロシージャ内のコードは、そうすることをルーチンまたはプロシージャに許容するようにタグ付けされるコードである。遅延オブジェクト色付けに関連して、メモリセルへの記憶は、上で述べられたような書込み者に基づくメモリセルの色付けをもたらす。たとえば、例530に戻って参照すると、フレーム531を有するルーチンfooの記憶命令は、アレイ532の中のメモリ位置に値を書き込み得る。有効な現在のスタック保護ポリシーによれば、fooのための呼出しフレームのアレイ532の中の位置へ記憶命令が書き込むには、記憶命令はRed1というタグを有することが必要とされ得る。ポリシーの第1のルールは、記憶命令のためのこの確認を実行するためにトリガされ得る。したがって、ある実施形態は、第1のルールがRed1で記憶命令をタグ付けすることをトリガするコードシーケンスを、コンパイラに生成

させることがある(たとえば、上記の変形として、Red1などのメモリセルのタグは、記憶命令または他の命令のタグに関連し得るが、同じではないことがある。たとえば、「Red1 code」CI tagは、このタグを有する命令がRed1でタグ付けされたメモリセルにアクセスでき、Red1でタグ付けされたメモリセルを作成できることを示し得る)。記憶命令が現在の命令であるとき、命令タグがRed1であることを確実にするために命令タグを確認する、前述の第1のルールがトリガされ得る。出力として、ルールは、Red1タグでアレイ532の中のメモリ位置をタグ付けし得る。

【０２３８】

特定のオブジェクトへのポインタを作成するプロシージャ内のコードは、そのオブジェクトのためのポインタをテイントまたは設定するためにタグ付けされる。ポインタは、後続の命令におけるプロシージャ固有の使用のためのものであることがあり、かつ/または、引数として別のプロシージャに渡されることがある。

【０２３９】

レジスタ値をフレームに記憶すること、またはレジスタ値をフレームから復元することは、フレームの権限に基づき得る。レジスタ値を記憶するスタックフレームのメモリ位置は、スタックフレームの中の固有のオブジェクトとして扱われ得る。命令のタグ付けは、そのようなタグ付けされた記憶およびロードの命令のための権限を与える。遅延オブジェクト色付けでは、データをメモリセルに記憶するための権限でタグ付けされた記憶命令はまた、書込み者に基づいてメモリセルをタグ付けする権限を与える(たとえば、記憶命令を含むプロシージャ)。

【０２４０】

スタック上で渡されるプロシージャ引数は、呼出者と被呼出者の両方がアクセスすることを許容するタグを用いてマークされ得る。リターンアドレスは特別にタグ付けされ得ることに留意されたい(たとえば、図46に関連するものなどの本明細書の他の箇所で説明される動的なCFIリターンポリシー)。したがって、リターンアドレスがスタックに記憶される場合(たとえば、ネストされた呼出しまたは再帰的な呼出しなどに関連して)、リターンアドレスのタグ付けが原因で、スタック上のリターンアドレスを上書きするための記憶は許容されない。スタックで導出されたポインタが別のプロシージャへの呼出しに関連して別のフレームに渡されるとき、ポインタを使用して実行されるメモリアクセスは、本明細書の他の箇所で説明されるようなメモリ安全性ポリシーのルールのトリガをもたらす。メモリ位置へのポインタを作成した命令は、特定のメモリ位置のタグに基づいてタグ付けされ得る。命令タグは、メモリ位置にアクセスするための権限を示し得る。命令は、メモリ位置にアクセスする権限を示すためにポインタをタグ付けするルールをトリガし得る。たとえば、このルールは、命令と同じタグを、または命令タグに基づく変形を、ポインタに割り当て得る。したがって、一態様では、ポインタを作成した命令はまた、ポインタを通じてメモリ位置にアクセスする能力を作り出しており、呼び出されるプロシージャに引数として渡されるポインタを通じてその能力を共有している。遅延オブジェクト色付けでは、ポインタは、フリースタックフレームセルをタグ付けする権限を与えるタグを有することが必要であり、これはヒープメモリ安全性ポインタでは許容されないことがあることに留意されたい。

【０２４１】

(たとえば、呼び出されるルーチンの完了などが原因で)スタックからのフレームの除去をもたらすリターンまたは他の動作に関連して、タグ付けされたコードはフレームをクリアし得る。そのようなコードのタグは、このフレームと関連付けられる任意のフレームオブジェクトタグをfree-stack-frame-cellタグに変更する権限を与える。

【０２４２】

本明細書の技法に従ったコンピュータシステムのある実施形態において実行されるプログラムのコードは、例外処理を実行するコードを含み得る。当技術分野において知られているように、例外処理は、例外ハンドラにより実行される特別な処理を必要とする異常な条件または例外条件の発生を示す例外に応答して実行される処理である。したがって、例外

がプログラムの中の第1の点において発生するとき、プログラム実行の普通のフローは、制御が例外ハンドラに移転されるように中断され得る。制御をハンドラに移転する前に、実行の現在の状態が所定の位置に保存され得る。例外がハンドラによって処理された後でプログラム実行が再開され得る場合、プログラムの実行が再開し得る(たとえば、次いで、プログラムの中の第1の点に後続する位置に制御が戻され得る)。たとえば、0で割る演算は継続可能な例外をもたらすことがあり、ここで、プログラム実行は例外がハンドラによって扱われた後で再開し得る。例外ハンドラを実装することに関連して、ある実施形態は、setjumpおよびlongjumpなどのライブラリルーチンを使用し得る。たとえば、setjumpおよびlongjumpはそれぞれ、以下で定義されるような、標準的なCライブラリルーチンsetjmpおよびlongjmpであり得る。

int setjmp(jmp_buf env)

ここで、setjmpはローカルのjmp_bufバッファをセットアップし、ジャンプのためにそれを初期化する。Setjmpは、longjmpにより後で使用するために、env引数により指定される環境バッファにプログラムの呼出し環境を保存する。リターンが直接の呼出しからである場合、setjmpは0を返す。リターンがlongjmpへの呼出しからである場合、setjmpは0ではない値を返す。

void longjmp(jmp_buf env,int value)

ここで、longjmpは、プログラムの同じ呼出しにおけるsetjmpルーチンの呼出しによって保存された環境バッファenvのコンテキストを復元する。ネストされた信号ハンドラからlongjmpを呼び出すことは定義されていない。valueにより指定される値は、longjmpからsetjmpに渡される。longjmpが完了した後で、setjmpの対応する呼出しがリターンしたばかりであるかのように、プログラム実行が継続する。longjmpに渡される値が0である場合、setjmpはそれが1を返したかのように振る舞い、そうではない場合、それがvalueを返したかのように振る舞う。

【０２４３】

したがって、setjmpはプログラムの現在の状態を保存するために使用され得る。プログラムの状態は、たとえば、メモリの内容(すなわち、コード、グローバル、ヒープ、およびスタック)、およびレジスタの内容に依存する。レジスタの内容は、スタックポインタ、フレームポインタ、およびプログラムカウンタを含む。Setjmpはプログラムの現在の状態を保存するので、longjmpは、プログラム状態を復元し、したがってプログラム実行の状態をsetjmpが呼び出されたときの状態に戻すことができる。言い換えると、longjmp()はリターンしない。むしろ、longjmpが呼び出されるとき、実行は、以前に保存されたプログラム状態により示される(setjmpにより保存されるような)特定の点にリターンし、または再開する。したがって、longjmp()は、標準的な呼出しまたはリターンの規約を使用することなく、信号ハンドラからプログラムの中の保存された実行点に制御を戻すために使用され得る。

【０２４４】

たとえば、図50への参照が行われる。例560において、ルーチンmain562はルーチンfirst563を呼び出すことができ、ルーチンfirst563はルーチンsecond564を呼び出すことができる。示されるように、main562は、ルーチンfirstを呼び出す前の点X1においてsetjmpへの呼出しを含み得る。setjmpが初めて点X1において呼び出されるとき、setjmpは0を返し、次いでルーチンfirstが呼び出される。longjmpが実行された後で、setjmpは1を返す。ルーチンsecond564は点X2におけるlongjmpへの呼出しを含み、これは、setjmpが呼び出された位置X1にあるmainへ制御が戻ることを引き起こす。ここでSetjmpが再び呼び出され、1を返すので、firstは呼び出されず、制御はNEXTに進む。

【０２４５】

スタック保護ポリシーに関連して、setjmpにより以前に保存された点X1へと実行を再開する前に、スタックをクリアするのが望ましいことがある。たとえば、上の呼出しチェーンmain-first-secondに基づいて、3つのスタックフレームがコールスタックの中に存在することがあり、longjmp呼出しとsetjmp呼出しとの間の呼出しチェーンにおける呼出し

と関連付けられるスタックメモリをクリアするために処理が実行され得る。具体的には、longjmpのコードは、この例ではfirst563およびsecond564のためのスタックフレームをクリアするコードを含み得る。ここで説明されるのは、スタック保護ポリシーに従ってそのようなスタックのクリアを実行することに関連して使用され得る技法である。

【０２４６】

プログラム状態をスタックメモリに保存するsetjmpを実行するときのスタック保護ポリシーに関連して、ある実施形態は、著名なタグ構成要素で現在のスタックポインタメモリセルをタグ付けすることができるので、後続のlongjmpに関連して、スタックがsetjmpから変化していないことをルールが確認し得る。現在のプログラム状態を示すデータは、setjmpデータ構造jmpbufに保存され得る。保存されるデータは、スタックポインタ、プログラムカウンタ、第1のポインタ(著名なタグ構成要素でタグ付けされるメモリ位置を指す(たとえば、現在のスタックポインタメモリセルを指す)ことが許容されるポインタとしてタグ付けされる)、および第2のポインタ(longjmp処理を実行する権限を与えるためにlongjmp-clearing-authority-pointerとしてタグ付けされる)を含み得る。少なくとも1つの実施形態では、longjmp-clearing-authority-pointerは、このプロシージャから再帰的に呼び出され得るプロシージャの集合の中のフレームと関連付けられるタグをクリアする権限だけを与え得る。

【０２４７】

longjmpを実行するときのスタック保護ポリシーに関連して、コードは、現在のスタックポインタがset jump構造(たとえば、setjmpデータ構造、jmpbuf)の保存されたスタックポインタより深いスタック場所を示すことを確認し得る。保存されたスタックポインタ(set jmpにより保存されるような)を格納するset jump構造のメモリセルが、(set jump構造の)タグ付けされた第1のポインタと調和するタグを有することを確認する、ルールがトリガされ得る。現在のスタックポインタと保存されたスタックポインタ(set jump構造においてset jumpにより以前に保存されたような)との間のすべてのスタックメモリ位置をクリアする、コードが実行され得る。そのようなコードは、スタックをクリアする権限を与えるlongjmp-clearing-authority-pointerとしてタグ付けされる、上で述べられた第2のポインタ(たとえば、クリアされたスタック位置を指すために使用される第2のポインタ)を使用して、クリアを実行し得る。ルールがクリアを実行するコードによりトリガされることがあり、ここでこのルールは、第2のポインタがlongjmp-clearing-authority-pointerとしてタグ付けされることを確認する。longjmpにおける命令は一意にタグ付けされるので、呼び出されたルールは、longjmp-clearing-authority-pointerとしてタグ付けされたポインタを、一意にタグ付けされた命令が使用することを可能にする。longjmpの中にない他のコードは、longjmp-clearing-authority-pointerとしてタグ付けされるポインタを使用することができない(たとえば、他のコードはlongjmp-clearing-authority-pointerの使用を許容するようにタグ付けされない)。

【０２４８】

少なくとも1つの実施形態では、命令のタグ付けは、所望の命令タグ付けおよび/またはメモリ位置タグ付けを実行するためのルールを呼び出す命令シーケンスをコンパイラに生成させることによって、実行され得る。たとえば、スタックメモリ位置のタグ付けのために、コンパイラは、ルールにスタック位置のタグを初期化またはリセットさせる、記憶命令を伴う命令シーケンスを生成し得る。命令をタグ付けするために、コンパイラは、ルールに命令をタグ付けさせる記憶命令を伴う命令シーケンスを生成することができ、ここで、命令のタグは、命令によってアクセスされるタグ付けされたメモリ位置と関連付けられる色に基づき得る。関連するスタックフレームを有する呼出しからのリターンに関連して、スタックからのフレームをクリアするコードが追加され得る。厳密なオブジェクト初期化が利用され、新しいフレームが呼出しに応答して作成されるとき、新しいフレームのオブジェクトを適切にタグ付けまたは色付けするコードが追加され得る。

【０２４９】

ここで図51～53を参照して説明されるのは、たとえば、悪意のあるコードなどにより行

われる、スタックに対して行われ得る様々な認可されないまたは意図されない変更(「スタック攻撃」はスタック変更を通じて行われる攻撃を指す)、または悪意のないコードによる意図されないスタック変更(たとえば、偶発的な上書きまたはバッファオーバーフロー)の例である。

【０２５０】

図51～図52は、サードパーティのコード(たとえば、呼び出されるライブラリルーチン)などのコードモジュールにより行われるスタック変更に関連するスタック攻撃を防ぐために行われ得る、かつ、任意の攻撃者モデルとして特徴付けられ得る、活動を示す。したがって、570および575のケースは、たとえば、認可されないまたは意図されないスタック変更を実行するコードを含む、呼び出されたサードパーティのライブラリルーチンの結果として発生し得る。加えて、スタック変更は、呼び出されたライブラリルーチンのコードによりさらに呼び出されるさらに別のルーチンによっても行われ得る。570および575の各行は、3列の情報を含む。行572a～hの各々に対して、列1は望まれないランタイム実行の挙動を示すのを防ぐための項目を特定し、列2は列1の望まれない挙動を防ぐためにとられ得る予防的な活動を特定し、列3は列2の予防的な活動を実装または実施するために使用され得る1つまたは複数の機構を特定する。一般に、列3において、各々独立に実装され、かつ具体的なシステムに応じて別々に実装され得る、代替的な機構が列挙される。たとえば、従来のシステムは第1の機構として別個のプロセスを使用し得るが、第2のシステムは代わりに能力を使用することができ、第2のシステムは代わりに特定のスタック位置の色付けまたはタグ付けを使用することができる。

【０２５１】

さらに例示すると、本明細書の他の箇所の議論と一貫して、呼出しが行われるときに実行されるプロローグコードなどのコードが、リターンアドレスおよびレジスタをスタックに書き込む。プロローグコードは、どのコードがスタック位置を変更でき、または一般的にアクセスできるかを制限するために、特別なタグでスタック位置をタグ付けするルールを呼び出し得る。たとえば、プロローグコードは、スタックフレームのメモリセルにリターンアドレス、レジスタなどを記憶するために、メモリ書込み/記憶を実行することができる。プロローグコードのそのような書込み/記憶命令は、メモリ位置を特別なものとしてマークし、どのコードがメモリセルを変更できるかを制限するために、特別なタグSTACK FRAME TAGでスタックフレームのメモリセルをタグ付けするルールを呼び出し得る。プロローグコードの書込み/記憶命令はまた、このタグ付けを実行できる命令を制限するために、PROLOG STACK TAGでタグ付けされ得る。以下は、メモリ位置を特別なものとしてマークし、どのコードがメモリセルを変更できるかを制限するために、特別なタグSTACK FRAME TAGでスタックフレームのメモリセルをタグ付けするプロローグコードの書込み/記憶命令により呼び出されるルールによって実施される論理の例である。

If(CI=PROLOG STACK TAG)AND(これがメモリ書込み動作である)then
出力またはRtag=STACK FRAME TAG
前述のルール論理において、出力タグはスタック位置に付けられるタグを指す。

【０２５２】

同様の方式で、リターンを実行することで呼び出されるエピローグコードなどの他のコードが、スタックまたはその一部をクリアすることを許容され得る。エピローグコードは、EPILOG STACK TAGという特別なタグ(たとえば、CIタグ)でタグ付けされることがあり、特別なタグSTACK FRAME TAGでタグ付けされたポインタのアクセスを通じて権限を与えられることがある。エピローグコードは、STACK FRAME TAGで特別にタグ付けされるポインタを使用して、書込み/記憶動作を使用した前述のスタックのクリアを実行することができる。スタックのクリアを実行するのをさらに制限するために、エピローグコードは上で述べられたようにタグ付けされ得る。そのような実施形態では、書込み/記憶命令はポリシーを実施するために以下の論理を実装するルールを呼び出すことができ、ここで、スタックのクリアのみが、特別にタグ付けされるポインタ(STACK FRAME TAGでタグ付けされる)を使用してエピローグコードによって実行され得る。

if(CI=EPILOG STACK TAG)AND(メモリ書込み動作)AND
(Mtag=STACK FRAME TAG)then出力またはRtag=Default tag

【０２５３】

スタックからリターンアドレスおよびレジスタを復元することが意図されるコードは、スタックのこれらの特別にタグ付けされたメモリセルに対する権限を与えられ得る。そのような権限は、たとえば、コードがスタックの特別にタグ付けされたメモリセルにアクセスすることを許容されることを示すためにコードをタグ付けすること(CI tag)、コードが権限を有することを示すためにPCをタグ付けすること、または、ポインタが特別にタグ付けされたメモリセルを指し、ポインタのタグがアクセス権限を示すように、コードによって使用されるポインタをタグ付けすることのうちのいずれかによって、与えられ得る。たとえば、STACK FRAME TAGでタグ付けされたスタックメモリセルを読み取るための権限を、読取り/ロード命令は与えられ得る。一実施形態では、読取り/ロード命令は、スタックメモリ位置から読み取ることを特別にタグ付けされたポインタ(STACK FRAME TAGでタグ付けされた)を使用した読取り/ロード命令のみに許容することによって、権限を与えられ得る。特別にタグ付けされたスタックメモリ位置(STACK FRAME TAGでタグ付けされた)から読み取ることを特別にタグ付けされたポインタ(STACK FRAME POINTERでタグ付けされた)を使用して読取り/ロード命令にのみ許容するルール論理は、次のようなものであり得る。

if(メモリ読取り動作)AND(R2tag=STACK FRAME POINTER)AND(Mtag=STACK FRAME TAG)then Rtag=DEFAULT TAG

【０２５４】

上記の変形として、読取り/ロード命令は、特別なタグSTACK FRAME TAGで読取り/ロード命令によって使用されるポインタをタグ付けすることによって権限を与えられ得る。

【０２５５】

スタックメモリ位置から読み取ることを特別にタグ付けされた命令(STACK FRAME INSTRUCTIONでタグ付けされた)を使用した読取り/ロード命令だけに許容するルール論理は、次のようなものであり得る。

if(メモリ読取り動作)AND(CItag=STACK FRAME INSTRUCTION)AND(Mtag=STACK FRAME TAG)then Rtag=DEFAULT TAG

【０２５６】

機構の例は、以下および他の箇所でより詳細に説明される。

【０２５７】

要素572aは、呼出しルーチン(呼出者)に決して返らない呼び出されるルーチン(被呼出者)の望ましくないランタイム挙動を特定する。この挙動を防ぐために、とられる行動は、呼び出されるルーチンを完了するために最長の時間が許容され得るような、各呼出しと関連付けられるタイムアウトを設けることであり得る。その最長の時間が経過した後で、呼び出されるルーチンのランタイム実行が終了する。タイムアウトを実装するための機構は、第三者のコードの呼び出されるルーチンが、タイムアウトを実施する別個のスレッドから作られるようにすること、または、時間もしくは命令が限られた呼出しを使用して呼び出されるルーチンの時間の長さを直接制限することを含み得る。

【０２５８】

要素572bは、メモリなどの利用可能なリソースを呼び出されたルーチンが使い果たし得る、リソース消耗という望ましくないランタイム挙動を特定する。この挙動を防ぐために、とられる行動は、呼び出されるルーチンに対して利用可能にされるリソースを制限することであり得る。タイムアウトを実装するための機構は、第三者のコードの呼び出されるルーチンが、最大のリソース制限を実施する別個のスレッドから作られるようにすること、または、特別な命令の限られた呼出しを使用して呼び出されるルーチンのリソースの量を直接制限することを含み得る。

【０２５９】

要素572cは、さらに別のルーチンへの予想されない呼出しを行うことなどによって予想さ

れない権限を行使する、呼び出されるルーチンの望ましくないランタイム挙動を特定する。この挙動を防ぐために、とられる行動は、呼び出されるルーチンの権限を許容可能な最低の特権に制限することであり得る。これを実装するための機構は、被呼出者または呼び出されるルーチンの権限および制御能力でPCをタグ付けすることと、呼び出されるルーチンがアクセス可能なファイルシステムまたは他のリソースの一部を制限することと、呼び出されるルーチンが行うことができる許容可能なシステム呼出しを制限することとを含み得る。

【０２６０】

要素572dは、呼び出されるルーチンにより続いて呼び出される他のルーチンによりレジスタに残された項目を読み取る、呼び出されるルーチンの望ましくないランタイム挙動を特定する(たとえば、mycodeはライブラリの中のP1を呼び出し、P1はさらにルーチンevilを呼び出し、P1はevilによりレジスタに残されたデータを読み取り得る)。この挙動を防ぐために、非入力レジスタおよび非リターンレジスタをクリアするための行動がとられ得る。これを実装するための機構は、明示的なレジスタのクリアを実行することと、呼び出されるルーチンが読み取れないように非リターンレジスタおよび非入力レジスタを含むスタックの部分を色付けすることと、呼び出されるルーチンを別個のプロセスに呼び出させることとを含み得る。

【０２６１】

要素572eは、呼び出されるルーチンにより続いて呼び出される他のルーチンによりスタックに残された項目を読み取る、呼び出されるルーチンの望ましくないランタイム挙動を特定する(たとえば、mycodeはライブラリの中のP1を呼び出し、P1はさらにルーチンevilを呼び出し、P1はevilによりスタックに残されたデータを読み取り得る)。この挙動を防ぐために、呼び出されるスタックをアクセス不可能にするための行動がとられ得る(たとえば、evilなどのさらに呼び出される他のルーチンにより使用されるスタック領域は、P1などの最初の呼び出されるルーチンがアクセス可能ではない)。これを実装するための機構は、別個のスタック(たとえば、最初の呼び出されるルーチンP1およびさらに呼び出されるルーチンevilのための)、能力(たとえば、スタックエリアを読み取ることが許容されるコードの能力または権限を制限するために、PCをタグ付けし、または特定のスタック領域にアクセスすることが許容される特別にタグ付けされたポインタを使用する)、色付け(たとえば、どのコードがアクセスできるかを制限するためにスタックのデータエリアをタグ付けする)を使用すること、および呼び出されるルーチンを別個のプロセスに呼び出させることを含み得る。

【０２６２】

要素572fは、スタックプレフィックスの中の項目を書き直す(たとえば、リターンアドレスを特定するリターンアドレスエリアを上書きする)呼び出されるルーチンの望ましくないランタイム挙動を特定する。スタックプレフィックスは、何らかの以前の呼出者にリターンすることが必要とされる情報を含むスタックのエリアであり得る。この挙動を防ぐために、呼び出されるルーチンに対してスタックプレフィックスをアクセス不可能または書込み不可能にするための行動がとられる。これを実装するための機構は、呼び出されるルーチンおよび呼び出されるルーチンを呼び出すユーザコードに別個のスタックを使用させること、能力を使用すること(たとえば、PCタグまたは特別にタグ付けされたポインタを通じて、特別にタグ付けされたコードまたは権限を与えられたコードを通じたアクセスを許容する)、色付けを使用すること(たとえば、呼び出されるルーチンがアクセスすることを許容されないように特別なタグでスタックプレフィックスのデータアイテムをタグ付けすること)、および呼び出されるルーチンを別個のプロセスに呼び出させることを含み得る。

【０２６３】

要素572gは、スタックプレフィックスの中のデータを読み取る、呼び出されるルーチンの望ましくないランタイム挙動を特定する。この挙動を防ぐために、572fで説明されたものと同様の機構を使用して、呼び出されるルーチンに対してスタックプレフィックスをアクセス不可能にするための行動がとられる。

【０２６４】
要素572hは、ポインタがスタックプレフィックスに記憶されるリターンアドレスへのポインタを上書きすることなどによって、スタックプレフィックスの中の制御フローをリダイレクトする、呼び出されるルーチンの望ましくないランタイム挙動を特定する。この挙動を防ぐために、スタックプレフィックスに記憶されるリターンアドレスを保護するために行動がとられ得る。一態様では、要素572hが572hの特定のインスタンスを特定するので、572hの機構は572fの機構と同様である。これを実装するための機構は、呼び出されるルーチンおよび呼び出されるルーチンを呼び出すユーザコードに別個のスタックを使用させること、能力を使用すること(たとえば、特別にタグ付けされたコードを通じて、または、PCタグもしくはアクセス権限によりタグ付けされる特別にタグ付けされたリターンポインタを通じて権限を与えられるコードを通じて、アクセスを許容する)、色付けを使用すること(たとえば、呼び出されるルーチンがアクセスすることを許容されないように、リターンアドレスを含むスタックプレフィックスのメモリ位置を特別なタグでタグ付けすること)、および呼び出されるルーチンを別個のプロセスに呼び出させることを含み得る。
【０２６５】
図53は、任意の入力攻撃者モデルに関連したスタック攻撃を防ぐためにとられ得る行動を示す。
【０２６６】
要素581aは、実行中のルーチンの現在のフレームの中の意図されない項目を書き直すコードを実行する望ましくないランタイム挙動を特定する。この挙動を防ぐために、とられる行動はオブジェクトの整合性を維持することであり得る。これを実装するための機構は、オブジェクトごとに能力を使用すること(たとえば、特別にタグ付けされたコードを用いて、または、PCタグもしくはアクセス権限によりタグ付けされる特別にタグ付けされたリターンポインタを通じて権限を与えられるコードを用いて与えられる能力を通じて、アクセスを許容する)、またはオブジェクトごとに色を使用すること(たとえば、オブジェクトのメモリ位置をタグ付けすること)を含み得る。
【０２６７】
要素581bは、実行中のルーチンの現在のフレームの中の項目を読み取る、望ましくないランタイム挙動を特定する。この挙動を防ぐために、とられる行動はオブジェクトの整合性を維持することであり得る。これを実装するための機構は、オブジェクトごとに能力を使用すること(たとえば、特別にタグ付けされたコードを用いて、または、PCタグもしくはアクセス権限によりタグ付けされる特別にタグ付けされたリターンポインタを通じて権限を与えられるコードを用いて与えられる能力を通じて、アクセスを許容する)、またはオブジェクトごとに色を使用すること(たとえば、オブジェクト固有のタグでオブジェクトのメモリ位置をタグ付けすること)を含み得る。
【０２６８】
要素581cは、(たとえば、実行中のコードを呼び出した他のルーチンの)前身のフレームの中の意図されない項目を書き直す(現在のフレームを有する)実行中のコードの望ましくないランタイム挙動を特定する。この挙動を防ぐために、とられる行動はスタックフレームを隔離または分離することであり得る。これを実装するための機構は、フレームごとに能力を使用すること(たとえば、特別にタグ付けされたコードを用いて、または、PCタグもしくはアクセス権限によりタグ付けされる特別にタグ付けされたリターンポインタを通じて権限を与えられるコードを用いて与えられる能力を通じて、アクセスを許容する)、またはフレームごとに色を使用すること(たとえば、フレームのメモリ位置をフレーム固有のタグでタグ付けすること)を含み得る。
【０２６９】
要素581dは、(たとえば、実行中のコードを呼び出した他のルーチンの)前身のフレームの中の項目を読み取る、(現在のフレームを有する)実行中のコードの望ましくないランタイム挙動を特定する。この挙動を防ぐために、とられる行動は、スタックフレームを隔離または分離することであり得る。これを実装するための機構は、要素581cについて説明され

たように、フレームごとに能力を使用することまたはフレームごとに色を使用することを含み得る。

【０２７０】

要素581eは、現在実行中のコードにより呼び出される別のルーチンによりスタックの残りの項目を読み取る、(現在のフレームを有する)実行中のコードの望ましくないランタイム挙動を特定する。予防的な行動は、呼び出されるルーチンの呼び出されるスタックを、現在実行中のコードに対してアクセス不可能にすることである。これを実装するための機構は、572gに関連して説明されたのと同様の方式で、別個のプロセス、別個のスタック、能力、および色付けを使用することを含み得る。

【０２７１】

要素581fは、リターンポインタ(たとえば、実行中のコードを呼び出したルーチンの中のリターンアドレスを含むスタックの中の位置)を変更する、(現在のフレームを有する)実行中のコードの望ましくないランタイム挙動を特定する。予防的な行動は、リターンポインタ、またはリターンアドレスを含むスタックの中の位置を保護することである。これを実装するための機構は、572gに関連して説明されたのと同様の方式で、能力および色付けを使用することを含み得る。

【０２７２】

本明細書の技法に従ったある実施形態は、ルールの新しい集合を学習および検証するために、別のハイブリッドシステムの一部として、PUMPルールメタデータ処理システムを使用し得る。たとえば、PUMPルールメタデータ処理システムは、許容される制御フローを(たとえば、ログをとることを通じて)学習し、したがって、実行中のプログラムについてのルールおよび許容される有効な制御の移転を判定するために、使用され得る。ルールおよび許容される有効な制御の移転は次いで、実行されたプログラムのために実施されるCFIポリシーの有効な制御の移転のルールおよび集合として使用され得る。

【０２７３】

プログラムのCFIポリシーのためのルールおよび制御移転を学習することをさらに例示すると、第1の訓練または学習段階が実行され得る。この第1の段階において、プログラムは、すべての制御点(たとえば、分岐または移転のソースおよびターゲット)がタグ付けされた状態で、かつ、制御移転命令に対してルールがないCFIポリシーの訓練バージョンとともに、実行される。したがって、分岐またはジャンプ命令などの制御の移転があるたびに、PUMPルールメタデータシステムのキャッシュミスハンドラへの制御の移転を引き起こす、PUMPルールキャッシュミスがある。キャッシュミスハンドラは、制御の移転に関する情報のログをとるために処理を実行し得る。ログをとられる情報は、たとえば、移転のソース位置および移転のターゲット位置を含み得る。他の情報は、たとえば、移転がそこから行われる(たとえば、ソース位置を含む)呼出しプロシージャまたはルーチン、および制御が移転される(たとえば、ターゲット位置を含む)呼び出されるプロシージャまたはルーチンを含み得る。より具体的には、学習または訓練段階において、最初に制御の特定の移転が発生するとき、キャッシュミスハンドラは、ソースからターゲットへの制御のその特定の移転のためのルールの学習された集合の新しいルールを計算する。同じソースから同じターゲットへの制御の後続のランタイム移転は、この計算されたルールを使用する。この方式では、プログラムにバグがなく、非攻撃プログラムである(悪意のあるコードではない)と推定され、すべての制御パスがプログラム実行の間に行われる場合、プログラム実行の終わりに学習されたルール集合によって示されるような、制御移転のログをとられた集合が、この特定のプログラムのためのすべての有効なまたは許容可能な制御の移転を表す。したがって、ルールの学習された集合は、プログラムのためのCFIポリシーに対する初期のまたは最初のルールの集合を示し得る。

【０２７４】

プログラムのためのCFIポリシーを示すルールの学習された集合を検証するために、処理が実行され得る。この検証は、これらのルールのいずれもが無効な制御の移転を許容しないことを確実にすることを含み得る。ルールの学習された集合の検証は、任意の適切な方

式で実行され得る。たとえば、ある実施形態は、各ルールを検証する分析ツールを実行し得る。このツールは、許容される移転に各ルールが対応することを検証するために、たとえば、バイナリまたはオブジェクトコード、シンボルテーブル、および元のソースコードなどを調査し得る。さらに例示すると、検証は、すべての制御点(たとえば、分岐または移転のソースおよびターゲット)がタグ付けされているバイナリコードを調査し得る。この方式では、タグ付けされたバイナリまたはソースコードは、すべての可能性のあるソースおよびターゲットの位置の有効な集合を示し、それにより、制御のランタイム移転において実際に使用され得る可能性のあるソースおよびターゲットの有効な集合を与える。ログをとられる制御の任意のランタイム移転は、ソースとターゲットの各々が有効な集合に含まれるようなソースからターゲットへのみ発生すべきである。たとえば、タグ付けされたバイナリまたはソースコードは、位置A1、A2、A3、およびA4を含み得る。制御の任意のログをとられる移転は、A1、A2、A3、またはA4であるソースと、A1、A2、A3、またはA4であるターゲットとを含むべきである。第1のルールにより示されるログをとられるランタイムの制御移転がA1からB7である場合、第1のルールは無効とされることがあり、それは、B7が制御移転のターゲットであるべきではないからである(たとえば、B7はA1、A2、A3、およびA4と矛盾せずにタグ付けされる、静的に判定された潜在的な制御点の集合に含まれない)。一態様では、ルールの学習された集合は、検証処理の結果としてルール除去を介してさらに減らされ得る、ルールの候補集合として特徴付けられ得る。

【０２７５】

検証されたプログラムのためのCFIポリシーについてのルールの初期のまたは学習された集合のすべてのルールが次いで、プログラムのために次いで実施されるCFIポリシーに含まれるルールの検証済の集合として使用され得る。

【０２７６】

図54を参照すると、ポリシールールを学習し、検証し、使用するための、本明細書の技法に従ったある実施形態によって実行され得るような、すぐ上で説明された例示的な要約処理が示されている。602において、プログラムは最初、実質的にCFIポリシールールなしで実行され得るので、制御の各々の新しい移転がルールキャッシュミスを引き起こし、ランタイムに遭遇する制御の移転に関する新しいルールをキャッシュミスハンドラに生成させる。新しいルールは、ソースおよびターゲットからの制御の移転を特定することができ、学習されたルール604の第1の集合に含まれることがある。プログラム実行の終わりに、学習されたルール604の第1の集合は、ランタイムに発生した制御の各々の異なる移転のためのルールを含む。学習されたルールの第1の集合は次いで、有効な制御の移転を各ルールが表すことを確実にするために、606の処理において検証され得る。606の処理は、自動化されたルール検証について上で説明されたようなツールを使用することがあり、他の処理も含むことがある。たとえば、606の検証処理は、制御の移転が有効であるというさらなる確認のために、ツールにより検証されたルールをユーザに提示することを含み得る。検証ルール608の第2の集合は、ルール検証処理606の結果として生成され得る。続いて、検証ルール608の第2の集合は、610において第2の時点でプログラムを実行するときに実施されるCFIポリシーとしてPUMPシステムにより使用され得る。

【０２７７】

したがって、602における前述の第1のプログラムの実行は、プログラムに対する有効な制御の移転の集合を判定するために使用され得る。しかしながら、この単一のプログラム実行がすべての制御パスを用い、それにより、608において有効であると特定された制御の移転がすべての潜在的な有効な制御の移転よりも少ないものを示し得ると仮定するのは、合理的ではないことがある。この場合、処理は、CFIポリシールールの検証された集合を使用して610に関連して上で説明されたように実行され得る。ランタイムの間、ルールキャッシュミスを引き起こす制御の移転に遭遇する場合(たとえば、608におけるルールを有しない予期されない制御の移転を示す)、たとえば、上で説明されたような制御の移転を検証するために(たとえば、バイナリコードにおいてタグ付けされる、またはソースプログラムにおいてアノテートされる、潜在的な制御点の集合を使用して)、追加の確認がランタ

イムに実行され得る。制御の移転が無効であると判定される場合、フォルトまたは例外がトリガされ得る。

【０２７８】

ある代替形態として、ルールキャッシュミスを引き起こす制御の移転に遭遇し、それにより予想されないランタイムの制御の移転を示す場合、キャッシュミスハンドラは、その後の検証のために予想されない移転ルールを記録し、また、予想されない制御の移転が有効な追加のまたは異なるポリシーとともに進行することを許容し得る。たとえば、未検証の制御の移転に対して、移転は信頼されないものと見なされ得るので、ポリシーは、未検証の制御の移転の信頼されない性質により、より高いレベルの保護を反映するように変更され得る。たとえば、予想されない移転は、ライブラリルーチンに制御を移転し得る。ライブラリルーチンは、予想されない移転の前に有効であったポリシーよりも高いレベルの保護および低い信用を反映するポリシーを使用して実行され得る。検証されない制御の移転に対して、第1のスタック保護ポリシーは、予想されない制御の移転の前の第1の時点において有効であることがあり、第2のスタック保護ポリシーは、予想されない制御の移転の後に有効であることがある。第1のスタック保護ポリシーは、静的プロシージャ権限を実施し得る。第1の保護ポリシーは、オブジェクト保護モデルについて本明細書の他の箇所で説明されるようなオブジェクトレベルにおけるどのような色付けも含まないことがある。予想されない制御の移転の後で、有効な第2のスタック保護ポリシーは、厳密なオブジェクト色付けについて本明細書の他の箇所で説明されるオブジェクト保護モデルに従ってスタック保護を提供し得る。したがって、予想されない制御の移転に遭遇すると実行されるコードは、より厳格で細かい粒度のスタック保護を提供する、より制約的な第2のスタック保護ポリシーを利用し得る。加えて、プログラム実行は、予想されない制御の移転が発生すると、より低いレベルの優先度で継続し得る。

【０２７９】

図55および図56を参照すると、上で説明されたプログラムに対するCFIポリシーのルールなどの、検証されたルールの集合を使用して本明細書の技法に従ったある実施形態において実行され得る処理ステップの、フローチャート620、630が示されている。フローチャート620は、実行中のプログラムのCFIポリシーの中にルールを有しない、予想されない制御の移転に関連して実行され得る処理ステップの第1の集合を説明する。フローチャート631は、実行中のプログラムのCFIポリシーの中にルールを有しない、予想されない制御の移転に関連して実行され得る処理ステップの第2の集合を説明する。

【０２８０】

フローチャート620を参照すると、ステップ622において、プログラムが検証されたルールの集合を使用して実行され得る。プログラム実行の間のステップ624において、ランタイムの制御の移転が実行される。626において、ルールキャッシュミスがあり、それにより移転が予想されないことを示すかどうかが判定される。具体的には、ランタイムの制御の移転に対する検証されたルールの第2の集合の中にルールがある場合、制御の移転は、ステップ626においてnoであると評価される場合に予想され、処理はステップ628へと続き、そこで制御の移転が実行されプログラムが実行を継続する。

【０２８１】

ステップ626においてyesであると評価される場合(たとえば、予想されない制御の移転を示すキャッシュミス)、プロセスはステップ632へと続き、そこでランタイムの検証処理が予想されない制御の移転のために実行される。具体的には、ミスハンドラは、予想されない移転を検証することを試みる処理を実行し得る。ルール検証処理の例は、タグ付けされたバイナリコード、元のソースプログラム、およびシンボルテーブルなどを使用して判定され得る、上で説明されたような潜在的な制御移転ポイントの集合に、ランタイムのソースおよびターゲットの位置が含まれるかどうかを判定することを含み得る。ステップ634において、ステップ632の検証処理が、予想されない制御の移転が有効であると判定したかどうかが判定される。ステップ634においてyesであると評価される場合、処理はステップ636へと続き、そこで、プログラムのためのCFIポリシーとして使用される第2の集合

に新しいルールが追加され、プログラムの処理が続く。ステップ634においてnoであると評価される場合、プログラム実行は、たとえばトラップを引き起こすことにより終了され得る。

【０２８２】

フローチャート631を参照すると、ステップ622、624、626、および628は、フローチャート620に関連して上で説明されたようなものである。ステップ626においてyesであると評価される場合、制御はステップ639に進み、そこで、予想されない制御の移転が後の検証のために記録され得る(たとえば、予想されない制御の移転のためのルール候補が記録される)。ステップ639において、プログラムは、制御の移転が予想されないときであっても実行を継続することが許容される。しかしながら、ステップ639において、プログラム実行は、たとえば、上で述べられたような、1つまたは複数の制約的なポリシーの集合、低減された実行優先度などを使用して、継続する。

【０２８３】

上で説明されたものなどの、上で説明された処理は、テイント追跡などの他のポリシーに関連して同様に実行され得る。たとえば、テイント追跡のために、第1の学習または訓練段階が、キャッシュミスハンドラに各キャッシュミスの「ログをとらせる」ことによって、プログラム実行を介してポリシーのルールを学習するために実行され得る。本明細書で説明されるように、テイント追跡は、データを産生し、またはデータにアクセスするコードに基づいて(たとえば、本明細書の他の箇所で説明されるようなCIを使用するなどして)、データをタグ付けすることを含み得る。コードまたはソースに基づいてデータをテイントする1つの理由は、プログラムが適切に格納され、不要なまたは不適切なデータアクセスを実行しないことを確実にするためである。たとえば、JPEGデコーダによりテイントされたデータが決してパスワードデータベースに流入しないこと、または、クレジットカードデータ、社会保障番号、もしくは他の個人情報が1つまたは複数の制約されたアプリケーションの特定の集合のみによってアクセスされることを確実にするために、ルールが使用され得る。テイント追跡ポリシーを判定すると、テストデータに対して、テイント追跡ルールが実行されない状態で、学習または訓練段階のために処理が実行されることがあり、このことは、キャッシュハンドラがデータの特定の流れ(たとえば、プログラムのどのルーチンがどのデータにアクセスするか、どのユーザ入力がどのデータベースに書き込まれるかなど)を初めて見るときにキャッシュハンドラミスを引き起こし、ルールを記録する。CFIポリシーについて上で説明されたのと同様の方式で、第1の学習段階のテストランの終わりには、動作の間にプログラムを保護するために適用すべき学習されたルールの集合がある。ルールの学習された集合の検証処理はまた、ルールのCFI学習された集合について上で述べられたようなツールまたは他の適切な手段を使用して実行され得る。テイント追跡のためのそのような検証処理は、各データフローまたはアクセスが適切であることを確実にすることを含み得る。

【０２８４】

また、フローチャート620および631に関連して説明されたのと同様の方式で、ルールの検証された集合はPUMPシステムとともに使用されることがあり、ここで、キャッシュミスハンドラは、検証された集合の中に対応するルールを有しない任意のデータアクセスのための処理を扱う。フローチャート620の処理と同様に、キャッシュミスハンドラは次いで、ランタイム検証処理(たとえば、ステップ632と同様)も実行して、データアクセスまたはデータフローのためのルール候補が有効であるかどうかを判定し、プログラム実行が継続することを許容する(たとえば、ステップ634、636と同様に)か、許容しない(たとえば、ステップ638と同様)ことがある。代わりに、フローチャート631の処理と同様に、キャッシュミスハンドラは、オフラインで(たとえば、ランタイムの間以外に)検証され得る予想されないデータアクセスまたはデータフローのためのルール候補を記録し、たとえば、より制約的なポリシー、低減された優先度など(たとえば、ステップ639と同様)を使用してプログラム実行を継続し得る。

【０２８５】

上の例は全般にバイナリの学習プロセスを説明する。本明細書の技法に従ったある実施形態はさらに、あるイベント(たとえば、制御移転またはデータアクセス)を許容するかどうかについての判断を行う際に統計の使用をサポートし得る。少なくとも1つの実施形態では、プログラム実行の間の各ルールの使用の数をカウントするために、カウンタが各ルールに追加され得る。ルールがPUMPキャッシュから削除されるとき、処理は、累積のルール使用量を、ルール使用量に関する追加の統計を提供するために使用され得るグローバルなソフトウェアカウントに追加し得る。このカウントはまた、何かが限られた回数発生することを許容するために使用され得る。たとえば、ソースからターゲットへのデータのフローを追跡するテイント追跡ルールに関連して、限られた閾値の量のデータが、ソースとターゲットとの間の予想されないデータフローに対して許容され得る(たとえば、特定のプログラムによって特定のデータベースから読み取られるXのデータ量)。閾値の量が転送されると、対応するルール候補の検証が成功するまで、ソースとターゲットとの間で追加のデータは転送され得ない。閾値の量の限られた使用事例では、PUMPシステム(たとえば、ミスハンドラ)は、ルールを欠いた命令が何らかの限られた回数発生することを許容し得る。閾値に適用されるような集約またはカウントは、異なる方法で行われ得る。たとえば、予想されない制御の移転を考える。集約されない場合、キャッシュミスハンドラは、検証されたルールを伴わない同じ予想されない制御の移転が5回よりも多く発生することを許容しないことがある。プログラムのためのすべての予想されない制御の移転などにわたって集約される場合、プログラムは、100回という最大の数の予想されない制御の移転を行うことが許容され得る。これは、たとえば、予想されない制御の移転または予想されないデータアクセスの単一のインスタンスが発生するのが受け入れ可能である場合に、有用であり得る。たとえば、ある特定のソースからのデータを調べるための単一のクエリは許容され得る。しかしながら、閾値の数を上回るクエリがデータソース(たとえば、特定のデータベース)に実行される場合、プログラムはフラグを付けられ、または停止されるべきである。

【０２８６】

より一般的な統計の事例が、普通の挙動の範囲を学習するために使用され得る。たとえば、ポリシーの異なるルールの相対的な使用量(たとえば、各ルールの使用量の比)を判定するために、プログラムが学習段階において実行され得る。たとえば、ランタイム制御移転のために呼び出される各ルールの相対的な使用量が記録され得る。理想的には、そのような実行は、何が平均的なまたは普通のプログラムの挙動であると見なされ得るかを学習するために、多数の異なるデータ集合を使用してプログラムのために行われ得る。ルールの学習および検証は次いで、(上で説明されたように)検証された制御の移転のためのルールの集合と、加えて、各々の検証されたルールの相対的な使用量を示す比とをもたらし得る。検証されたルールと、関連する使用量の比との両方が、実施されるポリシールールとして後続の処理の間に使用され得る。ポリシーが実施される後続のプログラム実行の間、PUMPシステムは、現在のルール使用量が予想される比から離れているかどうかを確認し得る。ある実施形態は、たとえば、ルールのための範囲または最大の予想される使用量を含むことがあり、ここで、最大よりも多くルールを呼び出す制御の移転はフラグを付けられ得る。たとえば、最大よりも多く特定の制御移転ルールを呼び出すプログラムは、さらなる検査または分析のためにフラグを付けられ得る。この機構を使用して、プログラムランタイムの挙動は、ネットワークの挙動がファイアウォールのルールを生成するために監視されるのと同様に監視され得る。ルール使用量を捉えるために統計的な学習アルゴリズムが使用されることがあり、場合によっては、普通の場合と攻撃の挙動を学習するために、メインメモリトラフィックおよびキャッシュミスレートのような他の標準的なランタイム特性が使用され得る。上で説明されたような使用制限の閾値を適用する実施形態では、異常である、または別様に信頼されないものと見なされ得る他のランタイム挙動をプログラムが示す場合、使用限度は大きく減らされることがあり、またはそうでなければ0に設定されることがある。代わりに、プログラムが普通のランタイム挙動を示す、または別様に信頼されるものと見なされ得る場合、使用限度は、信頼されないシナリオと比較してはるか

に高く設定され、または上げられることがある。
【０２８７】
上の技法は、コンパイラにどのような追加の情報も出力させることなく、CFIポリシーのルールにおいて反映される有効な制御の移転などの、ポリシーの有効なルールの集合を判定するために使用され得る。したがって、本明細書の技法に従ったある実施形態は、学習段階のために使用されるバージョンおよび後続の実施のために使用される別のバージョンという、各ポリシーの2つのバージョンを有し得る。学習段階は、テイント追跡のための許容可能なデータアクセスまたはフローを発見すること、CFIポリシーのための制御移転を発見することなどのために、自動化された診断モードとして使用され得る。
【０２８８】
ここで説明されるのは、RISC-Vプロセッサを使用した本明細書の技法に従った実施形態において使用され得るアーキテクチャの例である。加えて、以下で説明されるのは、プロセッサによって使用されるタグ付けされていないデータソースとタグ付けされたデータソースとの間で、プロセッサに基づいて仲介されるデータ転送を実行することに関連して使用され得る、技法である。そのような技法は、プロセッサによる使用のためにシステムに持ち込まれ得る外部の信頼されないデータをタグ付けすることと、また、システムの外部で使用するためのタグ付けされていないデータを生成するためにシステム内で使用されるタグ付けされたデータからタグを除去することとを行う。
【０２８９】
図57を参照すると、タグ付けされたデータとタグ付けされていないデータとの間を仲介するための、本明細書の技法に従ったある実施形態において使用され得る構成要素の例が示されている。例700は、RISC-V CPU702と、PUMP704と、L1データキャッシュ706と、L1命令キャッシュ708と、タグ付けされたデータ転送のためにシステム内で内部的に使用されるインターコネクトファブリック710と、ブートROM712aと、DRAMコントローラ(ctrl)712bと、タグ付けされたデータを記憶する外部DRAM712cとを含む。ハードウェア構成要素である追加タグ714aおよび検証ドロップタグ714b、プロセッサ702による使用のためにタグ付けされていないメモリ716から外部のタグ付けされていないデータを転送しタグ付けされていないデータをタグ付けされていないメモリ716へと転送するために使用されるインターコネクトファブリック715も、含まれている。タグ付けされていないメモリ716以外の、外部のタグ付けされていないデータの他のソース701が、タグ付けされていないファブリック715に接続され得ることに留意されたい。たとえば、要素701は、フラッシュメモリに記憶されているタグ付けされていないデータ、ネットワークからアクセス可能なタグ付けされていないデータなどを含み得る。DRAMコントローラ(ctrl)712bは、DRAM712cからデータを読み取り、それにデータを書き込むために使用されるコントローラである。ブートROM712aは、システムをブートするときに使用されるブートコードを含み得る。
【０２９０】
例700は、別個のタグ付けされたファブリック710およびタグ付けされていないファブリックを、それらの2つの間でデータを移動するために使用されるプロセッサ702とともに示す。追加タグ714aは、タグ付けされていないデータを入力として用い、データが公開であり(本明細書で説明されるシステムの外部で使用され得る)信頼されない(ソースが未知であり得るので、またはそうでなければ既知の信頼されるソースからのものではないので)ことを示すタグで入力をタグ付けする。少なくとも1つの実施形態では、タグ付けされていないメモリ716のタグ付けされていないデータは714aによって受信され得る。716からの受信されるタグ付けされていないデータは暗号化されることがあり、これにより、追加タグ714aは単に、信頼されていないタグを受信された暗号化されたデータに追加する。受信されたデータは、公開鍵と秘密鍵のペアを使用した公開鍵暗号などの非対称暗号、または当技術分野において知られている他の適切な暗号化技法を使用して、暗号化され得る。受信されたデータは暗号化された形式で記憶され得る。当技術分野において知られているように、所有者の公開鍵と秘密鍵のペアについて、秘密鍵は所有者だけに知られているが、

公開鍵は公にされ他者により使用される。第三者が、所有者の公開鍵を使用して、所有者に送られる情報を暗号化し得る。所有者は次いで、秘密鍵(誰とも共有されない)を使用して、受信された暗号化された情報を復号し得る。同様の方式で、所有者は、自身の秘密鍵を使用して情報を暗号化することができ、ここで、暗号化された情報は、所有者の公開鍵を使用して暗号化された情報を復号する第三者に送られる。

【０２９１】

検証ドロップタグ714bは、タグ付けされた暗号化されたデータを受信し、タグを除去することができ、これにより、タグ付けされていないメモリ716にエクスポートされているタグ付けされていない暗号化されたデータをもたらす。メモリ716に記憶されているそのようなタグ付けされていない暗号化されたデータは、たとえば、本明細書で説明されるPUMPを使用して実行することとして、タグおよび関連するメタデータルール処理を使用しない別のシステムおよびプロセッサ上で使用され得る。

【０２９２】

少なくとも1つの実施形態では、714aにおいて受信されるタグ付けされていないデータは、上で述べられたように暗号化されることがあり、また、データの整合性をもたらすために署名され得る。さらに、署名は、認証およびデータの整合性を保証する(たとえば、署名された元の送信者により送信されてから変更されておらず、データがデータに署名する送信者によって送信されたことを保証する)ために、受信されたデータアイテムを検証する際に使用され得る。たとえば、所有者は、ハッシュ値または「ダイジェスト」を生み出すためにメッセージをハッシュし、次いで、デジタル署名を生み出すために所有者の秘密鍵でダイジェストを暗号化し得る。所有者は、メッセージおよび署名を第三者に送信し得る。第三者は、署名を使用して受信されたデータを検証し得る。まず、第三者は、所有者の公開鍵を使用してメッセージを復号し得る。署名は、復号されたメッセージのハッシュまたはダイジェストを計算し、所有者/署名者の公開鍵で署名を復号して予想されるダイジェストまたはハッシュを取得すること、および、計算されたダイジェストを復号された予想されるダイジェストまたはハッシュと比較することによって、検証され得る。一致するダイジェストは、メッセージが所有者により署名されてから変更されていないことを裏付ける。

【０２９３】

動作において、ロード命令などの命令はタグ付けされていないメモリ716に記憶されているデータを参照することができ、このデータは次いで、命令実行において使用するためにデータキャッシュ706へと転送される。そのようなロード命令に対して、データは、タグ付けされた(信頼されておらず公開であるとタグ付けされた)データを出力する714aによる処理のために、715を通じて716から転送され得る。714aによって出力されるタグ付けされたデータは、処理のためにL1データキャッシュ706に記憶される。同様の方式で、記憶命令は、データキャッシュ706からタグ付けされていないメモリ716の中の位置にデータを記憶し得る。そのような記憶命令に対して、タグ付けされていないデータを出力するドロップタグ714bを検証するために、データが710を通じて706から転送され得る。714bにより出力されるタグ付けされていないデータは次いで、716における記憶のために715を通じて送信される。

【０２９４】

コードは、たとえばDRAM712cへの記憶のために、716からのタグ付けされていないデータをシステムへインポートするために、プロセッサ702上で実行され得る。以下は、タグ付けされていないデータをインポートするコードの論理を示し得る。

1．追加タグ714aにより出力されるタグ付けされたデータは信頼されないバッファに記憶され得る(公開であり信頼されないものとしてタグ付けされる)。

2．信頼されないバッファに記憶されているタグ付けされたデータを復号して復号バッファに記憶する。したがって、復号バッファは公開であり信頼されていないものとしてタグ付けされる復号されたデータを含む。

3．有効な汚染されていないデータを復号バッファが含むことを確実にするために検証処理を実行する。そのような検証処理は、本明細書の他の箇所で説明され当技術分野におい

て知られているようなデジタル署名を使用し得る。
4．復号バッファが検証データを含む場合、信頼されるコードの第2の部分が、公開であり信頼されていないものとしてタグ付けされる復号バッファのデータを信頼されるものとしてタグ付けされるデータに変換するために実行され得る。信頼されるコード部分は、実行されると、信頼され公開であるものとして復号バッファのデータを再タグ付けするためにルールを呼び出す、1つまたは複数の命令を含み得る。信頼されており公開であるものとしてここでタグ付けされる、再タグ付けされたデータは、外部DRAM712cの中に位置する信頼されるバッファに記憶され得る。
【０２９５】
信頼されるコードは、実行されると、参照されるメモリ位置を再タグ付けするルールを呼び出す権限をそのコードに与える特別な命令タグでタグ付けされた、メモリ命令を含み得る。たとえば、信頼されるコードは、公開であり信頼されていないものとしてタグ付けされているデータ(信頼されないバッファ)を、公開であり信頼されているという新しいタグとともに宛先メモリ位置(信頼されるバッファ)に記憶する、特別にタグ付けされた記憶命令を含み得る。前述の信用の記憶命令は、たとえばローダによって特別にタグ付けされ得る。
【０２９６】
以下は、「公開であり信頼されていない」から、「公開であり信頼されている」へとデータを再タグ付けする、信用コードの論理を示し得る。

```
for i=1 to N
temp=*untrusted buffer[i]
trusted buffer[i]=temp
```

ここで、Nは信頼されないバッファの長さであり、tempは実行される再タグ付けのために使用される一時バッファである。第1の命令であるtemp=*untrusted buffer[i]は、タグ付けされていないメモリ716からの信頼されないバッファの第1の要素をtempへとロードする、ロード命令をもたらし得る。第2の命令であるtrusted buffer[i]=tempは、公開であり信頼されていないものとしてtempにおいてタグ付けされるデータを、公開であり信頼されるという新しいタグとともにtrusted buffer[i]に記憶する、記憶命令であり得る。したがって、第2の命令は、「信頼されていない」から「信頼される」へとデータを再タグ付けすることを実行する権限を有するように、上で述べられたように特別にタグ付けされた命令である。
【０２９７】
同様の方式で、712cのタグ付けされたデータがタグ付けされていないメモリ716(または任意のタグ付けされていないメモリソース701)にエクスポートまたは記憶されているとき、データアイテムを暗号化して署名を生成するコードがプロセッサ702により実行され、暗号化されるデータアイテムおよび署名は714bに送信されることがあり、そこで、716における記憶のために715を通じた送信の前にタグが除去される。
【０２９８】
例700の変形として、メモリ716および712cは統合されることがあり、また、インターコネクトファブリック710および715は統合されることがある。そのような実施形態では、タグ付けされていないメモリソース701がアクセスすることを許容されるアドレス範囲は限られていることがある。たとえば、図58の例720に対して参照が行われる。例720は、700において番号を付けられたものと同様の構成要素を含み、違いは、コンポーネント714a～b、715、および716が削除され、メモリ712cが信頼されておらず公開のタグ付けされたデータを記憶するために使用されるメモリ712cの領域を示す部分U722を含むという点である。信頼されていないDMAおよびI/Oサブシステムなどの、タグ付けされていないメモリソース701は、メモリ722の下位16MBまたは256MBを使用することに制限され得る。一実施形態では、U722に記憶されるデータは、明示的にタグ付けされないことがあり、むしろ、この限られた範囲の中のアドレスを有するUに記憶されているすべてのデータが、公開であり信頼されないものとして暗黙的にタグ付けされ扱われ得る。変形として

、ある実施形態は、信頼されていない公開のデータを示す恒久的なタグで部分U722を事前にタグ付けすることができ、上述の関連する恒久的なメタデータを変更することはできない。ルールは、プロセッサが他のデータを領域U722に記憶するのを防ぎ得る。たとえば701に含まれるDMAによって実行される信頼されていないDMA動作は、領域U722へと書き込むことに限定され得る。

【０２９９】

移植されないI/O処理コードを実行する必要がある実施形態は、構成要素の信頼されない側の専用のI/Oプロセッサで実行され得る。たとえば、図59の例730に対する参照が行われる。例730は700において番号を付けられたものと同様の構成要素を含み、違いは構成要素732、732a、および732bが追加されているという点である。要素732は、PUMPおよびメタデータルール処理なしで実行する、追加のRISC-Vプロセッサである。要素732aは第2のプロセッサ732のためのデータキャッシュを示し、要素732bは第2のプロセッサ732のための命令キャッシュを示す。データキャッシュ732は、タグ付けされていないインターコネクトファブリック715に接続され得る。

【０３００】

本明細書の他の箇所でより詳細に説明されるように、タグ付けされていないデータソース(たとえば、701、716)と、プロセッサ702により使用されるタグ付けされたメモリ712cとの間を仲介するための別の代替形態として、別個のI/O PUMPが使用され得る。

【０３０１】

図60を参照すると、タグ付けされていないデータソース(たとえば、701、716)と、プロセッサ702により使用されるタグ付けされたメモリ712cとの間を仲介するための、本明細書の技法に関連して使用される本明細書のシステムに含まれ得る構成要素の別の実施形態が示されている。例740は例700と同様の構成要素を含み、違いは構成要素714a～bが除去されインターン742およびエクスターン744で置き換えられているという点である。この実施形態では、インターン742およびエクスターン744は、上で説明された処理を実行するハードウェア構成要素であり得る。具体的には、インターン742は、受信されたタグ付けされていないデータを処理し、信頼されており公開であるものとしてタグ付けされた検証されたデータアイテムを出力する、ハードウェアを含み得る。信頼されており公開のタグ付けされたデータアイテムは、実行中の命令に関連してプロセッサ702により使用されるデータキャッシュ706に記憶するために、ファブリック710を通じて伝えられ得る。インターン742は、タグ付けされていない暗号化されたデータの検証処理を実行し、検証が成功したとすると、受信されたタグ付けされていないデータを信頼され公開であるものとしてさらにタグ付けする、ハードウェアを含み得る。エクスターン744は、タグ付けされた暗号化されていないデータを処理し、署名された暗号化されたデータアイテムを出力する、ハードウェアを含み得る。エクスターンは、本明細書で説明されるようなメタデータルール処理を実行しない別のプロセッサ上で、署名された暗号化されたデータアイテムが使用されることになる場合、暗号化の前にタグを除去し得る。

【０３０２】

最も簡単な場合、インターン742およびエクスターン744のハードウェアは単一の公開鍵と秘密鍵の集合をホストすることがあり、ここで、署名および暗号化も単一の鍵の集合を使用して実行される。鍵の集合は、742および744により使用されるハードウェアにおいて符号化され得る。さらなる変形では、インターン742およびエクスターン744のハードウェアは複数の公開鍵と秘密鍵の集合をホストすることがあり、ここで、署名および暗号化も複数の鍵の集合のうちの1つを使用して実行される(各集合が異なる公開鍵と秘密鍵のペアを含む)。複数の鍵の集合は、742および744により使用されるハードウェアにおいて符号化され得る。入ってくるタグ付けされていないデータとともに含まれるクリアデータが、鍵のどの集合を使用すべきかをインターンユニット742に伝える。したがって、インターン742は、所望の鍵の集合を選択するために、複数の鍵の集合を含むハードウェアデータストア(たとえば、連想型メモリ)においてルックアップを実行し得る。複数の鍵の集合の各々は異なるタグと関連付けられ得るので、クリアデータにより示される特定の鍵の

集合は、タグ付けされたデータが含むであろう特定のタグも示す。このようにして、742により出力されるタグ付けされたデータアイテムのタグは、データアイテムが公開であり信頼されていることと、また、複数の鍵の集合のうちの特定の1つを使用してデータアイテムが暗号化/復号されることとを示す。複数の鍵の集合を伴う実施形態では、エクスターン744は、データアイテムを暗号化および署名することに関連して複数の鍵の集合のうちのどの特定の1つが使用されるかを判定するために、タグを調査し得る。したがって、インターンユニット742の処理は、受信されたタグ付けされていないデータを検証してタグ付けを実行し、それにより、データをタグ付けする能力を伴う、上で述べられた信用コード部分などのコードの部分を有する必要をなくす、隔離されたハードウェア構成要素を提供する。

【０３０３】

図1および図24に戻って参照すると、段階5におけるPUMP10への入力は、本明細書の他の箇所で説明されるようなタグを含む。命令のオペランドとしてメモリ位置を含む命令に対して、メモリ入力および関連するタグであるMR tag(本明細書ではMtagとも呼ばれることがある)を取得することは、余計なパイプラインのストールを引き起こすことがあり、これにより、段階5におけるPUMP10は、MR tagを含むすべてのその入力を有するまで、進むことができない。メモリから読み取られる実際のMR tag値を取り出すのを待つのではなく、予想または予測されるMR tagを判定するために本明細書の技法に従って処理を実行することができ、このMR tagは次いで、命令の結果に対するタグ値である(たとえば、もしあれば宛先レジスタまたはメモリ位置)R tagを判定するために使用されることが可能である。そのような実施形態では、最終的な確認が、予測されるMR tagが命令のオペランドのためにメモリから取り出されるアクションMR tagと一致するかどうかを判定するためのライトバック段階またはコミット段階(たとえば、図1の要素22または図24の最後の段階6としてのコミット段階を参照)である、段階6において行われることがある。オペランドとしてメモリ位置を有する命令に対するRtagを判定するための予測されるMR tagの前述の選択および使用は、Rtag予測アクセラレータ最適化と呼ばれ得る。

【０３０４】

図61を参照すると、Rtag予測アクセラレータ最適化のための本明細書の技法に従ったある実施形態の構成要素を示す例800が示されている。例800は、Rtag予測アクセラレータ最適化を実行するための追加の特徴を伴う、本明細書の他の箇所(たとえば、図1および図24)で説明されるような段階5におけるPUMP10に対応するPUMP802を含む。PUMP802は、MR tag 804aならびに本明細書の他の箇所で説明されるような他のPUMP入力804を入力として含む。PUMP802はまた、PUMP802が普通の処理モード(MR tag予測処理が実行されない非予測モード)で実行するか、またはそうではなく予測モード(MR tag予測処理が実行される)で実行するかを示す、予測モードセレクタ804bを別の入力として含む。少なくとも1つの実施形態では、予測モードセレクタ804bは、予測されるMR tag値が判定されない、PUMPの普通の処理モードを示す0であるか、予測されるMR tag値が判定される、PUMPの予測モードを示す1であるかのいずれかであり得る。予測モードセレクタが1であるとき、PUMP802は、MR tag 804aの入力がマスクまたは無視され得る予測モードで実行することがあり、PUMP802は予測されるMR tag 805cを出力として産生する。予測モードセレクタが0であるとき、PUMP802は、本明細書の他の箇所で説明されるような普通の処理モードで実行することがあり、ここでMR tag 804aはPUMP802への入力であり、生成される出力805cはない。

【０３０５】

例800に示されるように、段階5におけるPUMP802の追加の出力は、R tag 805aおよびPC new tag 805bを含む。予測されるMR tagを使用するとき、予測されるMR tagのためのルールが判定されることがあり、ここでこのルールはR tagのための関連するタグを指定する。予測モードで動作するとき、予測されるMR tag 805cはパイプラインの段階6 808への追加の入力である。要素808は、本明細書の他の箇所(たとえば、図1および図24)で説明されるようなコミットまたはライトバック段階を示し得る。したがって、要素808

aは一般に、本明細書の他の箇所で説明されるような、805a～c以外の他の段階6の入力を示し得る。

【０３０６】

段階6 808において、PUMP802が予測モードで動作するときに、追加の処理808bが実行され得る。要素808bは、予測されるMR tagを命令のオペランドのためのメモリから取得されるアクションMR tagと比較する、確認が段階6 808において実行され得ることを示す。言い換えると、808bは、PUMP802が、予測されるMR tagがメモリから取得されたMR tagと一致するかどうかを判定することによって、PUMP802がMR tag値を正しく予測したかどうかを評価する。予測されるMR tagがメモリから取得されたMR tagと一致しない場合、不正確な予測されたMR tagを用いてR tag 805aを判定する際に、不正確なルールがトリガされ、PUMP802により使用された。ここでは、改訂されたR tagを判定する際に、正しいルールが(実際のMR tagに従って)選択され使用されなければならない。したがって、予測されるMR tagがMR tagと一致しない場合、ルールキャッシュミスが判定され、キャッシュミスハンドリングが実行される。本明細書の他の箇所での説明と一貫して、キャッシュミスハンドリングは、MR tagを使用して正しいルールを選択し評価するための処理を含み得る。

【０３０７】

ロード/読取りおよび記憶/書込み命令は、予測されるMR tagの使用から利益を得るオペランドとしてメモリ位置を含み得るある実施形態の命令の例である。PUMPへの他の入力804は、MR tag 804a以外の他のまたは残りの入力タグの集合を含む。たとえば、図23に関連して示されるような一実施形態は、PC tag、CI tag、OP1 tag、OP2 tag、およびMR tagという5つの入力タグと、PC new tagおよびR tagという2つの出力タグを有し得る。したがって、残りの入力タグの集合(MR tag以外)は、PC tag、CI tag、OP1 tag、OP2 tagという4つのタグを含む。予測されるMR tagまたは命令を判定することは、命令の4つのタグ(たとえば、PC tag、CI tag、OP1 tag、OP2 tagのための)と一致するタグ値を有する1つまたは複数のルールの集合を判定することを含み得る。いくつかの事例では、単一のルールのみが、4つの入力タグに対して一致するタグ値を含み得る。この場合、単一の一致するルールはまた、予測されるMR tag 805cとして使用され得るMR tagに対する値を指定する。加えて、このルールは、R tag 805aをさらに判定するために、4つの入力タグおよび予測されるMR tagを使用して評価され得る。

【０３０８】

たとえば、典型的なロードおよび記憶動作を伴うメモリ安全性ポリシーを考える。ロード動作はポインタを使用してソースメモリ位置からデータをロードすることができ、ここで、第1のルールが、ソースメモリ位置でのタグまたは色がポインタのタグまたは色と一致すべきであることを示す。記憶動作はポインタを使用してターゲットメモリ位置にデータを記憶することができ、ここで、第2のルールが、ターゲットメモリ位置でのタグまたは色がポインタのタグまたは色と一致すべきであることを示す。ロード命令に対して、第1のルールが、ロード命令のPC tag、CI tag、OP1 tag、およびOP2 tagのための4つの入力タグと一致するタグ値を有する唯一のルールであり得る。第1のルールのMR tagは、予測されるMR tag 805cとして使用され得る。加えて、第1のルールのR tagは、4つの入力タグの集合および予測されるMR tagを使用して判定され得る。同様の方式で、記憶命令に対して、第2のルールが、記憶命令のPC tag、CI tag、OP1 tag、およびOP2 tagのための4つの入力タグと一致するタグ値を有する唯一のルールであり得る。第2のルールのMR tagは、予測されるMR tag 805cとして使用され得る。加えて、第2のルールのR tagは、4つの入力タグの集合および予測されるMR tagを使用して判定され得る。

【０３０９】

他の事例では、命令のPC tag、CI tag、OP1 tag、およびOP2 tagのための入力タグと一致するタグを有するポリシーのルールの集合は複数の一致するルールを含むことがあり、各々の一致するルールは、予測されるMR tag 805cとして使用され得る異なる許容可能なMR tagまたは候補のMR tagを特定する。ある実施形態は、予測されるMR tagとして

使用するために複数の許容可能なMR tagのうちの1つを選択するために、任意の適切な技法を使用し得る。たとえば、ある実施形態は、最も一般的に発生する、または最も発生する可能性の高い、許容可能なMR tagの集合のうちのMR tagを選択し得る。最も発生する可能性の高いMR tagは、以前の観察またはルールのプロファイリングに基づき得る。代替として、ある実施形態は、以前のまたは最新のMR tagとなるように予測されるMR tagを設定し得る。最悪の場合、予測されるMR tagが一度受信された実際のMR tagと一致しない場合、命令の他の入力タグとともに実際のMR tagを使用する正しいルールを判定するために、キャッシュミスハンドリングが本明細書で説明されるように実行され得る。

【０３１０】

少なくとも1つの実施形態では、PUMPが予測モードで動作するときに使用される、メモリ動作のためのルールのクラスが作成され得る。ルールのクラスは、「予測メモリタグ」ルールのクラスと呼ばれ得る。「予測メモリタグ」ルールに対して、MR tag 804aは、PUMP802への入力として使用されないので、PUMPにより実行される様々なルックアップに関連して使用されない。たとえば、「予測メモリタグ」ルールのためのcare/don't careビットベクトルは、MR tagをdon't careとして扱うことがある。加えて、「予測メモリタグ」ルールは、MR tagを入力として省くことがあり、むしろ、予測されるMR tagを出力として指定することがある。上で説明されたように、PC tag、CI tag、OP1 tag、およびOP2 tagのための入力タグの特定の集合と一致する、複数の一致する普通のルールがある場合、一致するルールの集合に対応する単一の「予測メモリタグ」ルールが、最も一般的なMR tagまたは期待されるMR tagである出力として、予測されるMR tagを指定し得る。一実施形態では、一致するルールの集合に対応する単一の「予測メモリタグ」ルールが、PUMP802によって受信された最後のまたは以前のMR tagを、予測されるMR tagとして指定し得る。

【０３１１】

ポリシー論理は、「予測メモリタグ」ルールを挿入または使用するかどうかを決めることができる。ある実施形態は各ポリシーの2つのバージョンを保つことができ、ここで、第1のバージョンは予測モードで動作するときに使用するためのポリシー「予測メモリタグ」ルールを含み、第2のバージョンは普通の処理モードまたは非予測モードで動作するときに使用するための普通のまたは非予測ポリシールールを含む。段階6の808bにおいて実行される確認が、「予測メモリタグ」ルールを使用するときに所与の命令に対して失敗する場合、キャッシュミスハンドリングが、普通のルール集合(たとえば、上で説明されたルールの第2のバージョン)を使用して命令に対する一致するルールを判定するための処理を実行し得る。

【０３１２】

RISC-Vプロセッサおよびアーキテクチャを使用するある実施形態では、予測モードセレクタ804bは対応するPUMP CSRを有し得る。RISC-Vアーキテクチャを使用するある実施形態におけるCSRの使用は、本明細書の他の箇所でより詳細に説明される。

【０３１３】

図62を参照すると、本明細書の技法に従ったある実施形態において実行され得る処理ステップのフローチャートが示されている。フローチャート840は、例800に関連して上で説明されたような処理を要約する。上で述べられたように、例800に示されるPUMP802は、プロセッサパイプラインの段階6への入力を与える段階5におけるPUMPを示す。少なくとも1つの実施形態では、フローチャート840のステップ842、844、846、848、および852は、PUMP内で具現化される上で説明されたような段階5において実行される処理ステップおよび使用される具体的なポリシールールを示すことがあり、ステップ854、856、および858は上で説明されたような段階6において実行されることがある。

【０３１４】

ステップ842において、予測モードがオン/有効であり、それにより、PUMPが「予測メモリタグ」ルールを使用して予測モードで動作していることを示すかどうかについての判定が行われる。ステップ842においてnoであると評価される場合、制御はステップ846に進

み、そこで、PUMPは普通のルールを使用して普通のモードまたは非予測モードで動作する。ステップ842においてyesであると評価される場合、制御はステップ844に進み、そこで、現在の命令がメモリ入力動作命令であるかどうかについての判定が行われる。ステップ842においてnoであると評価される場合、制御はステップ846に進む。ステップ844においてyesであると評価される場合、制御はステップ848に進み、そこで、PUMPは「予測されるメモリタグ」ルールを使用して予測モードで動作する。ステップ848において、命令のための一致する「予測されるメモリタグ」ルールが判定され得る。ステップ852において、現在の命令のためのR tagが、ステップ848からの一致する「予測されるメモリタグ」ルールを使用して判定され得る。ステップ854において、予測されるMR tagが実際のMR tagと一致するかどうかについての判定が行われる。ステップ854においてnoであると評価される場合、制御はステップ856に進み、ルールミスハンドラを呼び出すことによってルールキャッシュミス処理を実行する。ステップ856がyesであると評価される場合、制御はステップ858に進み、そこで、予測されるMR tagを含むルールを用いて判定されるようなR tagがR tag PUMP出力として使用される。

【０３１５】

例800からの変形として、普通の非予測モードで実行するPUMP802とともに予測モードで実行する第2のPUMP822も含む、ある実施形態の構成要素を示す図63に対する参照が行われる。この例では、予測モードで実行するPUMP822はMR tag予測PUMPとも呼ばれることがあり、このとき、予測モードセレクタ822bは常にオン(たとえば、1)である。同様に、PUMP802では、予測モードセレクタ804bもオフ(たとえば、0)であり得る。MR tag予測PUMP822は「予測メモリタグ」ルールだけを使用することがあり、PUMP802はポリシールールの普通のまたは非予測バージョンのみを使用することがある。そのような実施形態では、PUMP802および822は段階5において並列に動作し得る。要素828は、MR tag予測PUMP822と関連付けられる段階5および6の処理と構成要素を示し得る。要素829は、普通のモードで動作するPUMP802と関連付けられる段階5および6の処理と構成要素を示し得る。829において、PUMP802の出力は例800に関連したようなものであり、違いは、予測されるMR tag 805cがPUMP802によりもはや出力されないという点である。加えて、段階6 808は確認808bを実行しない。要素828は、例800と同様の方式で処理を実行する構成要素を含むことがあり、違いは、MR tag予測PUMP822が上で述べられたような「予測メモリタグ」ルールのみを使用するという点である。

【０３１６】

段階6(808)は、MR tag予測PUMP822からPUMP出力Rtag 805aおよびPCnewtag 805bをとり、PUMP802から出力Rtag805dおよびPCnew tag 805eをとるように改訂される。加えて、段階6において、Rtag805aと805dとの間で選択が行われ、また、予測されるMR tagが実際のMR tag(たとえば、808aにより示されるような)と一致するかどうかに基づいて、PCnew tag 805cと805eとの間で選択が行われる。予測されるMRtagと実際のMRtagとの間に一致がある(たとえば、808aが1または真と評価される)場合、予測されるPUMP822からのタグ(たとえば、Rtag805aおよびPCnew tag 805b)が使用され、予測されないPUMP802からのタグ(たとえば、Rtag805dおよびPCnew tag 805e)は廃棄される。予測されるMRtagと実際のMRtagとの間に不一致がある場合(たとえば、808aが0または偽と評価される)、予測されるPUMP822からのタグ805a～cは廃棄され、予測されないPUMP802からのタグ805d～eが使用される。予測されないPUMP802は、予測されるPUMP822の出力805a～cよりも遅いサイクルでその出力である出力805d～eを与えるので、PCnewtagおよびMRtagに関する段階5からのPUMP出力が処理のために段階6への入力として必要とされるとき、これは、前述の段階6の入力を待機する段階6へのストールをもたらす。予測されないPUMP802はまた、選択されるときにPUMPルールキャッシュミスを経験することがあり、この場合、これは、本開示の他の箇所で説明されるような典型的なルールキャッシュミスのように扱われる。

【０３１７】

段階6 808を参照すると、要素850および852はマルチプレクサを表す。要素808aは、

予測されるMRtagがMRtagと一致するかどうかの論理的な結果に基づいて、850および852の各々から入力を選択するために使用されるセレクタを示し得る。前述の2つのタグ値が一致する場合、Rtag805aは、段階6の最終的なRtag出力を示す選択されたRtag850aとして与えられる850への入力として選択される。そうではなく、前述の2つのタグ値が一致しない場合、Rtag805dが、選択されたRtag850aとして与えられる850への入力として選択される。加えて、前述の2つのタグ値が一致する場合、PCnew tag 805bが、段階6の最終的なPCnew tag出力を示す選択されたPCnew tag 852aとして与えられる852への入力として選択され、そうではなく、前述の2つのタグ値が一致しない場合、PCnew tag 805eが、選択されたPCnew tag 852aとして与えられる852への入力として選択される。

【０３１８】

ここで説明されるのは、本明細書の技法に従ったある実施形態において使用され得る、割り振られたメモリを色付けすることを使用する技法である。

【０３１９】

Cプログラミング言語でコーディングされたものなどのユーザプログラムが、メモリ割振りおよび割振り解除に関連して使用されるルーチンへの呼出しを含むことがある。たとえば、mallocおよびfreeは、Cの標準ライブラリのルーチンであり、ユーザプログラムの実行可能ファイルへとリンクされ得る。したがって、mallocおよびfreeは、mallocおよびfreeを呼び出し得る他のユーザコードとともにユーザプロセスアドレス空間の中のルーチンとして実行する。mallocは、コードを実行することによって使用されるメモリのブロックを割り振るために、動的なメモリ割振りのために呼び出される。少なくとも1つの実施形態では、mallocは、割り振られるべきメモリブロックのサイズを示す、呼出しの際に指定される入力を有することがあり、これにより、mallocは割り振られたメモリブロックへのポインタを返す。プログラムは、mallocにより返されたポインタを使用して、割り振られたメモリブロックにアクセスする。少なくとも1つの実施形態では、freeが、mallocを用いて以前に割り振られたメモリを解放するために、または割振り解除するために呼び出される。mallocを使用して割り振られたメモリブロックがもはや必要とされないとき、(mallocにより返されるような)ポインタが入力引数としてfreeに渡されることがあり、それにより、freeは、(ポインタにより示されるアドレスとして位置する)メモリが他の目的で使用され得るように、そのメモリの割振りを解除する。本明細書の技法に従ったある実施形態においてプロセッサで実行するユーザコードは、メモリ割振りおよび割振り解除を同様に実行するmallocおよびfreeまたは他のルーチンもしくは関数に対する、そのような呼出しを実行し得る。動的なメモリ割振りを実行するmallocおよびfreeなどのルーチンは、割り振られたメモリに関するメモリ管理メタデータを利用し得る。以下の段落では、メモリ管理のために使用されるそのようなメタデータは、mallocメタデータと呼ばれることがあり、タグとポインタタグにより指される他のメタデータとを含む本明細書で説明されるタグベースのメタデータとは別個であり、それに追加するものである(たとえば、ここでタグベースのメタデータは、実行中のユーザコード
がアクセス不可能であり、例1000および本明細書の他の箇所に関連して説明されるものなどのメタデータプロセッサまたはサブシステムにより処理される)。mallocメタデータは、たとえば、割り振られたメモリブロックのサイズ、後で割り振られるメモリブロックのためのmallocメタデータ部分へのポインタなどの、割り振られたメモリブロックについての情報を含み得る。

【０３２０】

図64を参照すると、mallocに関連したものなどの、メモリ割振りを示す例が示されている。例1100において、プログラムは、要求されたサイズのメモリの第1のブロックを割り振るために、mallocへの第1の呼出しを実行し得る。それに応答して、mallocは、要求されたサイズのメモリブロック1102bを割り振り、メモリブロック1102bのための開始アドレスを示すポインタP1を返し得る。次いで、ユーザプログラムが、ポインタP1またはP1からのオフセットに基づく別のアドレスを使用して、割り振られたメモリブロック1102b

にデータを記憶し、またはそれからデータを読み取り得る。加えて、動的なメモリ管理の目的で、mallocはまた、割り振られる各メモリブロックのための固有のmallocメタデータのために、ストレージ1102aを割り振り得る。要素1102aは、割り振られたメモリブロック1102bのためのmallocメタデータを記憶するために、mallocにより割り振られ使用されるメモリ部分を示す。同様の方式で、ユーザプログラムは続いて、第2のメモリブロックを割り振るためにmallocに対する第2の呼出しを実行し得る。要素1104aはこの第2の呼出しに応答してmallocにより割り振られるメモリ部分を示し、ここで、1104aはmallocメタデータを記憶するために使用される。要素1104bは割り振られた第2のメモリブロックを示し、ここでP2は第2のメモリブロックにアクセスするためにユーザプログラムに返されるポインタである。同様の方式で、ユーザプログラムは続いて、第3のメモリブロックを割り振るためにmallocに対する第3の呼出しを実行し得る。要素1106aは、この第3の呼出しに応答してmallocにより割り振られるメモリ部分を示し、ここで、1106aはmallocメタデータを記憶するために使用される。要素1106bは、割り振られる第3のメモリブロックを示し、ここで、P2は第3のメモリブロックにアクセスするためにユーザプログラムに返されるポインタである。

【０３２１】

1102bなどのメモリの割り振られたブロックが実行中のコードによりもはや必要とされなくなった後で、コードは、メモリブロック1102bが割振りを解除され他の目的で使用され得るように、メモリブロック1102bを解放するためにfreeに対する呼出しを実行し得る。ポインタP1は、freeに対するそのような呼出しを行うときに返され得る。同様の方式で、メモリブロック1104b～cがもはや必要とされないとき、それぞれポインタP2およびP3を指定する、freeに対する呼出しが行われることがある。

【０３２２】

実行中のユーザコードへmallocにより返されるP1などのポインタを通じて、ユーザコードは、不注意で、または意図せずにmallocメタデータにアクセスすることがあり、それは、mallocメタデータを保持するメモリ部分1102aのアドレスが実行中のコードのアドレス空間へとマッピングされるからである。たとえば、ユーザコードは、メモリ部分1102aの中のアドレスに別のポインタP4を割り当て(たとえばP4=P1-2)、次いでポインタP4により特定されるメモリ位置への読取りまたは書込みを行うことがある。したがって、ユーザコードは、たとえば、1102aに記憶されているmallocメタデータを上書きし、1102aに記憶されているmallocメタデータを読み取ることがある。このようにして、P4により特定されるアドレスにおけるメモリ位置への書込みを実行することは、mallocメタデータ部分1102aを破損させることがある。より一般的には、前述のことは、mallocメタデータ部分1102a、1104a、および1106aのいずれかに関連してユーザコードにより実行されることがある。

【０３２３】

freeへの呼出しに関連して、ユーザコードは、mallocを使用して以前に割り振られた、割り振られたメモリブロックの開始アドレスに対応するポインタを指定し得る。たとえば、ユーザコードは、P1、P2、またはP3以外の引数として、前述のポインタP4を指定するfreeへの呼出しを実行し得る。たとえば、mallocがmallocへの呼出しに関連して各mallocメタデータ部分1102a～cに対してXバイトのブロック(たとえば、Xは0ではない整数である)を割り振ると仮定する。ルーチンfreeは、第1のアドレス(P4-X)から第2のアドレス(P4-1)までのメモリ位置がそれぞれ、1102aなどのmallocメタデータ部分にわたる開始アドレスおよび終了アドレスを示すという仮定のもとで、処理を実行し得る。この場合、freeにより実行される処理は破損したmallocメタデータ部分1102aを使用することであることがあり、これはたとえば、予期されないランタイム実行および/または動的メモリ管理のエラーをもたらす。

【０３２４】

ある実施形態は、ユーザコードなどの他の実行中のコードにより実行される上書きを通じた破損を避けるために、mallocメタデータ部分1102a、1104a、および1106aを保護す

るために、本明細書で説明される技法を使用し得る。そのような技法は、コードおよび/またはデータを特定の色またはタグでタグ付けすることと、本明細書の他の箇所で説明されるものなどの所望のアクセスおよび動作のみを許容するようにルールを実施することとを含み得る。

【０３２５】

図65を参照すると、少なくとも1つの実施形態では、mallocおよびfreeにより使用されるメモリ部分は、本明細書で説明されるようなメタデータ処理により使用される第1のタグで色付けまたはタグ付けされることがあり、(mallocにより割り振られるような)ユーザコードにより使用される他のメモリ部分は、本明細書で説明されるようなメタデータ処理により使用される第2の異なるタグで色付けまたはタグ付けされることがある。例1100において、mallocおよびfreeにより使用されるデータ部分(mallocメタデータを含む)は赤に色付けまたはタグ付けされることがあり、ユーザデータ部分(ユーザコードによる使用のためにmallocにより割り振られるメモリブロック)は青に色付けまたはタグ付けされることがある。ある実施形態は、mallocおよびfreeにより使用されるメモリ位置を色付けまたはタグ付けする際に排他的に使用するために確保される少なくとも1つのタグまたは色を有し得る。この例では、赤はmallocおよびfreeにより使用されるメモリ位置をタグ付けするために使用される、確保された色である。本明細書の他の箇所で説明されるように、ある実施形態はまた、ユーザコードを実行するための1つまたは複数の色またはタグを確保し得る。少なくとも1つの実施形態では、ユーザプログラムによる使用のために割り振られるすべてのメモリが同じ色でタグ付けされ得る。変形として、ある実施形態は、mallocへの各呼出しのために異なるタグを使用し、したがって、割り振られる各々の別個のメモリブロックのために異なる色を使用し得る。この例1110では、説明を簡単にするために、単一の色である青だけが、ユーザプログラムのためにmallocにより割り振られるすべてのメモリブロックをタグ付けするために使用される。

【０３２６】

要素1111は、対応するメモリ位置1113のために指定されるタグを示し得る。要素1112a、1114a、および1116aはそれぞれ、mallocメタデータ部分1102a、1104a、および1106aのためのタグを示す。要素1112b、1114b、および1116bはそれぞれ、上で説明されたようにmallocに対して行われる呼出しを介してユーザコードによりユーザのためにmallocにより割り振られる、メモリブロック1102b、1104b、および1106bのためのタグを示す。

【０３２７】

要素1112a、1114a、および1116aは、1102a、1104a、および1106aの中にそれぞれある各メモリ位置が赤としてタグ付けされることを示す。要素1112b、1114b、および1116bは、1102b、1104b、および1106bの中にそれぞれある各メモリ位置が青としてタグ付けされることを示す。

【０３２８】

一般に、ある実施形態は、1111により示されるタグで1113のメモリブロックを色付けするルールをトリガすることに関連して、命令のタグ付け、色付けされたポインタ、またはこれらの組合せを使用することができ、また、mallocおよびfreeだけがmallocメタデータエリア1102a、1104a、および1106aにアクセスすることが可能でありユーザコードがそれらにアクセスできないようなメモリ安全性ポリシーを実施することができる。

【０３２９】

第1の実施形態では、mallocおよびfreeのコードは、特別な命令タグ(たとえば、CI tag)を用いてローダなどによってタグ付け(たとえば、命令タグ付け)され得る。mallocとfreeの両方が、同じ固有のもしくは特別な命令タグでタグ付けされることがあり(たとえば、mallocおよびfreeコードがtmemという同じCI tagでタグ付けされる)、または、各々が固有のもしくは特別な命令タグでタグ付けされることがある(たとえば、mallocコードはtmallocでタグ付けされ、freeコードはtfreeでタグ付けされる)。mallocのコードは、実行されると、例1110などにおける色付けを実行するルールをトリガする記憶命令を含み得

る。freeのコードは、実行されると、mallocメタデータ部分(たとえば、1102a、1104a、および1106a)または以前にmallocされたメモリブロック(たとえば、1102b、1104b、および1106b)を、空いているメモリを表すF tagでブロックの各メモリセルまたはmallocメタデータ部分を再タグ付けすることなどによって、再初期化または割振り解除するルールをトリガする、記憶命令を含み得る。また、第1の実施形態では、メモリ安全性ポリシーは、ロード命令および記憶命令などの特定の命令の実行によりトリガされるルールを含むことがあり、これにより、ルールは、上で述べられた特別な命令タグでタグ付けされた命令に、1)mallocメタデータ部分1102a、1104a、および1106aにアクセスすることと、2)例1110のようなメモリブロックの色付けを実行することとを許容するだけである。そのようなルールは一般に、1102a、1104a、および1106aのいずれかの中のメモリセルを色付けし、またはそれにアクセスする各命令が、mallocまたはfreeを示す特別な命令タグを有することを確実にするために、CI tagを確認し得る。

【０３３０】

第2の実施形態では、特別な命令タグを使用するのではなく、ある実施形態は、ロード命令および記憶命令などの特定の命令の実行によりトリガされるメモリ安全性ポリシーのルールとともに、色付けされたポインタを使用し得る。ローダは、色が赤であるmallocメタデータ部分1102a、1104a、および1106aを参照するmallocおよびfreeのポインタをタグ付けし得る。mallocのコードは、実行されると、例1110などの色付けを実行するルールをトリガする、記憶命令を含み得る。freeのコードは、実行されると、mallocメタデータ部分(たとえば、1102a、1104a、および1106a)または以前にmallocされたメモリブロック(たとえば、1102b、1104b、および1106b)を、空いているメモリを表すタグFでブロックのメモリセルを再タグ付けすることなどによって、再初期化または割振り解除するルールをトリガする、記憶命令を含み得る。メモリ安全性ポリシーは、ロード命令および記憶命令などの特定の命令の実行によりトリガされるルールを含むことがあり、これにより、ルールは、赤に色付けられたポインタを使用してメモリセルを参照する命令によるmallocメタデータ部分1102a、1104a、および1106aへのアクセスのみを許容する。そのようなルールは一般に、1102a、1104a、および1106aのいずれかの中のメモリセルにアクセスするメモリ命令が、メモリセルの第2の色と一致する第1の色を伴うポインタを使用することを確実にするために、MR tagを確認し得る。

【０３３１】

第3の実施形態では、特別な命令タグと上で説明されたような色付けされたポインタの両方が、組み合わせて利用され得る。以下は、そのような第3の実施形態において使用され得る命令およびルールの例である。本明細書の他の議論と一貫して、以下の例は、PC(プログラムカウンタ)、CI(現在の命令)、OP1(現在の命令のオペランド1)、OP2(現在の命令のオペランド2)、MR(もしあれば、現在の命令において参照されるメモリ位置)のためのメタデータ処理への5つの入力タグ、ならびに、PCnew(次の命令のための次のPCに対する新しいPC tag)、およびR(現在の命令の結果のためのタグ;現在の命令の結果が記憶される宛先レジスタまたはメモリ位置をタグ付けするために使用される)のための2つの伝播または生成されるタグに基づくルールを使用する。加えて、「-」はタグに対するdon't careを示す。そのような実施形態では、ローダは、特別なタグtmallocでmallocの命令をタグ付けすることができ、特別なタグtfreeでfreeの命令をタグ付けすることができる。色付けされたポインタは、下で述べられるトリガされるルールを使用して作成され得る。

【０３３２】

mallocに関連して、mallocのコード部分を実行することによりトリガされるメタデータルール処理は、mallocのコード部分の中の記憶命令の結果として呼び出される第1のルールを介して、1102bなどの新しく割り振られたメモリブロックへのポインタのためのタグを生成し得る。たとえば、malloc Cコードは「P1=next free」であることがあり、ここで、next freeは1113における次の空いているメモリ位置へのポインタであり、記憶命令は「move R1,R2」であることがあり、レジスタR1はアドレスnext freeを格納するソースレジスタであり、レジスタR2はポインタP1である宛先レジスタである。レジスタR1は

OP1(OP1 tagを有する)であることがあり、レジスタR2は結果または宛先レジスタ(発火したルールの結果として伝播または生成されるR tagを有する)であることがある。mallocのコード部分は、命令がmallocコードに含まれることを示す、前述のmove命令などの、特別なタグであるtmallocでもタグ付けされた命令を含み得る。少なくとも1つの実施形態では、ローダは、特別なコードタグであるtmallocでmallocの命令をタグ付けし得る。第1のルールは、割り振られたメモリブロック1102bへのポインタP1をタグblueでタグ付けし得る。mallocにおいて上のmove命令の結果としてトリガされる第1のルールは、

mv rule1A: (-,t malloc,blue-predecessor,-,-)→(-,blue)

であり得る。上のルールは、CI tagがtmallocであり、したがってmallocにおけるタグ付けされたmove命令のためのものであるときにのみ、発火する。mallocにより使用されるポインタがP1であると仮定すると、上のmv rule 1Aは、レジスタR2に記憶されているP1がblueのメモリ位置(たとえば、blueタグでタグ付けされたメモリ位置)へのポインタであることを示すために、タグまたは色blueでP1をタグ付けする。

【０３３３】

blueでタグ付けされたポインタP1は次いで、割り振られたメモリブロック1102bの中の各ワードに0または何らかの他の初期値を書き込むために、mallocの別の第2のstore命令とともに使用され得る。たとえば、「*P1=0」が「Store R3,(R2)」をもたらすmalloc Cコードに含まれることがあり、ここでR3は0を格納するソースレジスタオペランドOP1であり、R2はアドレスP1を格納するOP2レジスタである。このstore命令では、「(R2)」はオペランドMRであり、store命令のターゲットまたは宛先であるメモリ位置も示す。加えて、mallocにおける上のstore命令は、タグ付けされたtmallocであることもあり、mallocを呼び出したユーザコードにタグ付けされたポインタP1を返す前に、次のような第2の特別なstoreルールのトリガをもたらすことがある。

store 2A:(-,t malloc,-,blue,F)→(-,blue)

【０３３４】

上のstoreルール2Aは、CI tagがtmallocであり、R2の中のポインタまたはアドレス(P1を示す)がblueとしてタグ付けされるとき、かつ、P1により指されるメモリ位置MRがF tagを有するときにだけ、発火する。前述のメモリ位置*P1は、メモリ位置をblueで色付けする前に、「F」でタグ付けされると仮定される。この例では、Fは空いているメモリ位置を示す。メモリ位置のための得られるMR tagは、メモリ位置のためのblueタグを示す。

【０３３５】

したがって、mallocは、割り振られているメモリブロックの各メモリ位置に対する上で述べられた第2のルールのトリガをもたらすコードを含み得る。

【０３３６】

mallocは、mallocメタデータ部分1102a、1104a、および1106aを初期化する際に使用するための、以下で説明される追加のルール(たとえば、上のmove(mv)ルール1Aおよびstoreルール2Aと同様の)をトリガするコードも含み得る。たとえば、malloc Cコードは「(P1-2)=MD area」であることがあり、ここでMD areaはmallocメタデータエリア1102aへのポインタであり、move命令は「move R7,R8」であることがあり、ここでレジスタR7はアドレス「P1-2」を格納するソースレジスタであり、レジスタR8はポインタMD areaである宛先レジスタである。上のmove命令によりトリガされるルールは、MD areaポインタをredにタグ付けするために、

mv rule 1B: (-,t malloc_md,-,-,-)→(-,red)

であり得る。

【０３３７】

mallocはまた、それぞれmallocメタデータ部分1102a、1104a、および1106aのためのタグ1112a、1114a、1116aを記憶することなどのために、mallocメタデータ部分の各メモリ位置をタグ付けするために下で述べられるstoreルール2B(上のstore 2Aルールと同様の)をトリガするコードを含み得る。たとえば、sizeがmallocされたメモリブロック1102bのサイズを示す整数であり、「*(P1-2)=size」が「Store R6,(R7)」をもたらすma

lloc Cコードに含まれ、ここでR6がsize値を格納するソースレジスタオペランドOP1であり、R7がアドレスP1を格納するOP2レジスタであると仮定する。このstore命令では、「(R7-2)」はオペランドMRであり、store命令のターゲットまたは宛先であるMR1102aの中のメモリ位置も示す。storeルール2Bは、

store 2B:(-,t malloc_md,-,red,F)→(-,red)

store 2D:(-,t malloc_md,-,red,red)→(-,red)

であることがあり、これは、store命令がタグ付けされたtmallocである場合、アドレスP1を格納するR7レジスタがredにタグ付けされる場合、およびMRオペランドがFとしてタグ付けされる場合に、記憶を実行する。ある実施形態は、上で述べられたstore 2Bルールの変形である上で述べられたstoreルール2D(store 2D)も含むことがあり、これにより、store 2Dルールが、メタデータ値の更新が望まれる場合に使用されることがあることに留意されたい。

【０３３８】

より後の時点で、freeは、blueに色付けされたブロック1102bを解放するとき、または割振り解除するときなどに、以前にmallocされたメモリブロック(たとえば、ユーザコードの使用のために割り振られたメモリブロック)のメモリ位置を再タグ付けするために以下で述べられるstoreルール3のトリガをもたらす、「*P=0」などのコードを含み得る。ローダは、freeの命令をtfreeで色付けまたはタグ付けし得る。ルーチンfreeは、「Store R4,(R1)」をもたらすCコードステートメント「*P=0」を含むことがあり、ここでR4は0を格納するソースレジスタオペランドOP1であり、R1は初期化されるべきメモリ位置のアドレスを格納するOP2レジスタであり、「(R1)」はメモリオペランドMRを示し、R1はメモリ位置へのアドレスを格納する。storeルール3は、

store rule 3:(-,t free,-,blue,blue)→(-,F)

であり得る。

【０３３９】

したがって、freeは、割振り解除されたメモリブロックの各メモリ位置のために上で述べられた第3のルールのトリガをもたらすコードを含むことがあり、ここで、メモリブロックは以前に、ユーザコードにより使用するための(たとえば、mallocメタデータ以外のデータを記憶するために使用される)mallocを使用して割り振られている。storeルール3は、CI tag=t freeであることと、メモリ位置およびメモリ位置へのポインタの両方が同じ色であるblueを有することとを確実にするために、確認する。

【０３４０】

「blue」のMR tagは一般に、割り振られるユーザメモリブロックを色付けするためにmallocにより以前に使用される任意の色であり得ることに留意されたい。

【０３４１】

freeのコードはまた、1112aなどのmallocメタデータ部分の各メモリ位置を再タグ付けすることに関連して以下で説明されるmove(mv)ルール1Cおよびstoreルールをトリガするコードを含み得る。freeのコードは、上のmove(mv)ルール1Bと同様の以下で述べられるmove(mv)ルール1Cをトリガするコードを含み得る。move(mv)ルール1Cは、storeルール4を使用して再タグ付けすることに関連してfreeにより使用するためのredのポインタをタグ付けするために、

mv rule1C: (-,t free,-,-,-)→(-,red)

であり得る。

【０３４２】

以下のstoreルール4(上のstoreルール3と同様の)は、それぞれメタデータ部分1102a、1104a、および1106aのための再タグ付け1112a、1114a、1116aなどの、mallocメタデータ部分の各メモリ位置の再タグ付けを行うためにトリガされ得る。storeルール4は、

store rule 4:(-,t free,-,red,red)→(-,F)

であることがあり、これは、store命令がtfreeとタグ付けされる場合、かつMRオペランドがredとタグ付けされるポインタを使用する場合に、記憶を実行する。メモリ位置は「F

」でタグ付けされ、今やそのメモリ位置を空いているものとして示す。
【０３４３】
第4の実施形態では、PCタグ付けは、mallocメタデータ部分1102a、1104a、および1106aからデータを読み取り、そこにデータを書き込み、また、前述のメタデータ部分にアクセスすることから他のコードを排除するのに、十分な特権、アクセス権、または権限をmallocおよびfreeに与えるために使用され得る。PCタグ付けは、たとえば、異なるPC tag値を使用してプロセスごとに異なる特権、アクセス権、または権限を与えることについて例430に関連して、本明細書の他の箇所で説明される。同様の方式で、特別なまたは固有のPC tag値が、mallocメタデータ部分1102a、1104a、および1106aに関してロード動作および記憶動作を実行するための権限をmallocおよびfreeに与えるために使用され得る。さらに例示すると、mallocは、tmallocでタグ付けされた命令を含み得る(たとえば、命令が実行されるときCI tag=tmalloc)。mallocはまた、実行されると、mallocメタデータ部分1102a、1104a、および1106aにアクセスするための特権または権限を示す出力として特定のPC tagを伝播または産生するルールの適用をトリガする、コードを含み得る。mallocは、
Add 0,R2
などの第1の命令INS1を含むことがあり、ここでR2は、エリア1102aの中のアドレスP6などのmallocメタデータ部分の中のアドレスであり、(R2)はredで色付けされた1102aの中のアドレスP6を有するメモリ位置を示す。前述の命令INS1は、実行されると、X1などのタグ値を有するPCnewの生成をもたらすことがあり、ここでX1は1102aにアクセスするのに必要な特権を示す。この場合、上の第1の命令INS1のためにトリガされるルールは、R2を色redで色付けし、また、R2に記憶されているアドレス(たとえば、アドレスP6)を有するメモリ位置への読取り/書込みアクセス権を示すためにPCをX1に設定するために、
add:(-,tmalloc,--,red,-)→(X1,red)
であり得る。続いて、mallocは、アドレスP6(P6はR2に記憶されている)を有するメモリ位置へとレジスタR3からの値(たとえば、OP1)を記憶するために、「store R3,(R2)」という第2の命令INS2を含み得る。上の第2の命令INS2のためにトリガされるルールは、
store:(X1,tmalloc,-,red,red)→(PCdefault,red)
であることがあり、PCnewは、mallocメタデータ部分1102aにアクセスする特権を示さないデフォルトのPC tagであるPCdefaultへとクリアまたはリセットされる。したがって、この特定の例では、第1のADD命令は、1102aへの読取り/書込みアクセスのための特権または権限をmallocに付与するためにルールをトリガする。書込みを実行するmallocの上の第2の命令が実行された後で、伝播されるPC tagは、1102aへの読取り/書込みアクセスのための特権または権限をmallocから削除する。変形として、ある実施形態は、X1のPCnew tagを生成することによって1102aへの読取り/書込みアクセス権をmallocに付与するためのルールをトリガする命令を含む、プロローグを伴うmallocのバージョンを含み得る(たとえば、プロローグは上で述べられたルールをトリガするADD命令INS1を含む)。リターンの前のmallocの終わりに、実行されると、PCdefaultのPCnew tagを生成することによって1102aへのmallocの読取り/書込みアクセス権を削除するためのルールをトリガする命令を含む、エピローグが実行され得る(たとえば、エピローグは上で述べられたルールをトリガするstore命令INS2を含む)。
【０３４４】
同様の方式で、freeは、1102aへのアクセス権をfreeに与えるためのPCnew tag値を生成または伝播するためにルールを呼び出す命令を含み得る。適用されるルールは、具体的なプロセスに基づいて所望のアクセス権、特権、または権限を示す出力として特定のPC tagを伝播または産生することができ、それにより、具体的な許容される特権、アクセス権、または権限は、異なるPC tag値により表され得る。
【０３４５】
上記は、すべてのmallocされたメモリブロックに対するblueという単一の色と、すべてのmallocメタデータ部分に対するredという単一の色とを示すことに留意されたい。より一

般的には、本明細書の他の箇所で説明されるように、mallocは、ヒープメモリの異なる部分を色付けするために必要とされ得るような、無限の数の新しい色を生成する権限を与えられ得る。本明細書の他の箇所で論じられるように、たとえば、mallocは、1つまたは複数の色またはタグの初期の所定の集合を与えられることがあり、その初期の所定の集合から後で必要とされるタグを生成することがある。たとえば、mallocの初期の所定の集合は、yellowまたはYおよびredまたはRを含み得る。実行中のプロセスに対して、mallocは、ユーザコードにより使用される新しいメモリブロック(たとえば、mallocメタデータ記憶のためのもの以外の)を割り振るために、mallocの各呼出しのための未使用のYベースのタグ(たとえば、Y1、Y2、Y3...)を生成し得る。したがって、異なるYベースのタグが、各々のmallocされたメモリブロック1102b、1104b、および1106bを色付けするために使用され得る(たとえば、1102bはY1で色付けされ、1104bはY2で色付けされ、1106bはY3で色付けされる)。mallocは、mallocの各呼出しのために作成される各々の異なるmallocメタデータ部分のために未使用のRベースのタグ(たとえば、R1、R2、R3...)を生成し得る。したがって、Rベースのタグは、異なるRベースのタグでmallocメタデータ部分1102a、1104a、1106aを各々色付けするために使用され得る(たとえば、1102aはR1で色付けされ、1104aはR2で色付けされ、1106aはR3で色付けされる)。mallocにより使用される現在のまたは最後のRベースのタグおよび現在のまたは最後のYベースのタグは、malloc命令を実行するときにトリガされるルールを介して状態情報として記憶され得る。たとえば、mallocは、第1のメモリ位置のタグとして、最後のYベースのタグY9を記憶するルールをトリガする命令を含み得る。Y9はRtagとして生成され得る。後続の命令は再び、保存された最後のYベースのタグY9でタグ付けされた同じ第1のメモリ位置を参照することがあり、ここで、後続の命令は、1)最後のYベースのタグY9に基づく新しいタグY10を生成し、2)第1のメモリ位置のタグとしてタグY10を保存する、ルールをトリガする。Y10はRtagとして生成され得る。後続の命令によりトリガされるルールは、たとえばMRtag+1としてRtagを判定することを示すことがあり、ここでMRtagは後続の命令に対してはY9である。

【０３４６】

ここで説明されるのは、ハードウェアで加速されたミスハンドリングを使用するメタデータ処理に関連して最適化として使用され得る技法である。一般に、本明細書の実施形態において使用されるいくつかのポリシーは、頻繁なルールキャッシュミスを引き起こすことがあり、そのようなポリシーのためのキャッシュミスハンドラは、実行に多くのサイクルを要することがある。いくつかのポリシーでは、様々なルール入力間の関係は、結果または成果を論理的に判定するという点でいくらか単純であることがあり、したがって、専用のハードウェアを用いて迅速にハードワイヤリングされ解決されることがある。

【０３４７】

結果として、ハードウェア(HW)ルールキャッシュミスハンドラを使用して実装されるそのようなポリシーは、そのようなハードウェアアクセラレーションを使用しない他のものよりもはるかに短い時間で解決され得る。そのような実施形態では、1つまたは複数の選択されたポリシーのためのキャッシュミスハンドラなどのポリシー構成要素は、専用のハードウェアで実装され得る。したがって、本明細書の技法に従ったある実施形態は、そのようなハードウェアでサポートされるポリシーを単独で、または、ソフトウェアルールキャッシュミスハンドラを使用するソフトウェアで定義されるポリシー構成要素と組み合わせて、使用し得る。

【０３４８】

一例として、メモリ安全性の色付けを使用するメモリ安全性ポリシーを考える。本明細書の他の箇所で説明されるものなどのメモリ安全性ポリシーに関連して、メモリセルおよびポインタは色付けされることがあり、これにより、ロード動作と記憶動作の両方に関連して呼び出されるルールは、ポインタの色がメモリセルの色と一致するようなメモリ参照のみを許容し得る。たとえば、ロード命令のためにトリガされるルールは、(たとえば、レジスタがOP1などのオペランドである場合レジスタタグの)ポインタの色がメモリセルの色(

たとえば、Mtagなどのメモリ位置のタグ)に等しいようなポリシーを実施するために使用され得る。メモリ安全性ポリシーは、多くの色についてこの等しい色の関係を単純に捉える多くの異なるコンクリートルールでPUMPルールキャッシュを埋めることにより容量を逼迫させることがあり、容量ミスレートを上げる。ルールキャッシュをプリロードすることのない本明細書で説明されるようないくつかの実施形態では、これらのルールの1つ1つを挿入するために、強制的なルールキャッシュミスが必要とされる。メモリ安全性ポリシーのルールは一般に、ユーザコードを実行することに関連してトリガされ得るので、メモリ安全性ポリシーのルールは、ソフトウェアルールキャッシュミスハンドラではなくHWルールキャッシュミスハンドラを使用してサポートされ得る。

【０３４９】

そのような実施形態では、HWルールキャッシュミスハンドラは、ルールキャッシュミスが発生するとキャッシュへと挿入される新しいルールを生成または計算し得る。たとえば、メモリ安全性のためのミスハンドラは、ロード命令のためにOP1tagをMtagと比較するHWルールキャッシュミスハンドラとしてハードウェアを使用して実装され得る。OP1tagがMtagに等しい場合、HWルールキャッシュミスハンドラは、Mtagを割り当てられたRtagを伴う新しいルールを生成し得る。たとえば、ポインタPTRがredであり、PTRにより指されるメモリセルがredである場合、ルールを呼び出す命令は許容され、得られるタグRtagはredであるべきである。ルールキャッシュに挿入されるべき新しいルールとして前述のものを生成するために、HWキャッシュミスハンドラは、まずOP1tagをMtagと比較し得る。それらが等しくない場合、ルールの侵害があり、命令は許容されない(たとえば、プロセッサに実行を停止させる)。OP1tagがMtagに等しいとHWルールキャッシュミスハンドラが判定する場合、HWルールキャッシュミスハンドラは、opcode=load、OP1tag=red、Mtag=red、およびRtag=red(ルールのすべての他のタグの入力および出力はdon't careであり得る)を含む新しいルールをハードウェアの出力として生成することができ、ここで、生成されたルールは次いでルールキャッシュへと挿入され得る。

【０３５０】

図66を参照すると、本明細書の技法に従ったある実施形態において、ハードウェアで実装されるキャッシュミスハンドラを示す例が示されている。例1300は、入力1302aと一致するルールがキャッシュの中にあるかどうかを判定するためにルックアップを実行するための、PUMPルールキャッシュ1302(たとえば、図22)に入力される入力1302aを示す1301を含む。キャッシュの中にある場合、出力1302bはキャッシュに記憶されているルールに基づいて判定される。本明細書の他の箇所の議論と一貫して、入力1302aは、opcodeおよび入力タグであるPCtag、CItag、OP1tag、OP2tag、Mtagを含み得る。出力1302bは、PCnew tagおよびRtagなどのルールの出力タグを含み得る。ソフトウェアでキャッシュミスハンドラを実装する本明細書の技法に従ったある実施形態に関連して、ルールキャッシュミスが発生すると、ソフトウェアキャッシュミスハンドラが呼び出されることがあり、これにより、ミスハンドラのコードが、現在のルールキャッシュミスを引き起こす入力1302aのための新しいルールを実行して計算する。キャッシュミスハンドラはまず、入力が許容可能なルールと一致するかどうか(たとえば、メモリ安全性ロードルールについて、OP1tagがMtagと等しいかどうか)を判定し、一致する場合、特定の入力1302aのための出力を計算し(たとえば、RtagをMtagとして判定し)、それにより入力1302aのためのルールを生成する。新しいルール(入力1302aとミスハンドラの計算された出力との組合せに基づく)はルールキャッシュに挿入される。本明細書の他の箇所の議論と一貫して、新しいルールは、opcode、入力タグであるPCtag、CItag、OP1tag、OP2tag、Mtag、および出力タグであるPCnewtag、Rtagを含み得る。

【０３５１】

要素1303は、ソフトウェアルールキャッシュミスハンドラではなく、本明細書の技法に従ったある実施形態において使用され得るHWルールキャッシュミスハンドラ1304を示す。そのような実施形態では、HWルールキャッシュミスハンドラ1304は、たとえば、ゲートレベルの論理および他のハードウェア構成要素を含む、専用のハードウェアを使用して

実装され得る。そのような実施形態では、HWミスハンドラ1304は、PUMPルールキャッシュ1302と同じ入力1302aを受け取ることがあり、そのハードウェアを使用して、PUMPルールキャッシュに出力されるであろう同じ出力1302bを生成することがある。続いて、新しいルールが、上で述べられたようなopcode、入力タグ、および出力タグを組み合わせることによって、形成され得る。新しいルールは次いで、PUMPルールキャッシュ(たとえば、図22)に記憶され得る。

【０３５２】

少なくとも1つの実施形態では、メモリ安全性ポリシーのためのHWルールキャッシュミスハンドラは、ルールをキャッシュにロードするためにメモリセルタグをMtagからRtagに単純にコピーして直ちにPUMPルール挿入を実行し得る、ハードウェアで(たとえば、ゲートレベルの論理を使用して)上で説明されたように実装され得る。この単純な場合、メモリをデリファレンスしてメモリにおいてデータ構造の操作を実行する必要はない。

【０３５３】

加えて、少なくとも1つの実施形態では、メモリ安全性は、(1)メモリセル色タグ、(2)メモリセル中のポインタのポインタ色タグという、タグのペアとしてメモリセルのタグを実装し得る。メモリ安全性の加速は、記憶上で新しいRtagへとMtagおよびOP2tagを組み合わせるのを実行するために、かつ、Rtagへロードするための Mtagペアからのポインタタグの抽出を実行するために、専用のキャッシュを含み得る。これらのキャッシュに対するミスは、より簡単な専用のソフトウェアハンドラを使用し得る。前述はメモリ安全性などの単一の(非合成の)ポリシーに対して説明されるが、同じ一般的な技法は、UCPの合成ポリシーの構成要素に適用され得る。

【０３５４】

ある実施形態はまた、一般的に参照されることが予期されるものなどの、ルールの限られた一般的な部分集合のために、HWルールキャッシュミスハンドラを使用したハードウェアアクセラレーションを実行し得る。たとえば、メモリ安全性において、計算の間のロード/記憶および伝播のためのルールが最も標準的であり様式化されている。メモリ領域を最初に色付けして解放されるとメモリ領域を取り戻すための、他の一般的ではないルールも存在する。そのような一般的ではないルールは、ハードウェアのサポートで実装されるのではなく、本明細書で説明されるような典型的なルールミスハンドラの使用をもたらし得る。

【０３５５】

少なくとも1つの実施形態では、HWルールキャッシュミスハンドラは、ゲートレベルの論理としてマッピング関数を直接実装し得る。たとえば、そのようなゲートレベルの論理は、メモリ安全性ポリシーの記憶命令ルールのために、MtagをRtagにマッピングすることなどの、ルールのために入力タグを出力タグにマッピングすることを行い得る。別の例として、CFI(制御フロー整合性)ポリシーのためのHWルールキャッシュミスハンドラは、制御フローのターゲットまたは宛先のタグを許容される呼出者の集合へのポインタ(たとえば、タグ付けされた特定の制御フローのターゲットまたは宛先に制御を移転することが許容されるソース位置またはアドレス)にするために、ゲートレベルの論理を使用することがあり、CFI HWルールキャッシュミスハンドラが一致を求めてその集合全体を読み取ることを可能にする。さらに別の例として、スタック保護ポリシーが、stack-frame-codeタグおよび関連するstack-frame-memory-cellタグを、一方を他方からハードウェアが導出することが可能になる方式で符号化することができ(たとえば、それらは数ビットしか違わないことがあり、このことは、stack-frame-codeタグポインタおよびstack-frame-memory-cellタグポインタを一緒に割り振ることによって、それらのタグがポインタであった場合でも実行され得る)、その結果、スタック保護ポリシーを実施するHWルールキャッシュミスハンドラは、(スタックポインタからタグを作成する場合)作成すべきタグを判定することが可能であり、または、(読取りまたは書込みの場合)そのようなコード内でメモリ参照を要求することが可能である。

【０３５６】

PUMPルールキャッシュへと挿入される新しいルールを計算または判定するためにHWルールキャッシュミスハンドラを使用することからの変形として、ある実施形態は実際に、ポリシーの1つまたは複数のルールの論理をハードワイヤリングすることができ、このとき、そのようなルールはハードウェアで完全に具現化され実施されるので、PUMPルールキャッシュに記憶されない。たとえば、ポリシーのためにHWルールキャッシュミスハンドラおよびPUMPルールキャッシュを使用するのではなく、ある実施形態は、ハードウェアを使用してポリシーのルールを実施して符号化することができる(たとえば、ゲートレベルの論理および回路などのハードウェアで具現化されるポリシーのルール)。PUMPルールキャッシュとHWで指定されるルールの両方を使用するそのような実施形態では、PUMPルールキャッシュとHWで指定されるルールの両方の、ルールのルックアップが実行され得る。この場合、ミスハンドラ(たとえば、HWルールキャッシュミスハンドラまたはソフトウェアミスハンドラのいずれか)は、PUMPルールキャッシュまたはHWで指定されるルールのいずれかにおいて特定の入力のためのルールを発見しなかったことに応答して、新しいルールを判定/計算するために呼び出され得る。

【０３５７】

合成ポリシーは、追加の課題および好機をもたらす。本明細書の他の箇所の議論と一貫して、合成ポリシーは、命令のために同時に実施される複数のポリシーを含む。課題は、合成ポリシーがいくつかの異なるポリシー構成要素の解決を必要とするということである。好機は、合成ポリシーのための解決のシーケンス全体が、データキャッシュ、UCPキャッシュ(合成ポリシーの中のポリシー構成要素ごとのUCPキャッシュ)、およびCTAGキャッシュを用いて、合成ポリシーのすべての異なるポリシー構成要素のためのHWルールキャッシュミスハンドラを使用してハードウェアでサポートされ得るということである。以前の経験から、一般的な課題は、新たに割り振られるメモリ(たとえば、mallocを使用する)、したがって新しいメモリ色タグが、強制的なルールキャッシュミスを引き起こす場合である。これらの場合、メモリ安全性ポリシー構成要素は新しいルールを必要とするが、他の構成要素はUCPキャッシュの中にそれらのルールをすでに有する可能性が高いことがある。トップレベルの合成ポリシーのための、およびメモリ安全性の色の一致のための、HWルールキャッシュミスハンドラを通じたハードウェアアクセラレーションでは、メモリルールの解決は、ハードウェアで実行する小さい有限状態機械を用いて、かつ、ソフトウェアベースのミスハンドラコードを実行するルールを解決するために数百から数千のサイクルを必要とするのではなくキャッシュ(たとえば、データキャッシュ、UCPキャッシュ、およびCTAGキャッシュ)を調べることで、実行され得る。

【０３５８】

少なくとも1つの実施形態では、UCPキャッシュは、タグ結果の合成集合を産生して次いでCTAGキャッシュへとフィードバックされることと並列に、構成要素ポリシーにより分解されてすべて解決され得る。すべてのポリシーがそれらのUCPキャッシュにより解決され得るか、メモリ安全性のためのルールなどの簡単なハードウェアルールにより解決され得るかのいずれかである場合、UCPキャッシュのルックアップのための総時間は、ポリシーの数に比例するのではなく単一のポリシーのルックアップのための総時間となる。これは、構成要素ポリシーの数が固定されており、提供されるハードウェアに対して一致する場合、完璧に機能する。それでも、わずかな変化が単純に、利用可能な数が固定されているUCPキャッシュにわたって構成要素ポリシーを分散させるので、連続的なUCPキャッシュ解決の数は、物理的なUCPキャッシュに対する構成要素タグの数の比に過ぎない。

【０３５９】

図67を参照すると、本明細書の技法に従ってある実施形態において使用され得る合成ポリシーに関連してHWルールキャッシュミスハンドラの使用を示す例1310が示されている。この特定の例では、3つのポリシーが合成ポリシーを構成し、これにより、すべての3つのポリシーが同じ命令に対して同時に実施されるが、より一般的には、合成ポリシーは任意の数のポリシーを含むことがあり、3に限定されない。要素1314a～cは、合成ポリシーを構成する3つのポリシーのためのHWルールキャッシュミスハンドラである。入力1312は

、HWルールキャッシュミスハンドラ1314a～cの各々に与えられることがあり、HWルールキャッシュミスハンドラ1314a～cはそれぞれ、特定のポリシーのためのルール出力1316a～cを判定または計算する(たとえば、HWルールキャッシュミスハンドラ1314aはポリシーAのためのRtagおよびPCnew tagを含む出力1316aを判定し、HWルールキャッシュミスハンドラ1314bはポリシーBのためのRtagおよびPCnew tagを含む出力1316bを判定する)。続いて、出力1316a～cは、3つのポリシーのための合成のRtagおよびPCnew tagを示す単一の合成結果1318へと合成され得る。合成結果1318を判定するために出力1316a～cを合成することはまた、ハードウェアまたはソフトウェアを使用して実施され得る。新しいルールがキャッシュに挿入されることがあり、ここで、この新しいルールは、合成結果1318(たとえば、RtagおよびPCnew tagの合成値)とともに、ルールキャッシュミスハンドリングをトリガする特定の命令の入力1312(たとえば、オペコードおよび入力タグ)を含む。

【０３６０】

加えて、例1310には示されていないが、ある実施形態は、本明細書の技法に従ったある実施形態においてHWルールキャッシュミスハンドラ1314a～cと組み合わせてUCPキャッシュおよびCTAGキャッシュを使用し得る。本明細書の他の箇所で説明されるように(たとえば、図21、図23、および図24に関連して)、ポリシーA、B、およびCの各々が、最新のポリシー結果タグの固有のUCPキャッシュのキャッシング結果を有し得る(たとえば、ポリシーAのためのUCPキャッシュは、命令のオペコードおよび入力タグの組合せに基づいて、ミスハンドラ1314aにより最近計算された結果タグであるPCnewtagおよびRtagの結果を記憶する)。本明細書の他の箇所で説明されるように(たとえば、図21、図23、および図24に関連して)、CTAGキャッシュは、ポリシーA、B、およびCなどの複数の合成ポリシーから出力され得るような、個別のRtag値の特定の組合せのために、Rtagの合成結果を記憶し得る。CTAGキャッシュも、ポリシーA、B、およびCなどの複数の合成ポリシーから出力され得るような、個別のPCnew tag値の特定の組合せのために、PCnewtagの合成結果を記憶し得る。したがって、出力1316a～cから合成結果1318を生成するハードウェアは、合成結果1318を判定するためにCTAGキャッシュからの情報を使用し得る。加えて、HWルールキャッシュミスハンドラ1314a～cはまた、ポリシーA、B、およびCのためのUCPキャッシュからの情報を入力として有し得る。

【０３６１】

例1310におけるような合成ポリシーのすべての3つのポリシーのためにHWルールキャッシュミスハンドラを有することの代わりに、ある実施形態は、合成ポリシーを構成する、すべてではないが1つまたは複数のそのようなポリシーのために、HWルールキャッシュミスハンドラを実装することを選択的に選ぶことがある。そのような実施形態では、ルールキャッシュミスハンドラの一部分がハードウェアで実装されることがあり、合成ポリシーのルールキャッシュミスハンドラの残りの部分が本明細書の他の箇所で説明されるようにソフトウェアで実装されることがある。

【０３６２】

本明細書で説明されるようないくつかのポリシーは、たとえば、メモリ安全性ポリシーなどに関連して新しいタグを割り振り得ることに留意されたい。少なくとも1つの実施形態では、新しいタグを割り振り得るメモリ安全性などのポリシーのためのHWルールキャッシュミスハンドラは、HWベースのハンドラが使用し得る新しいタグ値のFIFOベースのキャッシュを与えられ得る(たとえば、生成される新しく割り振られたタグ値として使用され得るタグのキャッシュ。割り振られるタグがアドレスを示すポインタである場合、キャッシュはタグ値ではなくアドレスまたはポインタを含む)。このようにして、HWルールキャッシュミスハンドラは、FIFOベースのキャッシュから単純にトップのエントリーを読み取ることによって、割振りを実行し得る。定期的に、ソフトウェアハンドラは、割振りに利用可能な新しいタグでFIFOベースのキャッシュを埋め直すために、メタデータ処理領域で実行され得る。

【０３６３】

メタデータ処理領域とユーザコードまたはユーザ実行の「普通の」コード処理の領域との間に、完全で厳密な隔離がある実施形態が、本明細書で説明される。変形として、ある実施形態は、ユーザコードまたはユーザ実行領域によるメタデータ処理領域への情報の変更または書込みを依然として許容しないが、ユーザコードまたはユーザ実行領域へメタデータ領域により情報/値が返されることを許容し得る、より緩やかな手法を採用し、前述の厳密な隔離モデルを拡張することがある。

【０３６４】

ここで説明されるのは、前述のより緩やかな手法を利用し得る少なくとも1つの実施形態に含まれ得る技法であり、これにより、メタデータ処理領域は、普通のコード処理または実行領域におけるコード実行により使用または参照され得る値を返す(たとえば、メタデータ処理は普通のまたはユーザコード実行領域への入力である値を返す)。たとえば、本明細書の他の箇所で説明されるように、ある実施形態はmallocおよびfreeルーチンを使用することがあり、ここで、そのようなルーチンは、mallocおよびfreeのコードが、実行されると、mallocメタデータにアクセスすること、新しい色のタグを生成すること、そのような新しい色のタグでユーザデータエリアをタグ付けすることなどの処理を実行するための能力をおよびfreeに許容するルールをトリガするように、必要な固有の能力をmallocおよびfreeルーチンに与える命令タグでタグ付けされたコードを有し得る。上記は、ユーザコードなどの他のコードを排除して、mallocおよびfreeに固有に割り当てられるそのような特権または能力を提供する。ここで、mallocおよびfreeがそれらの処理を実行し、特別な実行特権を伴うmallocおよびfreeに属するものとしてmallocおよびfreeコードを一意に特定する特別なコードタグでローダによりmallocおよびfreeコードを特別にタグ付けするために、コードタグ付けが利用されるような実施形態を考える。そのような実施形態では、freeの呼出しを行うユーザコードが、たとえば、破損したポインタPTR1、またはそうでなければ、今では割振りが解除されている以前に割り振られていた記憶エリアの最初を指さないポインタPTR1を提供することがあり得る。PTR1は、freeによって、mallocにより以前に割り振られていたユーザデータエリアの第1の位置を指すと推定され得る。freeは、図64および図65に関連して説明されるものなどの、ユーザデータエリアおよびメモリヒープの関連するメモリ位置に対する所定の構造を仮定することがあり、たとえば、mallocメタデータは割り振られたユーザデータエリアに対して相対的に所定の位置に記憶される。

【０３６５】

ここで説明されるのは、PUMPにコード実行領域へ値を返させるためにある実施形態において使用され得る技法である。

【０３６６】

図68の例1200を参照して、以下で論じられるポインタPTR1およびPTR2でさらにアノテートされた図65の例1110に関連して説明されるような、要素1111および1113が示されている。ユーザコードは、メモリブロック1102の割振りを解除するための意図を伴って、PTR1を用いてfreeを呼び出すと仮定する。P1は、freeにより予期されるポインタまたはアドレスを示し得る。しかしながら、この例ではPTR1は、P1以外の異なるアドレスを一般に示す、破損したまたは不正確なアドレスを示し得る(たとえば、PTR1は、メモリ1102bの中の位置を特定することがあり、または、ヒープの中を指すことすらしないアドレスを示すことがある)。PTR1は破損しており、または正確なメモリ位置P1を別様に指さないが、freeは、PTR1を使用して処理を実行し、PTR1に対する相対的なアドレス指定を使用してmallocメタデータにアクセスすることがあり、ここで、mallocメタデータは、事前に定義された構造、フォーマット、またはレイアウトの中に存在すると仮定される。たとえば、freeにより使用されるmallocメタデータエリアは、図64および図65のように、割り振られたユーザデータ部分の直前に位置すると推定され得る。そのような場合、freeのコードは、特定のメモリブロックの割振りを解除するためにfreeが処理において使用するmallocメタデータが、所定のレイアウトに基づいてPTR1の前の特定のオフセットOFF1に位置すると判定し得る。たとえば、図68を参照すると、freeは、PTR1=P1であると

推定することがあり、ここで、PTR1はfreeの呼出しに際してユーザコードにより提供され得る。freeは、割振り解除されるべきメモリブロック1102bのための対応するmallocメタデータ1102aがアドレスPTR2=PTR1-OFF1を伴うメモリ位置で開始すべきであるという、所定のデータレイアウトに基づいて、上で説明されたような相対的なアドレス指定を使用し得る。この例では、PTR1はP1と等しくなく、PTR1は実際には、割り振られたメモリブロック1102bの中のどこかを指すので、アドレス計算PTR2=PTR1-OFF1も、ユーザにより割り振られたメモリブロック1102bの中にある(PTR2はfreeにより使用される関連するmallocメタデータの予期される始点を示す)。

【０３６７】

freeの呼出しに際してユーザコードにより提供されるPTR1が予期される位置P1を指さず、それによりPTR2がfreeにより使用されるmallocメタデータの推定される始点を示す場合、freeのコードは、侵害、中断、またはトラップを引き起こすmallocメタデータなどのデータを使用して、メモリブロック1102bに記憶されているデータに不正確にアクセスすることがある(たとえば、この侵害、中断、またはトラップは、PUMPにより検出されたルールの侵害、またはfreeの実行の間の他のコード実行エラー条件によるものであり得る)。したがって、ユーザプロセス空間または領域において実行するコードの実行は、ユーザコードからのPTR1を使用したfreeの呼出しにおいて呼び出されるような、ルーチンfreeの実行の間の前述の侵害により中止され得る。ルーチンfreeにユーザコードの前述の中止を引き起こさせるのではなく、PUMPにクエリすることをfreeのコードに許容すること、またはより一般的には、値を返すことをメタデータ処理に許容することが望ましいことがある。返される値は、たとえば、PTR2と関連付けられる色(mallocメタデータにアクセスするためにfreeのコードにより使用される)が実際に有効なまたは予期されるmallocメタデータエリアを指すかどうかを示す、ブーリアンであり得る。そのような返されるPUMPまたはメタデータ処理値を使用することは、アドレスPTR2におけるメモリ位置と関連付けられる色がredなどの有効なmallocメタデータの色を示すかどうかに基づいて、freeが異なる条件的な処理を実行することを可能にする。ルーチンfreeは、PTR2がPTR2の色を通じて判定されるような無効なmallocメタデータエリアを特定する場合、何らかの復元または他の活動を実行し得る。そのような活動は、ルールの侵害、トラップ、中断、または他の実行エラーにより中止されたユーザコードを有することよりも望ましいことがある。

【０３６８】

RISC-V命令セットを使用する少なくとも1つの実施形態では、メタデータ処理値を返すことを実施するために、新しい命令gmd(get-metadata-info)が以下のようにRISC-V命令セットに追加され得る。

gmd R1,R2,R3

ここで、

R1はPUMPまたはメタデータ処理により返される結果値を格納し、

R2はアドレスPTR2を有するメモリ位置の色でタグ付けされたアドレスPTR2を格納し、

R3は有効なmallocメタデータエリアに対して予期されるような有効な色でタグ付けされる。

したがって、R2およびR3は入力またはソースオペランドであるレジスタであることがあり、R1は結果または出力を格納するレジスタであることがある。この特定の例では、R3tagは有効なmallocメタデータエリアの色を示すredであることがあり、R2tagはblueであることがある。新しい命令により呼び出されるルールは、この例ではR2tag=R3tagであるかどうかを示すブーリアンとしてリターン値を出力することがあり、ここで前述のブーリアンの結果は、ユーザ実行コードのアドレス空間に含まれる、freeがアクセス可能なレジスタR1に記憶されるメタデータルール処理により出力される(たとえば、PUMP出力)リターン値であり得る。R1は本明細書の他の箇所の議論と一貫して、結果タグとしてRtagでタグ付けされ得ることに留意されたい。

【０３６９】

以下は、上で説明されたような、PTR1、PTR2、およびOFF1についてのC様の疑似コー

ドの記述を使用して、freeのコードによって実行され得る論理的な処理を記述する。
free(char *PTR1)
PTR2=PTR1-OFF1;/**PTR
if(IS_RED(PTR2))then
PTR2が有効に色付けされたmallocメタデータエリアを指す。割振り解除のための処理を実行する。
else
PTR2が有効に色付けされたmallocメタデータエリアを指さない。復元処理を実行する。
【０３７０】
上の論理的な処理において、IS_REDは、PTR2が色REDであるかどうかを確認し得る。
【０３７１】
上で述べられたelseブロックのコードにより実行される復元処理は、たとえば、PTR2から後方または前方に探索することによって、有効なmallocメタデータエリアの始点を位置特定することを試み得る。上で述べられたelseブロックのコードは、無効に色付けされたポインタPTR2を示すランタイムエラーメッセージ/条件などを用いて、より定義され予期される方式でユーザコードの終了を可能にし得る。
【０３７２】
新しい命令Get metadata info R1、R2、R3は、たとえば、上で述べられた論理的な処理を実行するためにCで書かれたfreeルーチンのコードのコンパイルおよびリンクの結果として生成される、命令に含まれ得る。ある実施形態は、どの特定のコード部分がこの新しい命令を実行することが許容され得るかを、制御または制約することを望むことがある。この新しい命令がどのルーチンによりいつ実行されることが許容されるかを仲介または制約するために、PUMPルールが使用され得る。たとえば、freeまたはmallocのコードは、新しいGet metadata info命令を実行することは許容されることがあるが、ユーザコードを実行することは許容されないことがある。一部が本明細書で説明される、任意の適切な技法が、PUMP値を返す新しい命令を実行するための必要な特権または権限をルーチンfreeに与えるために使用され得る。たとえば、freeのコードは、freeが新しい命令を実行することが許容されることを示す特別な命令タグでタグ付けされ得る。たとえば、ローダは、特別なタグNIでfreeのコードに現れる新しい命令をタグ付けし得る。どのコードが新しい命令を呼び出すことを許容され得るかを仲介し、またはNIの命令タグ(CI tag)を有するものに制約するために、ルールが使用され得る。
【０３７３】
図69を参照すると、本明細書の技法に従ったある実施形態においてメタデータルール処理の入力および出力を示す例1210が示されている。要素1212は一般に、本明細書で説明されるようなメタデータ処理を示し得る。メタデータ処理への入力1212aは、たとえば、本明細書で説明されるような様々なタグおよびオペコード情報を含み得る。メタデータ処理1212により生成される出力1214は、本明細書の他の箇所で説明されるようなRtag1214aおよびPCtag1214bを含み得る。加えて、メタデータ処理は、リターン値1214cである新しい出力を生成し得る。リターン値1214cは、ユーザプロセス空間/コード実行がアクセス可能なレジスタの集合の中にある、新しい命令について上で示されたR1などのレジスタの中に配置され得る。本明細書の他の箇所の説明と一貫して、1214aおよび1214bは、結果(たとえば、結果レジスタまたはメモリ位置)およびPCにそれぞれ付けられるタグを示し、それにより、1214a～bはユーザプロセス空間/コード実行がアクセス可能ではない。メタデータ処理がリターン値1214cを返すかどうかは、特定の命令またはオペコードについて条件的であり得ることに留意されたい。たとえば、本明細書の他の箇所で説明されるように、メタデータ処理出力は、リターン値1214cの出力を有効/無効にするためにマルチプレクサを使用してオペコードに基づいて図27～図33に関連して説明されるように、フィルタリングされ得る。この例の値1214cは、opcodeが新しい命令オペコードであるとき、R2tag=R3tagかどうかの論理的な結果を示す。そうではなく、opcodeが新しい命令オペコードを示さない場合、デフォルト値は、リターン値1214cとしてメタデータ処理

により条件的に返され得る。

【０３７４】

図70を参照すると、上で説明されたようなfreeのコードに含まれる新しい命令を実行するときなどの、メタデータ処理による値をユーザ実行領域に返すときに、本明細書の技法に従ってある実施形態において実行され得る構成要素および処理を示す例1220が示されている。説明を簡潔にするために、例1220は、この新しいリターン値のための宛先または結果レジスタR1および関連する結果レジスタだけとともに利用される、メタデータ処理の論理および構成要素を示す。要素1222aは一般に、メタデータ処理のために本明細書の他の箇所で説明されるようなPUMP入力(たとえば、この例ではR2tagおよびR3tagなどのタグ、オペコード)を表し得る。PUMP1222は、コードタグがNIであるかどうかを確認する新しい命令のためのルールを含むことがあり、R2tag=R3tagであるかどうかを示す論理的な結果を出力する(たとえば、この例ではOP1が第1の入力ソースオペランドR2を示し、OP2が第2の入力ソースオペランドR3を示す)。このルールは、前述の論理的な結果1221aの出力をもたらす。要素1225はマルチプレクサを示すことがあり、オペコードがそのマルチプレクサ1225のためのセレクタ1225aとして使用される。現在の命令のオペコードが新しい命令Get metadata infoのための特定のオペコードを示すとき、1225aはリターン値1214cとして出力されるべき選択1221aをもたらす。そうではなく、オペコードが新しい命令のオペコードではない場合、1225aはリターン値1214cとしてデフォルトのリターン値1222aの選択をもたらす。リターン値1214cは、宛先レジスタRD1228に記憶されるPUMP出力である(たとえば、1214cは、ユーザプロセスアドレス空間において実行するコードがアクセス可能なレジスタRDに記憶されている内容を示すD1 1228bに記憶される)。RD1228は結果レジスタであるので、ルールはRtagでRDをタグ付けすることももたらし得る(たとえば、Rtagはタグ部分T1 1228aに記憶され、T1はRDレジスタのタグワードである)。少なくとも1つの実施形態では、Rtagは、新しい命令の出力をRDが格納することを示す、特別なタグSPEC1であり得る。ルールへのタグ入力が(PCtag,CItag,OP1tag,OP2tag,MR tag)であり、ルール出力が、新しいリターン値1214cを示す第3の出力NEWOUTとともに(PCtag,Rtag)である、本明細書の他の箇所で説明されるようなシンボリック論理に基づくと、ルールは次のように表現され得る。

gmd:(-,NI,t1,t2,-)→(-,SPEC1,NEWOUT)

ここで、t1=t2の場合NEWOUT=1であり、それ以外の場合NEWOUT=0である。

【０３７５】

より一般的には、新しい命令の前述の使用法は、呼び出されたメタデータ処理ルールを介して許容可能な新しい命令の発生を示すために、(たとえば、NIで)特別にタグ付けされるコードにより使用され得る任意の適切なおよび望ましい値である値を返すために、本明細書の技法に従ったある実施形態において使用され得る。

【０３７６】

代替的な実施形態は、新しい命令の追加を避けることがある。これは、この挙動を制御するために既存の命令をコードタグ付けして、この場合に出力される値を選択するためにケアビットを設定することによって、行われ得る。別の代替形態は、出力される値がRD値の結果に流れるべき場合をルールが判定できるように、PUMPの出力でもあるvalue-output-care-bitを追加することができる。この第2のケースは、オペコードが、タグ付けされないときは普通に振る舞い、適切なコードタグを与えられるときにだけこの特別な挙動を示すことを可能にする。

【０３７７】

ここで説明されるのは、命令の特定のシーケンスが単一のユニットとしてアトミックに実行されることを保証するために、またはシーケンスの最初の命令から最後の命令まで指定された順序でシーケンスを完了するために使用され得る、技法である。加えて、そのような技法は、シーケンスの最初の命令以外への、命令のシーケンスへの制御の移転がないことと、シーケンスの最後の指定される命令を通らないシーケンスからの移転または脱出がないこととを、保証する。たとえば、図71の簡単な命令シーケンスを考える。

【０３７８】
例1400において、2つの命令1402および1404のシーケンスが示されている。第1の命令1402は、メモリ位置(メモリ位置のアドレスはR2に記憶されている)から内容を読み取り、またはメモリ位置からR1に内容をロードする。第2の命令は、同じメモリ位置(R2に記憶されているアドレスを有するメモリ位置)に0を書き込み、または記憶する。そのような命令シーケンスは、メモリ位置(R2において指定されるアドレスを有する)から読み取られる値が一度しか使用されないことを確実にすることを実現でき、これにより、古い値は、値がメモリ位置から読み取られた直後にメモリ位置から消去され、または0にされる。したがって、メモリ位置を0にすることは、シーケンスの第2の命令1404により実行され、メモリ位置から値を読み取る第1の命令1402の後にシーケンス中の次の命令として実行されることが必要とされる。
【０３７９】
RISCアーキテクチャを使用した本明細書の技法に従った少なくとも1つの実施形態では、データアイテムの前述の線形性および1400の命令シーケンスのアトミック性を実施するために、ルールが使用され得る。そのような実施形態では、PC tag(PC new tag)が、シーケンスの中の次の予期される命令のシーケンスの状態を伝えるために更新され得る。少なくとも1つの実施形態では、1つの解決法は、線形な読取り命令として命令1402を示すCIタグで命令1402をタグ付けすることである。加えて、R2に記憶されているアドレスを有するメモリ位置を示す(R2)は、固有のメタデータid X1(たとえば、X1はすべての他の線形な変数からこの線形な変数を一意に特定する)で線形な変数として分類されタグ付けされ得る。第1のルールは第1の命令1402の結果としてトリガされ得る。第1のルールは、線形であるものとしてタグ付けされた命令だけが、線形な変数から読み取ることを許容されることを示し得る。加えて、得られるPCnew tagは、実行される次の命令が線形な変数X1をクリアすることが必要であることを示すために、clear-linear-variable-X1-nextであり得る。第2のルールは第2の命令1404を実行した結果としてトリガされることがあり、ここで、オペランドの値0(メモリ位置に書き込まれる)が、メモリ位置を初期化またはクリアするために使用される特別な値を示す特別なEMPTYタグでタグ付けされる。加えて、メモリ位置は、直前の命令1402からの特定のタグ付けされた線形の変数を示す線形の変数X1であることが必要とされる。1402に続く第2の命令が、線形の変数X1にEMPTY値を書き込むこと以外の何かを行う場合、トラップが引き起こされる。したがって、第2のルールは、1400の命令のシーケンスの所望の連続性およびアトミック性を実施させる。
【０３８０】
より具体的には、第1の命令1402が、R2に記憶されているアドレスを有するメモリ位置からの内容をR1へとロードするload命令であると仮定する。load命令は次のようなものであり得る。
load R1,(R2)
加えて、第2の命令1404が、R2に記憶されているアドレスを有するメモリ位置へと0を移動するmove命令であると仮定する。move命令は次のようなものであり得る。
move 0,(R2)
本明細書の他の箇所で述べられる規約に従うと、ルールは、opcode、入力タグPCtag、CItag、OP1tag、OP2tag、Mtag、および出力タグPCnewtag、Rtagとして定義され得る。前述のルール規約に基づくと、第1のload命令に対して、OP1はR1であり、OP2はR2であり、(R2)はMtagとしてタグ付けされるメモリ位置である。第1のload命令によりトリガされる第1のルールは、
load: (-,linear read,-,-,linear variable X1)→
(clear-linear-variable-X1-next,-)
であり得る。前述のルール規約に基づくと、第2の記憶命令に対して、OP1は0であり、OP2はR2であり、(R2)はMtagとしてタグ付けされるメモリ位置である。第2のmove命令によりトリガされる第2のルールは、
move:(clear-linear-variable-X1-next,-,EMPTY,-,linear variable X1)→

(default tag,-)
であり得る。
【０３８１】
この例は、特定の命令シーケンスの不可分性を保証するためにタグおよびルールがどのように使用され得るかを示す。この一般的な技法は、命令の特定のシーケンスの一部として特定の方法でのみデータがアクセスされ得ることを実施するのが望ましい多くの他のシナリオにおいて適用され得ることを、当業者は容易に理解することができる。この技法は、命令シーケンスの厳密な実施が要求される任意の場合に対して採用され得ることも、当業者は容易に理解することができる。上で述べられたように、一般的な技法は、シーケンスの中の命令Nからの新しいPC(たとえば、PCnew tag)を、接着されたシーケンスの中の次の予期される命令N+1によりトリガされるルールにおいてPC tagとして確認される特別なタグでタグ付けすることを伴う。
【０３８２】
ここで説明されるのは、RISC-Vアーキテクチャに基づいて本明細書の技法に従ったある実施形態においてシステムをブートまたは始動することの一部として実行され得る技法である。以下の段落は、例900に関連するなどして本明細書の他の箇所で説明される様々なCSRを参照することがあり、ここでそのようなCSRはメタデータ処理領域に関連して使用されることがある。
【０３８３】
本明細書の他の箇所で説明されるように、ブートストラップタグは、ハードワイヤリングされることがあり、または特定のROM位置に記憶される値であることがある。システムをブートすることの一部として、様々なCSR、メモリなどの初期化を含む初期化を全般に実行する、ブートストラッピングコードのあるセグメントが実行され得る。初期化の一部として、そのような処理はまた最初に、初期のブートストラップタグから導出されるデフォルトタグ値でメモリ位置をタグ付けする。少なくとも1つの実施形態では、boottag CSR(たとえば、例900のようなsboottag CSR)などのCSRが、システムの中のすべての他のタグがそこから導出される初期の「シード」タグとして使用される、特別なブートストラップタグを用いて初期化され得る。ローダなどの様々なコードエンティティが、それらの命令が特別にタグ付けされるようにして(たとえば、CI tagが特別な命令タグに設定される)、それにより、特別な命令タグを有しない他のコードが行うことを許容されないタスクを実行するための特別な特権または権限を有するものとして、そのローダを指定することができる。上記は、CI tagが特別なタグであることを確実にするためにCI tagを調べるローダのコードによってトリガされるルールを使用して、そのようなトリガされるルールに所望のタグ付け動作を実行させるために実施され得る。したがって、たとえば、ローダの命令をタグ付けするために使用される特別なCI tagは、始動プロセスの一部としてコードを実行することによってトリガされる特別なルールの結果として、ブートストラップタグから生成または導出され得る。一般に、コードまたは記憶されている命令の何らかの部分がタグ付けされると、より望まれるタグを生成し、またそのような生成されたタグをコードおよびデータに付けるために、ルールがそのようなタグ付けされたコードの実行によりトリガされ得る。上記のおよび他の態様が以下でより詳細に説明される。
【０３８４】
システムの始動またはブートにおいて、tagmode CSR(たとえば、例900の901r)に記憶されているものなどのタグモードは最初はオフ(たとえば、例910の911a)であり得る。デフォルトタグCSR(たとえば、例900の901c)を特別なデフォルトタグ値にまず直接設定する、ブートストラップROMプログラムが実行され得る。続いて、ブートストラッププログラムは、tagmode CSRをあるモードに設定することができ、これにより、メタデータ処理領域は、default tag CSRに記憶されているようなデフォルトタグをすべての結果に書き込む。言い換えると、defaulttagタグモードにある間(たとえば、例900の911b)、PUMP出力Rtagは常にデフォルトタグ値である。
【０３８５】

続いて、メモリ位置がデフォルトタグで初期化されタグ付けされた後で、すべての他の後続のタグをさらに生成または導出するために使用されるタグの初期集合を生成するために、処理が実行され得る(たとえば、初期集合は、無制限にタグの1つまたは複数の他の生成を導出するためにさらに使用され得る)。そのような処理は、ルールにタグの初期集合を生成させる命令シーケンスまたはコードセグメントを実行することを含み得る。この場合、タグモードは、コードセグメントの実行の間にPUMPを関与させる適切なタグモードレベルに設定され得る。たとえば、例910を参照すると、ブートコードがハイパーバイザーモードで実行している場合、タグモードは、コードセグメントの実行の間にPUMPを関与させるために、911eに示されるようなx110または911fにより示されるようなx111のいずれかに設定されることがあり、これにより、ルールがコードセグメント命令の結果としてトリガされ実施される。

【０３８６】

上で述べられたコードセグメントを実行する前に、コードセグメントを検証または妥当性確認するために処理が実行され得ることに留意されたい。たとえば、少なくとも1つの実施形態では、上で述べられたコードセグメントは暗号化された形式で記憶されることがあり、ここで、実行の前に、コードセグメントが改竄または変更されていないことを保証するために、コードセグメントが復号され検証または妥当性確認される(たとえば、デジタル署名などを使用して)。

【０３８７】

さらに例示すると、ブートストラッププログラムは、PUMPが関与している間に実行され、それによりタグの初期集合を生成する、4つの命令を上で述べられたコードセグメントの中に含み得る。

1．R1←boottag CSRを読み取る
2．Add R2←R1+1
3．Add R3←R2+1
4．Add R4←R3+1

【０３８８】

上の命令1において、R1は汎用レジスタである。命令1はboottag CSRを読み取り、boottag CSRに記憶されている値とboottag CSRに記憶されているタグとの両方をR1に転送する。boottag CSRは、プロセッサのリセットの間、またはCSRのタグを含むCSRの特権モード書込みにより、特定のタグを保持するように設定された。boottag CSRからの読取りはまた、ブートの間のこの初期の取出しの後に取り出されることが可能ではないように、boottag CSRをクリアし得る。

【０３８９】

「Add Rn←Ry+1」という形式の上記の命令2～4を形成するadd命令の各々において、RnはAddの結果を記憶するためのターゲットまたは結果レジスタを示し、Ryもソースオペランドのレジスタを示す。上記のコードセグメントの命令2は、第1のタグから第2のタグを生成してR2により指されるメモリ位置に第2のタグを付ける、第2のルールをトリガし得る。上記のコードセグメントの命令3はさらに、第2のタグから第3のタグを生成してR3により指されるメモリ位置に第3のタグを付ける、第3のルールをトリガし得る。上記のコードセグメントの命令4はさらに、第3のタグから第4のタグを生成してR4により指されるメモリ位置に第4のタグを付ける、第4のルールをトリガし得る。このようにして、上記のコードセグメントは、レジスタにタグ値として記憶される4つのタグの初期集合を生成するために使用され得る。上記の一般的な技法は、初期集合の任意の所望の数のタグを生成するために同様の方式でさらに拡張され得る。

【０３９０】

一般に、少なくとも1つの実施形態においてタグの初期集合を生成する際、初期集合の中のタグの具体的な数は事前に定義された数であり得る。特別なタグの各々が、命令を実行するときにトリガされる異なる固有のルールの結果として生成され得る。上のコードセグメントの中などにある各命令は、キャッシュミスをもたらし、それにより、特定の命令の

ためのルール出力の一部としてRtagを計算するためにキャッシュミスハンドラの実行をもたらすことがあり、ここでRtagは初期集合のタグのうちの1つである。上のコードセグメントの命令と同様の方式で、初期集合のタグのうちの1つを使用して他のタグをさらに生成するために、異なる時点において異なるコードシーケンスが実行され得る。したがって、初期集合の中の各タグが、タグの別のシーケンスをさらに生成するために使用されるタグ生成源を示し得る。上記の例では、Add命令は、タグの別のシーケンス全体を生成するために使用され得る次のタグ生成源を生成する際に使用され得る。下で論じられるように、初期集合のタグ生成源(これ自体が別のシーケンスを生成するための始点として使用されるさらなるタグ生成源である)は、タグの別のシーケンスを生成するために生成源としてさらに使用することができない普通のタグまたは非生成タグとは区別され得る。したがって、ADDなどの特定の命令が、ルールおよびミスハンドリングをトリガしてタグ生成源の集合またはシーケンスを生成するために使用され得る。これは、MOVEなどの別の命令とは対照的であることがあり、MOVEはシーケンスの中の非生成タグを生成するためにルールおよびミスハンドリングをトリガし得る。mallocなどのコードに関連して、ADD命令が同様に、第1のアプリケーションのための異なる色のシーケンスを生成するために使用される新しいアプリケーションタグ色生成源を生成するために使用され得る(たとえば、新しいアプリケーションタグ色生成源は、特定のアプリケーションのための異なる色RED-APP1、BLUE-APP1、GREEN-APP1などのシーケンスを生成するために使用されるAPP1であり得る)。次いで、タグ付けされたADD命令が、RED-APP1-gen、BLUE-APP1-gen、またはGREEN-APP1-genのうちの1つなどの、特定のアプリケーション固有のシーケンスの中の次のタグを取得するために使用され得る。次いで、RED-APP1-gen、BLUE-APP1-gen、またはGREEN-APP1-genからそれぞれ、実際の色RED-APP1、BLUE-APP1、またはGREEN-APP1を生成するために、タグ付けされたMOVE命令が使用され得る(ここで、追加のタグシーケンスをさらに生成するために、RED-APP1、BLUE-APP1、GREEN-APP1を使用することはできない)。

【０３９１】

PUMPが関与している間に実行されるブートストラッププログラムのコードセグメントはまた、実行されると、カーネルコード/命令をタグ付けし、加えて任意の所望の特別な命令タグで他のコードモジュールまたはエンティティをタグ付けして、所望の特権または能力をそのような特別にタグ付けされたコードが有することを可能にするための、ルールをトリガする追加のコードを含み得る。たとえば、コードセグメントは、ルールにローダコードをタグ付けさせる命令と、特権的なタグ付け動作を実行するための特権または権限をルーチンmallocおよびfreeのコードに拡張する特別な命令タグを伴うルーチンmallocおよびfreeのコードとを含み得る。特別なコードタグは、ルールにさらに所望のタグを生成させ、また追加のコードおよび/またはデータを生成されたタグで適切にタグ付けさせる、命令の所定のコードシーケンス/集合を使用して、上で述べられたような方式と同様の方式でタグの初期集合から生成され得る。

【０３９２】

少なくとも1つの実施形態では、上で述べられたコードセグメントの部分に関連して、追加の対策または技法が採用され得る。たとえば、タグの初期集合を生成するために使用される上で述べられた4つの命令は、本明細書の他の箇所(たとえば、例1400)で説明されるものなどの、連続性およびアトミック性を実施させるために、「接着」ポリシーのルールを使用して第1の命令シーケンスに含まれ得る。

【０３９３】

上で述べられたコードセグメントが、タグの初期集合を生成するために、かつカーネルコードおよび任意の他の所望の命令をさらに特別にタグ付けするために実行された後で、制御は追加のブートコードに移転され得る。RISC-Vアーキテクチャに基づく少なくとも1つの実施形態では、追加のブートコードはハイパーバイザー特権レベルで実行され得る。そのような追加のブートコードは、たとえば、PUMPへのルールの初期集合のロードをトリガする命令を含み得る。ブートが完了すると、tagmode CSRにより示されるようなPUMP

タグモードは、ユーザ特権レベルで実行するなど、ユーザコードに関連してPUMPを関与させるのに適切なレベルに設定され得る(たとえば、PUMPが関与していることと、U(ユーザ)モードまたは特権レベルのみでの動作とを示すために、例910の911cのようにタグモードを設定する)。

【０３９４】

図72を参照すると、本明細書の技法に従ったある実施形態において実行され得る処理ステップのフローチャートが示されている。フローチャート1600は、上で説明された処理を要約する。ステップ1602において、タグモードはオフに設定され、ここで、tagmode CSRは例910の911aに関連して本明細書の他の箇所で説明されるようなPUMPオフ状態を示す。ステップ1604において、boottag CSRが特別なブートストラップタグに初期化される。ステップ1606において、ブートストラッププログラムの実行が開始される。ステップ1608において、ブートストラッププログラムはdefaulttag CSRをデフォルトタグに設定し得る。ステップ1610において、tagmode CSRはすべての結果にデフォルトタグを書き込む(たとえば、このタグモードにある間は各Rtag=デフォルトタグ)モードに変更され得る。ステップ1612において、ルールにメモリ位置を初期化させてデフォルトタグでメモリ位置をタグ付けさせる、命令が実行され得る。ステップ1614において、tagmode CSRは、ステップ1616における後続のコードセグメントの実行の間にPUMPを関与させるモードに変更され得る。ステップ1616において、PUMPが関与した状態で後続のコードセグメントが実行される。コードセグメントは、ルールに、タグの初期集合を生成させ、boottag CSRをクリアさせ、カーネルコードをタグ付けさせ、特別なコードタグで追加のコード部分をタグ付けさせてそのようなタグ付けされるコードに所望される通りの拡張された能力、権限、および特権を与える、命令を含む。ステップ1618において、実行される追加のブートコードに制御が移転され得る。ブートプロセスが完了すると、システムはユーザコードを実行する準備ができ、PUMPが実行中のユーザコードに対して関与しており動作可能である

【０３９５】

ここで詳細に説明されるのは、ブートストラップタグからタグをどのように生成するかである。ブートストラップタグを用いて開始するタグ生成処理は、タグツリーまたはツリーオブライフ(tree of life)とも呼ばれ得る。より一般的には、タグ生成処理は、図73の例1620に示されるような階層構造を形成する。

【０３９６】

例1620は、タグ生成プロセスのルートとしてboottag1621を示す。要素1621a～dは、上で説明されたように生成されるものなどの、タグの初期集合を示し得る。この例では、タグの初期集合1621a～dは、無限の数の特別な命令タグのシーケンス1622をさらに生成するために使用される初期OS特別命令タグ1621aを含むことがあり、この特別な命令タグは次いで、異なるコード部分またはモジュール1624の命令をタグ付けするために適用されることがある(1623)。初期OS特別命令タグ1621aから、タグ付けされるべき様々なモジュールのために追加のタグ1622が生成され得る。たとえば、第1のOS特別命令タグ1 1622aは、mallocコードに適用される(1623a)mallocのために生成されることがあり、これにより、mallocの命令は特別命令タグ1 1622aでタグ付けされる(1624a)。このようにして、mallocコードは、mallocをタグ生成源として特定する(たとえば、他の新しいタグをさらに生成し、新しく生成されたタグをさらに使用して他のメモリセルをタグ付けする特権を、mallocコードが有することを示す)特別命令タグでタグ付けされ得る。

【０３９７】

mallocに関するこの例では、1621bは、mallocタグ生成源アプリケーションタグ1626をユーザアプリケーションごとに1つさらに生成するために使用される初期mallocタグであることがあり、ユーザアプリケーションごとに1つなのは、mallocのインスタンスが各ユーザアプリケーションに含まれるからである。1625に含まれるような様々な色付けされたタグを生成する特権を、各ユーザアプリケーションの中の各々のそのようなmallocインスタンスに与えたい。

【０３９８】
一般に、例1620は、特別命令タグ1621a、Malloc1621b、CFI1621c、およびテイント1621dのためのタグの初期集合1621a～dを示す。したがって、第1の行の中のタグの垂直な陳列の中のタグ1621a～dの各々(boottag1621以外)は、無限のタグシーケンスを生成するために使用される異なる初期タグを示す。たとえば、値1621aは、無限の数の特別命令タグ1622をさらに導出または生成する際に使用される。値1621bは、無限の数の値1626をさらに導出または生成する際に使用される。1626の各インスタンスはさらに、各アプリケーションのためのタグの別の無限のシーケンスの生成源として使用され得る。たとえば、1626aは、単一のアプリケーションapp1のために使用される異なる色の別の無限のシーケンス1629をさらに生成するために使用される生成源値を示す。同様の方式で、1626の各々の異なる生成源値は、各アプリケーションのために無限の数の色をさらに生成するために使用され得る。

【０３９９】
値1621cは、無限の数の値1627をさらに生成する際に生成源として使用され得る。要素1627は、特定のアプリケーションまたはアプリNに対するCFIタグ生成源nの各々の発生が別の無限のシーケンスをさらに生成するための特権または能力を示すという点で、1626と同様である。たとえば、1627aは、単一のアプリケーションapp1のために使用される様々な色の別の無限のシーケンス1630をさらに生成するために使用される生成源値を示す。同様の方式で、1627の各々の異なる生成源値が、各アプリケーションに対して無限の数の色をさらに生成するために使用され得る。

【０４００】
値1621dは、無限の数の値1628をさらに生成する際に生成源として使用され得る。要素1628は、特定のアプリケーションまたはアプリNに対するタグ生成源nの各々の発生が別の無限のシーケンスをさらに生成するための特権または能力を示すという点で、1626および1627と同様である。たとえば、1628aは、単一のアプリケーションapp1のために使用される様々な色の別の無限のシーケンス1631をさらに生成するために使用される生成源値を示す。同様の方式で、1628の各々の異なる生成源値は、各アプリケーションのために無限の数の色をさらに生成するために使用され得る。

【０４０１】
示されるように、1621c～dからそれぞれ出てくるCFIおよびテイントのためのシーケンスまたはサブツリーは、1621bから出てくるMallocサブツリーと同様である。例1620において、nxtTagまたはTInxtTagが、生成される無限のシーケンスにおける次の要素を示すために使用され、getTagが、シーケンスメンバーから次のタグを抽出するために使用される。一般に、getTagは、それ自体はタグ生成源ではない使用すべきタグを抽出することを示すために使用され得る。使用可能なタグが使用すべき具体的なコード部分に与えられることになる場合、タグを生成する能力もそのコード部分に与えることは望まない。たとえば、各アプリケーションにそのアプリケーションのためのMallocタグ生成源(たとえば、App1ColorTagX)を与えることを望むが、そのアプリケーションに他のアプリケーションのためのMallocタグ生成源を生成する能力を与えることは望まない。よって、getTagはタイプを生成源からインスタンスに変更する。nxtTagとTInxtTagとの区別は、nxtTagは「タグ付けされた命令」なしで使用可能であるが、TInxtTagは適切にタグ付けされた命令によってのみ使用可能なものであるということである。

【０４０２】
Mallocアプリケーションタグシーケンス1626は、オペレーティングシステムまたはローダが各アプリケーションのための色タグ生成源を生成することを許容する。たとえば、要素1626aは、アプリケーション色シーケンス1629のタグを生成するために使用される、アプリケーション固有の色タグ生成源値を示す。アプリケーション内で、AppYColorTagシーケンス1629は、mallocが各色のための権限を生成することを可能にする。その色の権限は、割り振られたメモリのためにセルを色付けすること、割振りのためにポインタを色付けすること、その色のセルを解放すること(たとえば、freeが呼び出されるとき)のた

めに使用され得る。mallocおよびfreeなどともに色を使用することは、本明細書の他の箇所で説明される。

【０４０３】

このようにして、異なるタグが異なる使用のために確保され得る。上で述べられたような最初にタグ付けされるカーネル命令から、異なる能力または権限で他のコード部分をさらにタグ付けする、カーネルコードが実行され得る。たとえば、オペレーティングシステムのカーネルコードはさらに、他のコードおよびデータをさらにタグ付けすること、追加のタグ生成源を生成することなどのための能力をローダに付与することなどの、特別な特権でローダなどの他のコードエンティティをさらにタグ付けし得る。mallocを含むユーザプログラムをロードするときのローダは、異なるメモリ領域を色付けするために使用される他のタグをさらに生成するための能力をユーザプログラムに与える、ユーザプログラムをコードとして示す特別な命令タグでmallocコードをさらにタグ付けすることができる。したがって、ローダのコードに付けられる特定の命令タグは、特権の1つの集合をローダに与える。mallocコードに第2の異なる命令タグを付けることは、特権の別の異なる集合をmallocコードに与える。一般に、シーケンスのタグ生成を実行するとき、シーケンスの中の現在のタグは、シーケンスの中の次のタグを生成することに関連して参照され使用される状態情報として保存される。本明細書で説明されるように、シーケンスの中の現在のタグに関するそのような状態情報は、メタデータ処理領域に保存され使用され得る。現在のタグ、またはより一般的にはメタデータ処理状態情報は、ルール処理およびキャッシュミス処理の結果として保存され復元され得る。特定のアプリケーションのために使用するために割り振られる最後の色などの、シーケンスの中の現在のタグは、指定されるメモリ位置のタグとして、シーケンスの現在の状態として保存され得る。アプリケーションのための新しい次の色が割り振られる必要があるとき、mallocのコードは、アプリケーションの最後の割り振られた色を取り出し、最後の割り振られた色を使用してアプリケーション固有の色シーケンスの中の次の色を判定する、ルールをトリガし得る。一般に、タグの固有のシーケンスを生成することは、以下のことを実行するルールをトリガする命令を実行することを含み得る。

1．アトム(たとえば、レジスタ、メモリ位置)のタグ部分にシーケンス状態を記憶/保存すること、

2．保存/記憶されたシーケンス状態を使用してシーケンスの次のタグを生成するルールをトリガする命令を実行すること、および、

3．アトムのタグ部分に(2から生成された)シーケンスの次のタグを記憶/保存すること、

ここで次のタグはシーケンスの更新された現在の状態である。

【０４０４】

例1620に戻って参照すると、ローダは、mallocを使用する各アプリケーションに対して、1626のmallocタグ生成源アプリケーションタグのうちの特定の1つを割り振り得る。ローダは、たとえば、1626aなどの次のmallocタグ生成源タグを生成し、次いでこのタグをメモリ位置のタグ付けを介して状態情報として記憶する、ルールをトリガするコードを実行することができる。続いて、アプリケーションによるmallocへの最初の呼出しに際して、保存されたmallocタグ生成源タグを取り出し、保存されたタグを使用してアプリケーションのための第1の色を生成し、次いで、アプリケーションのために生成された最後のまたは最新の色として第1の色を記憶するように保存された状態情報を更新する、ルールをトリガするmallocのコードが実行することができる。アプリケーションによるmallocへの2回目の呼出しに際して、以前に保存された第1の色を取り出し、保存された第1の色を使用してアプリケーションのための第2の色を生成し、次いで、アプリケーションのために生成される最後のまたは最新の色として第2の色を記憶するように保存された状態情報を更新する、ルールをトリガするmallocのコードが実行することができる。同様の方式で、mallocへの他の後続の呼出しが、他のルールに、アプリケーションのための保存された状態情報(たとえば、直近に割り振られた色)に基づいて追加の色を割り振らせ得る。

【０４０５】

ここで説明されるのは、本明細書の技法に従ったある実施形態に含まれ得る、ダイレクトメモリアクセス(DMA)アーキテクチャの態様である。一般に、以下の段落で説明されるのは、タグ付けされたデータを使用する第2のインターコネクトファブリックのメモリに記憶されているデータにアクセスするために、タグ付けされていないデータを使用する第1のインターコネクトファブリックに接続される信頼されていないデバイスなどのソースから発行されたDMAを仲介するための、I/O PUMPの使用である。

【０４０６】

図74を参照すると、本明細書の技法に従ったある実施形態に含まれ得る構成要素の例が示されている。例1500は、例700の構成要素と同様に番号が付けられた構成要素と、本明細書の他の箇所で説明される他のもの(たとえば、図57～図60)とを含む。加えて、例1500はまた、I/O PUMP 1502と、メモリ712cに記憶されているデータにアクセスするためにDMA要求を発行し得るDMA要求ソースまたはイニシエータ1504a～cという追加のアクターとを含む。例1500は、イーサネット（登録商標）DMAデバイスA1504a、イーサネット（登録商標）DMAデバイスB1504b、およびタグ付けされていないファブリック715に接続されるUART(汎用非同期受信機/送信機)またはシリアル通信デバイス1504cを含む。データを読み取りまたは書き込むためのDMA要求は、デバイス1504a～cのうちの1つから出てくることがある。この要求はI/O PUMP 1502に送られ、I/O PUMP 1502は、DMA要求が許容されるかどうかを判定するための処理を実行し、許容される場合、要求が進行することを許容する。したがって、I/O PUMP 1502は、タグ付けされていないファブリック715から受信されるDMA要求を仲介するものとして特徴付けられることがあり、これにより、一般的な仮定は、そのようなDMA要求を発行する715に接続されるデバイスは信頼されないことがあるというものになる。

【０４０７】

少なくとも1つの実施形態では、I/O PUMP 1502は本明細書(たとえば、図22)で説明されるようなPUMPの具体化であることがあり、違いは、実施されるルールがメモリ1712cへのDMAアクセスを制御するDMAポリシーのルールであるということである。I/O PUMP 1502の上記の使用法は、メモリ動作を含むすべての命令がルールによって仲介されることを保証することの一般的なアーキテクチャに則している。自律的なDMAデバイス1504a～cがメモリへの直接の仲介されないアクセスを許容された場合、DMAデバイス1504a～cは、ルールが実施している不変条件および安全性プロパティを損なうことがある。その結果、DMAを許容するために、本明細書の技法に従ったある実施形態はまた、メモリ712cへのDMAアクセスについてのルールを実施することがある。プロセッサ命令のためのルールを実施するPUMPに類似して、I/O PUMP 1502は、1504a～cなどのDMAデバイスからのメモリのロードおよび記憶のためのルールを実施する。一般に、I/O PUMPはすべてのロードおよび記憶を仲介する。RISC-Vアーキテクチャに基づいて本明細書で説明される少なくとも1つの実施形態では、I/O PUMPは、RISC-Vアーキテクチャにおいて使用されるPUMPに関連して本明細書の他の箇所で説明されるような方式と同様の方式で、CSRを使用してルールキャッシュミスハンドリングを実行する。I/O PUMP 1502は、PUMPと同様のCSRの集合を有するが、メモリのマッピングされたアドレスを介してそれらにアクセスする。例1520に関連して以下の段落で説明されるものなどの、I/O PUMP CSRへのアクセスは、ルールを使用して保護されるタグでもあり得る。I/O PUMPにおいてルールを見つけることを試みるときに遭遇するルールキャッシュミスは、プロセッサRISC-V CPU 702によりサービスされることになる中断をトリガする。I/O PUMPは、同じルール解決プロセスをプロセッサ702として使用するが、メモリ712cの中のデータにアクセスするためのDMAのロードおよび記憶のためのルールだけを含む単一のDMAポリシーがある。I/O PUMPは、メモリ712cへアトミックに書き込む(たとえば、単一のアトミックな動作としてタグおよび値を書き込む)。しかしながら、いくつかの実施形態では、Mtagを読み取ることからMtagを書き込むことへの完全なプロセス(たとえば、タグの確認または検証を実行して書き込むための処理)は、標準的な記憶を用いるとアトミックではないことがある。

【０４０８】
I/O PUMP 1502はSDMPのためのルールキャッシュである。I/O PUMPは、DMA動作に関与するタグの集合と動作の結果との間のマッピングを提供する。少なくとも1つの実施形態では、I/O PUMPはプロセッサ702とは独立に実行する。I/O PUMP 1502はキャッシュであるので、前に入力の集合を見たことがなかったとき(強制的)、またはルールを維持することが可能ではなかったとき(容量、または場合によっては競合)、ミスが発生する。これは、PUMPについて本明細書で説明されるようなルールキャッシュミスと同様の方式で、I/O PUMPに関するルールキャッシュミスをもたらす。I/O PUMPルールキャッシュ1502に関するミスは中断を引き起こし、この中断は次いで、プロセッサ702のミストラップをサービスするものと同じルールキャッシュミスハンドラシステムによってソフトウェアにより扱われる。I/O PUMP 1502に関するルールミスに際して、以下で説明されるI/O PUMP CSR(たとえば、例1520)を通じて入力がミスハンドラ(メタデータ処理領域においてプロセッサ702のコードで実行されるものなど)に伝えられ、ルール挿入がCSRを通じてI/O PUMPに戻される。I/O PUMPミスは、I/O PUMPをプロセッサ702によりサービスされるまで無効にする。少なくとも1つの実施形態では、I/O PUMPの無効状態は、I/O PUMPにより仲介されるすべてのDMA転送が、I/O PUMPミスがサービスされるまでストールすることを意味する。
【０４０９】
PUMPについての本明細書の他の箇所の議論と一貫して、I/O PUMP入力は、オペグループ(opgrp)、DMA命令およびそのオペランドのためのタグ(たとえば、PCtag、CI tag、OP1 tag、OP2 tag、Mtag(本明細書ではMRtagとも呼ばれることがある))を含む。I/O PUMP出力は、本明細書で説明されるようなRtagおよびPCnew tag(次の命令のPCのためのタグ)を含み得る。I/O PUMPに関連して、そのような入力および出力はさらに、一実施形態において以下で説明されるような意味および値を有し得る。
以下が、一実施形態におけるI/O PUMP入力である。
1．Opgrp-現在は2つある:ロードおよび記憶
2．PCtag-DMA IOデバイスの状態(コードのためのPCtagに類似する)
3．CItag-DMA IOデバイスを特定するタグ(コードの指定された領域の命令タグに類似する)
4．OP1tag-常に「公開であり信頼されない」(IOPUMPキャッシュにおいて物理的に表されないが、ルールのために使用される)と仮定する
5．OP2tag-OP1tagと同じ
6．Mtag-DMA動作へのメモリ入力のタグ
7．byteenable-どのバイトが読み取られている/書き込まれているか?
以下が、一実施形態におけるI/O PUMP出力である。
8．Rtag-記憶のためのメモリ結果のタグ
10．PCnew tag-この動作の後のDMA I/Oデバイスの状態
【０４１０】
I/O PUMPでは、プログラム可能なオペグループマッピングテーブル(たとえば、420)はないことがある。むしろ、ルールを探すためにI/O PUMPにより使用されるオペグループは、DMAロード動作およびDMA記憶動作のための単一のオペグループを示す固定されたオペコードであり得る。少なくとも1つの実施形態では、I/O PUMPのためのケアマスキングはない。
【０４１１】
図22などにおいて、本明細書で説明されるように、PUMPに関連してルールキャッシュミスがあるとき、プロセッサ702が自動的に、ミスを引き起こした命令を、その対応するルールがPUMPルールキャッシュへと挿入された後に再び発行することが期待され得る。結果として、ルール挿入は単に、PUMPキャッシュの中にルールを置き、タグ付けされた結果を得るために命令が再び発行されることを期待する。しかしながら、DMA動作についての挙動は上記のことから変化する。DMA動作は、中断されて再試行動作を必要とするもの

とは予想されていない。これらのDMA動作をサポートするために、ルール挿入はI/O PUMPに対して異なるように扱われ得る。具体的には、I/O PUMPがミスによりフォルトすると、処理は保留中のDMA動作を保持し、プロセッサ702(たとえば、ルールミスハンドリングを実行して新しいルールのための出力タグRtagおよびPC new tagを計算する)が(そのことが許容されると想定して)ルールのための欠けている出力タグを供給するのを待つことがある。出力が供給されると、IOPUMPへのルールの書込みをトリガすることに加えて、動作がI/O PUMPに再び発行されることを強制することなく動作が継続できるように、出力がI/O PUMPから来たかのように、出力がDMAパイプライン(たとえば、下の例1540に関連して説明される)に転送される。ルールの侵害は、動作が許容されないことをシグナリングする、更新されたPCtag、PCnew tagのための指定されるdisabled-DMA-deviceタグを供給することによって扱われることがあり、その特定のDMAデバイス1504a～cからのさらなるDMA動作は、そのPCtagがリセットされるまで許容されない。一般に、DMA動作または要求を発行する、1504a～cのうちの1つなどの特定のDMAデバイスに対するデバイスタグは、発行側のDMAデバイス(たとえば、DMA要求のソース)を一意に特定するCIの特定の値、およびDMAデバイスの現在の状態を示すPC tagであり得る。少なくとも1つの実施形態では、PC tagは、CI tagにより特定される特定のDMAデバイスからのDMA要求のさらなる処理を無効にするある特定の値に、ある時点において設定され得る。

【０４１２】

図75を参照すると、本明細書の技法に従ったある実施形態においてI/O PUMPにより使用され得るCSRのテーブルが示されている。テーブル1520は、アドレス列1524(CSRのメモリマッピングされたアドレスを示す)、名前列1526、および説明列1528を含む。テーブル1520の各行は、I/O PUMPにより使用される定義されたCSRのうちの1つに対応する。行1522aは、CSR transaction idがアドレス0x00を有することを示す。transaction id CSRへの書込みは、記憶されている現在のtransaction id(たとえば、プリフェッチのための)をインクリメントし、transaction id CSRからの読取りは、transaction id CSRに記憶されている現在のtransaction idを返す。行1522bは、CSR opgrpがアドレス0x01を有することを示す。opgrp CSRは、現在のDMA命令のためのオペグループを格納し、ルールミスハンドラへの入力としてルールミスの際に使用される。行1522cは、CSR byteenableがアドレス0x02を有することを示す。byteenable CSRは、ワードの中のバイトのいずれにDMA動作が影響するかを示し、ルールミスハンドラへの入力としてルールミスの際に使用される。本明細書の他の議論と一貫して、このことは、バイトレベルの保護をポリシーが提供することを可能にし、トリガされるルールは、DMA要求されたデータのバイトへの、要求を開始する特定のDMAデバイスによるアクセスが、異なるDMAデバイスがアクセス可能であるメモリ部分を特別にタグ付けすることなどによって可能にされることを確実にするために、確認を行い得る。行1522dは、CSR pctagがアドレス0x03を有することを示す。pctag CSRは、現在のDMA命令のためのPC tagを格納し、ルールミスハンドラへの入力としてルールミスの際に使用される。行1522eは、CSR citagがアドレス0x04を有することを示す。citag CSRは、現在のDMA命令のためのCI tagを格納し、ルールミスハンドラへの入力としてルールミスに際して使用される。行1522fは、CSR mtagがアドレス0x07を有することを示す。mtag CSRは、現在のDMA命令のためのM tagを格納し、ルールミスハンドラへの入力としてルールミスの際に使用される。行1522gは、CSR newpctagがアドレス0x08を有することを示す。newpctag CSRは、現在のDMA命令の完了(たとえば、PUMPの出力およびキャッシュミスハンドリング)の後でPCに付けられるPC new tagを格納する。行1522hは、CSR rtagがアドレス0x09を有することを示す。rtag CSRは、現在のDMA命令のメモリ結果(たとえば、PUMPの出力およびキャッシュミスハンドリング)に付けられるタグを格納する。行1522iは、CSR commitがアドレス0x0Aを有することを示す。commit CSRへの書込みは、commit CSRに書き込まれた値と現在のtransaction id(transaction id CSRに記憶されているような)との比較をもたらす。上記の2つが一致する場合、この一致はI/O PUMPへのルールの書込みをトリガする。書き込まれるルールは、現在のDMA命令のためのオペコードならびにタグの入力

および出力(ミスハンドリングにより判定されるような)を含む。行1522jは、CSR statusがアドレス0x0Eを有することを示す。status CSRは、I/O PUMPのステータスを示す値を格納する。たとえば、本明細書で説明されるような一実施形態では、status CSRは、I/O PUMPが有効であるか無効であるかを示し得る。I/O PUMPは、本明細書の他の箇所で説明されるようなPUMP I/Oルールキャッシュミスの場合には無効にされ得る。行1522kは、CSR flushがアドレス0x0Fを有することを示す。flush CSRは、書き込まれると、I/O PUMPのフラッシュ(たとえば、I/O PUMPキャッシュからのルールをフラッシュまたはクリアする)をトリガする。

【０４１３】

少なくとも1つの実施形態では、status CSRのビット0が1である場合、これは、I/O PUMPが無効であることを意味し、そうではなくビット0が0の値を有する場合、これはI/O PUMPが無効であることを意味する。PUMP I/Oミスはpumpを無効にする。status CSRのビット1は、PUMPがフォルトし、サービスを待機しているかどうかを示す(たとえば、ビット1=1は、I/O PUMPのフォルト/キャッシュミスおよびサービスの待機を示唆する)。status CSRのビット2は、I/O PUMPルールミスが現在ルールキャッシュミスハンドラにより解決されているかどうかを示し、transaction idが一致する場合、挿入された結果を保留中のミス動作に直接提供する。status CSRのすべての上記のビットは、(成功するまたは不成功の)コミット動作によりリセットされる(たとえば、ビット0=有効、ビット1=フォルトなし、ビット2=保留中のミスなし)。status CSRへの書込みはまた、たとえば、I/O PUMPを最初に有効にするために、始動時に必要に応じて上記のビットをリセットするために実行され得る。不成功の書込みに対するstatus CSRのリセットは、DMAデバイスが動作を再試行してフォルトを再びトリガすることを可能にする。

【０４１４】

I/O PUMP CSRへのプロセッサ702によるロード/記憶メモリ動作は、iopump CI tagでタグ付けされるべきである。ポリシールールは、iopump CI tagを有する命令に動作を制約するために実施されるべきである。個々のI/O PUMP CSRはタグを持たない。

【０４１５】

タグ付けされていないまたは信頼されないファブリック715上の各デバイス1504a～cは、プロセッサがデバイスへのロードまたは記憶を実行するときにデバイスタグとして提示される固有のタグを用いて構成され得る(たとえば、デバイスタグが下で説明されるデバイスレジスタファイルに記憶され、特定のデバイスがDMAのロードまたは記憶を実行するときにCI tagとして指定される、1534b参照)。これは、どのコードおよび権限がどのデバイスに直接アクセスできるかを、細かい粒度で制御することを可能にする。同じタグがデバイスへのすべてのロードおよび記憶に際して提示され、タグはロードおよび記憶動作に基づいて変化しない。各デバイス1504a～cと関連付けられ、これらを特定する特定のデバイスタグは、デバイスレジスタファイルに記憶され得る。デバイス1504a～cのために指定される特定のデバイスタグは、デバイスレジスタファイルを修正することによってのみ変更され得る。デバイスレジスタファイルは、各デバイス1504a～cに対して、固有のターゲットデバイスid(タグ付けされていない、または信頼されないファブリック715上のデバイスを特定するために使用される)と、固有のターゲットデバイスidに対するターゲットデバイス固有タグとを示し得る。少なくとも1つの実施形態では、デバイスレジスタファイルは、それ自体が、固有のデバイスタグを伴う信頼されないファブリック715上のデバイスとしてアクセスされ得る。デバイスレジスタファイルの使用をブートストラップするために、デバイスタグレジスタファイルの固有のタグ(デバイスレジスタファイルに記憶される)は、PUMPが有効にされる前に始動の間にファイルへ書き込まれ得る。たとえば、デバイスタグレジスタファイルの固有のタグは、PUMPがオフである間にブート処理の一部としてファイルに書き込まれ得る(たとえば、例910の911aによるtagmodeの表記)。命令のCI tagは、ロードまたは記憶命令を実行するDMAターゲットデバイスのターゲットidを特定することができ、ここでCI tagは、指定されるDMAデバイスによる特定のロードまたは記憶動作を制約する(たとえば、許容する、または許容しない)ために、そのような

ロードおよび記憶動作によりトリガされるルールにおいて使用され得る。加えて、特定のDMAデバイスが許容されないロードおよび/または記憶動作を実行する場合、特定のDMAデバイスと関連付けられる状態は、さらなる要求(たとえば、DMAのロードおよび記憶)が無視されるように、無効へと変更され得る。

【０４１６】

上で述べられたように、DMA要求もしくは命令を開始する、またはそのソースであるDMAデバイスは、DMAデバイスのPCtagにより示される関連するステータスを有し得る。具体的には、固有のPCtagが、DMAデバイスから許容されるDMA動作(CI tagにより特定される)に関して、無効ステータスを示すために使用され得る。無効にされたイニシエータは、自身のDMA要求が、下で説明されるDMAまたはTrustbridgeパイプライン(たとえば、例1530および1540)の最初に拒絶される。

【０４１７】

ある実施形態は、すべてのDMAトラフィックを仲介する単一のI/O PUMP、DMAエンジンごとのI/O PUMP、または複数のDMAエンジンのためのDMAトラフィックを仲介する複数のI/O PUMPを有し得ることに留意されたい。例1510に示されるのは、単一のDMAエンジン(たとえば、単一のメモリ712c)のための単一のI/O PUMPである。例1500のように単一のI/O PUMPを使用することは、ボトルネックになることがあるので、ある実施形態は、複数のI/O PUMPにI/Oトラフィックを仲介させることを選ぶことがある。複数のI/O PUMPがあるそのような実施形態では、各々が独立に有効または無効にされ得るので、複数のI/O PUMPの1つまたは複数の第1の部分が無効にされ得る(I/O PUMPミスが原因で)としても、複数のI/O PUMPの残りの第2の部分が有効にされ、DMA要求をサービスすることを続けることがある。

【０４１８】

少なくとも1つの実施形態では、DMA動作のイニシエータまたはソースとして活動する異なるDMAデバイスが各々、メモリ712cの指定される部分にのみアクセスすることを許容され得る。DMAを介してアクセス可能なメモリ712cの異なる部分は各々、別個のタグでタグ付けされ得る。たとえば、デバイス1504aはメモリ712cのアドレスの第1の範囲へのアクセス権を有することがあり、デバイス1504bはメモリ712cのアドレスの異なる第2の範囲へのアクセス権を有することがある。第1の範囲に対応する712cのメモリ位置は第1のタグでタグ付けされることがあり、第2の範囲に対応する712cのメモリ位置は第2のタグでタグ付けされることがある。このようにして、デバイス1504aのアクセスを第1の範囲の中のメモリ位置に強制または制約し、デバイス1504bのアクセスを第2の範囲の中のメモリ位置に強制または制約するために、ルールが使用され得る。変形として、異なるタグが許容されるアクセスのタイプ(たとえば、読取り専用、書込み専用、読取りおよび書込み)と関連付けられ得る。同様の方式で、同じメモリ712cにアクセスする複数のDMAエンジンを有するある実施形態では、DMAエンジンの各々が排他的にアクセス可能な単一のメモリ712cの異なる部分が一意にタグ付けされることがあり、それにより、ルールは各DMAエンジンのアクセスをメモリ位置の指定されるアドレス範囲に強制または制約する。

【０４１９】

図76を参照すると、本明細書の技法に従ったある実施形態における、信頼されるファブリック1532(たとえば、タグ付けされたインターコネクトファブリック710に対応する)と信頼されないファブリック1536(たとえば、タグ付けされていないインターコネクトファブリック715に対応する)との間のデータフローを示す例が示されている。要素1534は一般に、1532と1536との間でのDMAの仲介に関連してI/O PUMP 1534aにより実行される処理を表す。要素1534は、DMA仲介の一部としてDMA動作を検証してサービスするために実行される、トラストブリッジまたはDMAパイプライン1534cを示し得る。要素1538aは、信頼されないファブリック1536から(たとえば、例1500のDMAデバイス1504a～cなどへ)の出力チャネルを示し得る。要素1538bは、信頼されないファブリック1536への(たとえば、デバイス1504a～cのうちの1つからの)入力チャネルを示し得る。一般に、I/O PUMP 1534aは、ターゲットメモリのタグが要求されたDMAアクセスを許容すること

を検証するために、DMA読取りおよび書込み動作の間に読取り要求を発行する必要がある。I/O PUMPは、(処理段階と処理段階の間に例1540において以下で説明されるように)要求をバッファリングして、タグ付けされた通信動作の主制御を実行する必要がある。

【０４２０】

要素1537は、例1520において説明されるようなI/O PUMP CSRをロードする(または取り出す)ために入力として与えられる値を示す。加えて、異なるDMAデバイスイニシエータに対するデバイス状態情報は、DMAデバイスイニシエータ(たとえば、信頼されないファブリック715上の1504a～cなど)のための、PCtag(たとえば、このDMAデバイスからの要求が無効であるかどうかなどのDMAデバイスの状態)、CItag(たとえば、DMAデバイスの固有の識別子)を含む信頼されないファブリックデバイスレジスタファイル1534bに記憶され得る。DMAのロードまたは記憶を実行する特定のDMAデバイスのためのデバイスレジスタファイル1534bの中のエントリーは、現在のDMAのロードまたは記憶のためのCItagおよびPCtagの値を提供し得る。要素1535aは、1534の仲介されたDMA処理要求を行うために、信頼されないファブリック1536上のデバイスのために使用されるチャネルを示し得る。要素1535bは、1534の仲介されたDMA要求の結果を信頼されないファブリック1536に返すために使用されるチャネルを示し得る。

【０４２１】

要素1531a～bは、信頼されないファブリック1536から(1534のDMA仲介処理を介して)信頼されるファブリック1532にDMA要求を転送するためのチャネルを示す。具体的には、チャネル1531aは初期のタグ読取り(未検証)DMA要求を信頼されるファブリック1532に転送するためのチャネルであり、チャネル1531bはタグが更新されたデータの最終的な書込みを転送するための第2のチャネルである。2つのチャネルを使用することは、例1540に関連して以下で説明されるDMAまたはトラストブリッジパイプラインのさらなる議論を考慮すると、より明らかになり得る。要素1531cは、DMA仲介処理1534を介した、信頼されるファブリック1531cから信頼されないファブリックへのチャネルを示す。

【０４２２】

一実施形態では、要素1534は、図77の例1540に示されるようなDMA処理パイプラインを表し得る。例1540は、信頼されないまたはタグ付けされていないファブリック(例1500では1506、および例1530では1536)からDMAデバイス1504a～cにより行われるようなDMA動作をサービスするための、4段階の処理パイプラインを示す。要素1542、1544、1546、および1548は、DMA要求の結果としてトリガされるルールを示し得る。要素1545は、他の図に関連して説明されるものなどのI/O PUMPを示す(たとえば、例1500の1502)。要素1543は、DMW処理パイプラインの段階を示す。第1の段階1541aでは、DMA要求が信頼されないファブリックから受信され、要求されたDMAデータおよびその関連するタグをメモリ712cから取得するために、第2のメモリフェッチ段階1541bにおいて、未検証の要求がルール1542を介して行われる。DMA要求されたデータに対するメモリからのフェッチされるタグ情報は、第3の検証段階1541cへの入力として与えられ、ここで、現在のDMA要求に対応するルールのためのI/O PUMPキャッシュ1545においてルックアップが実行される。ルールがI/O PUMPにおいて発見されない場合、ルールミスハンドラが、DMA要求のための出力RtagおよびPCnew tagを計算するために、またはそうでなければ、現在のDMA要求が許容されないと判定する(それによりフォルトまたはトラップをトリガする)ために、プロセッサ702において実行している間に、I/O PUMP処理は段階1541cにおいてストールして無効になることがある。現在のDMA要求のためのルールがI/O PUMPの中に位置し、DMA要求の実行が許容されることが判定されると仮定する。DMA要求が書込み要求である場合、DMA要求の書込みデータは、そのタグ情報とともに、段階4 1541dにおいてメモリ712cにライトバックされる。DMA書込み動作では、書込みが完了すると、応答1548aが信頼されないファブリックに(次いでDMA要求を開始したDMAデバイスに)提供され得る。DMA読取り動作では、応答1546aが信頼されないファブリックに(次いでDMA要求を開始したDMAデバイスに)返されることがあり、ここで応答は段階2 1541bにおいてフェッチされる要求されるデータを含む。

【０４２３】
メモリフェッチが段階2 1541bにおいて実行されている間、要素1542は、信頼されないファブリックからの要求を伝え、(段階3 1541cにおける)I/O PUMP 1545のためのI/O要求に関する(段階1 1541aからの)情報を伝える、ルールを示し得る。要素1544は、信頼されるファブリックからのタグ応答を集め、I/O PUMPへと入力される実際のルールを編成し、I/O PUMPの出力とマージされるように段階1541bからライトバック段階1541dに情報を伝播する、ルールを示し得る。
【０４２４】
前述の実施形態の変形として、例1500に戻って参照が行われる。少なくとも1つの実施形態では、上で説明されたようにI/O PUMPキャッシュへルールを記憶させるのではなく、I/O PUMPはハードワイヤリングされたI/O PUMPとして実装されることがあり、ここで、ルールは、I/O PUMPのロードおよび記憶のDMAルールの固定された集合を具現化するようにワイヤリングされる論理ゲートなどの、専用のハードウェアを使用して実装され得る。
【０４２５】
さらなる変形として、I/O PUMPは代わりに、例1500に関連して説明されたようにプログラム可能であるキャッシュとしてさらに別の実施形態において定義されることがあり、違いは、ルールキャッシュとしてのI/O PUMPが有限の容量を有し、I/O PUMPキャッシュにすべて記憶されているルールの固定された集合で埋められるという点である。この後者の実施形態では、I/O PUMPはすべてのDMAルールの完全な集合で埋められ得るので、I/O PUMPについてのルールキャッシュミスは決してない。したがって、I/O PUMPルールキャッシュミスをサービスする必要は決してない。
【０４２６】
ここで説明されるのは、メモリ位置と関連付けられ得るものなどのタグを、初期化し、設定し、またはリセットすることに関連して使用され得る技法である。本明細書の他の箇所の説明と一貫して、そのような技法に関連して使用されるタグは、非ポインタタグ(ここで、非ポインタタグは関連するメモリ位置のための実際のタグ値である)またはポインタタグ(ここで、ポインタタグは実際の1つまたは複数のタグ値を含む別のメモリ位置のポインタまたはアドレスである)を示し得る。たとえば、メモリ位置と関連付けられるポインタタグは、合成タグに関連して使用されることがあり、ここで、ポインタは、並列に実装される複数の合成ポリシーなどのための複数のタグ値を含むメモリの中のアドレスを特定する。本明細書の他の箇所で説明されるように、並列にサポートされ得る例示的な合成ポリシーは、本明細書の他の箇所で説明されるメモリ安全性ポリシーおよび制御フロー整合性(CFI)ポリシーを含む。
【０４２７】
メモリ安全性およびスタックポリシーに関連して実行される処理は、たとえば、メモリ位置と関連付けられる多数のタグを特定の値に設定または初期化することを含み得る。たとえば、特定の色と関連付けられ得るものなどの、メモリの領域を割り振るとき、領域の中のメモリ位置と関連付けられる各タグは、特定の色値を有するように初期化される必要がある。別の例として、メモリ領域を解放するときなどの、メモリの領域を取り戻すときに、解放または割振り解除された領域のすべてのメモリ位置が、メモリ位置を解放されたものまたは割り振られていないものとして示す特定のタグ値に初期化され得る。
【０４２８】
ある領域の中のすべてのメモリ位置のタグを初期化またはリセットするために実行される処理は、許容不可能な長さの時間を費やすことがあり、タグ付けされるべきメモリ領域のサイズが増大するにつれて特に許容不可能になる。したがって、以下の段落で説明されるのは、メモリ位置のタグを効率的に初期化または設定する(たとえば、タグ付けする)ことを実現する技法である。少なくとも1つの実施形態では、タグの初期化または設定は、たとえば、メモリの領域の割振りまたはメモリの領域の解放に関連して実行され得る。本明細書で説明されるそのような技法は、大きなメモリ領域とともに使用するために拡張可能

である。そのような技法はメモリ位置のタグに関連して下で示されるが、より一般的には、そのような技法は、データアイテムまたはエンティティと各々関連付けられる値を初期化し、設定し、またはリセットすることに関連して使用され得る。

【０４２９】

少なくとも1つの実施形態では、タグおよびメモリの領域の関連するメモリ位置は、階層的な構造または配置で表されることがあり、ここで、階層のリーフはメモリ位置のためのタグを示す。例示を目的に、以下の議論では階層構造としてツリーを参照する。しかしながら、より一般的には、メモリ位置の領域と関連付けられるアドレス空間を表すために、任意の適切な階層構造が使用され得る。

【０４３０】

ある極端な場合、一実施形態では、ツリーまたは階層構造のリーフは、メモリの中の個々のワードを表し、タグを保持することがある。しかしながら、サブツリー全体が同じタグ値で一様にタグ付けされる場合、本明細書の技法は単に、サブツリーのいずれの子孫ノードもさらに表すことなく、その特定のノードにおけるタグ値およびツリーにおける関連するレベルを記憶し得る。この場合、ノードのタグ値は、特定の領域(たとえば、連続的なまたは切れ目のないメモリアドレスの範囲など)の複数のメモリ位置のためのタグ値を指定する。このようにして、大きな一様にタグ付けされる領域がある場合、タグ値を記憶する際にストレージを節約することができる。一様なタグ値がない(たとえば、連続するアドレスを有するいずれの2つのメモリ位置も同じタグ値を有しない)最悪の場合のシナリオでは、ツリーのリーフは各々、領域の中の単一のワードなどの単一のメモリ位置のためのタグ値を表す。

【０４３１】

以下の段落で説明されるようなツリーなどのそのような階層構造では、ツリーの中の1つのノードを単に書き直すことによって、2のべき乗のメモリ領域を再タグ付けし、または初期化するために、処理が実行され得る。2のべき乗ではない領域に対して、2のべき乗の領域の最小の集合へと領域を区分するために処理が実行され得る(たとえば、最小の集合の中に最大でも$2*\log_2$(領域サイズ)個のそのような領域)。特定のワードまたはメモリ位置のタグが必要とされるとき(たとえば、関連するメモリ位置のためのタグを読み取る)、ツリーを使用してタグを判定するために処理が実行され得る。以下で説明される少なくとも1つの実施形態では、ツリーの異なるレベルのためにキャッシュメモリの階層が利用され得る。タグ値は、所望のメモリ位置に関してキャッシュヒットを有するツリーにおける最高レベルと関連付けられるキャッシュによって提供され得る(たとえば、所望のメモリ位置のアドレスのタグ値のためにキャッシュルックアップを実行する)。メモリ位置と関連付けられるタグ値を書き込み、または変更するために実行される処理に関連して、処理は、サブツリーをマークするために単一の書込みを、または複数の書込み(たとえば、$2*\log_2$(領域サイズ)回のログの書込み)を実行することを含み得る。そのような複数の書込みは、たとえば、第1のメモリ領域に含まれる第1のメモリ位置のタグを変更または設定したことに応答して実行されることがあり、この第1のメモリ位置のタグはタグ値をそのように変更または設定する前は一様にタグ付けされている。この場合、タグ値を設定または変更することは、第1のメモリ領域がもはや一様にタグ付けされないようにし、第1のメモリ領域のタグ値を示す階層構造はそれに従って、第1のメモリ領域のタグ値を示すサブツリーをさらに分解するように更新される。

【０４３２】

図78を参照すると、本明細書の技法に従ったある実施形態においてメモリのある領域に対応するアドレス空間のタグ値を表すために使用され得る階層構造の例が示されている。例100は、例示を簡潔にする目的で、8個のメモリ位置を含むメモリ領域を表すために使用される階層構造としてツリーを示す。より一般的には、本明細書の技法は、任意の数のレベル、各レベルにおける任意の適切な数のノード、親ノード当たりの任意の適切な数の子ノードなどを有する、任意の適切な階層を使用して、任意のアドレス空間またはメモリ領域のタグ値を表すために使用され得る。

【０４３３】
例100は、両端を含めてアドレス0から7を有する8個のメモリ位置のタグ値のバイナリのツリー表現を示す。この例のツリーは、構造100の同じサブツリーを使用して一様にタグ付けされるものがもしあれば、それが8個のメモリ位置のいずれであるかに応じて、最高で4つのレベルのノードを含み得る。この例では、8個のメモリ位置のメモリ領域全体が2のべき乗個のより小さいメモリ領域へと繰り返し区分されることがあり、ここで、各々のそのようなより小さいメモリ領域の区分は、ツリーの中の異なるレベルのノードに対応する。たとえば、レベル1 104はアドレス空間0から7全体に対応するノードA1を含み、アドレス空間0から7はレベル2 106におけるノード(ノードB1およびノードB2)により各々表される2つのより小さい領域へと区分される。レベル2 106は、アドレス0～3と関連付けられるノードB1およびアドレス4～7と関連付けられるノードB2を含む。
【０４３４】
レベル2 106におけるノードB1およびB2の各々はさらに、レベル3 108におけるノードにより各々表される2つのより小さい領域へと区分され得る。ノードB1およびその関連するアドレス範囲0～3は、ノードC1およびC2により表される2つの領域へと区分され、ここでC1はアドレス範囲0～1と関連付けられ、C2はアドレス範囲2～3と関連付けられる。同様に、ノードB2およびその関連するアドレス範囲4～7はノードC3およびC4により表される2つの領域へと区分され、ここでC3はアドレス範囲4～5と関連付けられ、C4はアドレス範囲6～7と関連付けられる。
【０４３５】
レベル3 108におけるノードC1～C4の各々はさらに、レベル4 110におけるノードにより各々表される2つのより小さい領域へと区分され得る。この例では、レベル4におけるノードは各々、単一のワードまたはメモリ位置のタグ値を表す。ノードC1およびその関連するアドレス範囲0～1は、ノードD1およびD2により表される2つの領域へと区分され、ここでD1はアドレス0と関連付けられ、D2はアドレス1と関連付けられる。ノードC2およびその関連するアドレス範囲2～3は、ノードD3およびD4により表される2つの領域へと区分され、ここでD3はアドレス2と関連付けられ、D4はアドレス3と関連付けられる。ノードC3およびその関連するアドレス範囲4～5は、ノードD5およびD6により表される2つの領域へと区分され、ここでD5はアドレス4と関連付けられ、D6はアドレス5と関連付けられる。ノードC4およびその関連するアドレス範囲6～7は、ノードD7およびD8により表される2つの領域へと区分され、ここでD7はアドレス6と関連付けられ、D8はアドレス7と関連付けられる。
【０４３６】
すべてのノードA1、B1～B2、C1～C4、およびD1～D8が、8個のメモリ位置の領域のタグ値の階層的な表現において存在し得る、最大の数の可能なノードを表し得る。しかしながら、以下でより詳細に説明されるように、メモリ位置0～7に記憶されているタグ値を示すツリーに含まれる特定のノードは、様々な時点において表される具体的なタグ値および一様なタグ領域および非一様なタグ領域に依存して変わり得る。階層のレベルは、ルートノードまたはレベル1 104ノードに対応する最高のレベルから、最下位のレベル4 110に対応する最低のレベルまでランク付けされ得る。
【０４３７】
第1の例についての本明細書の技法に関連して、図79の120への参照が行われる。この第1の例では、階層120の中の特定のレベルにおけるノードと関連付けられるすべてのメモリ位置が同じタグ値T1を有し、これにより一様にタグ付けされるメモリ位置のサブツリーを示し、その特定のレベルにあるノードがすべてのそのようなメモリ位置のタグ値を有し、一様にタグ付けされるメモリ位置のいずれのタグ値を判定するためにも、サブツリーの中のさらなる子孫ノードを調べる必要はないと仮定する。たとえば、すべてのメモリ位置0～7が、同じタグを伴うメモリの領域を初期化することなどに関連して同じタグ値T1を含む場合、メモリ位置0～7のタグ値T1はノードA1に記憶され得る(たとえば、ノードA1により「tag=T1」の指示によって示されるように)。少なくとも1つの実施形態では、ア

ドレス0～7に対する領域全体のタグ値が単一のノードA1により表されるので、ツリーの他のノードのために追加のタグ値を記憶するさらなる必要はない。この場合、要素122は、アドレス0～7を伴うメモリ位置に記憶されているタグ値の階層的な表現に含まれ得る単一のノードを示し、残りのノードB1、B2、C1～C4、およびD1～D8は階層的な表現から省略され得る。

【０４３８】

第2の例では、図80の130に対する参照が行われる。この第2の例では、メモリ位置0～3が同じ第1のタグ値T1を有し、メモリ位置4～7が同じ第2のタグ値T2を有する(第1のタグ値と第2のタグ値は異なる)と仮定する。この場合、ノードA1は、ノードA1がメモリ位置0～7のための一様なタグを指定せず、メモリ位置0～7のタグ値は階層の1つまたは複数のより低いレベルにおけるノードにより指定されることを示す、インジケータ(たとえば、ノードA1により「TAG VALUE=NO TAG VALUE」の指示により示される)を含み得る。階層のレベル2において、第1のタグ値T1はノードB1に記憶されることがあり(ノードB1により「TAG VALUE=T1」の指示により示されるように)、第2のタグ値T2はノードB2に記憶されることがある(ノードB2により「TAG VALUE=T2」の指示により示されるように)。B1がルートであるサブツリー(B1、C1、C2、D1～D4)は、一様にタグ付けされるメモリ位置0～3の集合を示す。B2がルートであるサブツリー(B2、C3、C4、D5～D8)は、一様にタグ付けされるメモリ位置4～7の別の集合を示す。少なくとも1つの実施形態では、アドレス0～7に対する領域全体のタグ値がレベル2におけるノードB1およびB2により表されるので、レベル3および4におけるツリーの他のノード(たとえば、レベル3のノードC1～C4およびレベル4のノードD1～D8)のために追加のタグ値を記憶するさらなる必要はない。この場合、要素132はアドレス0～7を伴うメモリ位置に記憶されているタグ値の階層的な表現に含まれ得るノードを示し、残りのノードC1～C4およびD1～D8は階層的な表現から省略され得る。

【０４３９】

第1の時点で、タグの階層は単一のノード122のみを伴う例120に関連して説明されるようなものであることがあり、それは、すべてのタグが同じタグ値を有するからである。後続の第2の時点において、アドレス0～3のタグ値は同じ第1のタグ値T1となるように変更されることがあり、アドレス4～7は同じ第2のタグ値T2となるように変更されることがある。上記のタグの変更の結果として、例130において上で説明されたような2つの追加のノードB1およびB2が階層に追加され得る。ここで、後続の第3の時点で、アドレス0～3のタグ値が例130と同じままであると仮定する。しかしながら、アドレス4～7のタグ値は図81に関連して下で説明されるように変更されることがあり、これにより、追加のノードC3～C4およびD5～D6がタグの階層に追加される。

【０４４０】

第3の例において、図81の140への参照が行われる。この第3の例では、メモリ位置0～3が上で説明されたような同じ第1のタグ値T1を有すると仮定する(ここで、第1のタグ値T1はノードB1に記憶されることがあり、B1がルートであるサブツリー(B1、C1、C2、D1～D4)は一様にタグ付けされるメモリ位置0～3の集合を示す)。さらに、メモリ位置4～5は各々異なるタグ値を含み、ここでメモリ位置4はタグ値T3を有し、メモリ位置5はタグ値T4を有し、メモリ位置6～7は一様にタグ付けされており同じタグ値T5を含むと仮定する。この場合、上の議論と一貫して、ノードA1、B2、およびC3は各々、特定のノードがタグ値を指定せず、それにより、階層の中の1つまたは複数のより低いレベルにあるノードがノードA1、B2、およびC3と関連付けられる特定のメモリ位置のタグ値を指定することの、インジケータ(たとえば、TAG=NO TAG)を含み得る。たとえば、メモリ位置4～5に対応するノードC3は、ノードC3がタグ値を指定せず、それにより、階層の中の1つまたは複数のより低いレベルにあるノードがメモリ位置4～5のタグ値を指定することの、インジケータを含み得る。レベル4にあるノードD5は、メモリ位置4のタグ値T3を指定することがあり(たとえば、ノードD5によるTAG=T3インジケータ)、レベル4にあるノードD6は、メモリ位置5のタグ値T4を指定することがある(たとえば、ノードD6によるTAG=T4インジ

ケータ)。メモリ位置6～7に対応するノードC4は、メモリ位置6～7に共通する一様なタグ値であるタグ値T5を指定することがあり(たとえば、ノードC4によるTAG=T5インジケータ)、ノードD7およびD8の中の追加のタグ値を記憶するさらなる必要がない(たとえば、C4の子孫ノードD7、D8をさらに調査する必要がない)ことを示す。この場合、要素142はアドレス0～7を伴うメモリ位置に記憶されているタグ値の階層的な表現に含まれ得るノードを示し、残りのノードC1～C2、D1～D4、およびD7～D8は階層的な表現から省略され得る。

【０４４１】

120、130、および140の上記の例示は、メモリ位置と関連付けられるタグ値が経時的に変化し得るときの、異なる時点におけるアドレスまたはメモリ位置0～7のためのメモリ領域のタグ値の階層的な表現を示し得る。上で説明されたような階層にノードを追加することと同様の方式で、既存のノードのサブツリーがすべて同じタグ値を有するように変更されるにつれて、必要に応じてノードが階層から除去され得る(たとえば、ノードの子孫がすべて同じタグ値を有する場合、すべての子孫ノードが階層から除去されることがあり、ノードおよびその子孫の単一の一様なタグ値を示すためにノードがサブツリーの唯一のノードとして使用され得る)。

【０４４２】

少なくとも1つの実施形態では、ツリーの中のあるレベルにある第1のノードが第1のノードと関連付けられる1つまたは複数のメモリ位置の値を指定するとき、第1のノードの子孫ノードをさらに表す必要はない(たとえば、第1のノードを超えてサブツリーのノードをさらに表す必要はない)。さらに例示すると、メモリ領域0～7の単一の一様なタグ値を表すためにノードA1のタグ値のみが必要とされる、図79の120における上で述べられた第1の例に対する参照が行われる。さらに例示すると、実施形態がノードC1～C2、D1～D4、およびD7～D8をさらに表さないことがある、上で述べられた図81の第3の例140に対する参照が行われる。このようにして、メモリ位置および関連するタグ値のそのような階層的な表現を使用することで、メモリ位置のタグ値に関連してストレージを節約することができる。言い換えると、少なくとも1つの実施形態では、メモリ位置の各々の個々のタグ値を常に割り振り記憶するのではなく、階層の中の単一のタグ値が連続するまたは切れ目のないアドレスを有する複数のメモリ位置のための一様なタグ値を示すように、ストレージが割り振られることがある。120において述べられる第1の例を参照すると、同じタグ値を各々含むメモリ位置0～7の8つのタグ値のためにストレージを割り振るのではなく、メモリはノードA1の単一のタグ値を記憶するために割り振られることがある。

【０４４３】

アドレス0～7を有するメモリ領域の中に一様にタグ付けされたメモリ位置がないことを仮定する最悪の場合のシナリオでは、図78のノードの階層構造全体が、アドレス0～7に記憶されているタグ値を表すために使用される。たとえば、階層のリーフの各々がメモリの中の異なるワードを表すことがある。したがって、階層の底のレベル4 110はアドレス空間0～7のタグ値を示し得る。

【０４４４】

図78に戻って参照すると、メモリ位置0～7のアドレスを表すために使用される8ビットのアドレス空間があると仮定する。少なくとも1つの実施形態では、8ビットのアドレス空間全体が、8つのメモリ位置を各々含む異なるメモリ領域へと区分されることがあり、ここで、異なるメモリ領域の各々は、タグ値の階層の異なるインスタンスにより表されるタグ値を有することがある。したがって、すぐ上で説明されたアドレス0～7のメモリ領域に対して、高位または上位5ビットはすべて0であり、アドレス0～7は残りの下位3ビットにおいて表され得る。したがって、0に等しい高位または上位5ビットは、アドレス0～7のメモリ領域を示すために使用され得る。そのような実施形態では、8つのメモリ位置の各メモリ領域は、特定のメモリ領域のタグ値を示す図78に示されるものなどの別個のタグ値の階層を有し得る。この例では、8つのアドレスまたはメモリ位置の異なる範囲を示す異なるメモリ領域の各々が、メモリ位置の8ビットアドレスの上位5ビットを調査することによ

って区別され得る。

【０４４５】

本明細書の技法に従った少なくとも1つの実施形態では、一連のタグキャッシュメモリが使用されることがあり、ここで、タグキャッシュの数はタグ値を示すノードの階層の中のレベルの数に対応することがある。上で論じられた例について続けると、かつ図78の100に戻って参照すると、8つのメモリ位置のメモリ領域のためのタグ階層の各インスタンスは4つのレベルを有する。そのような場合、ある実施形態は、メモリ位置のタグを記憶するために、図82の例150に示されるような4つのタグキャッシュメモリ152、154、156、および158を使用し得る。一般に、4つのタグキャッシュメモリ152、154、156、および158の各々が、タグ値の階層の中の異なるレベルと関連付けられ、タグ値階層の関連する異なるレベルの中の各ノードについての情報を記憶し得る。たとえば、タグレベルキャッシュ152は、メモリ領域(これは上で述べられるようなこの特定の例では8つのメモリ位置のメモリ領域である)の各々のためのタグ値階層のレベル1 104のノードまたはルートの情報を含み得る。タグレベルキャッシュ154は、メモリ領域の各々のためのタグ値階層の、レベル2 106のノードのための情報を含み得る。タグレベルキャッシュ156は、メモリ領域の各々のためのタグ値階層の、レベル3 108のノードのための情報を含み得る。タグレベルキャッシュ158は、メモリ領域の各々のためのタグ値階層の、レベル4 108のノードのための情報を含み得る。この例ではレベル4 110である、階層の中の最低のまたは最下位のレベルは、データキャッシュに記憶され得るメモリ位置のためのキャッシュラインに対応し得る(たとえば、図1の要素20により示されるものなどのL1-D$として示される)。ある実施形態は、タグキャッシュのレベル4 158を有することがあり、加えて、データキャッシュのキャッシュラインに別々に記憶され得るメタデータタグを有することがある。ノードのキャッシュ152、154、156、および158の各々は、メインメモリの中に関連する表現を有する。

【０４４６】

8ビットのアドレス空間について本明細書で説明される例示的な実施形態に関連して、メモリ位置のアドレスの上位または高位5ビット152aは、キャッシュ152がメモリ位置のアドレスのための任意のレベル1のノードを含むかどうかをルックアップするために、レベル1キャッシュ152によって使用され得る。メモリ位置のアドレスの上位または高位6ビット154aは、キャッシュ154がメモリ位置のアドレスのための任意のレベル2のノードを含むかどうかをルックアップするために、レベル2キャッシュ154によって使用され得る。メモリ位置のアドレスの上位または高位7ビット156aは、キャッシュ156がメモリ位置のアドレスのための任意のレベル3のノードを含むかどうかをルックアップするために、レベル3キャッシュ156によって使用され得る。メモリ位置のアドレスの8ビット154aは、キャッシュ158がメモリ位置のアドレスのための任意のレベル4のノードを含むかどうかをルックアップするために、レベル4キャッシュ158によって使用され得る。

【０４４７】

ある特定のアドレスに対して、最下位のレベル以外のレベルと関連付けられるキャッシュの各々が、以下を返し得る。

1)特定のアドレスのタグ値(複数のアドレスに対してタグ値がそのレベルにおける一様なタグ値であることを示す)、

2)キャッシュが特定のアドレスのタグ値を指定せず、階層の中のより低いレベルにあるキャッシュが特定のアドレスのタグ値を取得するために調べられる必要があることのインジケータ(この特定のレベルは特定のアドレスのための一様なタグ値を指定しない)、または

3)特定のアドレスに対応するノードまたはタグ情報を含むキャッシュ位置がその特定のレベルのキャッシュの中にないことを示す、ヌルまたは第2のインジケータ。第2のインジケータは、最下位ではないより下位のレベルのキャッシュにおけるキャッシュがアドレスのためのノード情報またはタグ情報を含まないことも示す。これは以下でより詳細に論じられる。

【０４４８】

上の議論と一貫して、上の項目2)において返されるインジケータは、例120、130、および140において示されるものなどのノードと関連付けられる「NO TAG」インジケータであり得る。たとえば、図80の例示130を参照して、メモリ位置5のタグを判定するために処理が実行されると仮定する。この場合、レベル1キャッシュ152は、メモリ位置5のタグ値が他のより低位のレベルキャッシュ154、156、または158のうちの1つにより指定されることを示す、NO TAGインジケータを返し得る。レベル2キャッシュ154は、上の返されるキャッシュ項目1)を示すメモリ位置5のタグ値T2を返し得る。第2のインジケータが返される場合の上の返される項目3)を例示するために、レベル3キャッシュ156を考える。レベル3キャッシュ156は、メモリ位置5に対応するノード情報を何ら含まないことがある(たとえば、キャッシュ位置がメモリ位置5のルックアップと関連付けられるノード情報またはタグ情報を含まない)ので、このレベル3キャッシュの中にメモリ位置5のノード情報がないことを示す、3)において上で説明された第2のインジケータが返され得る。そのような実施形態では、処理は一般に、最高のタグレベルのキャッシュから返されるタグ値を利用し得る。たとえば、例130を参照してメモリ位置5に関連して、レベル2キャッシュ154は、メモリ位置5のタグ値を返すタグキャッシュの最高のレベルである。

【０４４９】

L1(レベル1)データキャッシュに記憶されるメモリ位置の内容について、キャッシュされる情報は、現在のタグ値と、また、タグ値が定義されるタグキャッシュの階層におけるレベルとを含み得る。図80の階層130を使用するメモリ位置5についての上の例を再び参照すると、メモリ位置5の内容がデータキャッシュにも記憶される場合、データキャッシュは、タグ値T2と、また、レベル2キャッシュ154に自身のノード情報が記憶されているレベル2ノード(たとえば、B2)により現在のタグ値T2が定義されるという情報とを含み得る。したがって、例150はタグキャッシュの階層における4つのタグキャッシュを示し、ここで、タグ値が記憶されることがあり、実施形態がタグキャッシュの階層に記憶されているいずれのタグ情報とも別のタグ付けされたデータキャッシュ(たとえば、L1データキャッシュ)を追加で含むことがある。

【０４５０】

本明細書の技法に従ったある実施形態では、特定のアドレスを有する特定のメモリ位置のタグ値を解決または判定するために、処理がPUMPによって実行され得る。特定のメモリ位置のタグ値および内容を取得するために処理を実行するとき、メモリ位置の内容およびそのタグがデータキャッシュに記憶されている、データキャッシュヒットがあることがある。あるメモリ位置に対するデータキャッシュヒットが発生すると、タグキャッシングレベルから得られる第1のキャッシュされたタグ値がデータキャッシュに記憶されているようなメモリ位置の第2のタグ値と一致することを確実にするために、このメモリ位置のタグ値を定義するタグキャッシュ階層の記憶されているレベルを調べるために、処理が実行され得る。これらの2つが一致しない場合、これは、データキャッシュに記憶されているような第2のキャッシュされたタグ値が古く、最新ではなく、変更されていることを示す。この場合、メモリ位置のためのデータキャッシュに記憶されている第2のキャッシュされたタグ値と、メモリ位置のためのタグキャッシュから取得されるような第1のタグ値とが一致しない場合、(たとえば、タグキャッシュ階層に記憶されているようなタグ値と一致するように)メモリ位置のためのデータキャッシュに記憶されているような第2のキャッシュされたタグ値を更新することを含む、処理が実行され得る。少なくとも1つの実施形態では、データが、したがってデータとデータのタグがデータキャッシュにおいてキャッシュされているメモリ位置に対して、タグが定義されるタグキャッシュ階層のレベルを含む情報が、メモリ位置のタグのためのデータキャッシュにおいて追跡され得る。キャッシュタグ階層へのレベルの上記の記憶は、記憶されているレベルがタグ階層からのタグ値に容易にアクセスするために使用され得る(たとえば、階層のルートまたは最上位から下方へリーフノードに向かう探索などにおいて、すべてのタグレベルキャッシュを調べること、または別様に既存のノードの階層を探索することを必要とするのではなく)ような最適化であり得る。したがって、データキャッシュヒットが発生すると、かつ、メモリ位置のための

データキャッシュに記憶されているタグ値がメモリ位置のためのタグ階層に記憶されているタグ値と一致しない場合、処理は、データキャッシュに記憶されているようなタグ値を更新することと、メモリ位置のタグがタグ階層においてどこで定義されるかについて、データキャッシュに記憶されている階層レベル情報を追加で更新することとを含み得る。続いて、メモリ位置のタグ値を解決または判定するためのPUMPによって実行される処理が再開され得る。

【０４５１】

メモリ位置に対するデータキャッシュミスが発生すると(たとえば、メモリ位置の内容およびタグがデータキャッシュにおいて見つからない場合)、(たとえば、最下位のタグキャッシュレベルではなく)タグキャッシュの複数のレベルにおけるタグ値に対するタグキャッシュのルックアップを並列に実行するために、処理が実行され得る。たとえば、メモリ位置のタグ値のためのルックアップが、タグキャッシュのレベル1、2、3、および4のために、4つのキャッシュ152、154、156、および158をそれぞれ並列に調べることによって、実行され得る。上で論じられたように、タグキャッシュ152、154、156、および158の最高のレベルにより返されるタグ値は、メモリ位置のタグ値として使用される。加えて、適切に表現されるタグ値の階層において、152、154、156、および158のうちの単一のものだけが、メモリ位置のタグ値を返し得ることに留意されたい。少なくとも1つの実施形態では、キャッシュ152、154、156、および158は、並列なアクセスを可能にするためにインデクシングされ得る。

【０４５２】

ある実施形態はまた、すべての4つのタグキャッシュ152、154、156、および158に関して、特定のメモリ位置のタグのための並列なルックアップまたは探索を実行しないことがある。変形として、ある実施形態は、ルートノードレベル(レベル1)から下方へリーフノード(たとえば、レベル4)に向かって、階層のタグキャッシュを横断し得る。レベルNにおけるタグキャッシュミスに対して、ツリーまたは階層が横断されて、特定のメモリアクセスのためにタグキャッシュの異なるレベルへとノードを挿入し得る。タグキャッシュの並列の探索に対するレベルキャッシュミスに関連して、ある実施形態は、タグがあるときにだけノードをレベルキャッシュに挿入することを選び得る。あるレベルキャッシュがタグを提供する限り、すべての他のレベルキャッシュがNO TAGエントリーを有することは必要とされない。

【０４５３】

本明細書の他の箇所で論じられるように、メモリ位置のタグが変更され得る。メモリ位置のタグを変更したことに応答して、メモリ位置のタグ値を指定する階層をそれに従って更新するために、処理が実行され得る。そのような更新は、もはや一様ではない階層の任意の1つまたは複数のレベルを無効にすることを含み得る。加えて、図82の例150に示されるものなどのキャッシュのレベルをそれに従って更新するために、処理が実行され得る。

【０４５４】

メモリ位置のタグを設定または初期化するための動作を実行するとき、そのような処理は、所望の動作を実行することの有効性をPUMPが確認することを含み得る。たとえば、メモリ領域の中のすべてのメモリ位置を新しいタグで再タグ付けする場合を考える。処理は、領域の中のすべてのメモリ位置の現在のタグ値を取得することと、再タグ付けの有効性についてPUMP処理を介して確認することとを含み得る。処理は、許容される場合、領域の中のメモリ位置のタグをクリアすることと、次いで、許容される場合、領域の中のメモリ位置のタグ値を更新することとを含み得る。上の議論と一貫して、メモリ領域のタグを更新すること、変更すること、または設定することは、それに従って、メモリ領域の中の異なるメモリ位置のタグ値を反映するように階層および関連するノードを変更すること(たとえば、タグ値の変更を反映するために追加される追加の子ノードまたは子孫ノードがあり得るように、変更の前は一様であり変更の後は一様ではない領域の部分を分解すること)を含み得る

【０４５５】

少なくとも1つの実施形態では、メモリ領域のタグ値の階層的な表現はツリーであり得る。たとえば、ツリーは、各ノードが0個、1個、または2個のいずれかの子ノードを有する、バイナリのツリーであり得る。変形として、階層的な表現はツリーであり得るがバイナリのツリーではないことがある。たとえば、ツリーの中の各ノードは、指定される最大値までの任意の適切な数の子ノードを有することが許容され得る。階層的な表現は、任意の適切な数のレベル、各レベルにおける任意の適切な数のノード、親ノード当たりの任意の適切な数の子ノードなどを含み得る。当技術分野において知られているように、レベルおよび各レベルにおけるノードの深さまたは数/親ノード当たりの子ノードの数などの、階層的な表現の様々なパラメータの間にはトレードオフがある。たとえば、各レベルにおけるノードが多いほど、レベルの数は少なく、したがって、メモリ位置のタグ値を判定するときに調べられるべき時間/レベルが少なくなる。しかしながら、そのような場合、領域をクリアするためにより多数の書込みが実行されることが必要である。

【０４５６】

ここで説明されるのは、本明細書の技法に従ったある実施形態においてCFIポリシーに関連して、ローダコードの結果としてトリガされるルールなどによって実行され得る技法である。タグ情報にアクセスするメタデータ処理ルールを使用してCFIポリシーを実施するために、許容可能な制御フローに関する情報は、メタデータ処理領域に伝えられる必要がある。この目的で、本明細書の技法に従ったある実施形態は、以下の段落で説明される手法を使用し得る。一般に、分岐のソースからターゲットまたは宛先への制御の移転が行われる。許容可能な制御フローに関連して、特定の制御フローのターゲットまたは宛先に対して、特定の制御フローのターゲットまたは宛先へ制御を移転することが許容されるソースの集合が特定され得る。各々の可能な制御フローのターゲットのためのソースの集合は、記憶されているメタデータタグ情報などのメタデータ処理領域に伝えられることがあり、このメタデータタグ情報は次いで、ユーザコード(たとえば、コード実行領域または非メタデータ処理領域において実行するコード)のランタイム実行の間のCFIポリシー実施に関連してCFIポリシーのルールによって使用されることがある。

【０４５７】

実行される処理は、各ソースを一意にタグ付けし、次いで、その特定のターゲットに制御を移転することが許可される許容可能なソース(たとえば、ソースのアドレス)の集合で各ターゲットをタグ付けすることを含み得る。たとえば、図83への参照が行われる。例1700において、要素1701は、コード実行領域または非メタデータ処理領域において実行されるアプリケーションの命令のコード部分を示し得る。要素1702aおよび1704a～cはコード部分の中の命令の位置を示す。要素1702aは制御フローのターゲットAを示す。要素1704a～cは、ターゲットA1702aへ制御を移転することが許容される制御フローのソースを示す。1704a～cの各々からの制御のそのような移転は、JMP(ジャンプ)A命令によって示される。要素1706は、ターゲットAへ制御を移転することが許可される許容可能なソースの集合を示す。D7は命令1704aの固有のソースタグを示す。C3は命令1704bの固有のソースタグを示す。E8は命令1704cの固有のソースタグを示す。1710によって示されるように、JMP(ジャンプ)命令1704a～cはそれぞれ、D7、C3、およびE8としてタグ付けされる。やはり1710により示されるように、命令1704a～cもそれぞれ、アドレス1020、1028、および1034に記憶される。ターゲット位置Aはアドレス800を有する。この場合、許容可能なソースの集合、またはターゲットAに制御を移転することが許容されるソース命令のアドレスが、1706により示される{1020,1028,1034}に設定され得る。したがって、集合1706は、メタデータ処理領域に伝えられる必要がある許容可能な制御フローの情報の例であり、ここで、そのような許容可能な制御フローの情報はCFIポリシーのルールによる使用のためにタグメタデータとして記憶される。本明細書の技法に従った少なくとも1つの実施形態では、ローダのコードは、コード部分1701を含むアプリケーションのためのCFIポリシーを実施するために、メタデータ処理領域により必要とされる制御フロー情報を収集するための処理を実行するルールを発火し得る。ローダコードはアプリケーションをロードすること(たとえば、アプリケーションのための実行可能コードをロード

すること)に関連して実行されることがあり、これにより、ローダコードは、アプリケーションをロードするために実行している間、必要な処理を実行するルールに、制御フロー情報(アプリケーションの実行の間にCFIポリシーを実施するためにメタデータ処理により後で使用されるような)を収集させる。

【０４５８】

本明細書の他の箇所の説明と一貫する少なくとも1つの実施形態では、カーネルコードの実行は、ローダのタグ付けされた命令が、実行されると、ソースをタグ付けする(たとえば、ソースタグD7、C3、およびE8を生成する)ために使用されるソースタグのシーケンス(シーケンスの各タグは固有である)を生成するルールをトリガすることを可能にする、特別な命令タグでローダのコードをタグ付けするルールをトリガし得る。たとえば、ローダの実行中コードの結果として発火されるルールにより実行される論理的な処理を含めて、図84への参照が行われる。論理的な処理はC様の疑似コードの記述を使用して1720において記述され、ここで、そのような処理は、A 1702aなどの各制御フローのターゲットのために実行され得る。ステップ1721において、ソース集合は空集合に初期化される。ステップ1722において、ターゲットに制御を移転することが許容される各ソースに対して、ステップ1723、1724、および1725が実行され得る。ステップ1723において、tが新しく割り振られたCFIソースタグを割り当てられる。ステップ1724において、(ターゲットに制御を移転する命令の)ソース位置は、ステップ1723において生成される新しく割り振られたタグtでタグ付けされる。ステップ1725において、ソース集合はタグtも含むように更新される。一態様では、ステップ1725の動作は、ターゲットのための許容可能なソースの集合の和集合を形成するものとして特徴付けられることがあり、ここで、この和集合の動作は、各ソースのために実行されるような、1722において開始するループ処理の各々の繰り返しのために1725において実行される。ステップ1726はソース集合でターゲットをタグ付けまたはマークする。

【０４５９】

要素1723は、新しいCFIソースタグを生成する、または割り振るルールをトリガする、ローダコードに含まれる以下の命令であり得る。

ADD R1←R1+R1

ここで、ADD命令(たとえば、RISC-V命令セットにおけるADDIなど)はカーネルコードによりタグ付けされており、CFI-alloc-tagの特別なCI tagが、許容可能なタグ生成源命令としてこの命令をマークする。少なくとも1つの実施形態では、ソースタグの異なるシーケンスが、CFIポリシーに関連して各アプリケーションのためのローダによって生成され得る(たとえば、例1620において、ローダは、アプリケーションのためのCFIソースタグの固有のシーケンスとして、CFIタグ1630の異なるシーケンスを使用することがあり、ここでCFIタグのシーケンスは、1627のCFI tag生成源App-nタグのうちの特定の1つから生成され得る)。CFI-alloc-tagは、ADD命令は、ADD命令がアプリケーション固有のCFIシーケンスにおいて次のタグを割り振り、または生成することを許容されることを示す、上のローダADD命令に付けられるCI tagである。CFI-alloc-tagは、例1620に関連して説明されるような1624の特別な命令タイプのうちの1つであり得る。上のADD命令は、R1のタグがCFIシーケンスの状態を保持することを示し、ここで、状態は以前に生成されたシーケンスの最後のタグであり得る。上のADD命令の実行は、CFIシーケンスにおける次の新しいタグを生成し、その新しく生成されたタグとなるようにR1のタグを更新する、ルールをトリガする。本明細書の他の箇所で説明されるようなルール規約を使用すると、以下のADDルールは上のADD命令によりトリガされるルールを示し得る。

ADD:(-,CFI-alloc-tag,t1,t1,--)→(-,t1next)

これは、ADDI命令のためのCI tagがCFI-alloc-tagであることを確実にする。このADDルールにおいて、t1は、シーケンスの中の次のタグt1nextを生成するために使用されるシーケンスの中の以前のタグ(アプリケーションのためのCFIタグシーケンスの現在の状態として保存される)を示し、ここでt1nextはRD(宛先または結果レジスタ)のためのタグとして記憶される。CFIシーケンスの中の上記のタグt1nextは、ソース点に付けられる固有のCF

Iソースタグとして使用され得る。

【０４６０】

要素1724は、固有のCFIソースタグでソース位置をタグ付けするルールをトリガするために使用される、下のST(記憶)命令などのローダコードの命令であり得る。

ST R1→R3

ここで、R3はタグ付けされているユーザプログラムコードの中の制御フローのソース位置(たとえば、例1700における1704a)へのポインタであり、R1のタグはソース位置に付けられるべき固有のCFIソースタグである。上のST命令はまた、ST命令が以下のルールSTをトリガするローダコードに含まれることを示す、CI-LDRなどの特別なCI tagでタグ付けされ得る。

ST: (-,CI-LDR,t1,-,codetag)→(-,t1)

ここで、CI tag=CI-LDRであり、t1はR1のタグとして現在記憶されているCFIソースタグであり、codetagはアドレスR3におけるソース位置にある命令タグである(たとえば、ソース位置がコードとして現在タグ付けされることを確実にする)。結果として、宛先(R3)は固有のCFIソースタグt1でタグ付けされる。

【０４６１】

要素1725は、ソースのアドレス(たとえば、R3により現在指されている、ここでR3はソースのアドレスを格納する)を、ターゲットに制御を移転することができる許容可能なソース位置を示すCFIソースタグの累積された集合に追加する、ルールをトリガするために使用される以下のADD命令などのローダコードの命令であり得る。

ADD R2←R2+R3

ここで、R2のタグは許容可能なソース位置の累積された集合を示すメモリ位置を指す。上のADD命令は、このADD命令がCFIソースの和集合動作を実行することと、この和集合がR2のタグとして記憶されることとを示す、特別なCFI UNION命令タグでタグ付けされ得る。ADDのための以下のルールが、上のADD命令の結果として発火され得る。

ADD: (-,CFI UNION,tset,tsrc,-)→(-,tunion)

これは、CI tagがCFI UNIONであることと、tsetがターゲット集合であることと、tsrcがソースタグであることとを確実にするために確認を行う。これは、tsetへのtsrcの追加を表す新しいCFI集合tunionを産生する。

【０４６２】

要素1726は、ターゲットに制御を移転することができる許容可能なソース位置の和集合または累積されたリストでターゲットをタグ付けする、ルールをトリガするために使用される下のST命令などのローダコードの命令であり得る。

ST R2→R17

R17はターゲット位置のアドレスを格納するレジスタであることがあり、R2は、上で述べられたように、許容可能なソース位置の現在の累積された和集合でタグ付けされたレジスタであることがある(たとえば、R2のタグは、そのアドレスがR17に格納されているターゲット位置のための許容可能なソース位置の集合を示す)。上のST命令は、制御移転のターゲット位置をタグ付けすることが許容される特別なものとして命令を示す、特別な命令タグCFI MARK TARGETでタグ付けされ得る(たとえば、ローダコードのこのSTORE命令1726は、1720の処理をルールに実行させるロードコード命令上の他のコードタグと同様の方式で、カーネルコードによってタグ付けされていることがある)。以下のSTルールは、1726のための上のSTORE命令の結果としてトリガされ得る。

ST: (-,CFI MARK TARGET,tset,-,codetag)→(-,tset)

これは、CI tagがCFI MARK TARGETでありターゲット(R17により指される、ここでR17はターゲットアドレスを含む)が命令を示すcodetagでタグ付けされているときにトリガされ、tsetのアノテーションをターゲットに置く。

【０４６３】

ソース、ターゲット、および許容可能なソース位置の集合とともに使用するために定義され得る様々なタグの構造またはレイアウトは、本明細書の他の箇所で、ならびに任意の他

の適切な構造の定義において説明される(たとえば、より一般的に任意の命令とともに、ならびにタグ付けされたワード当たり複数の命令に関連して使用され得る、タグ付けされたソースおよびターゲット位置とともに使用するためのタグのレイアウトを説明する例240、250、260、267、270、および280を参照)。

【０４６４】

したがって、例1720のような上で説明される処理ステップは、ローダのコードが実行されるときに、例1720のステップが本明細書の技法に従ったある実施形態においてメタデータ処理領域により実行されるようにするルールが発火されるように、ローダのコードが適切にタグ付けされるようにすることにより実行されることがある。命令の上記のシーケンスおよび命令の結果として発火されるルールは、本明細書の技法を使用するある実施形態において使用され得る命令およびルールの一例に過ぎないことに留意されたい。たとえば、ある実施形態は、上で説明されたような処理(たとえば、要素1725)を実行するルールをトリガするローダコードの中のADD以外の異なる命令を含み得る。

【０４６５】

上記の説明において、簡潔性、明瞭性、および理解のためにいくつかの用語が使用されている。そのような用語は説明を目的に使用されており、広く解釈されることが意図されているので、従来技術の要件を超えるような不必要な限定が、そこから意図されるものではない。その上、本開示の好ましい実施形態の説明および図示は例であり、本開示は示されるまたは説明される厳密な詳細に限定されない。

【０４６６】

本明細書で説明される技法の様々な態様は、任意の1つまたは複数の異なる形態のコンピュータ可読媒体に記憶されるコードを実行することにより実行され得る。コンピュータ可読媒体は、取り外し可能または取り外し不可能であり得る様々な形態の揮発性ストレージ(たとえば、RAM)および不揮発性ストレージ(たとえば、ROM、フラッシュメモリ、磁気ディスクもしくは光学ディスク、またはテープ)を含み得る。

【０４６７】

本発明は詳細に示され説明される様々な実施形態に関連して開示されたが、それらに対する変更および改善が当業者には容易に明らかになるであろう。したがって、本発明の趣旨および範囲は以下の特許請求の範囲だけによって限定されるべきである。

【０４６８】

PUMPのプログラミング

セキュリティのためのハードウェアで支援されるマイクロポリシー

概要

広範なセキュリティポリシーが、ISAレベルでメタルールについてのルールとして策定され、プログラム可能なハードウェアにおいて効率的に実施され得る。主計算とともに、解釈されないメタデータに対するフレキシブルなルール評価をサポートする、メタデータ処理のためのプログラム可能ユニット(PUMP)アーキテクチャに基づいて、そのようなポリシーのためのプログラミングモデルを精緻化する。空間的および時間的なメモリ安全性、テイント追跡、制御フロー整合性、ならびにプリミティブの分類という4つの特定の領域において、様々な複雑さの安全性およびセキュリティポリシーの多様な集合を実装することによって、モデルの一般性を示す。単純なRISC ISAのためのこれらのポリシーの性能を、単独と組合せの両方で特徴付ける。大半のポリシーの平均のランタイムオーバーヘッドはわずか8%である。このことは、PUMPモデルがソフトウェア実施のフレキシビリティおよび適応性を専用ハードウェアの性能により実現できることを示している。

【０４６９】

1．導入

プログラムの意図を覆すことは攻撃者にとってとても簡単である。こうした現状において、実行する動作の意図される高水準のセマンティクスに対して作動するように設計されている現代のプロセッサは共犯であり、これはトランジスタが高価で主要な設計の目標がランタイム実行であった技術の時代の遺産である。コンピュータシステムが重大な仕事をま

すます任せられるにつれて、システムのセキュリティがついに重要な設計の目標になった。同時に、プロセッサは今や、中程度のシステムオンチップダイと比較しても小さく、セキュリティを増強するハードウェアによりそれらを拡張することは現実的であり安価である。管理するデータのプライバシーおよび整合性を未来のコンピュータが適切に保護するために、現代の脅威およびハードウェアのコストを考慮したセキュリティ機構とともにコンピューティングスタック全体を設計し直さなければならない。

【０４７０】

セキュリティの文献は、悪意のある誤ったコードが原因の脆弱性を減らすことができる、広範囲のランタイムポリシーを提供する。これらのポリシーはしばしば、高水準言語の抽象化(これは数列であり、これはコードポインタであり、...)またはユーザレベルのセキュリティ不変条件(この文字列はネットワークから来た)を、プログラムのデータおよびコード上のメタデータアノテーションへと符号化する。高水準のセマンティクスまたはポリシーは、計算が進行するにつれてこのメタデータを伝播し、適切な点において侵害について動的に確認することによって、実施される。これらの低水準で細粒度の実施機構を、マイクロポリシー(または略式には単に「ポリシー」)と呼ぶ。

【０４７１】

マイクロポリシーのソフトウェアによる実現は、マイクロポリシーについての任意のメタデータおよび任意の強力な計算を定義することができる。ソフトウェア実装は、新しいポリシーの高速な展開を容易にするが、ランタイムおよびエネルギーコストの観点で法外に高価であることがあり(1.5倍～10倍)[42]、望ましくないセキュリティと性能のトレードオフにつながる。簡単なマイクロポリシーは、小さいオーバーヘッドでハードウェアにおいてサポートされ得る[41,?]。しかしながら、単一のポリシーをサポートするようにカスタマイズされるハードウェアは、展開するのに数年かかることがあり、適応が遅い。今日の活動的なサイバー攻撃の情勢は、進化する脅威に対する迅速な現場での対応をサポートする機構を必要とする。

【０４７２】

より大きなフレキシビリティに対する要望は、ポリシー実施ハードウェアをよりプログラム可能にするためのいくつかの最近の努力を促した[18,45,19,13](5章参照)。ここで、広範な低水準のランタイムポリシーが任意のメタデータに対する命令粒度の計算に関して定義されることを可能にする、PUMP[7]と呼ばれる設計である「メタデータ処理のためのプログラム可能ユニット」を考える。ハードウェアレベルにおいて、データの1つ1つのワードがワードサイズのメタデータタグと関連付けられる。これらのタグは、ハードウェアにより解釈されず、ソフトウェアにおいて、それらは、それらが添付されるデータのタイプ、由来、分類レベル、または信頼性などの情報の表現にマッピングされ得る。タグはポインタを表現するのに十分大きいので、タグは、メタデータのタプルを含む任意のサイズおよび複雑さのデータ構造を参照することができ、複数の直交するポリシーが並列に実施されることを可能にする。プログラムカウンタが、プログラムの制御状態の履歴の追跡をサポートするためにタグ付けされ、プログラムコードが、コードの由来、制御フロー、およびコンパートメント化についてのポリシーをサポートするためにタグ付けされる。プロセッサコアは、命令実行と同期した高性能なルールの分解を可能にするルールキャッシュと、このキャッシュにおけるルックアップのミスのときにポリシーハンドリングコードへの高速なコンテキストの切替えのための特別な動作モードとによって拡張される。このことは、PUMPが、ソフトウェアの表現性および適応可能性とハードウェアの性能による、広範な低水準のポリシーの実施を容易にすることを可能にする。

【０４７３】

本文書の目標は、PUMP様のタグ付けおよびルール処理が現実の脅威に対して有用であることと、ルールの形式でポリシーを書き込むことが扱いやすいこととの両方を示すことである。このことを、PUMPが低水準のセキュリティおよび安全性ポリシーの多様な集合をサポートするためにどのようにプログラムされ得るかを精緻化することにより行う。ポリシーの4つの群(すべて文献においてよく知られている)、すなわち(i)プリミティブ型、弱

い形の型安全性を実施する、(ii)空間的および時間的なメモリ安全性、ヒープに割り振られたデータに対する境界およびuse-after-freeエラーを捉える、(iii)制御フローの整合性(CFI)[2]、コード再使用攻撃を防ぐ、(iv)テイント追跡、テイントは所与のデータに寄与した可能性のあるデータソースまたはコンポーネントを表し得る、の詳細な実装および評価を提示する。これらのポリシーの大半は、ソフトウェアにおいて現在のシステムが効率的にサポートできるものを超えている。最後に、これらのポリシーがどのように同時に適用され得るかを示す。これらのポリシーは既存の文献においてよく研究されているので、主な論点はポリシーが提供するセキュリティの保証ではなく、ポリシーがどのようにルールとして表現され、PUMPで実施され得るかを探究することである。PUMPが簡単なインオーダーRISCプロセッサ(Alpha[1])に取り付けられるときのSPEC CPU2006ベンチマークスイートにわたるこれらのポリシーのランタイムへの影響を推定するために、命令トレースのシミュレーションを使用する。PUMPが、広範な複雑さを伴うポリシーをサポートし、性能への影響を定量化できることを示す。複雑さのこの広い範囲は、脅威が進化するにつれてポリシーを精緻化するための能力と、この進化が性能にどのように影響し得るかとを示すものである。

【０４７４】

本文書は、今後この夏のワークショップで発表される短い論文である[7]の拡張された、強化された、かつ焦点が定め直されたバージョンである。以前の論文は、RISCプロセッサへのPUMPの単純なハードウェア統合に注目し、大半のベンチマークで適度な性能を確立し、改善すべき領域を特定した。今回の研究では、[7]においてよく説明されているマイクロアーキテクチャの検討を避け、代わりに、プログラミングモデルおよびポリシー自体のより詳細な説明および評価に注目する。また、PUMPソフトウェアサービスがそれらを悪用からどのように保護するかを説明する。報告する性能は、(i)オペグループの使用(2章)、(ii)ミスコストのより正確な推定(3章)、および(iii)必要なときにだけポインタタグを使用することによるDRAMアクセスの低減(4章)により、[7]を改善する。

【０４７５】

まとめると、今回の研究の主な貢献は、(i)このアーキテクチャによりサポートされるポリシーをコンパクトにかつ正確に記述するためのプログラミングモデルおよびサポートするインターフェースモデル(2章および3章)、(ii)よく研究されているポリシーの4つの多様なクラスを使用したポリシーの符号化および合成の詳細な例、ならびに(iii)これらのポリシーの要件、複雑さ、および性能の定量化(4章)である。5章および6章において、関連する研究および今後の研究について論じる。いくつかの追加の材料は、http://git.io/8K7IKAから匿名の形式で入手可能である。これらは、研究されたポリシーの完全な定義、実験のソースコード、および[7]の匿名化されたバージョンを伴う付録を含む。

【０４７６】

2. ポリシープログラミングモデル

PUMPポリシーは、タグ値の集合と、何らかの所望の追跡および実施機構を実装するためにこれらのタグを操作するルールの集合とで構成される。ルールには、システムのソフトウェア層について話しているか(シンボリックルール)、またはハードウェア層について話しているか(コンクリートルール)に応じて、2つの形式がある。

【０４７７】

例.PUMPの動作を示すために、プログラム実行の間にリターン点を制約するための簡単な例示的なポリシーを考えよう。このポリシーの動機は、リターン指向プログラミング(ROP)[39]として知られているあるクラスの攻撃から来ており、ROPでは、攻撃者は攻撃されているプログラムのバイナリ実行可能ファイルの中で「ガジェット」の集合を特定し、これらを使用して、各々が何らかのガジェットを指すリターンアドレスを格納するスタックフレームの適切なシーケンスを構築することにより複雑な悪意のある挙動を組み立て、次いで、所望のシーケンスでスタックの上部を上書きするためにバッファオーバーフローまたは他の脆弱性が悪用され、断片が順番通りに実行されるようにする。

【０４７８】

ROP攻撃を制限する1つの簡単な方法は、リターン命令のターゲットをよく定義されているリターンポイントに制約することである。これは、有効なリターンポイントである命令をメタデータタグターゲットでタグ付けすることにより、PUMPを使用して行うことができる。リターン命令を実行するたびに、リターンが発生したばかりであることを示すために、確認すべきPC上にメタデータタグを設定する。次の命令で、PC tagがcheckであることに気付き、現在の命令のタグがtargetであることを検証し、そうではない場合にはセキュリティ侵害をシグナリングする。メタデータをよりリッチにすることによって、どのリターン命令がどのリターン点にリターンできるかを正確に制御できることが、このセクションの後において分かる。メタデータをさらにリッチにすることによって、完全なCFI確認[2]を実施することができる(4.3章参照)。

【０４７９】

シンボリックルール。ポリシー設計者およびPUMPのソフトウェア部分の観点から、ポリシーは小さな領域固有の言語で書かれたシンボリックルールを使用してコンパクトに記述される。各シンボリックルールは以下の形式を有する。

opgroup:(PC,CI,OP1,OP2,MR)→(PC',R')if guard?

これは、命令オペコードの集合(opgroup)と、プログラムカウンタ(PC)、現在の命令(CI)、レジスタファイルからの2つまでのオペランド(OP1、OP2)、および命令により参照されるメモリ位置(MR)がもしあれば、それらのメタデータタグとで、ルールが一致することを述べている。すべての関連するタグ表現が一致し、guard?が当てはまる場合、ルールが適用される。この場合、右手側は、PCのタグ(PC')および動作の結果のタグ(R')をどのように更新するかを判定する。大半のポリシーでは、同一のルールを伴う多数のオペコードがあるので、オペコードの代わりにオペグループを使用する。無視される入力または出力フィールドを示すために、「-」と書く(「ワイルドカード」)。guard?条件が単に真であるとき、省略する。

【０４８０】

すぐ上で描かれた簡単なROPポリシーでは、オペコードを、return(単一のオペコードだけを格納する)および

【０４８１】

【数３】

$$\overline{return}$$

【０４８２】

(すべての残り)という2つのオペグループに分割し、可能性のあるタグ値は、check、target、および⊥である。PCは常にcheckまたは⊥のいずれかでタグ付けされ、各命令はtargetまたは⊥のいずれかでタグ付けされる(命令タグは信頼されるローダにより供給される、3章参照)。シンボリックルールは次の通りである。

return: (⊥,-,-,-,-)→(check,-) (1)

【０４８３】

【数４】

$$\overline{return}: \ (\text{check}, \text{target}, -, -, -) \rightarrow (\bot, -) \qquad (2)$$

【０４８４】

【数５】

$$\overline{return}: \ (\bot, -, -, -, -) \rightarrow (\bot, -) \qquad (3)$$

【０４８５】
return: (check,target,-,-,-)→(check,-) (4)
【０４８６】
ルール1は、現在の動作がreturnである(かつPCがまだcheckとタグ付けされていない)とき、PCのタグをcheckに変更することを述べている。checkとタグ付けされたPCを伴う命令を実行するとき(ルール2)、命令タグCIがtargetであることを確認し、そうである場合、動作を許容してPCのタグをクリアする。現在の動作がreturnではなくPCタグが⊥である場合、単純に(ルール3)に進む。ルール4は、リターンの有効なターゲット自体がリターンである特別な場合を扱う。どのルールも適用されない場合、動作は許容されない(たとえば、PC=checkおよびCI=⊥という構成は許容されない)。シンボリックルールは重複しないと仮定する。
【０４８７】
次に、このポリシーのより正確な変形を考える。ここで、1つ1つのリターンが何らかの有効なリターンターゲットに到達することだけではなく、リターンが実際にそこから呼び出されることが可能であるコードポイントをリターンがターゲットとしていることを確実にする。このポリシーは、コンパイラが、リターンポイントの完全な知識を持っており、各々に対して、どの呼出し場所にリターンする可能性があり得るかを分析できると、仮定している。この情報を使用して、固有のタグを各リターンおよび各々の潜在的なリターンターゲットに添付することができる。リターンに遭遇すると、PUMPは(汎用的なタグcheckではなく)命令のタグをPCへとコピーする(ルール1'および4')。次のステップで、PUMPは、実際のリターンポイントが予想されたもののうちの1つであること、すなわち、PCからCIへのリターンが許容される(ルール2'および4')ことを確認する。
return:(⊥,ci,-,-,-)→(ci,-) (1')
【０４８８】
【数６】

$$\overline{return}:\ (pc, ci, -, -, -) \rightarrow (\perp, -) \text{ if } (pc, ci) \in \chi \qquad (2')$$

【０４８９】
【数７】

$$\overline{return}:\ (\perp, -, -, -, -) \rightarrow (\perp, -) \qquad (3')$$

【０４９０】
return:(pc,ci,-,-,-)→(ci,-)if(pc,ci)∈X (4')
【０４９１】
これらのルールにおいて、コードにおけるリターンを介して許容される間接的な制御フローを示すために、X(コンパイラによって提供されるコード位置識別子のペアの集合)を使用する。ここで示されるように、シンボリックルールにおいてタグを記述する表現は、一定値には限定されない。タグの大きな集合をコンパクトに記述する、より一般的な表現を書くことができる。
【０４９２】
コンクリートルール。シンボリックルールは、多種多様なメタデータ追跡機構をコンパクトに符号化することができる。しかしながら、ハードウェアレベルでは、主要な計算を遅くすることを避けるために、効率的な解釈のために調整されるルール表現が必要である。この目的で、コンクリートルールと呼ばれる低水準のルールフォーマットを導入する。直観的には、所与のポリシーのための各シンボリックルールは、コンクリートルールの等価

な集合へと拡張され得る。しかしながら、単一のシンボリックルールは無限の数のコンクリートルールを一般に生成し得るので、この精緻化を遅延して実行し、システムが実行する間に必要とされるようなコンクリートルールを生成する。

【０４９３】

PUMPハードウェアは、プロセッサのALU演算と並列に調べられ得るコンクリートルールのキャッシュを含む。命令が発行されるとき、ルールキャッシュは、キャッシュの中のすべてのコンクリートルールに対する、現在のマシン状態からのタグ(現在のPCタグ、現在の命令のオペランドのタグなど)の連想的な照合を実行する。一致が見つかる場合、キャッシュは、PCの新しいタグおよび命令の結果のタグを返す。それ以外の場合、プロセッサは、ルールミスハンドラへとフォルトし、ルールミスハンドラは、ポリシーのシンボリックルールを調べて、フォルトしたマシン状態が進行することを許容されるべきかどうかを判定するソフトウェアルーチンであり、されるべきである場合、ルールミスハンドラは適切なコンクリートルールを生成し、それをキャッシュにインストールし、フォルトした命令を再開する。許容されるべきではない場合、ルールミスハンドラは適切なセキュリティフォルトハンドラを呼び出す。コンクリートルールの一般的なフォーマットは次の通りである。

opgroup:(PC,CI,OP1,OP2,MR)⇒(PC',R')

ここで、入力および出力フィールドは固定されたタグである。ミスハンドラがあらゆるコンクリートルールをキャッシュに追加する前に対応する条件を確認するので、シンボリックルールフォーマットの「guard?」フィールドは必要ではないことに留意されたい。

【０４９４】

1つの便利な符号化の要領が、コンクリートルールの数を大きく減らす。所与のオペグループのためのすべてのシンボリックルールが、特定の入力または出力を「ワイルドカード」としてマークするのは、非常に一般的であることが認められる。たとえば、我々のROPポリシーでは、リターンオペグループおよび

【０４９５】

【数８】

$$\overline{return}$$

【０４９６】

オペグループのためのルールは、OP1、OP2、およびMRの入力が一致する必要はなく、R'の結果を産生する必要はない。使用されない入力フィールドのすべてのあり得る値に対してコンクリートルールを生成するのを避けるために、各オペグループおよび入力フィールドのためのdon't-careビットを格納するビットベクトルを定義し、これが、ルールキャッシュのルックアップにおいて対応するタグが実際に使用されるかどうかを判定する。同様に、don't-careベクトルは、デフォルトのタグが返されない未使用の出力をマークする(下ではこのために⊥を使用する)。

【０４９７】

たとえば、ROPポリシーでは、

【０４９８】

【数９】

$$\overline{return}$$

【０４９９】

オペグループは、OP1、OP2、MR、およびR[1]'に対してdon't-careビット集合を有するので、t1とタグ付けされたリターン命令がコードにおける唯一のリターンであり、コンパイラがt2およびt3とタグ付けされたリターンターゲットにのみリターンできることを、コンパイラが知っている場合、ルール2'の結果は以下の2つだけのコンクリートルールをもたらす。

【０５００】

【数１０】

$$\overline{return}: (t1, t2, \perp, \perp, \perp) \Rightarrow (\perp, \perp)$$

【０５０１】

【数１１】

$$\overline{return}: (t1, t3, \perp, \perp, \perp) \Rightarrow (\perp, \perp)$$

【０５０２】

「don't-care」の場所は⊥へとマスクされた。一方、シンボリックルール3'は以下の4つのコンクリートルールに対応する。

【０５０３】

【数１２】

$$\overline{return}: (\perp, \perp, \perp, \perp, \perp) \Rightarrow (\perp, \perp)$$

【０５０４】

【数１３】

$$\overline{return}: (\perp, t1, \perp, \perp, \perp) \Rightarrow (\perp, \perp)$$

【０５０５】

【数１４】

$$\overline{return}: (\perp, t2, \perp, \perp, \perp) \Rightarrow (\perp, \perp)$$

【０５０６】

【数１５】

$$\overline{return}: (\perp, t3, \perp, \perp, \perp) \Rightarrow (\perp, \perp)$$

【０５０７】
CIは
【０５０８】
【数１６】

$$\overline{return}$$

【０５０９】
のための「don't-care」の場所ではない(ルール3'はCIをワイルドカードとしてマークするがルール2'はマークせず、両方のルールが同じオペコードについてのものである)ので、とり得る可能な値の各々、すなわち⊥およびすべての識別子(この例ではt1、t2、およびt3のみ)に対して異なるコンクリートルールを得る。
【０５１０】
オペコードからオペグループへのマッピングおよびdon't-careベクトルはプログラム可能である。ROPポリシーは2つだけのオペグループ(returnおよび
【０５１１】
【数１７】

$$\overline{return}$$

【０５１２】
)を使用するが、他のポリシーはより多くを必要とすることがある。たとえば、プリミティブ型ポリシー(4.1章)は10個を使用する。
【０５１３】
構造化されたタグ。ROPよりリッチなメタデータタグを伴うポリシーでは、シンボリックルールからコンクリートルールへの変換は同じ一般的な傾向に従うが、詳細はより少し複雑になる。たとえば、テイント追跡ポリシー(4.4章)は、各々がテイントの任意のサイズの集合を記述する(所与のデータに寄与した可能性のあるデータソースまたはシステム構成要素を表す)メモリデータ構造へのポインタとして、タグを解釈する。loadオペグループのためのシンボリックルールは、ロードされた値のテイントが、命令自体、ロードのためのターゲットアドレス、およびそのアドレスにおけるメモリの、テイントの和集合であるべきであることを述べている。
load:(-,ci,op1,-,mr)→(-,ci∪op1∪mr)
何らかの瞬間において、(i)実行されるべき次の命令がld r0 r1でありそのタグがtciであり、レジスタr0がtpとタグ付けされたポインタpを格納し、アドレスpにおけるメモリがtφとタグ付けされた値を格納し、(ii)tciが集合{TA,TB}を表すデータ構造(たとえば、テイントidのアレイ)を指し、(iii)tpが{TC,TD}の表現を指し、(iv)tφが空集合を指すと仮定する。さらに、テイント{TA,TB,TC,TD}にこれまで遭遇したことがない、すなわち、ロードの結果をテイントするために使用すべき集合を表すデータ構造が現在、メモリの中にないと仮定する。この場合、ルールキャッシュルックアップはミスし、実行はルールミスハンドラへとフォルトし、ルールミスハンドラが適切なコンクリートルールを生成してそれをキャッシュにインストールし、場合によっては空間を作るために別のルールを除去する。このことは、新しいメモリ(たとえば、アドレスtnewにおける)を割り振ることと、{TA,TB,TC,TD}を表すようにそのメモリを初期化することとを必要とする。そうすると、生成されるコンクリートルールは、
load:(⊥,tci,tp,⊥,tφ)⇒(⊥,tnew)

である。命令が再開された後で、次のキャッシュルックアップが続き、r1の中のロードされた値はtnewとタグ付けされる。

【０５１４】

別個のタグの数(および、したがってルールキャッシュに対する圧力)を減らすために、メタデータ構造は標準的な形式で内部的に記憶され、タグは不変であるので共有が完全に利用される(たとえば、集合が共通のプレフィックスの部分集合を共有してコンパクトに表現され得るように、集合の要素は標準的な順序を与えられる)。これらの構造は、もはや必要とされないとき、(たとえば、ガベージコレクションにより)回収され得る。

【０５１５】

合成ポリシー。さらに1ステップ進むと、タグをいくつかの構成要素ポリシーからのタグのタプルへのポインタにすることによって、複数の直交するポリシーを同時に実施することができる(一般に、複数のポリシーは直交しないことがあり、6章においてこの点に戻る)。たとえば、上で概説したばかりのテイント追跡ポリシーと第1のROPポリシーを合成するには、各タグをタプル(r,t)の表現へのポインタにし、ここでrはROPタグ(コード位置識別子または⊥)であり、tはテイントタグ(テイントの集合へのポインタ)である。キャッシュルックアッププロセスは厳密に同じであるが、ミスが発生すると、ミスハンドラが、タプルの構成要素を抽出し、シンボリックルールの両方の集合を評価するルーチンにディスパッチする。この動作は、適用されるルールを両方のポリシーが有するときにだけ許容され、この場合、得られるタグは、2つのサブポリシーからの結果を格納するペアへのポインタである。

【０５１６】

命令修飾子および一時的なルール。一部のポリシー(たとえば、メモリ安全性)は、未使用のタグが動的に生成されることを必要とする。この効果を達成するための1つの方法は、moveなどの命令のタグを、未使用のタグに対する要求をポリシー管理システムに伝えるための修飾子として使用することである。

【０５１７】

【数１８】

$$\mathrm{Move} : (-, tpolicygen, -, -, -) \xrightarrow{1} (-, tnewtag)$$

【０５１８】

これは、tpolicygenでタグ付けされたmove命令が、未使用のタグを生成するための要求として解釈されることを述べている。結果tnewtagは、指定されるポリシーと関連付けられる固有のタグである。命令のタグtpolicygenはまた、このサービス要求に対する承認または能力として機能し、このタグがなければ、呼出しを行うことは可能ではない。信頼されるローダは、特別に指定されたコード領域(たとえば、4.2章のメモリ安全性ポリシーにおけるmallocルーチン)のみがこのタグでアノテートされることを保証する。「1」は一時的なルールを示し、その結果はハードウェアルールキャッシュへ恒久的には記憶されない(呼出しのたびに変化するので)。

【０５１９】

タグを初期化するためのコードは、「安定状態」のルールを覆すことも必要であり得る。たとえば、メモリ安全性ポリシーにおいて、mallocは、新しく割り振られたメモリ領域のタグを初期化する必要がある。標準的なルールは、ポインタがポインタと一致するように適切にタグ付けされたメモリ領域にのみ書き込むことができるというものである。しかし、mallocは、新しく作り出されたタグを新しい領域の中の各ワードへと書き込む間、このルールを覆すことを許容されなければならない。特別な修飾子タグ(mallocのみで使用される)をstore動作に与えることによって、これを行う。

store:(-,tmallocinit,t1,c2,F)→(-,(c2,t1))

【０５２０】
3．ポリシーシステムおよび保護
ポリシーシステムは、各ユーザプロセス内のメモリの別個の領域として存在する。これは、ミスハンドラのためのコード、ポリシールール、およびポリシーのメタデータタグを表すデータ構造を含む。このポリシーシステムをプロセスにおいて位置付けることは、既存のUnix（登録商標）プロセスモデルに対して最小限に侵害的であり、ポリシーシステムとユーザコードとの間の軽量な切替えを容易にする。ポリシーシステムは、次に説明される機構を使用してユーザコードから隔離される。
【０５２１】
メタデータ脅威モデル。明らかに、PUMPにより提供される保護は、攻撃者がメタデータタグを書き直せる場合、またはその解釈を変更できる場合、無意味である。我々のシステムは、そのような攻撃を防ぐように設計される。我々は、カーネル、ローダ、および(一部のポリシーに対しては)コンパイラを信頼する。具体的には、初期タグをワードに割り当てるために、また、必要な場合、ルールをポリシーシステムに伝えるために、コンパイラに依存する。ローダがコンパイラにより提供されるタグを保存すること、およびコンパイラからローダへの経路が、たとえば暗号学的な署名を使用して、改竄から保護されることを仮定する。各プロセスのための初期メモリイメージをセットアップする、標準的なUnix（登録商標）型のカーネルを仮定する(これらの仮定の一部を除去するためにマイクロポリシーを使用し、TCBのサイズをさらに低減することは可能であり得る-6章参照)。ルールキャッシュミスハンドリングソフトウェアは、正しく実装されるとさらに仮定する。これは小さいので、正式な検証の良いターゲットである。最近の成果[8]は、PUMPと類似したプログラミングモデルの実現可能性を実証している。
【０５２２】
主な関心事は、プロセスにおいて実行するユーザコードが、プロセスのポリシーにより提供される保護を損ねるのを防ぐことである。ユーザコードは、(i)タグを直接操作することが可能であるべきではなく、すなわちすべてのタグの変更が現在有効なポリシールールに従って実行されるべきであり、(ii)ミスハンドラにより使用されるデータ構造およびコードを操作することが可能であるべきではなく、(iii)ハードウェアルールキャッシュにルールを直接挿入することが可能であるべきではない。
【０５２３】
アドレス指定。ユーザコードによるタグの直接の操作を防ぐために、64bワードごとに添付されるタグは、それら自体が別々にアドレス指定可能ではない。具体的には、タグを読み取り、または書き込むために、タグまたはタグの一部分のみに対応するアドレスを指定することは可能ではない。すべてのユーザアクセス可能命令は、アトミックな単位として(データ,タグ)のペアに対して動作し、標準的なALUは値部分に対して動作し、PUMPはタグ部分に対して動作する。
【０５２４】
ミスハンドラアーキテクチャ。ポリシーシステムは、PUMPキャッシュへのミスに際してのみアクティブにされる。ポリシーシステムとユーザコードとの分離を実現するために、ミスハンドラ動作モードをプロセッサに加える。また、レジスタの保存および復元を避けるために、ミスハンドラだけが利用可能な16個の追加のレジスタで整数レジスタファイルを拡張する。フォルトした命令のPC、ルール入力(オペグループおよびタグ)、およびルール出力は、ミスハンドラモードにある間にレジスタとして現れる。また、コンクリートルールをキャッシュへとインストールすることを終えてユーザコードに戻る、ミスハンドラリターン命令を追加する。
【０５２５】
プロセッサがミスハンドラモードである間、PUMPの普通の挙動は関与しない。代わりに、単一のハードワイヤリングされたルールが適用される。ミスハンドラにより扱われるすべての命令およびデータは、いずれのポリシーによって使用されるタグとも別個の事前に定義されたミスハンドラタグでタグ付けされるべきである。これは、同じアドレス空間に

おけるミスハンドラコードおよびデータとユーザコードとの隔離を確実にする。ユーザコードはポリシーシステムデータまたはコードを扱うこと、または実行することができず、ミスハンドラはユーザデータおよびコードを偶発的に扱うことができない。ミスハンドラリターン命令は、ミスハンドラモードにおいてのみ発行することができ、ユーザコードがルールをPUMPに挿入するのを防ぐ。

【０５２６】

4．ポリシーおよび実験

このセクションでは、セキュリティ不変条件の多様な集合を実施するポリシーの4つの群を実装するためにどのようにPUMPを使用するかを示す。各群に対して、まず脅威モデルを概説する。次いで、脅威を軽減するポリシーおよび対応するルールを説明する。公開の脆弱性一式[10]からの例を使用して、各ポリシーがどのように典型的な悪用を捉えるかを示す。最も重要なことには、各ポリシーがシステムに課す負荷を説明する。文献からの類似するポリシーとの比較で終了する。

【０５２７】

ポリシーの負荷を評価するために、SPEC CPU2006[25]ベンチマークスイートからの28個のC、C++、およびFortranアプリケーションを使用し、それらをgem5シミュレーション環境[9]で64ビットAlpha ISA[1]についてシミュレーションする(gem5が失敗するtontoおよびxalancbmkベンチマークは除く)。gem5シミュレーションはPUMPを直接モデル化しない。むしろ、別々のPUMPシミュレータを通じて実行する命令トレースを産生する。

【０５２８】

【表１】

	なし		型		メモリ安全性					CFI						テイント追跡						最小限の構成要素	完全な構成要素
	ⓐミスハンドラなし	ⓑ重要ではないハンドラ	ⓒ insn/addr/other	ⓓ i/a/o + retaddr	ⓔサンドボックス	ⓕ1色	ⓖ8色	ⓗ32色	完全な保護 (ⓘ2^{64}色)	ⓙ 1ID [2]	ⓚ 2ID [2]	ⓛ CCFIR [51]	ⓜ PUMP JOP	ⓝ PUMP ROP	ⓞ PUMP CFI (ⓜⓝ)	ⓟ1テイント	ⓠストリームごと	ⓡライブラリごと	ⓢファイルごと	ⓣ関数ごと	ストリームおよびライブラリごと (ⓠⓡ)	ⓤ (ⓒⓔⓙⓟ)	ⓥ (ⓒⓔⓙⓟⓤ)
tag usage	-	-	✓		✓	✓	pp1	op2	mr	✓	r		pc	ci	✓	✓	✓	pc	✓		✓	ci	pp1	op2	mr	✓	r	all	
opgroups	-	-	10	11	7					2	3	3	2			7						13	
symbolic rules	-	-	15	16	13					2	3	4	4			8						49	53
initial tags	1	-	3	4	34	6	6	6	6	2	4	6	351	1.7K	2.1K	2	3	34	371	1.2K	36	16	2.2K
dyn. alloc. tags	0	-	0	0	2	2	18	241	102K	0	0	0	0	0	0	0	1	76	381	2.3K	77	4	252K
final tags	1	-	3	4	36	8	26	247	102K	2	4	6	351	1.7K	2.1K	2	4	110	752	2.5K	113	20	255K
concrete rules	1	-	17	19	14	18	93	578	386K	5	11	21	426	1.8K	2.3K	10	14	1.1K	6.5K	47K	1.2K	79	1.1M
metadata struct	-	-	-	-	2	2	2	2	2	-	-	-	-	-	-	1	2	5	13	50	5	4	5
metadata space	-	-	0	0	2	2	24	418	51K	0	0	0	2K	26K	28K	3	9	510	13K	573K	517	76	652K
policy-depend instrs	0	10	116	136	64					24	44	64	682			810						1240	1800
(dynamic)	0	10	25	26	20	20	32	36	38	16	26	31	117	120	120	56	174	430	1076	4110	430	211	710
runtime overhead (%)	8	8	8	8	8	8	8	8	13	8	8	8	8	8	8	8	8	8	9	314	8	8	38

図1: ポリシーおよび特性(28 SPEC CPU2006ベンチマークにわたる平均)

【０５２９】

この段階的なシミュレーションは、ここで説明されるポリシーに対しては十分であり、それは、計算へのポリシーの唯一の影響が、ポリシー侵害が発生したときに実行を中断することであるからである。4096エントリーのミスハンドラ前のルールキャッシュをシミュレーションする。

【０５３０】

2章で説明された抽象的なプログラミングモデルは、固有のタグの数、コンクリートルールの数、またはソフトウェアレベルでメタデータを表すために使用されるデータ構造のサイズについて、制限を課さない。PUMPがどのように実際に実行するかを理解するには、いくつかの質問を考慮しなければならない。どれだけの固有のメタデータタグを、所与のポリシー、アプリケーション、およびデータセットが実際に生成するか?O個のオペグループおよびT個のタグにより、理論的にはプログラムは$O \cdot T^5$個のコンクリートルールを必要とし得るが、典型的なケースは何か?メタデータタグの総数およびメタデータ表現のサイズがどのように性能に影響するか?タグおよびルールの使用法にどれだけの局所性があるか

?コンクリートルールの分解がどれだけ高コストか、およびルールキャッシュミスがどれだけ性能に影響するか?タグ、ルール、メタデータサイズ、またはルール分解時間が増大するにつれて、性能のグレイスフルデグラデーションがあるか?

【０５３１】

これらの影響の理解を始めるために、ポリシーの各々に対して、ランタイムオーバーヘッド以外のいくつかの特性を測定する(図1参照)。Tag usageは、どのタグがポリシーの中のルールのいずれによっても使用されないかを示す。Opgroupsは、ポリシーを捉えるために必要なオペグループの最小の数であり、使用するオペグループが少ないほど、コンクリートルールに対して得る圧縮は大きくなり、したがって実効的なPUMP容量が大きくなる。Symbolic rulesは、ポリシーを表現するために書くシンボリックルールの数である。Initial tagsは、実行が開始する前の初期メモリイメージの中のタグの数である。実行の間、より多くのタグが動的に割り振られる(dyn.alloc.tags)。さらに、テイント追跡のようなポリシーはテイントの集合の和集合を表すためのタグを作成し、合成ポリシーは個々のポリシータグのタプルを形成する。Final tagsは、10億個の命令シミュレーション期間の終わりに存在するタグの数を特定し、これは、ポリシーの複雑さについて何らかの感覚を与え、タグ作成のレートを推測するために使用され得る。Concrete rulesは、シミュレーション期間の間に生成される固有のコンクリートルールの数であり、シンボリックルールをコンクリートルールに分解するために必要な強制ミスの数、および実質的に、強制ミスレートを特徴付ける。Metadata structは、各タグにより指されるデータ構造のワード単位の平均サイズであり、無制限のメタデータを有する値を示す。Metadata spaceは、メタデータタグが指すポリシー関連の情報を保持するデータ構造のすべてに対して必要とされるワードの数であり、タグ自体を超えるメモリオーバーヘッドを特徴付ける。Policy-depend instrsは、シンボリックルールをコンクリートルールに分解するコードのために必要とされる命令の総数であり、これはポリシーの複雑さの理解に有用である。Policy-depend instrs(dynamic)は、シンボリックルールからコンクリートルールに分解するために実行されるポリシー依存命令の平均の数であり、これは、ポリシーの各々のためのミスハンドラのランタイムの複雑さを示すものである。ポリシー依存部分の影響は、ルールの複雑さ、メタデータのデータ構造、メタデータのデータ構造の局所性、および新しい結果タグを割り振る必要性に依存する。ミスハンドラのポリシー依存部分は、数十個の命令しか必要としない(図1の列

【０５３２】

【数１９】

Ⓑ

【０５３３】

参照)。Runtime overheadは、PUMPのないベースラインAlphaと比較した、ポリシーを実行するアプリケーションの実時間ランタイムの比である。ポリシーが使用されていない場合でも、タグおよびPUMPのためのハードウェア構造を追加するためだけに、いくらかのランタイムオーバーヘッドがある。具体的には、タグで拡張されたプロセッサ上のL1キャッシュは、より大きなタグ付けされたワードの幅に対応しながら同じサイクル時間を達成するために、PUMPのないベースラインAlphaの実効容量の半分である。このことは、タグで拡張されたプロセッサに対するより高いL1ミスレートをもたらす。このオーバーヘッドは第1の列

【０５３４】

【数２０】

(Ⓐ)

【０５３５】
において捉えられ、ここで、すべてのタグがデフォルトであり、単一のルールがあり、ミスハンドラが実質的に決して呼び出されない。
【０５３６】
図1の平均の数は、コンパクトさのための必須の簡略化である。ベンチマークはある範囲の効果を示す。これらは図3～図6において示されており、これらの図において、SPEC CPU 2006ベンチマークセットにおける複数のアプリケーションにわたる特性の分布を示すために箱ひげ図を使用する。図6は
【０５３７】
【数２１】

Ⓐ

【０５３８】
を超過するランタイムオーバーヘッドをプロットする。
【０５３９】
ランタイム性能のみを測定し、いくつかの他の重要なコストは脇に置いておく。具体的には、単純な実装形態では、ワードサイズのタグをキャッシュおよびメモリの1つ1つのワードに追加することは、最低で2倍の面積オーバーヘッドを課す。PUMPキャッシュおよびより大きなメモリの影響を加えると、これは4倍のエネルギーオーバーヘッドに変換され得る。注意深い最適化により、これらの数字を約30%の面積および50%のエネルギーへと、または場合によってはより低い水準に減らすことができることについて我々は楽観的であり、この主張を実証することに取り組んでいる。
【０５４０】
4.1 プリミティブ型
脅威モデル。データの誤解釈は、プロセッサを騙して意図されない動作を実行するようにするための一般的な方法である。ここで、敵のために実行されるコードがポインタとして任意のデータ値を使用すること、または命令としてワードを実行することを試みることができる、ある形態の低水準タイプの混乱が気がかりである。ランタイムにおいてデータが実行できないこと、およびコードが作成または変更できないことを、強制する(4.3章参照)。
【０５４１】
ポリシーおよびルール。ポリシー
【０５４２】
【数２２】

Ⓒ

【０５４３】
において、命令(tagged insn)、アドレス(addr)、およびすべての他のデータ(other)を分離するためにタグを使用する。命令を作成または変更することはできず、命令を実行す

ることだけができる。アドレスだけが、メモリアクセス命令とともに使用することができる。他のタイプのタグは、命令またはアドレスではないワードのための包括的なものとして使用される。以下のルールは、nop(たとえば)が実行される前には実際にはtagged insnであることを検証する。

nop: (-,insn,-,-,-)→(-,-) (5)

【０５４４】

アドレス計算が許容される。たとえば、addへの引数のうちの1つがアドレスであるとき、結果がアドレスである。

add: (-,insn,addr,other,-)→(-,addr) (6)

【０５４５】

load命令およびstore命令はポインタのみをデリファレンスし、命令を読み取らず、または書き込まないことも強制する。

load: (-,insn,addr,-,t)→(-,t)if t/=insn (7)

store: (-,insn,t,addr,-)→(-,t)if t/=insn (8)

【０５４６】

リターンアドレスが上書きされる(たとえば、スタックスマッシングを通じて)攻撃を防ぐのを助けるために、リターンアドレスのための第4のタグ(retaddr)を追加する拡張されたポリシー(

【０５４７】

【数２３】

Ⓓ

【０５４８】

)を考える。これを使用して、呼出しのリターンアドレスをタグ付けする(ルール9)。Alpha ISAにおける呼出しは、reg26にリターンアドレスを置くが、リターンは制御をこのレジスタの中のアドレスに移転する(レジスタはさらなる呼出しに際してスタックにスピルされる)。ルール10は、return命令が実行されるときにreg26の中の値がretaddrと分類されることを確認する。

call:(-,insn,addr,-,-)→(-,retaddr) (9)

return:(-,insn,retaddr,-,-)→(-,-) (10)

【０５４９】

インストルメント化されたコンパイラは、これらのタイプタグを推測し、それらをバイナリの初期メモリイメージ、すなわち、すべての生成された命令get tagged insn、スタックを割り振られたメモリへのポインタget tagged addr、および他のすべてgets tagged otherに適用することができ、新しいaddrタイプのワードは動的なメモリの割振りを通じて存在するようになる。しかしながら、現在はそのようなコンパイラがないので、異なる方法を使用して、シミュレーションおよび分析のためにこれらのタグを推測する。まず、バイナリ実行可能ファイルinsnの中のすべての命令をタグ付けする。addrとタグ付けされるべきワードを推測するために、実行トレースのafter-the-fact分析を使用し、各レジスタがいつどこからロードされるか、およびロードまたは記憶に対するポインタオペランドとしてレジスタが後で使用されるかどうかの記録をとる。他のすべてがotherとタグ付けされる。初期タグを取得するこの方法は、SPECベンチマーク上でポリシーを分類することのランタイムの影響を測定することを可能にする。しかしながら、このセットアップは、分類のポリシーが不必要な警告をもたらすことなくすべてのベンチマークを受け入れるのに十分寛容かどうかについて、どのような主張をすることも許容しない。これは、分類に必要な厳しいコンパイラの統合により引き起こされ、以下で提示する他のポリシーについては発生しない。

【０５５０】
保護の実証。プログラマが整数を関数ポインタにタイプキャストし、後でこの関数を呼び出す、CWE-843(分類の混乱)[30]の事例を使用する。このことは、otherとしてタグ付けされた即刻の値をレジスタにロードし、後の時点で、そのレジスタにより指されるアドレスにジャンプすることに変換される。ポリシー
【０５５１】
【数２４】

【０５５２】
を使用して、フォルトした命令を捉えることが可能であり、それは、ポリシーがaddrとタグ付けされた値のみへの間接的なジャンプを許容するからである。
【０５５３】
特性。ポリシー
【０５５４】
【数２５】

【０５５５】
および
【０５５６】
【数２６】

【０５５７】
は新しいタグを作成しない。
【０５５８】
【数２７】

【０５５９】
は、17個のコンクリートルールしか作成しない15個のシンボリックルールで符号化され得るが、
【０５６０】
【数２８】

Ⓓ

【０５６１】
は、16個のシンボリックルールおよび19個のコンクリートルールを必要とする。ルールの総数が小さいので、無視できるランタイムオーバーヘッドしか見えない(ミスハンドラなしポリシー
【０５６２】
【数２９】

Ⓐ

【０５６３】
と比較して0.01%未満)。したがって、PUMPは、ポリシーをハードウェアへと焼くことなく、簡単なハードワイヤリングされたタイプタグの実行を実現する。
【０５６４】
関連する研究。コンピュータアーキテクチャにおけるタグの最初の使用の1つは、マシンにおいてワードのタイプを区別することであった[34,23]。Symbolics LISP Machines[31]は、それらの36bのプリミティブワードの中からタグ付けのために2～8bを割り振り、命令、ポインタのいくつかのフレーバー、整数、浮動小数点数、および初期化されていない値を含む、プリミティブ型の集合を区別した。Berkeley SPUR[43]は6bのオブジェクト型のタグを使用した。
【０５６５】
4.2 空間的および時間的なメモリ安全性
脅威モデル。ポリシーの次のグループは、ヒープに割り振られたデータのメモリ安全性を対象とし、オブジェクトの境界を超えて参照すること(空間的な侵害)、領域が解放された後でポインタを通じて参照すること、または無効なポインタを解放すること(時間的な侵害)などの、プログラミングのエラーを攻撃者が悪用することを防ぐ。これは、ヒープスマッシングおよびポインタの偽造などの、典型的なヒープベースの攻撃を含む。ここで研究するポリシーは、ヒープに割り振られたデータのみを保護し、このデータについて、mallocおよびfreeへの呼出しがメモリ領域をどのようにセットアップし破壊するかを伝え、スタックの割振りまたはボックス化されていない構造体については扱わない。これらも原則として扱うことができ、何らかのコンパイラのサポートを仮定する([32]参照)。
【０５６６】
ポリシーおよびルール。直観的には、各々の新しい割振りに対して、未使用のblock id、たとえば(「色(color)」のための)cを用意し、(memset式で)cを新しく作成されたメモリブロックの中の各メモリ位置にタグとして書き込む。新しいブロックへのポインタもcとタグ付けされる。その後、ポインタをデリファレンスするとき、ポインタのタグが、ポインタの指すメモリセルのタグと同じであることを確認する。ブロックが解放されるとき、ブロックのすべてのセルのタグが、自由なメモリを表す定数Fに変更される。
【０５６７】
非ポインタに対して追加のタグ⊥を使用し、色cまたは⊥のいずれかであるタグのためにtを書き込む。1つの追加の詳細、すなわちメモリセルがポインタを格納し得ることに注意する。よって、メモリの中のワードは2つのタグと関連付けられなければならない。各メモリセルのタグをペア(c,t)へのポインタにすることによってこれを扱い、ここで、cはこのセルが割り振られたメモリブロックのidであり、tはセルに記憶されているワードのタグ

である。loadおよびstoreのためのルールは、これらのペアをパックおよびアンパックすることとともに、各メモリアクセスが有効である(すなわち、アクセスされるセルがこのポインタにより指されるブロック内にある)ことを確認することを引き受ける。

load: (-,-,c1,-,(c2,t2))→(-,t2)if c1=c2 (11)

store: (-,-,t1,c2,(c3,t3))→(-,(c3,t1))if c2=c3 (12)

【０５６８】

アドレス計算の動作はポインタタグを保存する。

add:(-,-,c,⊥,-)→(-,c)

【０５６９】

ポインタのタグが割振りからのみ発生し得るという不変条件を維持するために、(定数のロードのように)無からデータを作成する動作は、自身のタグを⊥に設定する。

【０５７０】

2章の終わりで説明された命令修飾子および一時的なルールを使用してメモリ領域をタグ付けするために、mallocおよびfreeを拡張する。mallocにおいて、一時的なルールを介して新しい領域へのポインタのための未使用のタグを生成する。次いで、新しくタグ付けされたポインタを使用して、タグ付けされたポインタを返す前に、特別なstoreルールを使用して割り振られた領域の中の1つ1つのワードに0を書き込む。

store:(-,tmallocinit,t1,c2,F)→(-,(c2,t1)) (14)

逆に、freeは、修飾されたstore命令を使用して、メモリ領域をfreeのリストに返す前に、その領域を割り振られていないものとして再タグ付けする。

store:(-,tfreeinit,t1,c2,(c3,t4))→(-,F) (15)

【０５７１】

このポリシーのいくつかの変形を実装して、様々な性能/セキュリティのトレードオフを示した。第1の(

【０５７２】

【数３０】

Ⓔ

【０５７３】

)では、単一の色を、所与のソースモジュールにより割り振られるすべてのメモリ領域に割り当てる。このサンドボックスポリシーは、ソフトウェアベースのフォルトの隔離[46]と同様の、プロセス内でのモジュールごとの隔離を実現する。次の変形において、異なる数の色を使用して、空間的および時間的なメモリ安全性の最弱の形態をもたらし割り振られたメモリを割り振られていないメモリと区別するだけである単一の色(

【０５７４】

【数３１】

Ⓕ

【０５７５】

)だけから、8個の色(

【０５７６】

【数３２】

Ⓖ

【０５７７】
)および32個の色(
【０５７８】
【数３３】

Ⓗ

【０５７９】
)へと、mallocへの連続的な呼出しにより返される領域をタグ付けする。色の数を増やすことは、色の再使用により生じるエイリアシングの影響を減らす。最終的に、色のために64ビットのタグ空間全体を使用して、正確で完全なメモリ安全性ポリシー(
【０５８０】
【数３４】

Ⓘ

【０５８１】
)を実装する。
【０５８２】
保護の実証。Julietスイート[10]からの2つの攻撃を使用する。第1は、Fとタグ付けされたメモリ位置からのロードを試みるアプリケーションがポリシー
【０５８３】
【数３５】

Ⓘ

【０５８４】
を使用して捉えられる、CWE-416(解放後の使用)のケースである。第2は、バッファが割り振られてその境界を超えて後で書き込まれ(strcpyを使用して)、有効な領域を上書きする、CWE-122(ヒープベースのバッファオーバーフロー)[27]のケースである。
【０５８５】
【数３６】

Ⓘ

【０５８６】
を使用して、PUMPはFとタグ付けされたメモリ位置に文字を置くことを試みる命令を中断

する。
【０５８７】
特性。(
【０５８８】
【数３７】

【０５８９】
)およびポリシーを少数の色(
【０５９０】
【数３８】

【０５９１】
および
【０５９２】
【数３９】

【０５９３】
)でサンドボックスすることは、いくつかのタグを割り振り、少数のルール(32色の場合600未満)を作成するだけである。これらは、ランタイムオーバーヘッドを追加せず、ルールはすべてキャッシュに収まる。完全なメモリ安全性(
【０５９４】
【数４０】

【０５９５】
)はより高価である。これは、メモリの割振りごとに1つのタグを割り振り、メモリの割振りに対して、新しいコンクリートルールがキャッシュに追加されなければならない。これは、ミスハンドラを通じたより多くのトリップを必要とし、ベンチマークのいくつかでは、コンクリートルールの集合がキャッシュより大きいことを意味する。それでも、ルールの局所性は高く(図7参照)、平均のランタイムオーバーヘッドは13%にすぎない。GemsFDTDに対して、約130%の最大のオーバーヘッドを確認する。
【０５９６】
関連する研究。Clause他[16]は初めて、メタデータのテインティングを使用した空間的および時間的なメモリ保護を実証した。Deng他[19,20]は、ハードウェアのタグ管理を用いてこのテインティングを支持した。HardBound[21]は、監視されているコードと監視されていないコードとの間のデータ構造レイアウトの互換性を維持するために影の空間に境

界情報を置く、空間的なメモリ安全性に対する手法である。HardBoundのランタイムオーバーヘッドは10～20%である。Watchdog[32]は、各割振りのために固有の識別子を生成することにより時間的な侵害をさらに防ぐHardBoundの追跡調査であり、これは24%の平均のランタイムオーバーヘッドを有する。SoftBound[33]は、HardBoundのように、Cに対する空間的なメモリ安全性を提供するが、ランタイムオーバーヘッドの増大(SPECおよびOldenベンチマークで67%)というコストを伴う、ソフトウェアの手法である。Baggy Bounds[3]も、空間的な侵害のみを対象とし、SPEC2000で60%のランタイムオーバーヘッドを達成する。

【０５９７】

4.3 制御フロー整合性

脅威モデル。このポリシーのグループは、コード再使用攻撃を対象とする。攻撃者は、データを実行できず、またコードのインジェクションもしくは変更もできないという標準的な仮定[2]を行う(この仮定を実施するために、合成されたポリシーについて4.5章で行ったように、4.1章からのプリミティブ型ポリシーを使用することができる)。代わりに、攻撃者は、悪意のある挙動を引き起こすために、既存のコードの断片(ガジェット)を一緒につなぐことを試みる。

【０５９８】

ポリシーおよびルール。すべてのコード再使用攻撃の共通の要素は、元のバイナリに存在しない制御フローを導入することである。プログラムの制御フローグラフに対して各々の間接的な制御フロー(計算されたジャンプ)を検証する、CFIポリシー群を実装する。コードは不変であるので、直接のジャンプは動的に確認される必要がない[2]。まず、[2,51]の粗粒度のCFIポリシー(

【０５９９】

【数４１】

【０６００】

および

【０６０１】

【数４２】

【０６０２】

)を実装する。

【０６０３】

【数４３】

【０６０４】

は、すべての間接的な呼出しをタグ付けし、ジャンプを間接参照し、命令およびその潜在的なターゲットを単一のタグ{f}とともに返す。間接的な制御フローのソースである命令を実行すると、このタグをPCに転送する。

indir: (-,{f},-,-,-)→({f},-) (16)

【０６０５】

すべての他の命令がφとタグ付けされる。PCが{f}とタグ付けされるときはいつでも、現在の命令は同じタグを有しなければならない。

【０６０６】

【数４４】

$$\overline{indir} : (pc, ci, -, -, -) \rightarrow (\emptyset, -) \text{ if } pc \subseteq ci \quad (17)$$

【０６０７】

ポリシー

【０６０８】

【数４５】

Ⓚ

【０６０９】

は、(そのタグがrを格納する)リターンから発生する制御フローを、(そのタグがcを格納する)間接的な呼出しおよびジャンプから発生する制御フローとは別々に追跡するために、より多くのタグ(φ、{r}、{c}、および{r,c})を使用する。ポリシー

【０６１０】

【数４６】

Ⓛ

【０６１１】

は、(そのタグがpを格納する)特権コードへのリターンのために、2つの追加のタグ({p}および{p,c})を用いて

【０６１２】

【数４７】

Ⓚ

【０６１３】

を拡張し、重要なコードの断片に対する追加の保護を可能にする[51]。

【０６１４】

Goktas他[22]の攻撃が示すように、これらの緩いCFIポリシーは、洗練されたコード再使用攻撃に対しては十分な保護とはならない。我々は、Goktas他が「理想的なCFI」として説明した、細粒度のCFIポリシーの集合も実装した。まず、2つの直交するポリシーである、間接的なジャンプおよび呼出しとそれらのターゲットとの関連を正確に追跡するPUMP JOP(

【０６１５】

【数４８】

【０６１６】
)と、2章(ルール1'～4')で提示されたように、リターンのために同じことを行うPUMP ROP(
【０６１７】
【数４９】

【０６１８】
)とを導入する。最後に、これらの2つのポリシーをPUMP CFI(
【０６１９】
【数５０】

【０６２０】
)へとマージし、これは、すべての間接的な制御フローを正確に追跡して検証する単一のポリシーである。すべてのこれらのポリシーにおいて、コンパイラまたはリンカは、間接的な制御フローの妥当な過大近似を計算し、それに従って命令をタグ付けすると仮定される。
【０６２１】
保護の実証。「無害な」関数への単一の呼出しからなる特別に作られたプログラムに対して、これらのポリシーをテストした。コードは、決して呼び出されない「悪い」関数も含み、実行パスの一部ではないが意図されない挙動を引き起こすために悪用され得る休眠中のガジェットを模擬する。リターン指向攻撃をシミュレートするために、無害な関数の中のインラインアセンブリが、悪い関数のアドレスでスタックポインタを上書きし、実行を騙して悪い関数へリターンさせる。ポリシー
【０６２２】
【数５１】

【０６２３】
は、悪いリターンが有効な制御フローの集合の中にないことに気付くことによって、このシミュレートされた攻撃を検出する。
【０６２４】
特性。上のCFIポリシーの各々は、わずか2～4個のシンボリックルールを用いて非常にコンパクトに符号化され得る。より簡単なポリシー(
【０６２５】
【数５２】

ⒿⓀ

【０６２６】
および
【０６２７】
【数５３】

Ⓛ

【０６２８】
)も、非常に少数のタグおよびコンクリートルールしか必要としない。図1に示されるように、これらのうちの最大のものである(
【０６２９】
【数５４】

Ⓛ

【０６３０】
)は、6個の定数タグを使用し、わずか21個のコンクリートルールしか必要としない。そのような小さなワーキングセットサイズにより、これらのポリシーは、空のポリシーと比較して、観測可能なランタイムオーバーヘッドを招かない。より強いCFIポリシー(
【０６３１】
【数５５】

Ⓜ,Ⓝ,Ⓞ

【０６３２】
)をSPECベンチマークに適用することは、これらのポリシーのために最大で数千個のコンクリートルールを生み出し、これは4096エントリーのミスハンドラ前のPUMPキャッシュに完全に収まる。その結果、追加のランタイムオーバーヘッドなしで、理想的なCFIの追加の保護を得る。完全なCFI(
【０６３３】
【数５６】

Ⓞ

【０６３４】
)ポリシーは、アプリケーションのための制御フローグラフを記憶するのに平均で28000個のワードを必要とする(このシミュレーションでは、gem5によって生成された命令トレ

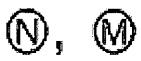
ースから制御フローグラフを抽出する。実際には、制御フローグラフは、このシミュレーションで決して行われない許容される制御フローパスを含む、示されているものよりも多くの空間を要する)。

【０６３５】

関連する研究。CFI[2]は、一般的なコード再使用攻撃に対する魅力的な防御をもたらすが、高価すぎると考えられることが多かった。最近の研究[51]は、分岐ターゲットの確認を行うことと、ガジェットの構築の成功に対するさらなる防御としてランダム化を行うこととの両方のために、「跳躍台」を使用する低オーバーヘッドのCFI方式を実証した。しかしながら、この研究は、ポリシー

【０６３６】

【数５７】



【０６３７】

および

【０６３８】

【数５８】

【０６３９】

が行うような特定のターゲットを伴う特定のリターンポイントではなく、2章のルール1～4の単一ターゲットの例と同様に、許容される呼出しおよびリターンターゲットを閉じ込めるだけであり、攻撃[22]に対して脆弱なままであり、我々の

【０６４０】

【数５９】

【０６４１】

および

【０６４２】

【数６０】

【０６４３】

が行うようなプロシージャ内のCFIを扱わない。

【０６４４】

4.4　テイント追跡

脅威モデル。このポリシーは、入力のサニタイゼーションを行わないプログラムに攻撃者

が歪曲されたデータを入力し、意図されないまたは悪意のある挙動(たとえば、SQLまたはOSコマンドインジェクション)を呼び出すケースを扱う。

【０６４５】

ポリシーおよびルール。テイント追跡は、信頼されないデータが慎重に扱うべき動作へといつ流れ込み得るかを検出することによって、これらの脅威を軽減する。PUMPは、無限の数のソース、ソースごとの別個のテイント、および各データ上の複数のテイントを用いて、細粒度のテイント追跡を容易にし、各タグがソースidの集合に対するポインタとなることを可能にする。ある値の上のテイントは、それを計算するために使用される値の上のテイントの和集合である。典型的なテイントの伝播ルールは以下を含む。

add: (-,ci,op1,op2,-)→(-,ci∪op1∪op2) (18)

load: (-,ci,op1,-,mr)→(-,ci∪op1∪mr) (19)

store: (-,ci,op1,op2,-)→(-,ci∪op1∪op2) (20)

【０６４６】

我々が研究するすべてのポリシーは、初期テイントの数およびソースのみが異なる、シンボリックルールの同じ集合を使用する。入力ソース(

【０６４７】

【数６１】

【０６４８】

および

【０６４９】

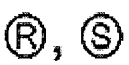
【数６２】

【０６５０】

)によるもの、ならびに、コード領域(

【０６５１】

【数６３】



【０６５２】

および

【０６５３】

【数６４】

【０６５４】

)によるものという、2つの異なる方法でテイントを導入する。
【０６５５】
【数６５】

【０６５６】
において、すべての入力ソース(すなわち、SPECプログラムのための、標準的なI/Oストリームおよび入力ファイル)のために単一のテイントを使用する。これは、単一ビットのテイントtが単に、信頼されないソースからのいずれのデータも、tとタグ付けされた値を計算するために使用されているかどうかを示す、最も以前の研究[41]と同様である。ポリシー
【０６５７】
【数６６】

【０６５８】
は、固有のテイントidを各入力ストリームに割り当てることによって
【０６５９】
【数６７】

【０６６０】
を拡張し、ストリームの数に制限はない。プログラムコードによるテインティングは、信頼されないライブラリおよびバグのある構成要素からの保護を行う。(i)各ライブラリ(
【０６６１】
【数６８】

【０６６２】
)、(ii)各々の含まれるヘッダファイル(
【０６６３】
【数６９】

【０６６４】

)、または(iii)コードの中の各関数(
【０６６５】
【数７０】

【０６６６】
)に対して、固有のテイントを使用することによって、粒度を変化させる。これらのポリシーは、命令を関連するテイント識別子でタグ付けすることをコンパイラに要求する。最終的に、
【０６６７】
【数７１】

【０６６８】
と
【０６６９】
【数７２】

【０６７０】
を組み合わせて、ポリシー
【０６７１】
【数７３】

【０６７２】
を形成する。
【０６７３】
保護の実証。systemシステムコールに渡されるlsコマンドの引数をパラメータ化することのみをユーザが許容されている、CWE-78(OSコマンドインジェクション)[29]のケースを考える。悪意のあるユーザは、コマンド終了文字で開始するパラメータ文字列を、任意のコマンドとともに追加する。これにより、「信頼されない」ものとしてタグ付けされているサニタイゼーション後のデータが、execveシステムコールに引数として渡されることになる。ポリシー
【０６７４】
【数７４】

Ⓤ

【０６７５】
を使用すると、PUMPは、信頼されていないものをシステムコールテイントと組み合わせようとしていることを認識すると、実行を停止する。
【０６７６】
特性。すべてのこれらのポリシーは、7個のオペグループに関して定義される、8個のシンボリックルールの同じ集合を使用する。最初の2つ(
【０６７７】
【数７５】

Ⓟ

【０６７８】
および
【０６７９】
【数７６】

Ⓠ

【０６８０】
)は、入力ストリームをテイントソースとして使用する。
【０６８１】
【数７７】

Ⓠ

【０６８２】
に対しては、すべてのSPECプログラムにわたり、平均で2つのソースしかなく、10個および14個のコンクリートルールしか必要としない。その結果、これらのポリシーは、顕著なランタイムオーバーヘッドを招かない。ポリシー
【０６８３】
【数７８】

Ⓡ-Ⓤ

【０６８４】
に対しては、より大きなワーキングセットがある。
【０６８５】

関数によるテイントの実験(
【０６８６】
【数７９】

【０６８７】
)は、意図的に機構を極限まで押し出し、現実に有用である可能性が高い粒度よりも細粒度のタグ付けを実現する。その多数のテイントは、PUMPキャッシュが一度に保持できるものよりも1桁多いルールをもたらす。さらに、タグを扱うオーバーヘッドが大きくなる(4110個の命令)。これらの要因は、314%の平均のランタイムオーバーヘッドをもたらす。これは、PUMP機構が複雑なポリシーに耐えながら、それでもそれらをサポートできることを示している。ファイル(
【０６８８】
【数８０】

【０６８９】
)ごとのテイントはまた、有用である可能性が高いものよりも細粒度であり、より小さなルールの集合およびより堅いミスハンドラの分解能により、9%という低いランタイムオーバーヘッドを達成する。
【０６９０】
ライブラリ全体にテイントが割り当られる、ポリシー
【０６９１】
【数８１】

【０６９２】
および
【０６９３】
【数８２】

【０６９４】
は、より合理的な使用法を代表する。ここで、平均のランタイムオーバーヘッドは、ミスハンドラなしの場合とは区別不可能なままである。これは、基本的に追加のランタイムオーバーヘッドなしで、PUMPが(1bまたは4bのテイントを使用する従来の成果と比較して)よりリッチなモデルを表現しサポートすることが可能であることを示している。さらに、これらの様々なテイントのケースにわたり、最終的なタグは初期の動的に割り振られるタ

グの2～3倍にすぎず、これは、単集合ではないタグの集合を作成しながら、理論的に最悪の場合の冪集合の効果に近いものが何もないことを示している。

【０６９５】

関連する研究。テイント追跡を使用して対処された脆弱性は、フォーマットストリング攻撃[48,17,41,18,12]、クロスサイトスクリプティング[48,18,12]、メモリ悪用[48,17,41,14,18,36,12]、コードインジェクション[48,17,18,12]、およびその他[49,18]を含む。大半の既存の研究は、プログラムがインストルメント化されるソフトウェア技法に注目している。典型的には、これらは、大きなランタイムオーバーヘッド(しばしば2倍を超える、一部は最大で20倍)を、他の障壁(たとえば、動的なバイナリ変換におけるレース条件の取扱い[15])とは別にもたらす。

【０６９６】

DIFT[41]、Minos[17]、およびSIFT[35]のようなハードウェア手法は、単一のテイントビットを使用する。Rakshaは、on-core[18]とdedicated-coprocessor[26]の変形の両方で、4ビットのタグを使用して最大で4つの同時ポリシーをサポートする。対照的に、我々は、場合によっては異なるレベルの信頼性を伴う、複数の信頼されないソースに対応するテイントの任意の集合を許容する。よりフレキシブルなタグ付け方式が5章で論じられる。

【０６９７】

4.5 合成ポリシー

これらのポリシーの各々は潜在的に有用であるが、一度に実施するポリシーを1つだけ選ばなければならない、たとえば、バッファオーバーフローの脆弱性に対する保護を行うか、またはコマンドインジェクションの脆弱性に対する保護を行うかの選択を行わなければならないとすれば残念である。そうではなく、通常は、複数のポリシーを合成することに由来する保護を望む。実際に、我々の個々のポリシーの一部は、完全な脅威から守るために相互の保護を必要とする(データを実行できないことと、コードを作成または変更できないこととを確実にするために、CFIはタイプ保護に依存する)。

【０６９８】

合成は、タグの数ならびに作成されるルールの数を潜在的に増やすことがあり、それにより、性能をかなり低下させる。この合成の影響を特徴付けるために、2つの合成ポリシーを実装する。まず、3つのプリミティブ型(

【０６９９】

【数８３】

【０７００】

)、単純なメモリ安全性(

【０７０１】

【数８４】

【０７０２】

)、CCFIR(

【０７０３】

【数８５】

【０７０４】
)、およびシングルビット入力テイント(
【０７０５】
【数８６】

【０７０６】
)という、4つの保護クラスの各々の最も簡単な事例に基づいて、相当に最小限のポリシーを実装する。第2に、4つのプリミティブ型の合成(
【０７０７】
【数８７】

【０７０８】
)であるより完全で強力な保護を実装し、その4つのプリミティブ型は、完全な空間的および時間的なメモリ安全性(
【０７０９】
【数８８】

【０７１０】
)、PUMP CFI(
【０７１１】
【数８９】

【０７１２】
)、およびストリーム入力テイントごとおよびライブラリコードテイントごとの合成(
【０７１３】
【数９０】

【０７１４】
)を実装する。
【０７１５】
特性。単純な合成ポリシー
【０７１６】
【数９１】

【０７１７】
はキャッシュに収まり、構成要素であるポリシーと同じ性能を有する。より大きな合成ポリシー
【０７１８】
【数９２】

【０７１９】
に対して、すべてのポリシーを解決する必要性は、ミスハンドラにおけるルールの分解に必要とされる命令の数を、38から710へとかなり増やす。最終的なタグの増大は2.5倍にすぎず、合成タグからのいくらかの積集合の影響があったが最悪の場合のシナリオに近いものではまったくないことを示唆している。さらに、コンクリートルールは約3倍に増えただけであり、これらはともに、タグのより大きな集合および追加のオペグループによるものである。より大きなコンクリートルールの集合(ここではPUMPキャッシュ容量よりはるかに大きい)と、増大したミスハンドラのコストとの組合せは、平均で38%のオーバーヘッドをもたらし、最悪の場合のオーバーヘッドは280%(GemsFDTD)に達する。これは、PUMPが、性能に対する影響という犠牲を伴って、合成に起因するルールの大きな集合を扱うことができることを示している。多くのアプリケーションでは、オーバーヘッドは穏当なままであるが、一部に対しては、法外に大きくなる。このことは、関数によるテイントの実験(
【０７２０】
【数９３】

【０７２１】
)とともに、我々の進行中の研究の焦点である、このようなリッチな合成に対して適当な性能を達成するために、ミスハンドラサービス時間を減らすための追加のソフトウェアおよびマイクロアーキテクチャの最適化が必要であることを示している。
【０７２２】
4.6　議論
ルールの総数は、ルールの局所性を完全には捉えず、その結果、実効的なワーキングセットのサイズを完全には捉えない。図7は、gccベンチマークに対する10億回の命令シミュレーション内の各々の100万回の命令シーケンス内で使用される固有のルールの数の累積分布関数(CDF)を示す。これは、フルのメモリ安全性(
【０７２３】
【数９４】

【０７２４】
)が非常にタイトなワーキングセット(大半が3000未満)を有することを示している。このことは、フルのメモリ安全性が任意の非合成ポリシーのうちで最大の数のコンクリートルールを有するので、重大である。この局所性は、ルールのはるかに大きな集合にもかかわらず、性能のオーバーヘッドが低いままである理由を説明するのを助ける。完全なCFI(
【０７２５】
【数９５】

【０７２６】
)はより大きなワーキングセットを有するが、ルールの完全な集合は4096エントリーのキャッシュに完全に収まるので、削除はない。
【０７２７】
以前の研究は、安全性およびセキュリティのポリシーをコンパクトに表現または近似するための賢い方式を使用してきたが(たとえば、[42])、これは意図されるポリシーに対する危殆化であることが多く、コンパクトさと引き換えに複雑さを大きくすることがある。我々は、追加のランタイムオーバーヘッドをほとんどまたはまったく伴わずに、より完全にかつより自然に、セキュリティポリシーの必要性を捉えるよりリッチなメタデータを含めることが、可能であることを示す。メタデータの表現およびポリシーの複雑さに固定された限界を課すのではなく、PUMPは性能のグレイスフルデグラデーションを実現する。これは、ポリシーが、一般的なケースの性能およびサイズに影響することなく、必要な場合により多くのデータを使用することを可能にする。これはさらに、より複雑なポリシーを簡単に表現して実行できるので、ポリシーの付加的な改良および性能調整を可能にする。
【０７２８】
4.7　他のマイクロポリシー
我々は、このプログラミングモデルが他のポリシーのホストを符号化できると考えている。情報フロー制御(たとえば、[6,37,40,24,8])は、ここでは簡単なテイント追跡モデルよりリッチであるが、暗黙的なフローを追跡することは、RIFLE型のバイナリ変換[44]により、または、コンパイラからのある程度のサポートとともにPCタグを使用することにより、サポートされ得る。マイクロポリシーは、軽量なアクセス制御およびコンパートメント化をサポートすることができる[47]。偽造不可能なリソースを区別するためにタグが使用され得る[50]。固有の生成されたトークンが、データを検印して保証するための鍵として機能することができ、これを次いで強力な抽象化のために使用することができ、データが認証されたコード構成要素のみによって作成され破壊されることを保証する。マイクロポリシールールは、不変性および線形性などのデータ不変条件を強制することができる。マイクロポリシーは、データまたは将来のためのフル/エンプティビットなどの同期プリミティブに対するアウトオブバンドメタデータとして(たとえば、[5])、または、ロック状態でレース条件を検出するための状態として(たとえば、[38,52])、並列処理をサポートすることができる。システムアーキテクトは、1つ1つの行を検査することなく、または書き直すことなく、既存のコードに固有のマイクロポリシーを適用することができる。
【０７２９】
5．関連する研究
我々の例示的なポリシーに関する研究が、4章で触れられている。ここで、ハードウェアタグの確認および伝播に、より全般的に関連する研究を論じる。下で述べられる少数の例

外を除き、従来の成果の研究は、ハードワイヤリングされた、または大きく制約されたポリシーを伴う、タグビットの小さな集合を使用する(図2参照)。初期のテイントハードウェアは、各ワードに添付された単一のテイントビットを、ハードワイヤリングされたテイント伝播論理とともにサポートしていた。より後のシステムには、複数の独立なテイントタグ(たとえば、[18])、複数のビットタグ(たとえば、[45])、およびよりフレキシブルなポリシー(たとえば、[19])を扱うための能力が加わった。一度に2つ以上のポリシーをサポートするための唯一の設計であるRakshaは、最大で4つのテイント追跡ポリシーをサポートしていた[18]。

【０７３０】

我々のものに最も近い従来のシステムは、Aries[11]、FlexiTaint[45]、Log-Based Architecture(LBA)[13]、およびHarmoni[20]であり、これらはすべて、ソフトウェアハンドラにより支持されるプログラム可能なルールキャッシュを提案する。FlextTaintおよびLBAのみが、プログラム可能なルールキャッシュを使用する具体的なセキュリティポリシーの例を詳述している。LBAを除くすべてのケースにおいて、ルールキャッシュは、動作の2つのオペランドのための2つの入力に基づき、単一の出力を産生し、一方で、PUMPは潜在的に最大で5つの入力を受け、2つの出力を産生する。図1は、これらのタグソースおよび宛先が我々のセキュリティポリシーにおいてどのように使用されるかを要約する。LBAは潜在的に複数の入力を受けるが、ハードウェアにおけるメタデータの産生を扱わない。LBAを高速にしたLBAにおける革新の一部(たとえば、テイントの合成をあきらめることを含む、一般的な伝播追跡を単項継承の追跡に制約すること)は特に、我々の解決法が提供する一般性をあきらめている。これらの制約されたポリシーでも、LBAは大半の単一のポリシーに対して、8%という我々の平均のオーバーヘッドと比較して、約50%のランタイムオーバーヘッドを有する。我々がここで示したポリシーは、追加のタグ入力および出力と、よりリッチなタグメタデータとの両方が原因で、FlexiTaintによりサポートされるポリシーよりリッチである。

【０７３１】

【表２】

参照番号	HW/SW	タグ サイズ	タグ IO	セキュリティポリシー
[14, 17, 35]	H	1b	2/1	ハードワイヤリングされたテイント
[41]	H+S	1b	2/1	ハードワイヤリングされたテイント
[12]	H+S	1b	2/1	限定されたプログラム
[18, 26]	H+S	4b	2/1	4つの1bポリシー 限定されたSWプログラム
[45]	H+S	4b	2/1	限定されたプログラム
[4]	H+S	1-8b	1/-	伝播なし、大半がメモリ
[19]	H	var	2/1	FPGA再構成可能
[20]	H+S	var	2/1	限定されたプログラムおよびテーブル
[13]	H+S	64b	5/-	別個の拡張されたコア上のSW
PUMP	H+S	64b	5/2	細粒度のSWで定義されるポリシー

図2: ハードウェアタグ付け手法

【０７３２】

6. 今後の研究

PUMP設計は、フレキシビリティと性能の魅力的な組合せを実現し、多くの場合において専用の機構に匹敵する単一ポリシーの性能を伴う低水準で細粒度のセキュリティポリシーの多様な集合体をサポートしながら、ルールの複雑さが増すときに性能の概ねグレイスフ

ルデグラデーションとともによりリッチな合成ポリシーをサポートする。この設計空間をより完全に理解するために、いくつかの問題にはさらなる調査が必要である。まず、実行中のハードウェア実装形態があるとき、PUMPハードウェアおよび低水準ソフトウェアを、ホストオペレーティングシステムおよびソフトウェアツールチェーン(たとえば、コンパイラ、リンカ、およびローダ)に統合する必要がある。第2に、PUMPにより提供される機構が、その固有のソフトウェア構造を保護するために使用され得るかどうか思案している。我々は、PUMPを使用して「コンパートメント化」マイクロポリシーを実装し、これを使用してミスハンドラコードを保護することによって、特別なミスハンドラ動作モードを置き換えることができると考えている。最後に、各々により提供される保護が他者とは完全に独立である、ポリシーの直交する集合を合成することは簡単であることがここでわかった。しかし、ポリシーはしばしば相互作用する。たとえば、情報フローポリシーは、メモリ安全性ポリシーにより割り振られる未使用の領域にタグを付ける必要があり得る。ポリシー合成は、表現と効率的なハードウェアサポートの両方において、さらなる研究が必要である。

【０７３３】

【表３】

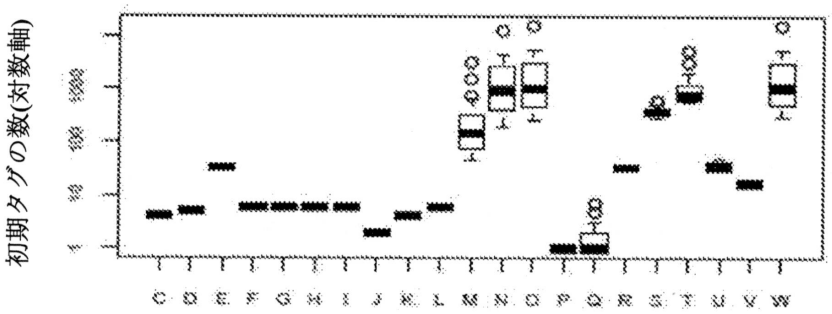


図3: 初期タグの分布

【０７３４】

【表４】

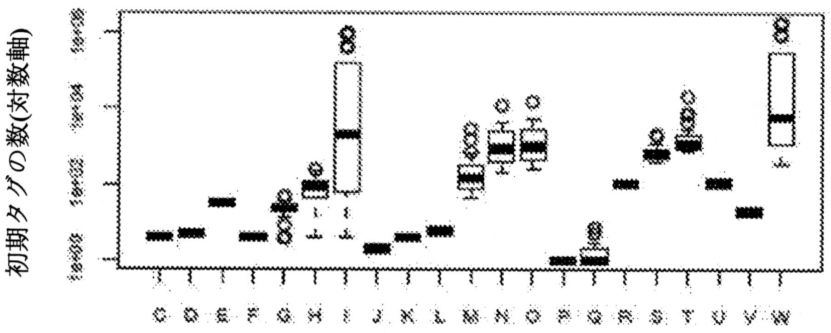


図4: 最終的なタグの分布

【０７３５】

【表５】

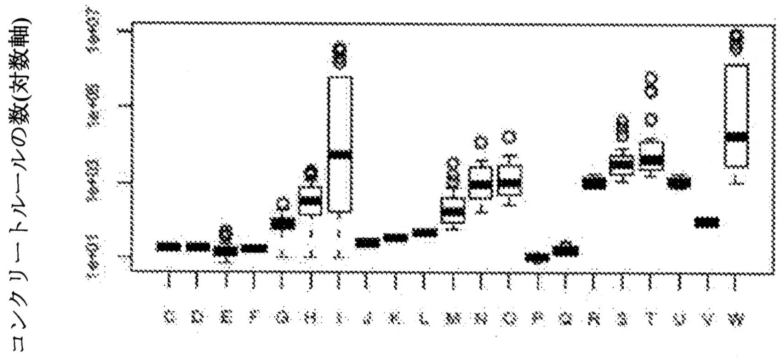


図5: コンクリートルールの分布

【０７３６】
【表６】

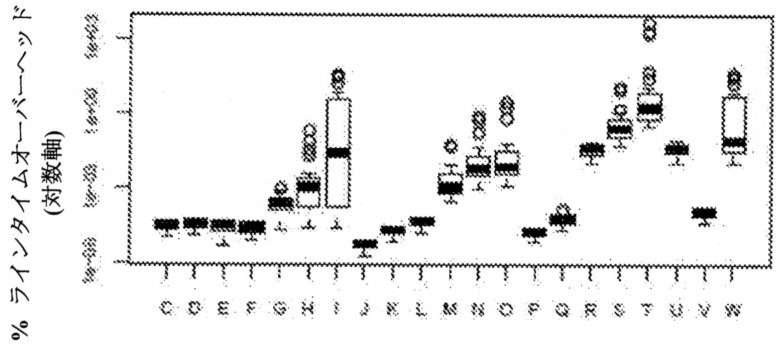
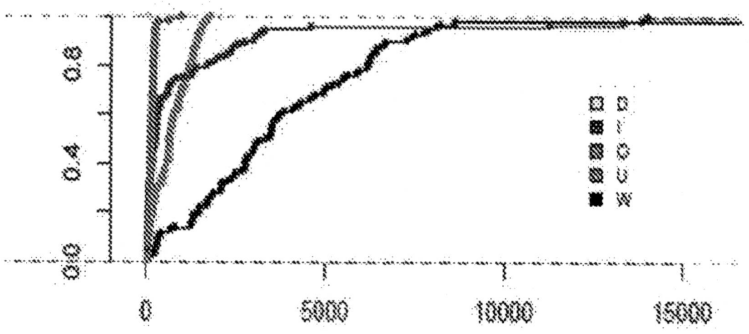


図6: ランタイムオーバーヘッドの分布(上のポリシーA)

【０７３７】
【表７】



[ワーキングセット]/1e6命令

図7: gccに対するワーキングセットサイズのCDF

【０７３８】

7．参考文献
【０７３９】
8．シンボリックルール
8.1 プリミティブ型
nop,ubranch:(-,insn,-,-,-)→(-,-) (1)
ar2s1d:(-,insn,other,other,-)→(-,other) (2)
ar2s1d:(-,insn,addr,other,-)→(-,addr) (3)
ar2s1d:(-,insn,other,addr,-)→(-,addr) (4)
ar2s1d:(-,insn,addr,addr,-)→(-,other) (5)
ar1s1d:(-,insn,other,-,-)→(-,other) (6)
ar1s1d:(-,insn,addr,-,-)→(-,addr) (7)
ar1d,flags:(-,insn,-,-,-)→(-,other) (8)
cbranch:(-,insn,other,other,-)→(-,-) (9)
ijump,return:(-,insn,addr,-,-)→(-,-) (10)
dcall,icall:(-,insn,addr,-,-)→(-,addr) (11)
load:(-,insn,addr,-,t)→(-,t)if t≠insn (12)
store:(-,insn,t,addr,-)→(-,t)if t≠insn (13)
move:(-,insn,other,-,-)→(-,other) (14)
move:(-,insn,addr,-,-)→(-,addr) (15)
リターンアドレスを確認するための代替的なルール
ijump:(-,insn,addr,-,-)→(-,-) (10)
return:(-,insn,retaddr,-,-)→(-,-) (10)
dcall, icall:(-,insn,addr,-,-)→(-,retaddr) (11)
【０７４０】
8.2 メモリ安全性
フルのメモリ安全性に対するN=2^{64-k}を用いたN-色付け。色をcと書き、ヒープへのポインタをタグ付けするためにそれらを使用する。色とは異なる特別なタグ⊥を仮定し、これは、ヒープへのポインタではないすべてのデータをタグ付けするために使用される。レジスタのタグは色または⊥(tと書かれる)である。メモリのタグは、色と色か⊥のいずれかとのペア((c1,t2)と書かれる)またはF(割り振られていない)である。ヒープは最初はすべてFとタグ付けされる。最後に、命令のタグが集合{tmalloc,tmallocinit,tfreeinit,tsomething else}から導かれる。
nop,cbranch,ubranch,ijump,return:(-,-,-,-,-)→(-,-) (1)
ar2s1d:(-,-,⊥,⊥,-)→(-,⊥) (2)
ar2s1d:(-,-,c,⊥,-)→(-,c) (3)
ar2s1d:(-,-,⊥,c,-)→(-,c) (4)
ar2s1d:(-,-,c,c,-)→(-,⊥) (5)
ar1s1d:(-,-,t,-,-)→(-,t) (6)
ar1d,dcall,icall,flags:(-,-,-,-,-)→(-,⊥) (7)
load:(-,-,c1,-,(c2,t2))→(-,t2)if c1=c2 (8)
store:(-,ci,t1,c2,(c3,t3))
→(-,(c3,t1))if c2=c3∧ci/∈{tmallocinit,tfreeinit} (9)
store:(-,tmallocinit,t1,c2,F)→(-,(c2,t1)) (10)
store:(-,tfreeinit,t1,c2,(c3,t4))→(-,F) (11)
【０７４１】
【数９６】

$$\text{move: } (-, \text{tmalloc}, t, -, -) \xrightarrow{1} (-, \text{tnewtag}) \qquad (12)$$

【０７４２】
【数９７】

$$\text{move: } (-,\ \overline{tmalloc}\ ,\ t,\ -,\ -) \rightarrow (-,\ t) \qquad (13)$$

【０７４３】

```
primitive_malloc=malloc;
malloc(int size){
void *p=primitive_malloc(size);//元のポインタ
void *tp;//タグ付けされたポインタ
void *tmp;//個々のワードへのタグ付けされたポインタ
asm:malloc move r1 =p, r2=tp//未使用のタグの割振り
tmp=tp;
for(int i=0;i〈size;i++){
//新しい領域asm mallocinitの中のワードに領域タグを設定する
store r1 =0,r2=tmp
tmp++;
}
return(tp);
}
primitive_free=free;
free(void*p){
size=size(p);//ポインタ領域のサイズ
void*tmp=base(p);//ポインタ領域のベース
for(int i=0;i〈size;i++){
//解放された領域の中のワードに領域タグを設定する
asm freeinit store r1=0,r2=tmp
tmp++;
}
return;
}
```

【０７４４】

8.3 CFI

8.3.1 CFI-1ID[2]

集合φおよび{f}と書かれる2つのタグを使用する。タグ{f}は、すべての間接的な制御フロー、ならびにそれらのすべての潜在的な宛先をタグ付けするために使用される。タグφは他のすべてのもののために使用される。

return,ijump,icall:(-,{f},-,-,-)→({f},-) (1)

【０７４５】
【数９８】

$$\overline{return\ ,\ ijump, icall}:$$
$$(pc,\ ci,\ -,\ -,\ -) \rightarrow (\emptyset,\ -) \text{ if } pc \subseteq ci \qquad (2)$$

【０７４６】

8.3.2 CFI-2ID[2]
このポリシーでは、rはリターンおよびそれらの潜在的なターゲットをマークするために使用され、cは間接的な呼出しおよびジャンプならびにそれらの潜在的なターゲットのために使用される。これらの2つのケースは重複することがあるので、集合:φ、{r}、{c}、および{r,c}と書かれる4つのタグを使用している。
return:(pc,ci,-,-,-)→({r},-)if r∈ci,pc⊆ci (1)
ijump,icall:(pc,ci,-,-,-)→({c},-)if c∈ci,pc⊆ci (2)
【０７４７】
【数９９】

$$\overline{return, ijump, icall}:$$

$$(pc, ci, -, -, -) \rightarrow (\varnothing, -) \text{ if } pc \subseteq ci \qquad (3)$$

【０７４８】
8.3.3 CCFIR[51]
rはreturn-idであり、cはcall-idであり、pはreturn-into-privileged-code-idである。
集合:φ、{r}、{p}、{c}、{r,c}、および{p,c}と書かれる6つのタグを仮定する。
return:(pc,ci,-,-,-)→({r},-)if r∈ci,pc⊆ci (1)
return:(pc,ci,-,-,-)→({p},-)if p∈ci,pc⊆ci (2)
ijump,icall:(pc,ci,-,-,-)→({c},-)if c∈ci,pc⊆ci (3)
【０７４９】
【数１００】

$$\overline{return, ijump, icall}:$$

$$(pc, ci, -, -, -) \rightarrow (\varnothing, -) \text{ if } pc \subseteq ci \qquad (4)$$

【０７５０】
8.3.4 CFI-ROP
リターンIDおよび潜在的な宛先IDのペアを含む、許容される制御フローグラフXを仮定する。以下ではIDをciまたはpcと書く。タグは有効なIDまたは⊥のいずれかである。
return: (⊥,ci,-,-,-)→(ci,-) (1')
【０７５１】
【数１０１】

$$\overline{return}: \quad (pc, ci, -, -, -) \rightarrow (\perp, -) \text{ if } (pc, ci) \in \chi \qquad (2')$$

【０７５２】
【数１０２】

$$\overline{return}: \quad (\perp, -, -, -, -) \rightarrow (\perp, -) \qquad (3')$$

【０７５３】
return: (pc,ci,-,-,-)→(ci,-)if(pc,ci)∈X (4')
【０７５４】
8.3.5 CFI-JOP
許容される制御フローグラフXを仮定すると、
ijump,icall: (⊥,ci,-,-,-)→(ci,-) (1)
ijump,icall: (pc,ci,-,-,-)→(ci,-)if(pc,ci)∈X (2)
【０７５５】
【数１０３】

$$\overline{ijump, icall}:$$
$$(\perp, -, -, -, -) \rightarrow (\perp, -) \qquad (3)$$

【０７５６】
【数１０４】

$$\overline{ijump, icall}:$$
$$(pc, ci, -, -, -) \rightarrow (\perp, -) \text{ if } (pc, ci) \in \chi \qquad (4)$$

【０７５７】
8.3.6 Complete-CFI
許容される制御フローグラフXを仮定する。
return,ijump,icall: (⊥,ci,-,-,-)→(ci,-) (1)
return,ijump,icall: (pc,ci,-,-,-)→(ci,-)if(pc,ci)∈X (2)
【０７５８】
【数１０５】

$$\overline{return, ijump, icall}:$$
$$(\perp, -, -, -, -) \rightarrow (\perp, -) \qquad (3)$$

【０７５９】
【数１０６】

$$\overline{return, ijump, icall}:$$
$$(pc, ci, -, -, -) \rightarrow (\perp, -) \text{ if } (pc, ci) \in \chi \qquad (4)$$

【０７６０】

8.4 テイント追跡
nop,cbranch,ubranch,ijump,return:(-,-,-,-,-)→(-,-) (1)
ar2s1d:(-,ci,op1,op2,-)→(-,ci∪op1∪op2) (2)
ar1s1d:(-,ci,op1,-,-)→(-,ci∪op1) (3)
ar1d,dcall,icall,flags:(-,ci,-,-,-)→(-,ci) (4)
load:(-,ci,op1,-,mr)→(-,ci∪op1∪mr) (5)
store:(-,ci,op1,op2,-)→(-,ci∪op1∪op2) (6)
【０７６１】
【数１０７】

$$\text{move} : (-, ttaint, -, -, -) \xrightarrow{1} (-, tnewtag) \qquad (7)$$

【０７６２】
move:(-,ci≠ttaint,op1,-,-)→(-,ci∪op1) (8)
【０７６３】
8.5 サブワード操作
我々の実験で使用した上のルールは、サブワード操作を考慮していない。サブワード操作を適切にサポートするには、loadオペグループおよびstoreオペグループを、ワード操作のための2つのオペグループ(wloadおよびwstore)とバイト操作のための2つのオペグループ(bloadおよびbstore)に分解する必要がある。
【０７６４】
ロードまたは記憶について明確に述べるすべてのポリシーについてのルールが、変化する必要があるであろう(単純型、メモリ安全性、およびテイント追跡)。ここにあるのは、単純型ポリシー(のno retaddr変形)がどのように変化するかである(wオペグループは以前のルールに対応する)。
wload: (-,insn,addr,-,other)→(-,other) (1)
wload: (-,insn,addr,-,addr)→(-,addr) (2)
wstore: (-,insn,other,addr,-)→(-,other) (3)
wstore: (-,insn,addr,addr,-)→(-,addr) (4)
bload: (-,insn,addr,-,other)→(-,other) (5)
bload: (-,insn,addr,-,addr)→(-,other) (6)
bstore: (-,insn,other,addr,-)→(-,other) (7)
bstore: (-,insn,addr,addr,-)→(-,other) (8)
【０７６５】
ここにあるのは、メモリ安全性のためのbルールである。
【０７６６】
【数１０８】

$$\text{bload:} \quad (-, -, c1, -, (c2, c_3^{\perp})) \rightarrow (-, \perp) \text{ if } c1 = c2 \qquad (1)$$

【０７６７】
【数１０９】

$$\text{bstore:} \quad (-, ci, c_1^{\perp}, c2, (c3, c_4^{\perp})) \qquad (2)$$

【０７６８】
ここにあるのは、テイント追跡のためのbstoreルールである。
bstore: (-,ci,op1,op2,mr)→(-,ci∪op1∪op2∪mr) (1)
【符号の説明】
【０７６９】
10 PUMP
12 RISCプロセッサ
14 フェッチ段階
16 復号段階
18 実行段階
20 メモリ段階
22 ライトバック段階
24 Alpha ISA
26 命令トレース
28 PUMPシミュレータ
30 アドレストレース
31 アドレス追跡シミュレータ
104 レベル1
106 レベル2
108 レベル3
110 レベル4
122 単一のノード
132 要素
152 レベル1キャッシュ
154 レベル2キャッシュ
156 レベル3キャッシュ
158 レベル4キャッシュ
142 要素
201 要素
202 タグ
204 命令
205 命令ワード
221 ポインタ
222 メモリ位置
223 要素
224 メモリ位置
231 fooルーチン
232 ソース位置
233 ルーチンbar
234 ターゲット位置
242 タグ
243 要素
245 フィールド
246 要素
247 リスト構造
248 要素
251 要素
252 要素
254 要素
256 要素

262 タグ
265 ワード
272 ルートノード
274 ノード
276 ノード
278 ノード
281 ルートノード
266 構造
402 要素
404 要素
421 オペコード
422 オペコードマッピングテーブル
424 出力
431 タグ
432 物理ページ
434 アドレス空間
436 アドレス空間
502 ルーチンfoo
504 ルーチンbar
506 ルーチンbaz
522 スタックフレーム
524 スタックフレーム
526 スタックフレーム
532 領域
533 タグ
534 領域
536 領域
538 領域
540 要素
562 ルーチンmain
563 ルーチンFirst
564 ルーチンSecond
572 要素
581 要素
604 学習されたルールの第1の集合
606 ルール検証処理
608 検証されたルールの第2の集合
701 他のタグ付けされていないメモリソース
702 RISC-V CPU
704 PUMP
706 L1データキャッシュ
708 L1命令キャッシュ
710 インターコネクトファブリック
712a ブートROM
712b DRAM制御
712c 外部DRAM
714a 追加タグ
714b 検証ドロップタグ
715 インターコネクトファブリック
716 タグ付けされていないメモリ

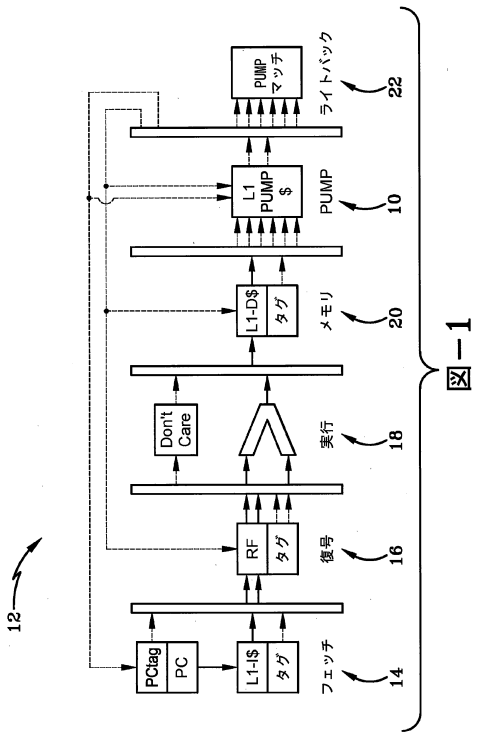
732 RISC-V CPU
732a データキャッシュ
732b 命令キャッシュ
742 インターン
744 エクスターン
802 PUMP
804 PUMP入力
805a R tag
805b PCnew tag
805c 予測されるMR tag
805d R tag
805e PCnew tag
808 段階6
822 MR tag予測PUMP
828 要素
829 要素
850 要素
852 要素
900 表
901 行
902 列
904 列
906 列
908 列
910 表
911 行
912 列
914 列
916 列
1001 要素
1002 サブシステム
1002a タグ
1002b データ
1003 PUMP
1004 メタデータ処理サブシステム
1006a メモリ
1006b I-ストア
1006c レジスタファイル
1006d ALU
1007 PUMP入力
1008a メモリ
1008b I-ストア
1008c レジスタファイル
1008d ALU
1008e PC
1012 PUMP入力
1014 PUMP出力
1016 PUMP入力
1018 PUMP出力
1022 入力

1024 出力
1026 入力
1028 出力
1031 PUMP制御
1032 マスキング
1034 ハッシュ
1036 ルールキャッシュルックアップ
1038 出力タグ選択
1041a PC new tag
1041b PC tag
1042 セレクタ
1043 マルチプレクサ
1044 セレクタ
1045a R tag
1045b 他の入力
1047 R tag
1049 要素
1051 Op1data
1052 要素
1053 Rdata
1054 出力
1055 出力
1056 要素
1061 Mtag
1062 タグ値
1063a Iキャッシュ
1063b Dキャッシュ
1064 PUMPハッシュ
1066 PUMPルールキャッシュ
1068 要素
1102a 要素
1102b メモリブロック
1104a 要素
1104b メモリブロック
1106a 要素
1106b メモリブロック
1111 要素
1112 要素
1113 要素
1114 要素
1116 要素
1212 メタデータ処理
1214 出力
1221a 論理的な結果
1222 PUMP
1225 マルチプレクサ
1228 ポインタ
1302 PUMPルールキャッシュ
1304 ハードウェアルールキャッシュミスハンドラ
1312 入力

1314 HWルールキャッシュミスハンドラ
1316 出力
1318 合成結果
1402 命令
1404 命令
1502 I/O PUMP
1504a イーサネット（登録商標）DMAデバイスA
1504b イーサネット（登録商標）DMAデバイスB
1504c UARTまたはシリアルデバイス
1524 アドレス列
1526 名前列
1528 説明列
1531 無効な要求
1532 信頼されるファブリック
1534a I/O PUMP
1534b デバイスタグ
1534c トラストブリッジ
1535 要素
1536 信頼されないファブリック
1537 CSR
1538 要素
1541a 要求受信段階
1541b メモリフェッチ段階
1541c 検証段階
1541d ライトバック段階
1542 要素
1543 要素
1544 要素
1545 IOPUMP
1546 要素
1548 要素
1621 boottag
1622 OS特別命令タグ
1624 異なるコード部分
1625 ユーザモード
1626 mallocタグ生成源アプリケーションタグ
1627 要素
1628 要素
1629 アプリケーション色シーケンス
1630 無限のシーケンス
1631 無限のシーケンス
1701 要素
1702 要素
1704 要素
1706 集合

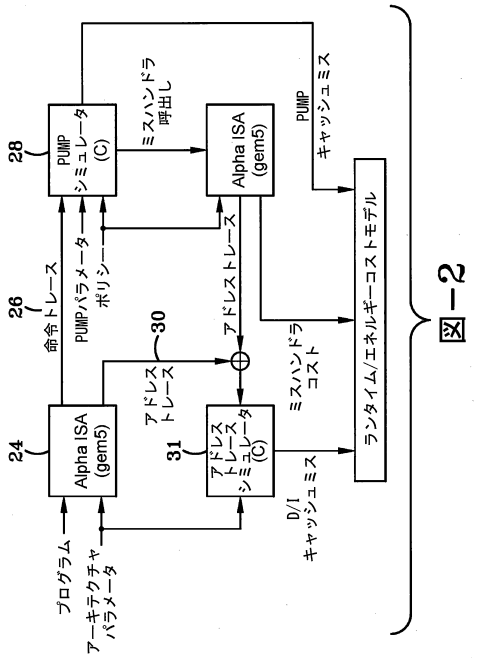
【図面】

【図１】



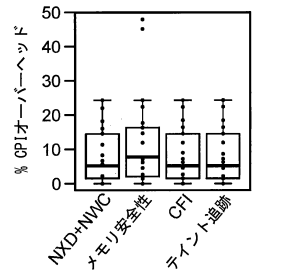
図－1

【図２】



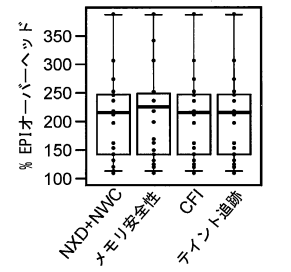
図－2

【図３Ａ】



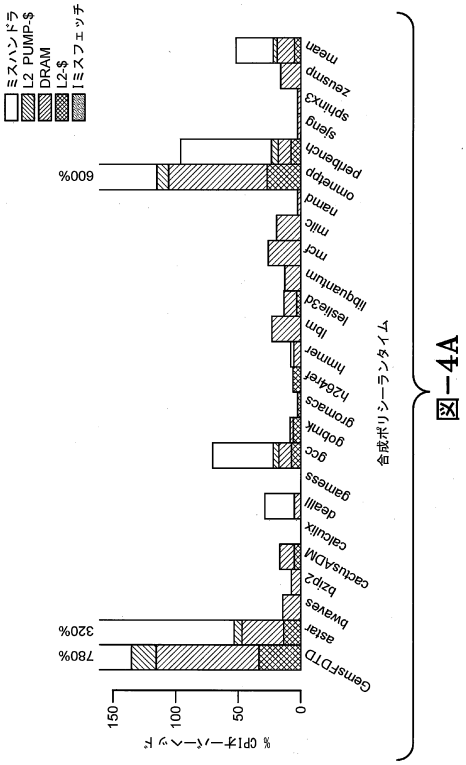
図－3A

【図３Ｂ】



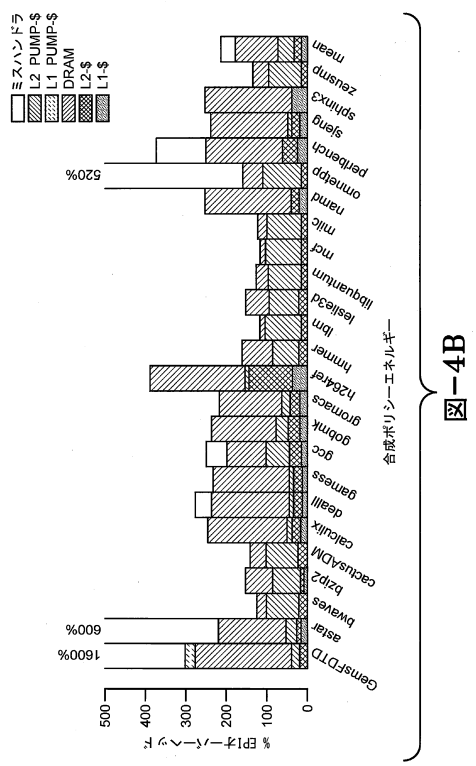
図－3B

【図４Ａ】



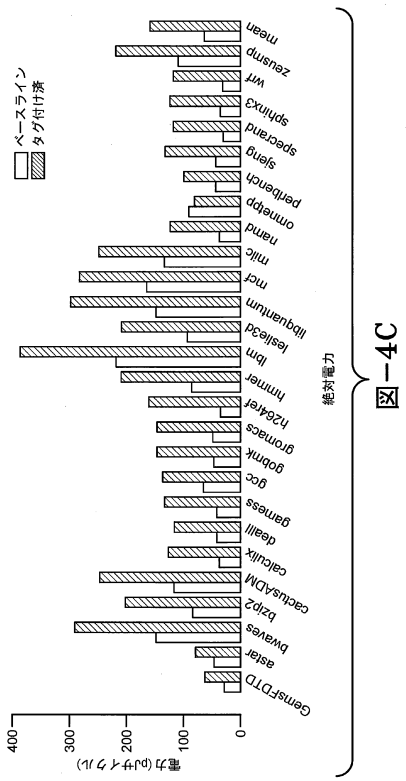
図−4A

【図４Ｂ】



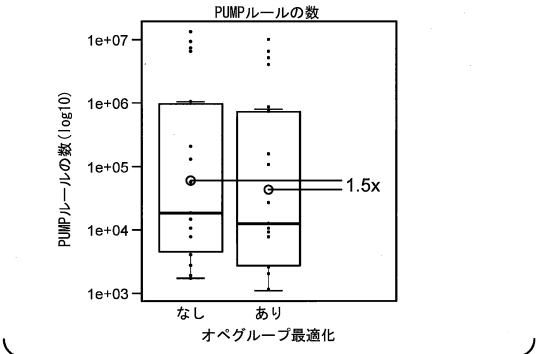
図−4B

【図４Ｃ】



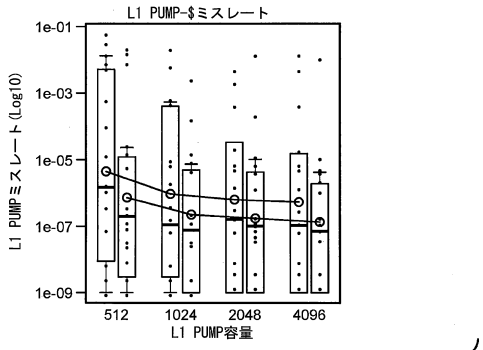
図−4C

【図５Ａ】



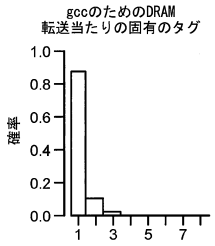
図−5A

【図５Ｂ】



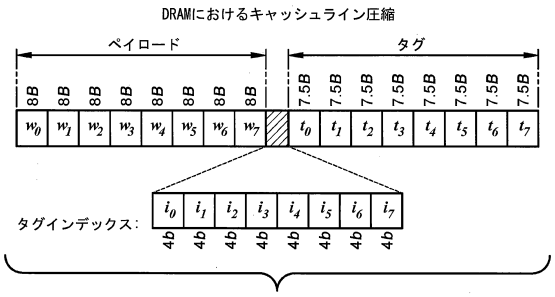
図−5B

【図６Ａ】



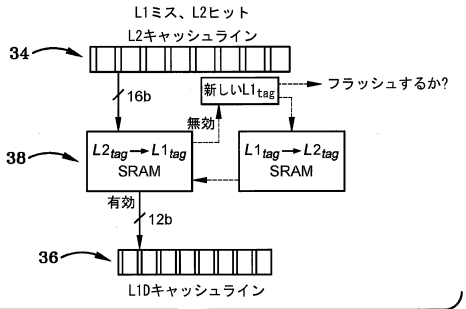
図−6A

【図６Ｂ】



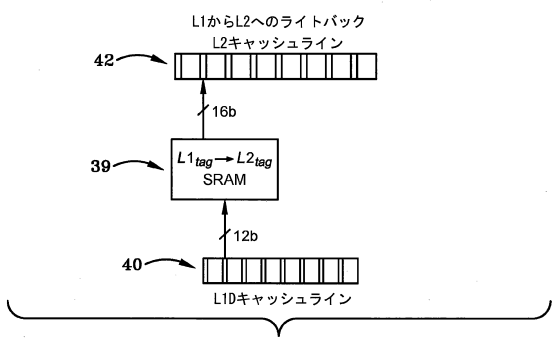
図−6B

【図７Ａ】



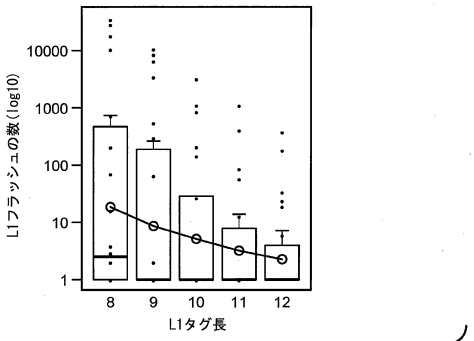
図−7A

【図７Ｂ】



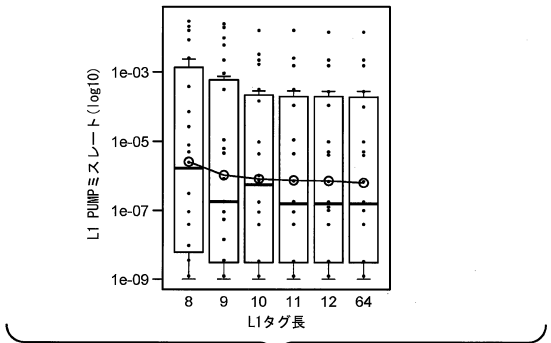
図−7B

【図８Ａ】



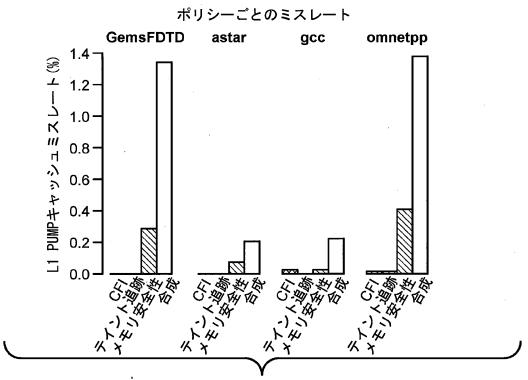
図−8A

【図８Ｂ】



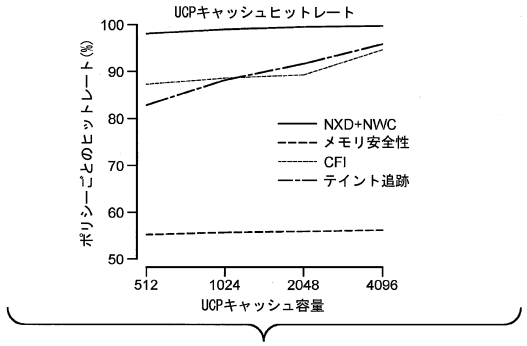
図−8B

【図９Ａ】



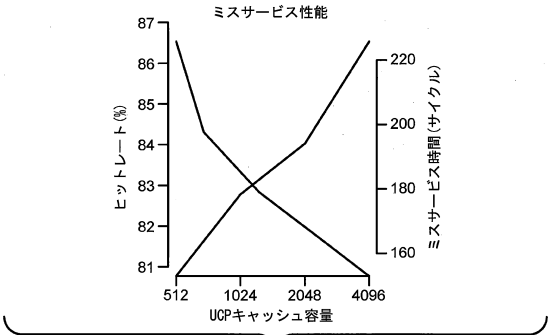
図−9A

【図９Ｂ】



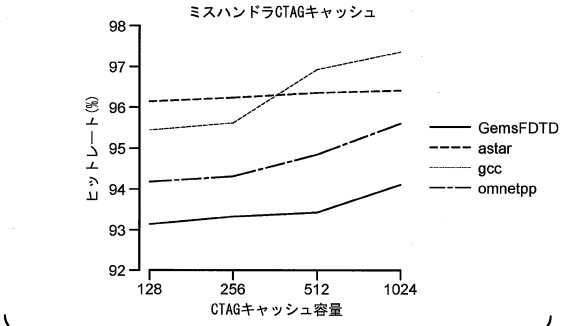
図－9B

【図９Ｃ】



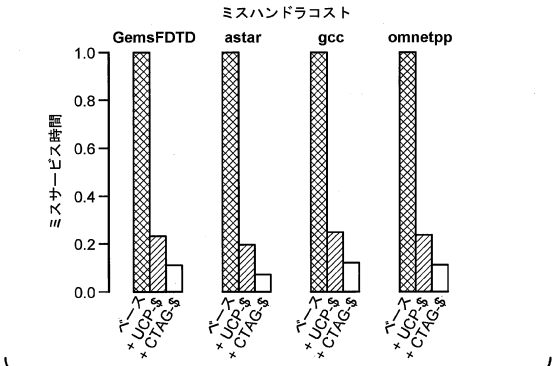
図－9C

【図９Ｄ】



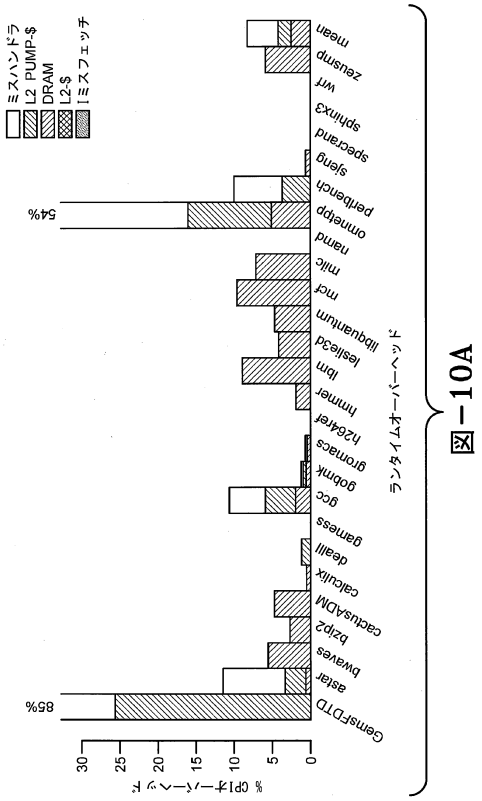
図－9D

【図９Ｅ】



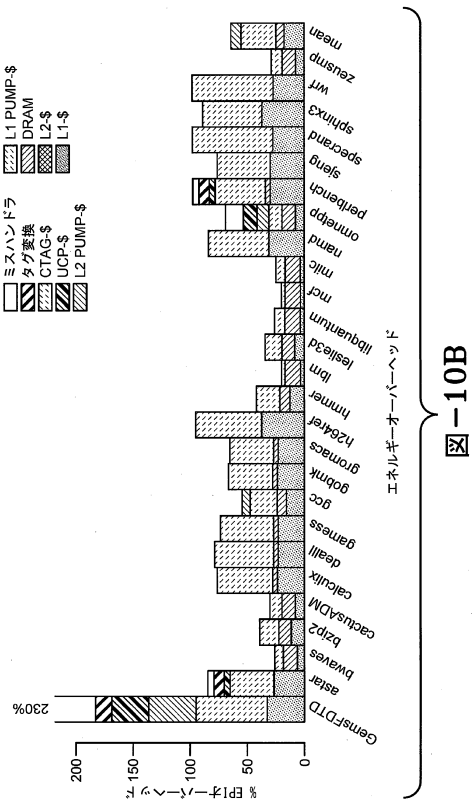
図－9E

【図１０Ａ】



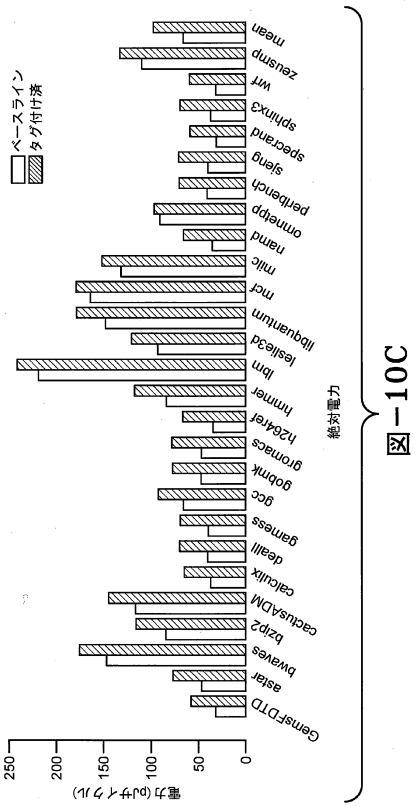
図－10A

【図１０Ｂ】



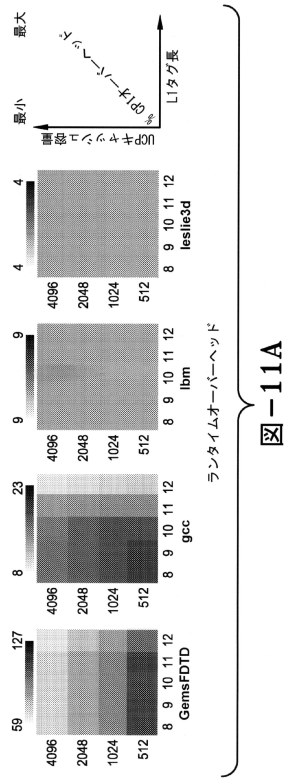
図－10B

【図１０Ｃ】



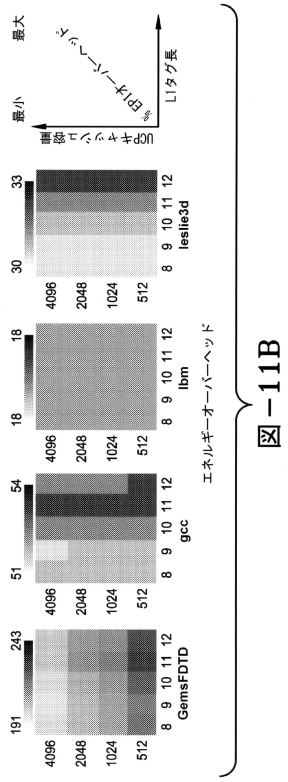
図－10C

【図１１Ａ】



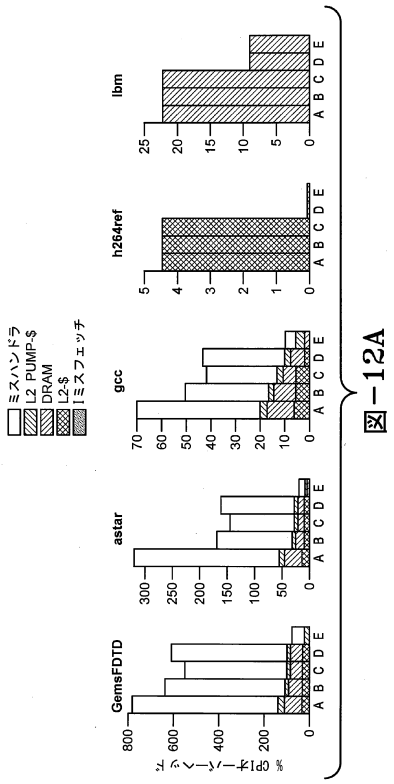
図－11A

【図１１Ｂ】



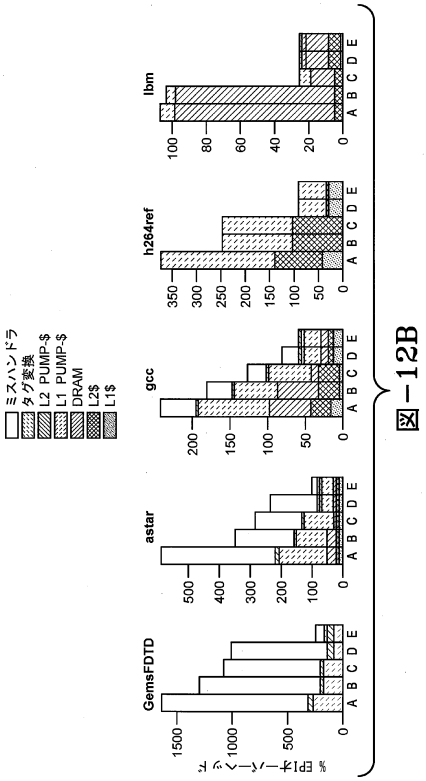
図－11B

【図１２Ａ】



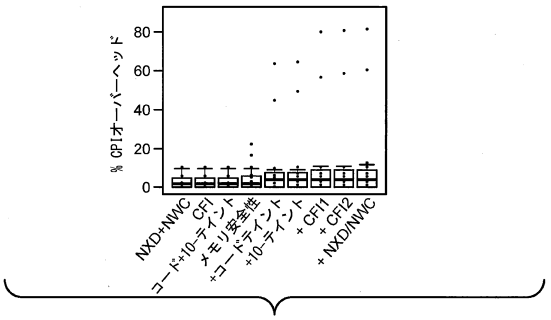
図－12A

【図１２Ｂ】



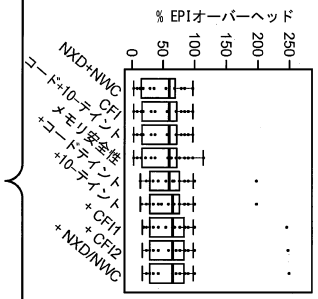
図－12B

【図１３Ａ】



図－13A

【図１３Ｂ】



図－13B

【図１４】

表1-調査されるポリシーの概要

ポリシー	脅威	メタデータ	最大の固有タグ	タグ確認(許容?)	タグ伝播ルール	参考文献
NXD+NWC	コード インジェクション	メモリ位置上の DATAまたはCODE	2	*CI*=CODE; *MR*=CODE 書込み時	なし	
メモリ安全性	ヒープ上での 空間的/時間的な メモリ安全性の侵害	ポインタの色; メモリ位置での領域の 色+ペイロードの色	(malloc数)^2	ポインタの色== 参照される領域の色	*R* ← mov/add/sub時 *R* ← ペイロード(*MR*)ロード時	[17]
CFI	制御フローの ハイジャック(JOP/ ROP/コード再使用)	各々の間接な制御 フローのソースおよび ターゲットの固有id	ソースの数+ ターゲットの数	(*PC*,*CI*) $\in$プログラムの 制御フローパス	*PC* ← 間接参照ジャンプ時 (呼出し、リターンを含む)	[4]
テイント 追跡	信頼されないコード、 IOからの整合性の 低いデータ	命令およびIOのソース テイントid;ワード上の 入力テイントidの集合	$2^{(コードの数+IO\ id)}$	ユーザにより 定義される確認	*R* ← *CI* ∪ *OP1* ∪ *OP2* ∪ *MR*	[47]
合成	上のすべて	4つの要素がある構造	上の積	上のすべて(N=4のアルゴリズム2)		

図－14

【図１５】

表2-タギング方式の分類

タグ ビット	伝播?	出力 許容?	出力 R(結果)	出力 PC	入力 PC	入力 CI	入力 OP1	入力 OP2	入力 MR	用途
2	X	ソフトウェアで定義される	X	X	X	X	X	X	✓	メモリ保護
ワード	X	限定的なプログラム可能性	X	X	X	X	X	X	✓	メモリ健全性、スタック、隔離
32	X	限定的なプログラム可能性	X	X	X	X	X	X	✓	偽造不可能なデータ、隔離
2	X	固定	固定	X	X	X	X	X	✓	細粒度の同期
1	✓	固定	X	X	X	X	✓	X	X	能力
2-8	✓	固定	固定	X	X	X	✓	✓	X	タイプ
128	✓	固定	コピー	X	X	X	✓	X	✓	メモリ安全性
0	✓	ソフトウェアで定義される		X	1つだけ伝播する					不変条件確認
1	✓	固定	固定	X	X	X	✓	✓	✓	テイント
4	✓	限定的なプログラム可能性		X	X	X	✓	✓	X	テイント、介入、フォルト隔離
10	✓	限定的なプログラム可能性	固定	X	X	X	✓	✓	✓	テイント、隔離
未指定	✓	ソフトウェアで定義される		X	X	X	✓	✓	✓	フレキシブルなテイント
32	✓	ソフトウェアで定義される		X	X	X	✓	✓	✓	プログラム可能、テイント、メモリ確認、参照カウント
0-64	✓	ソフトウェアで定義される		✓	✓	✓	✓	✓	✓	情報フロー、タイプ
無制限	✓	ソフトウェアで定義される		✓	✓	✓	✓	✓	✓	完全にプログラム可能、ポインタサイズのタグ

伝播=タグがデータとともに伝播する(✔)、またはメモリアドレスの特性である(×)
許容?=動作を許容する/許容しないための論理
R=ポリシーが結果のタグに影響する
PC=プログラムカウンタのプログラム可能なタグ
CI=現在の命令タグが確認または伝播されるか
OP1, OP2, MR=これらの入力のタグが確認または伝播されるか

図－15

【図１６】

表3-32nmノードにおけるベースラインプロセッサおよび簡単なPUMP拡張されたプロセッサに対するメモリリソース推定

ユニット	設計	編成	面積 (mm^2)	アクセスエネルギー 読取り/書込み(pJ)	静的電力 (pJ/サイクル)	レイテンシ (ps)	サイクル
レジスタファイル (整数および 浮動小数点数)	ベースライン	64b、2R1W、[32整数、32浮動小数点数]	0.002	0.3/0.5	0.08	264	1
	64bタグ拡張	128b、2R1W、[48整数、32浮動小数点数]	0.007	1.0/1.4	0.23	295	1
L1-$ (IおよびD)	ベースライン	64KB、4ウェイ、64B/行	0.236	17/11	14.4	880	1
	64bタグ拡張	64KB、4ウェイ、128B/行(実質的に32KB、64B/行)	0.244	19/14	14.5	880	1
L2-$ (統合)	ベースライン	512KB、8ウェイ、64B/行	1.207	393/481	0.111	4000	5
	64bタグ拡張	1MB、8ウェイ、128B/行(実質的に512KB、64B/行)	2.350	758/1223	0.214	4930	5
TLB (IおよびD)	いずれか	1KB、2ウェイセット アソシアティブ	0.04	3.6/4.5	2	800	1
DRAM	ベースライン	1GB、転送ごとに64Bの行にアクセス	-	15,000	-	-	100
	64bタグ拡張	1GB、128Bの行にアクセス(実質的に転送ごとに64Bの行)	-	31,000	-	-	130
L1 PUMP-$	64bタグ	完全に連想型の1024エントリー、328bの一致、128bの出力(使用されない; 参照のためのみに示される)	1.500	750/900	-	3000	4
		マルチハッシュ1024エントリー、328bの一致、128bの出力	0.683	51/62	32	500	1
L2 PUMP-$	64bタグ	マルチハッシュ4096エントリー、328bの一致、128bの出力	0.994	173/444	0.085	3300	4

ベースライン総面積 **1.485mm^2**
64bタグ付け済総面積 **4.318mm^2**(ベースラインより+190%)

図－16

【図１７】

表4-PUMPパラメータ範囲テーブル

ユニット	パラメータ	範囲	最終
L1 PUMP-$	容量	512-4096	1024
	タグビット	8-12	10
L2 PUMP-$	容量	1024-8192	4096
	タグビット	13-16	14
UCP-$	容量	512-4096	2048
CTAG-$	容量	128-1024	512

【図１８】

表5-32nmノードにおけるPUMP最適化されたプロセッサに対するメモリリソース推定

ユニット	設計	編成	面積 (mm^2)	アクセスエネルギー 読取り/書込み (pJ)	静的電力 (pJ/サイクル)	レイテンシ (ps)	サイクル
レジスタファイル	拡張10b	74b、2R1W、[49要数、32浮動小数点数]	0.005	0.4/0.5	0.13	360	1
L1キャッシュ	10bタグ	74KB、4ウェイ、74B/行(実質的に64KB、64B/行)	0.272	19/13	16.4	975	1
L2キャッシュ	14bタグ	592KB、8ウェイ、76B/行(実質的に512KB、64B/行)	1.247	343/640	0.133	4600	5
TLB	-	1KB、2ウェイセットアソシアティブ	0.040	3.6/4.5	2	800	1
DRAM	64bタグ	1GB、128Bの1行にアクセス(76B移動)	-	17,500	-	-	112
L1 PUMPキャッシュ	10b L1タグ	マルチハッシュ1024エントリー、58bの一致、20bの出力	0.095	15/43.2	2.22	520	1
L2 PUMPキャッシュ	14b L2タグ	マルチハッシュ4096エントリー、78bの一致、20bの出力	0.287	99.4/267	0.032	2800	3
フル→L2タグ	64b→14b	8Kエントリー、64bの一致、14bの出力、マルチハッシュ	0.432	166/436	0.052	3400	4
L2タグ→フル	14b→64b	16Kx64 SRAM	0.216	55.5/31.5	0.027	1700	2
L2タグ→L1タグ	14b→10b	16Kx11 SRAM	0.038	8.8/4.7	0.004	1420	2
L1タグ→L2タグ	10b→14b	4Kx14 SRAM	0.004	0.8/1	0.0004	780	1
UCPキャッシュ	64bタグ	マルチハッシュ2048エントリー、203bの一致、12bの出力	0.377	196/479	0.035	2730	3
CTAGキャッシュ	64bタグ	マルチハッシュ512エントリー	0.107	56/139	0.009	1700	2

総面積 **3.120mm²** (ベースラインより+110%)

図−18

【図１９】

アルゴリズム1　テイント追跡ミスハンドラ(フラグメント)

```
1: switch (op)
2: case  add, sub, or:
3:       PC_new ← PC
4:       R ← canonicalize (CI ∥ OP1 ∥ OP2)
5: (他の命令に対するcaseは省略される)
6: default: エラーハンドラにトラップする
```

図−19

【図２０】

アルゴリズム2　N-ポリシーミスハンドラ

```
1: for  i=1 to N do
2:       M_i ← {op, PC[i], CI[i], OP1[i], OP2[i], MR[i]}
3:       {pc_i , res_i} ← policy_i (M_i)
4: PC_new ← canonicalize ([pc_1 , pc_2 , ..., pc_N])
5: R ← canonicalize ([res_1 , res_2 , ..., res_N])
```

図−20

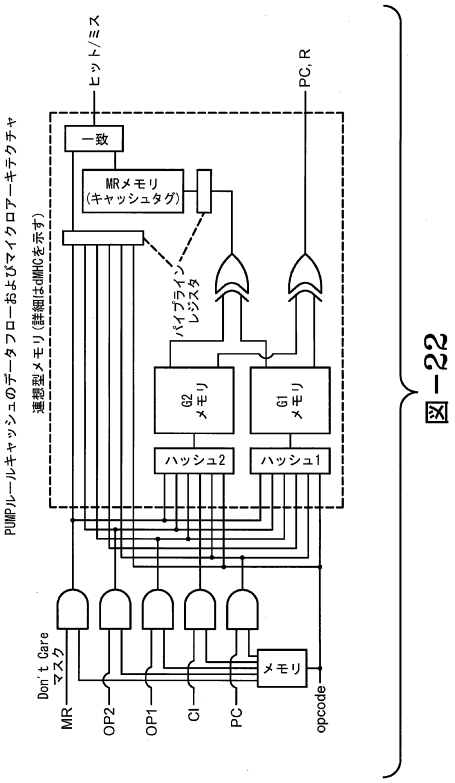
【図２１】

アルゴリズム3　HWサポートを伴うN-ポリシーミスハンドラ

```
1:  for  i=1 to N do
2:      M_i ← {op, PC[i], CI[i], OP1[i], OP2[i], MR[i]}
3:      {hit, pc_i , res_i} ← UCP-$ (i, M_i)
4:      if !hit then
5:          {pc_i , res_i} ← policy_i (M_i)
6:          {i, pc_i, res_i} をUCP-$に挿入する
7:  pc[1...N] ← {pc_1 , pc_2 , ..., pc_N}
8:  {hit, PC_new} ← CTAG-$ (pc[1...N])
9:  if !hit then
10:     PC_new ← canonicalize (pc[1...N])
11:     {pc[1...N], PC_new} をCTAG-$に挿入する
12: res[1...N] ← {res_1 , res_2 , ..., res_N}
13: {hit, R} ← CTAG-$ (res[1...N])
14: if !hit then
15:     R ← canonicalize (res[1...N])
16:     {res[1...N], R} をCTAG-$に挿入する
```

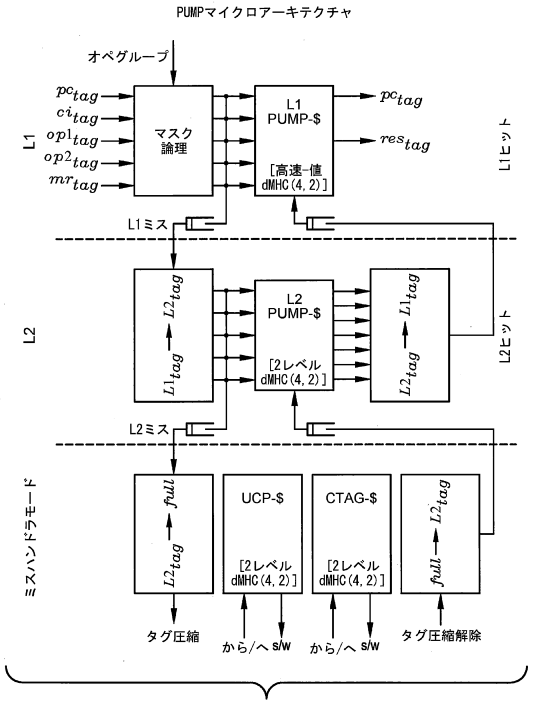
図−21

【図２２】



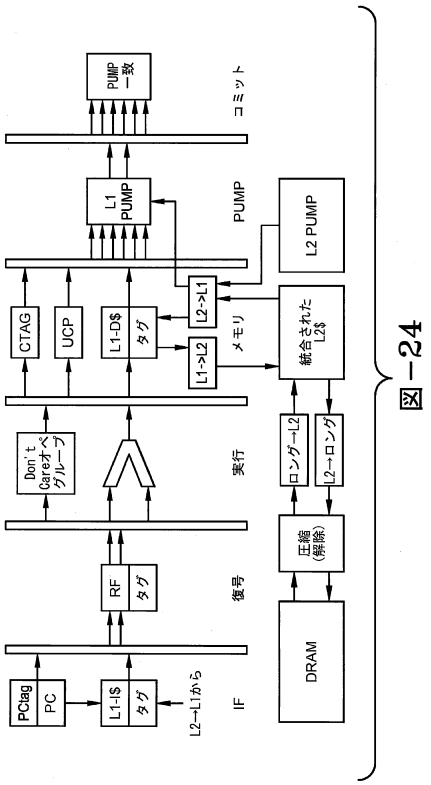
図−22

【図２３】



図−23

【図２４】

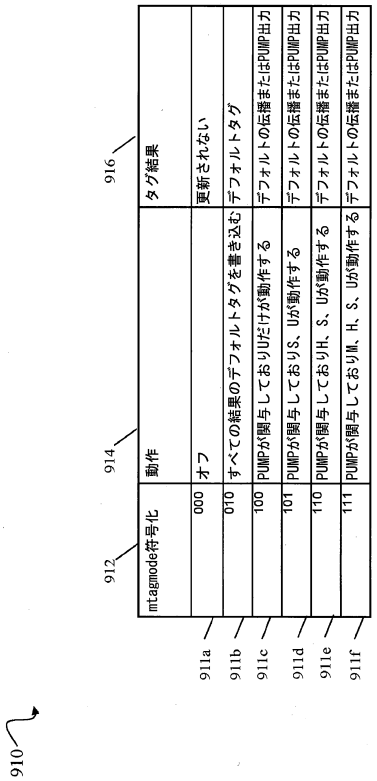


図−24

【図２５】

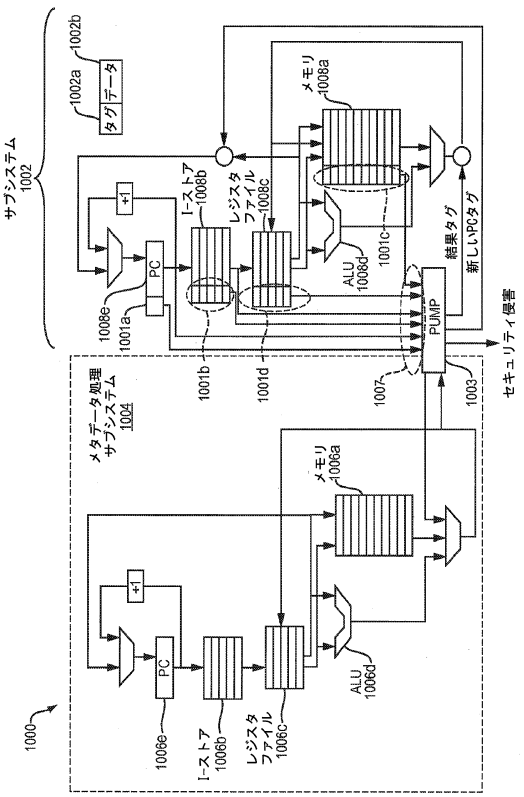
	アドレス	特権	名称	説明
901a	0x5C0	SRW	shootstag	OSのためのブートストラップタグ（読み取られるとクリアされる）
901b	0x5C1	SRW	spubuntrust	公開の信頼されていないタグの符号化
901c	0x5C2	SRW	sdefaulttag	デフォルトタグ
901d	0x5C8	SRW	sopgrpaddr	オペグループ/ケアテーブルへの書込みのためのアドレス（{opcode<3}:funct3）
901e	0x5C9	SRW	sopgrpvalue	オペグループ/ケアテーブルへの書込みのためのデータ このCSRへの書込みはテーブルへの挿入をトリガする
901f	0x5CC	SRW	spumpflush	このCSRへの書込みはPUMPフラッシュをトリガする
901g	0x5D0	SRW	sopgrp	ルールミスのためのオペグループ
901h	0x5D1	SRW	spctag	ルールミスのためのPC tag
901i	0x5D2	SRW	sctag	ルールミスのためのCI（現在の命令）tag
901j	0x5D3	SRW	srs1tag	ルールミスのためのRS1 tag
901k	0x5D4	SRW	srs2tag	ルールミスのためのRS2 tag
901l	0x5D5	SRW	srs3tag	ルールミスのためのRS3 tag
901m	0x5D6	SRW	smtag	ルールミスのためのメモリタグ
901n	0x5D7	SRW	sfunct12	命令からのfunct12
901o	0x5D8	SRW	ssubinstr	単一のタグが複数の命令に適用されるとき、 どの命令を実行しているか（PCの少数のビット） （命令長=32／タグ付けされたワード長=64であるとき、これは奇数／偶数である）
901p	0x5E0	SRW	snewpctag	命令が完了するとPCに付けるべきタグ
901q	0x5E1	SRW	srtag	命令からRDレジスタまたはメモリ結果に付けるべきタグ このCSRへの書込みはPUMPへのルールの書込みをトリガする
901r	0x7C0	MRW	mtagmode	PUMP動作のためのモード

【図２６】

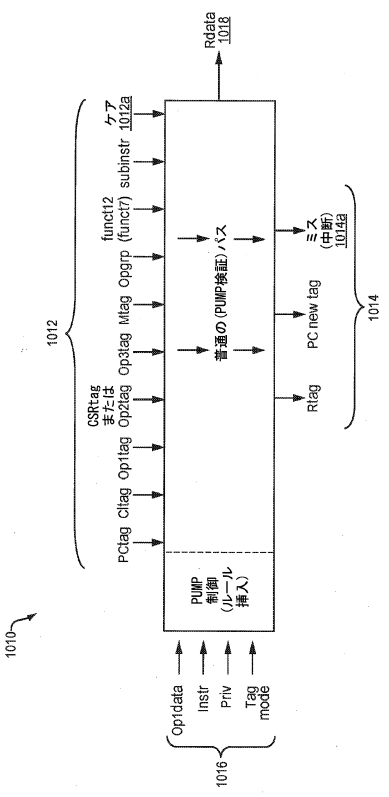


	mtagmode符号化	動作	タグ結果
911a	000	オフ	更新されない
911b	010	すべての結果のデフォルトタグを書き込む	デフォルトタグ
911c	100	PUMPが関与しておりUだけが動作する	デフォルトの伝播またはPUMP出力
911d	101	PUMPが関与しておりS、Uが動作する	デフォルトの伝播またはPUMP出力
911e	110	PUMPが関与しておりH、S、Uが動作する	デフォルトの伝播またはPUMP出力
911f	111	PUMPが関与しておりM、H、S、Uが動作する	デフォルトの伝播またはPUMP出力

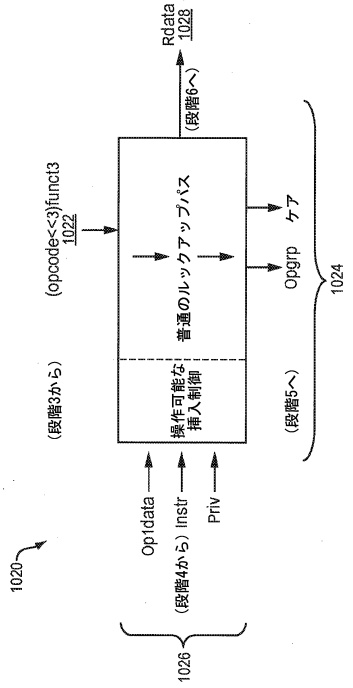
【図２７】



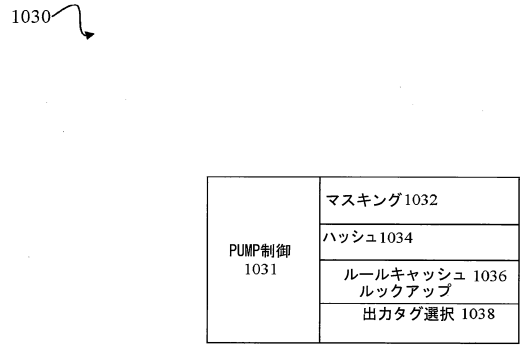
【図２８】



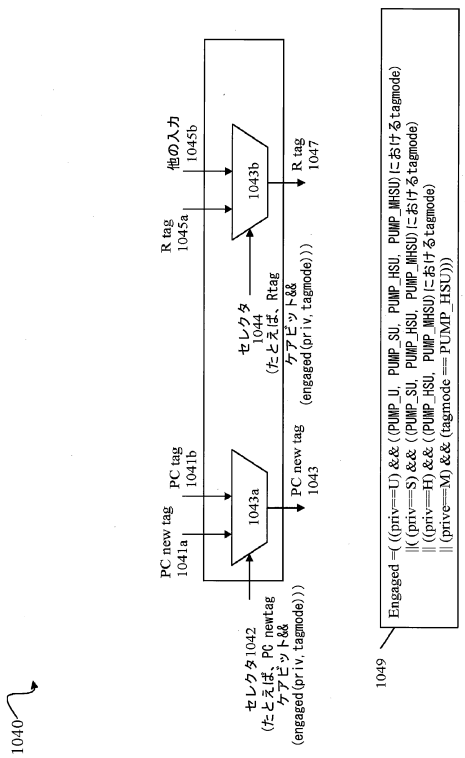
【図２９】



【図３０】



【図３１】



【図３２】

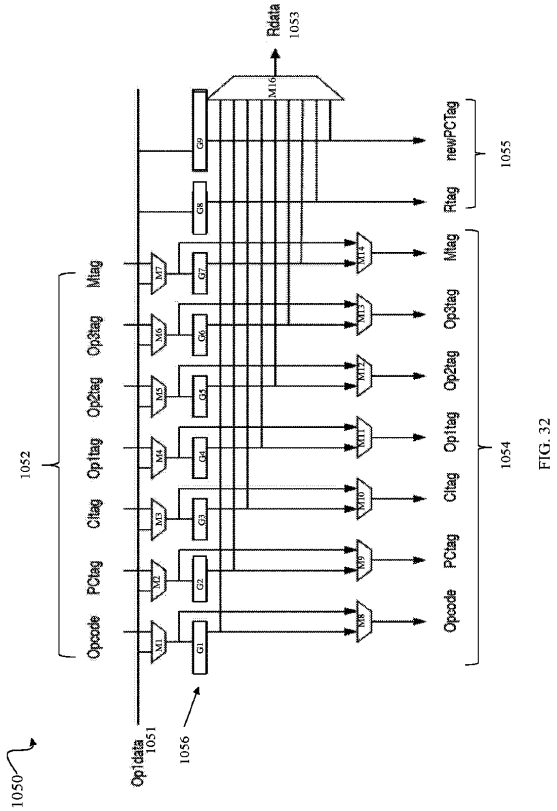
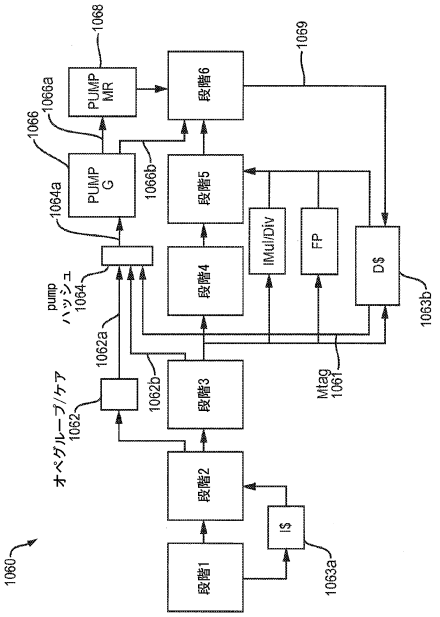
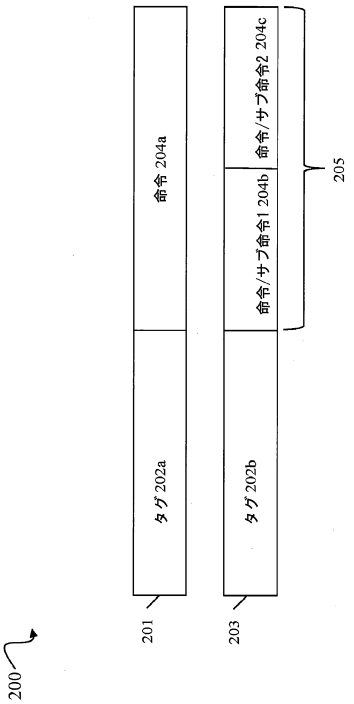


FIG. 32

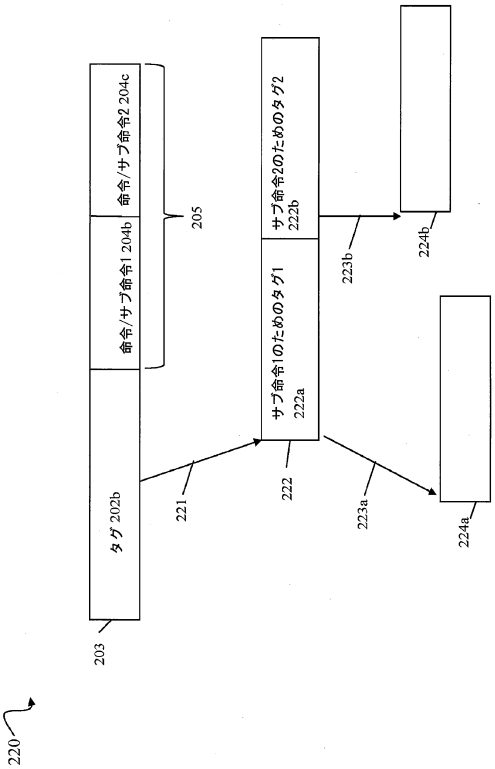
【図３３】



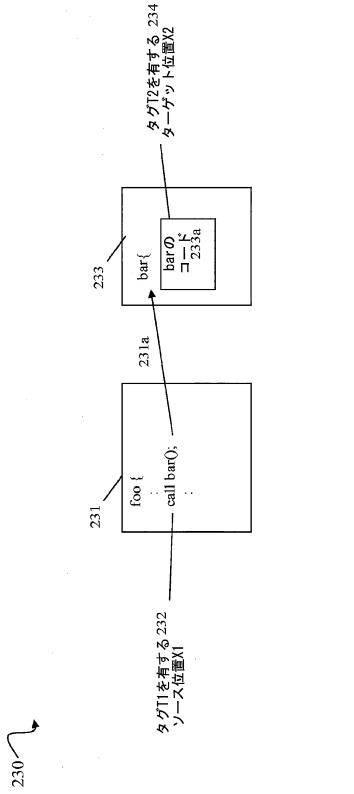
【図３４】



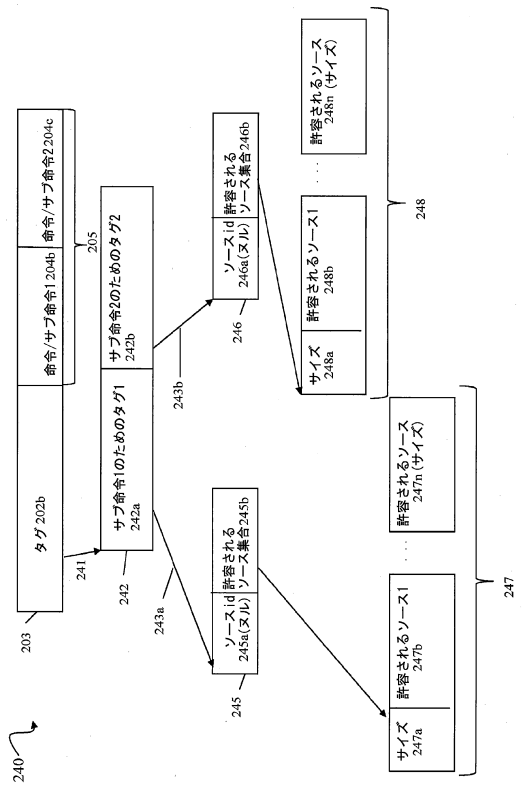
【図３５】



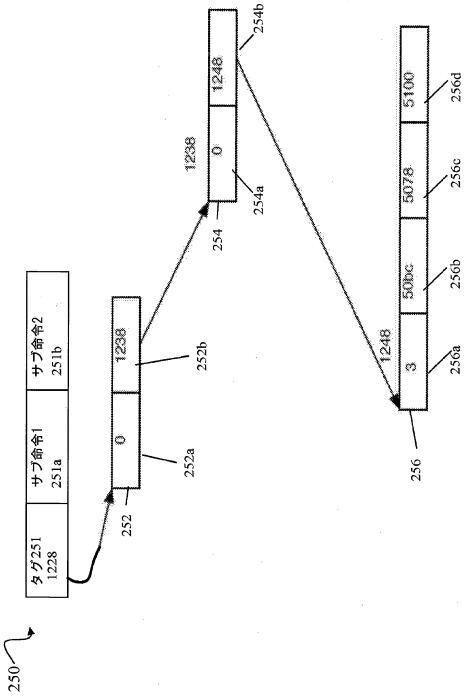
【図３６】



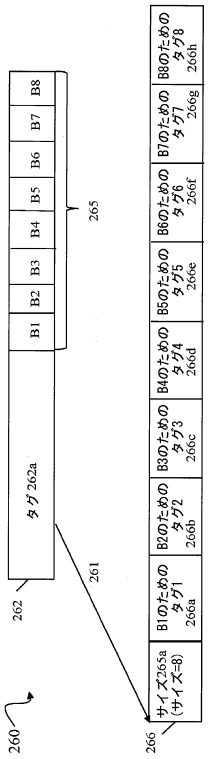
【図３７】



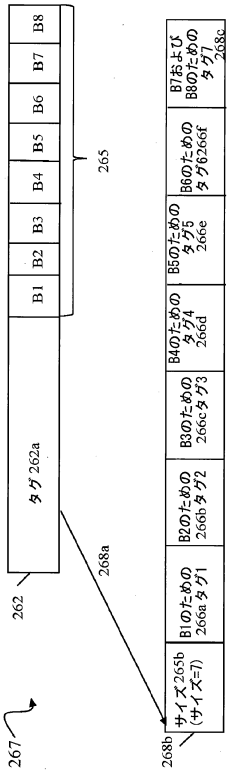
【図３８】



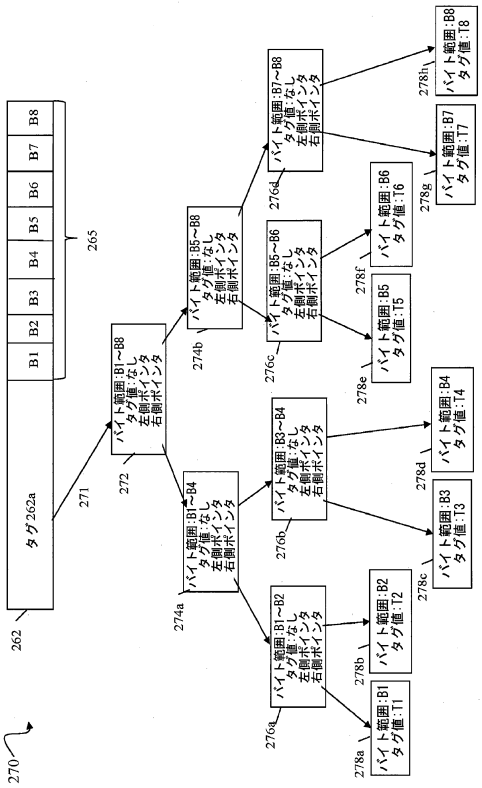
【図３９】



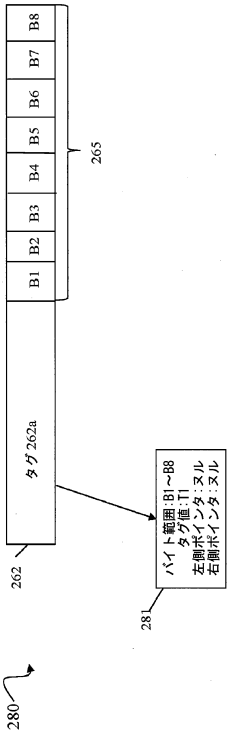
【図４０】



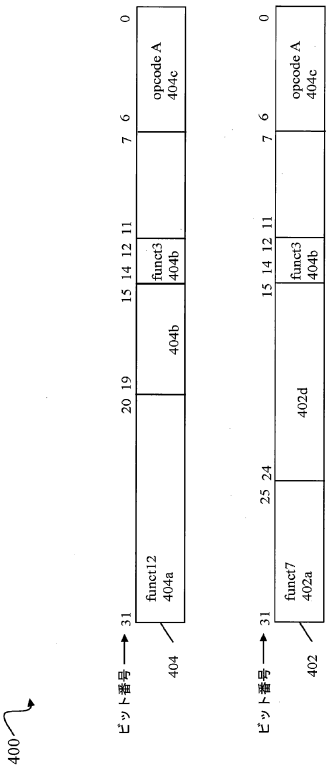
【図４１】



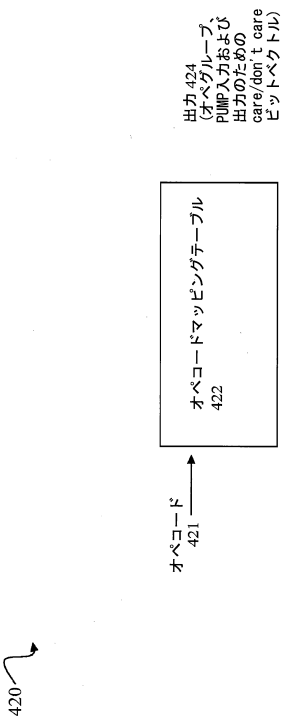
【図４２】



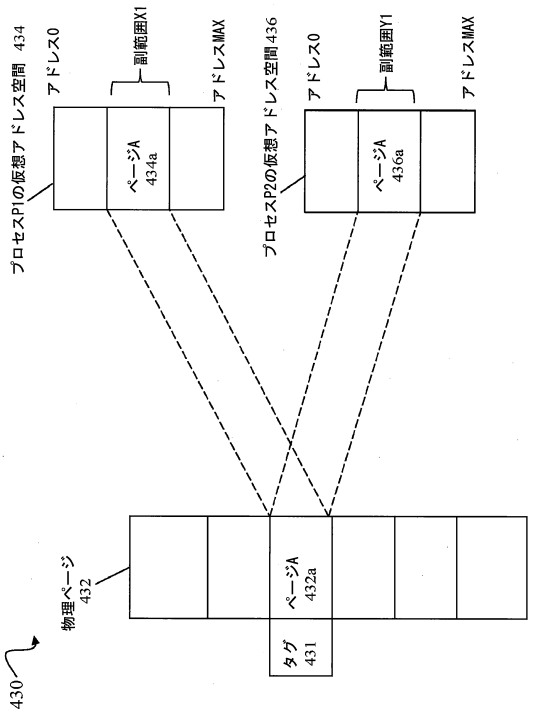
【図４３】



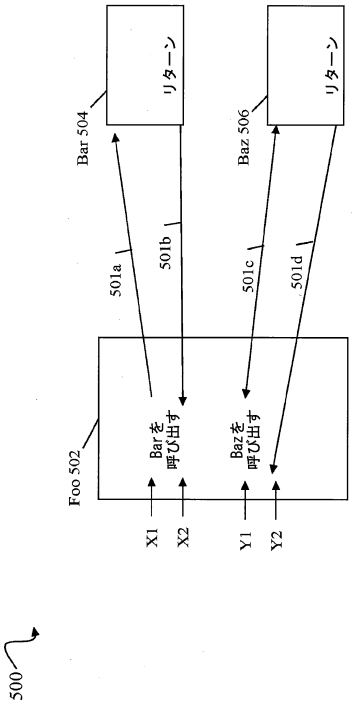
【図４４】



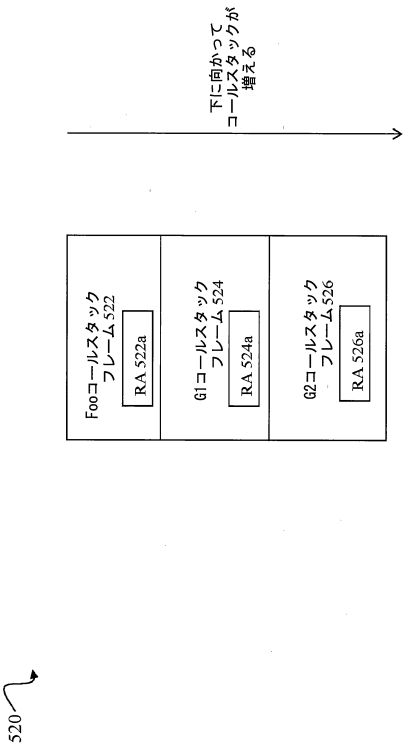
【図４５】



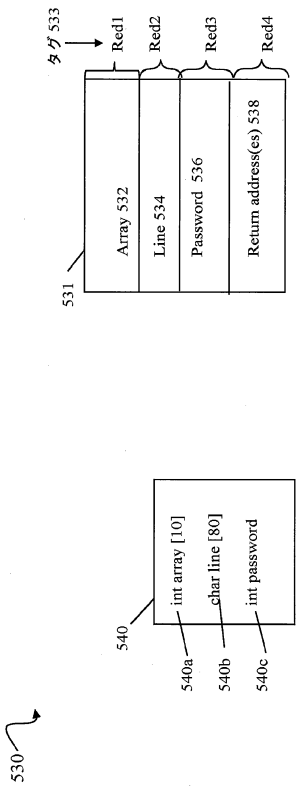
【図４６】



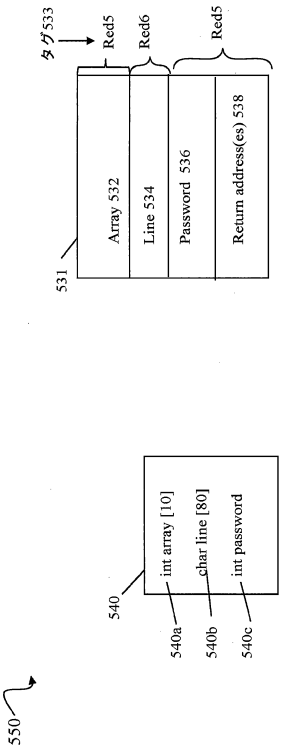
【図４７】



【図４８】



【図４９】



【図５０】

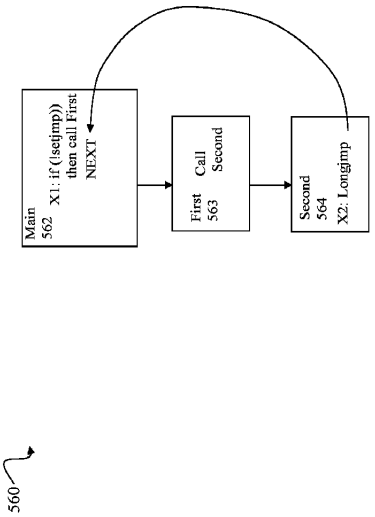


FIG. 50

【図５１】

570

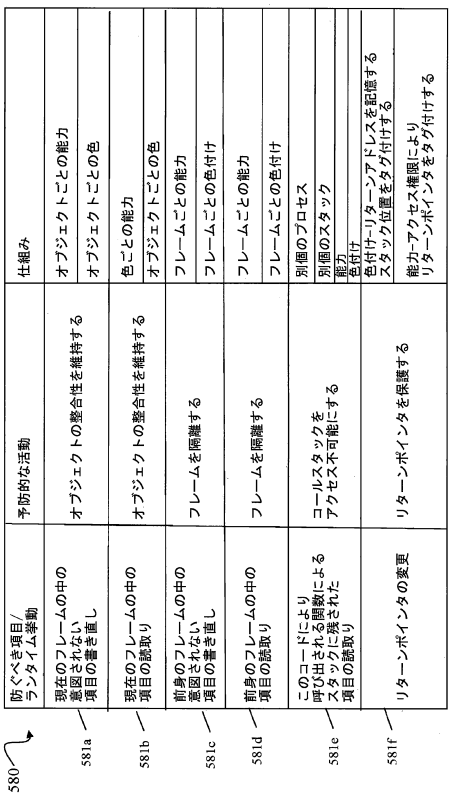
	防ぐべき項目/ランタイム挙動	予防的な活動	仕組み
572a	決してリターンしない	タイムアウト	別々のスレッド
			時間/命令が限られる呼出し
572b	リソースの消耗	被呼出者のリソースを制限する	別個のスレッド
			リソースが限られた呼出し
572c	予期されない権限の行使	権限を最低の特権に限定する	被呼出者に対して見える権限および制御の能力としてPdをタグ付けする
			アクセス可能なファイルシステムの部分を制限し、行われ得るシステム呼出しを制限する
572d	このコードによって呼び出される関数によるレジスタに残されている項目を読み取る	非入力/非リターンレジスタをクリアする	別個のプロセス
			明示的なレジスタのクリア
			色付け
572e	このコードにより呼び出される関数によるスタックに残された項目の読取り	呼び出されるスタックをアクセス不可能にする	別個のプロセス
			別個のスタック
			能力
			色付け

【図５２】

575

	防ぐべき項目/ランタイム挙動	予防的な活動	仕組み
572f	スタックフレフィックス上の項目の書き直し	スタックプレフィックスをアクセス不可能にし、または別様に書込み不可能にする	別個のプロセス
			別個のスタック
			能力
			色付け
572g	スタックフレフィックスの中の項目の読取り	スタックプレフィックスをアクセス不可能にする	別個のプロセス
			別個のスタック
			能力
			色付け
572h	プレフィックスの中の制御フローのリダイレクト（たとえば、リターンポインタの上書き）	リターンアドレスの保護	別個のプロセス
			別個のスタック
			色付けリターンアドレスを記憶するスタック位置をタグ付けする。
			能力：アクセス権限によりリターンポインタをタグ付けする

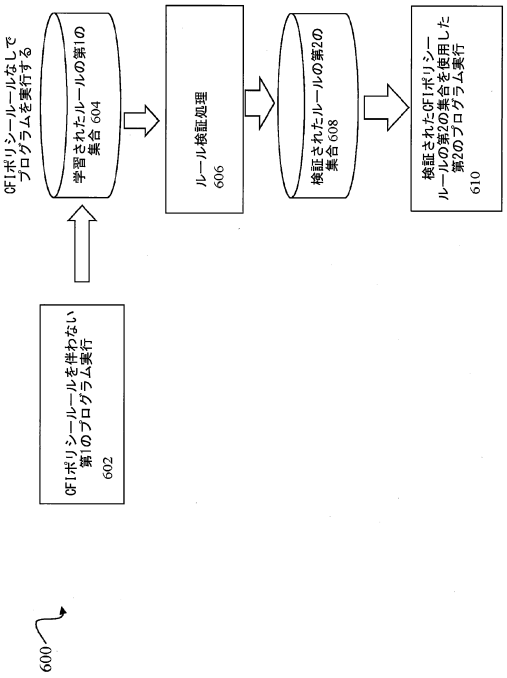
【図５３】



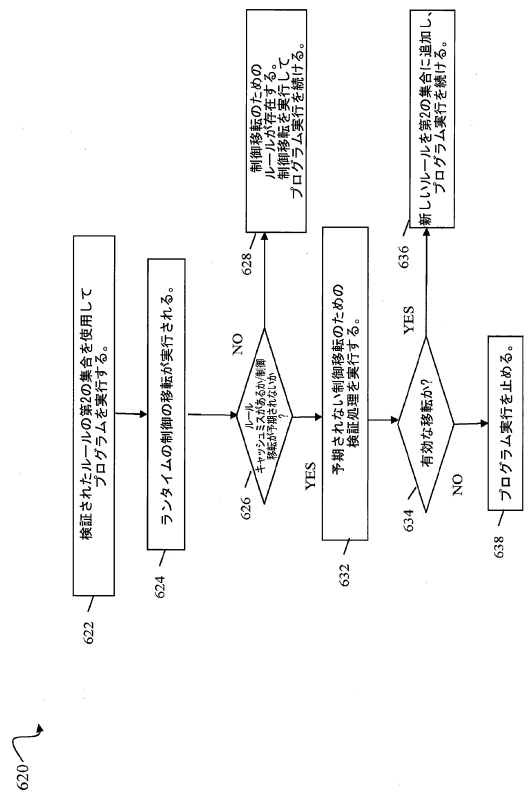
防ぐべき項目／ランタイム挙動	予防的な活動	仕組み
現在のフレームの中の意図されない項目の書き直し	オブジェクトの整合性を維持する	オブジェクトごとの能力
		オブジェクトごとの色
現在のフレームの中の項目の読取り	オブジェクトの整合性を維持する	色ごとの能力
		オブジェクトごとの色
前身のフレームの中の意図されない項目の書き直し	フレームを隔離する	フレームごとの能力
		フレームごとの色付け
前身のフレームの中の項目の読取り	フレームを隔離する	フレームごとの能力
		フレームごとの色付け
このコードにより呼び出される関数によるスタックに残された項目の読取り	コールスタックをアクセス不可能にする	別個のプロセス
		別個のスタック
		能力
		色付け
リターンポインタの変更	リターンポインタを保護する	色付け-リターンアドレスを記憶するスタック位置をタグ付けする
		能力-アクセス権限によりリターンポインタをタグ付けする

580
581a
581b
581c
581d
581e
581f

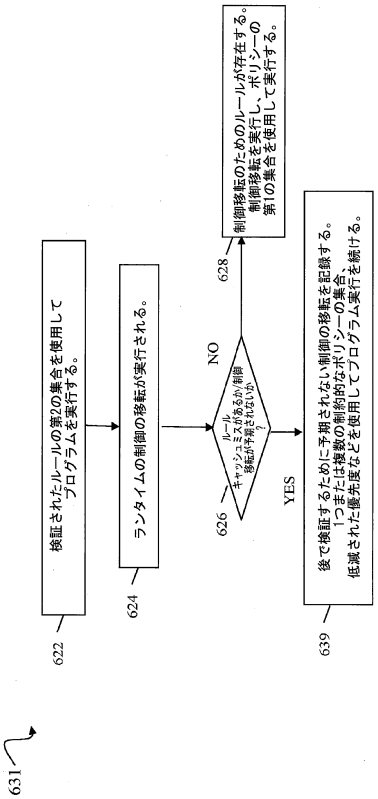
【図５４】



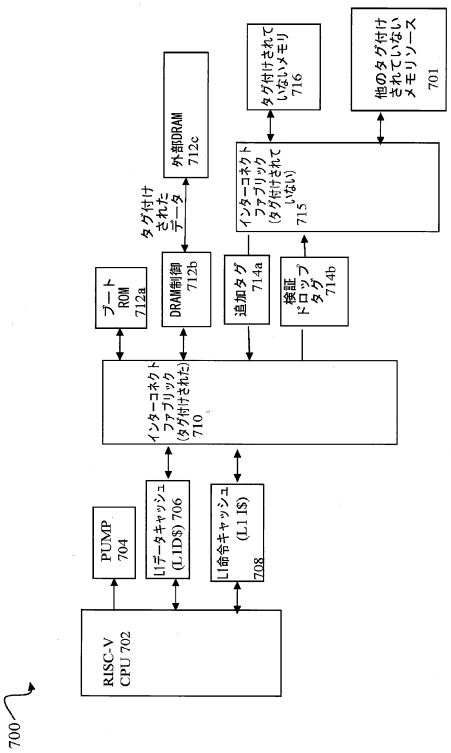
【図５５】



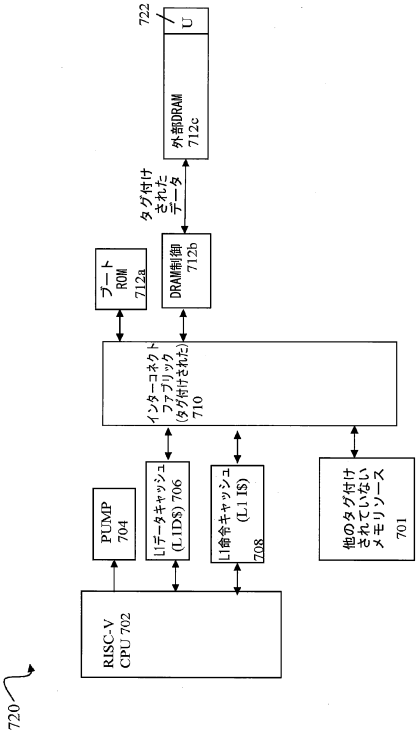
【図５６】



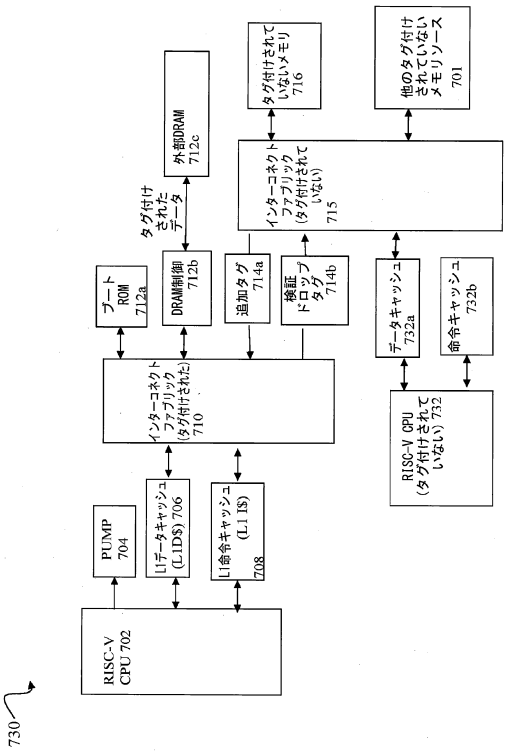
【図５７】



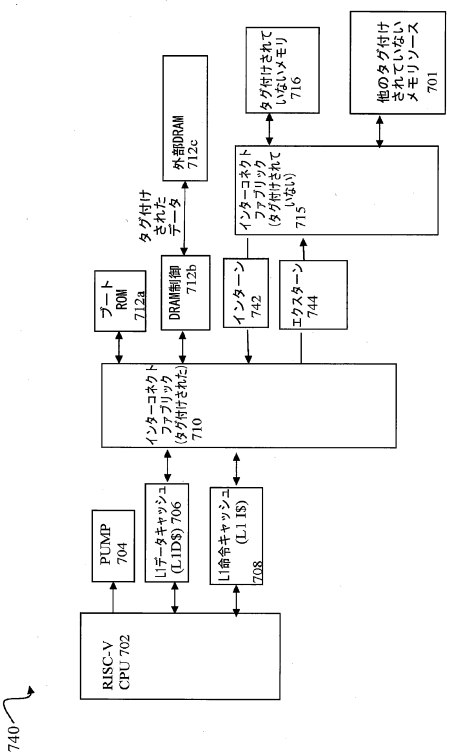
【図５８】



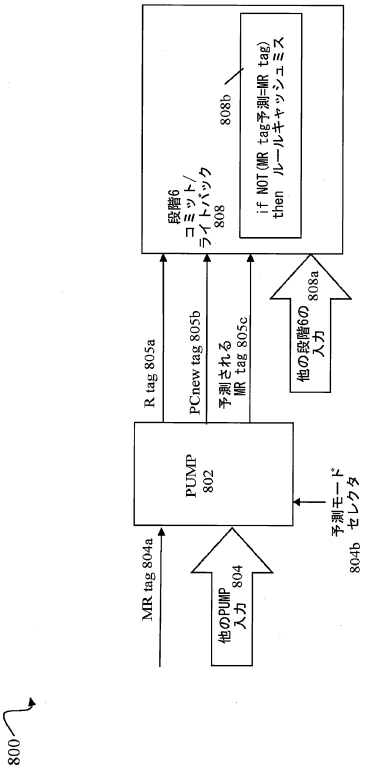
【図５９】



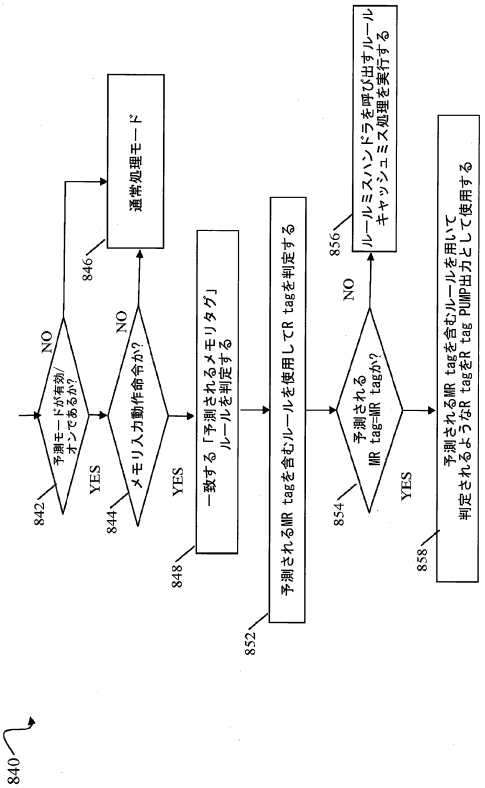
【図６０】



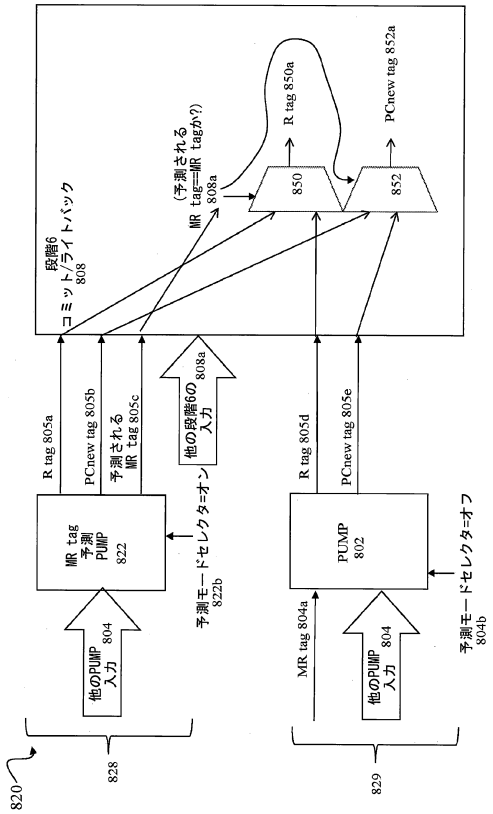
【図６１】



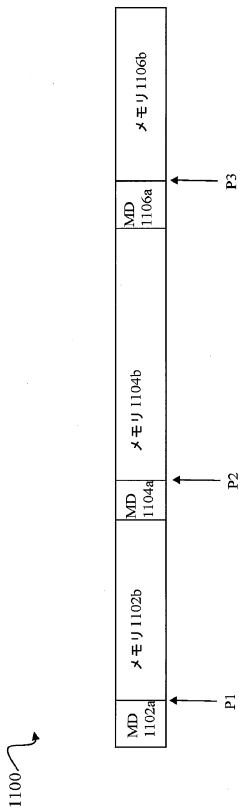
【図６２】



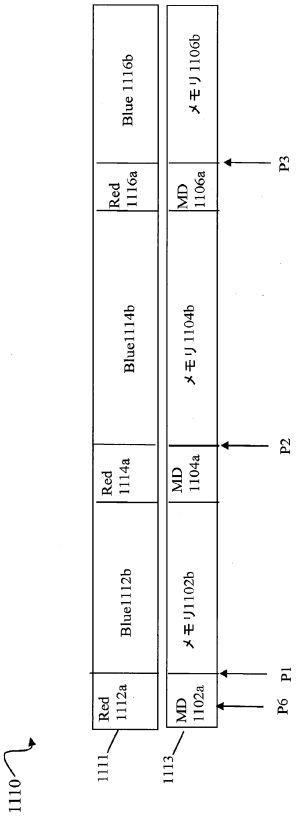
【図６３】



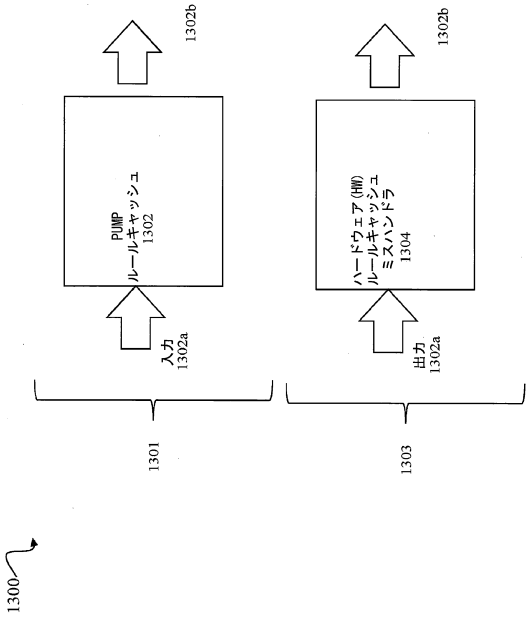
【図６４】



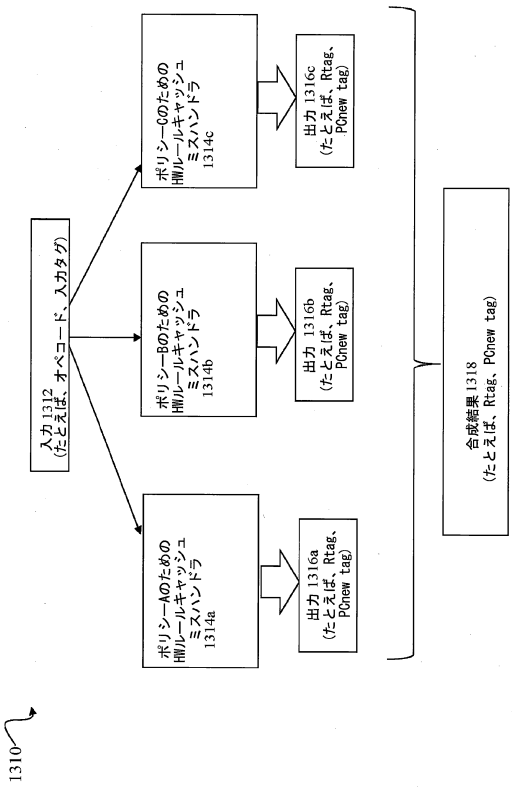
【図６５】



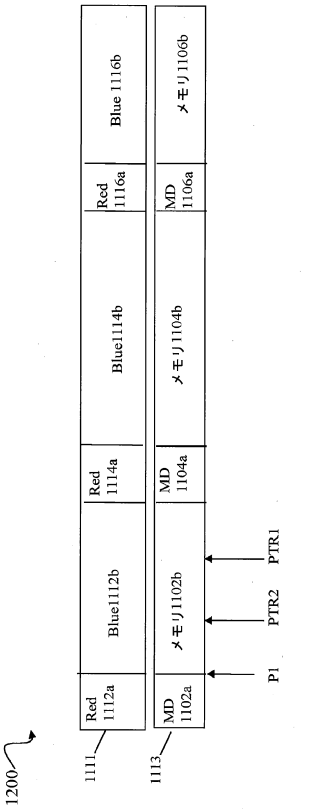
【図６６】



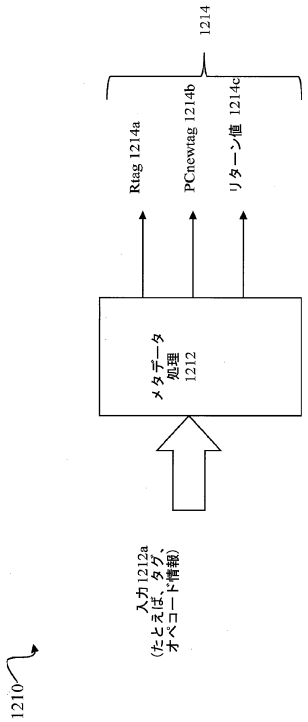
【図６７】



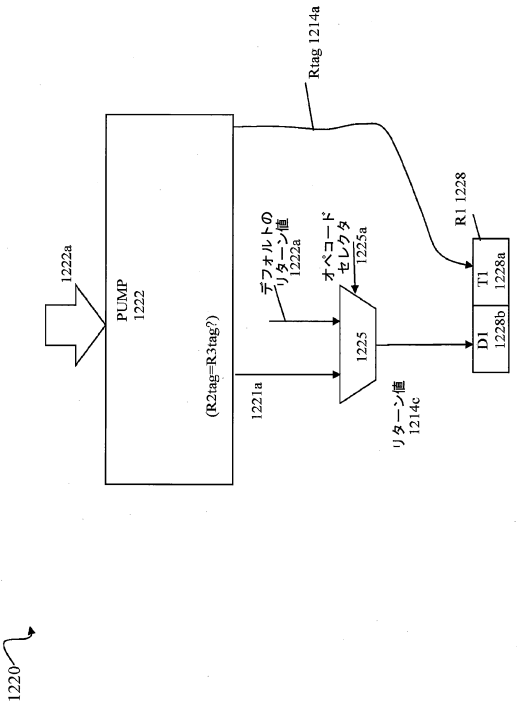
【図６８】



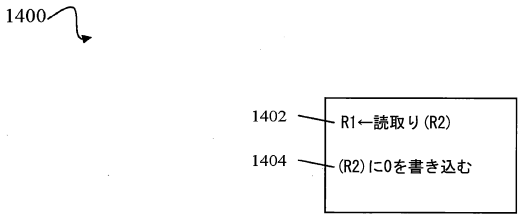
【図６９】



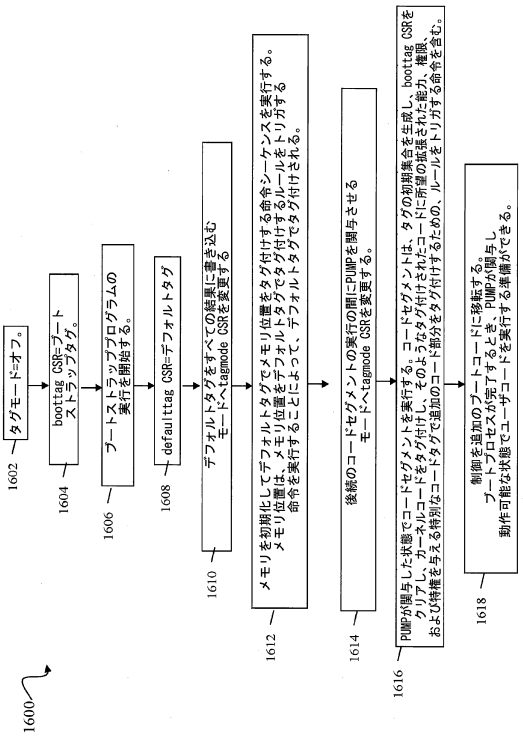
【図７０】



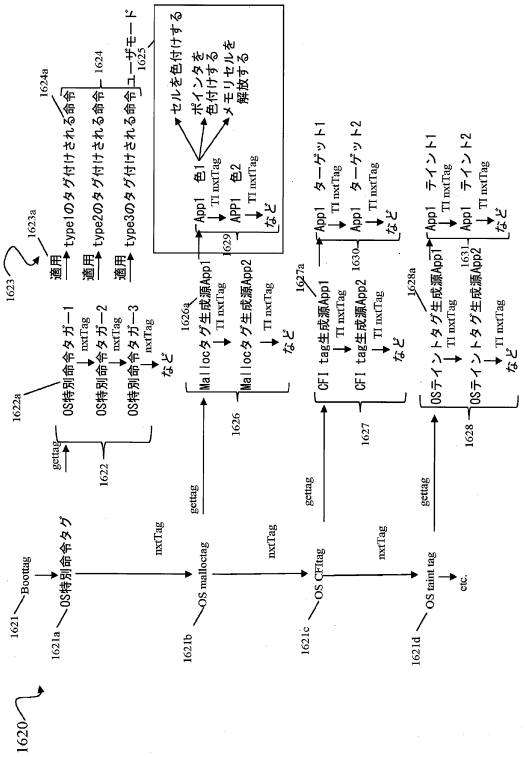
【図７１】



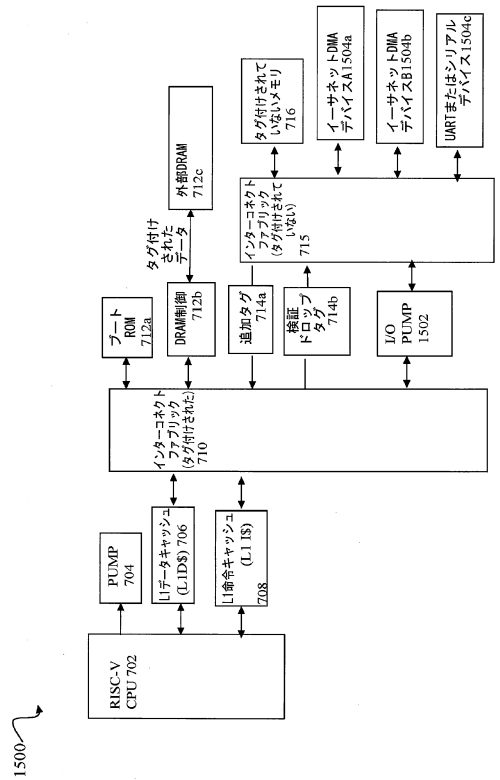
【図７２】



【図７３】



【図７４】



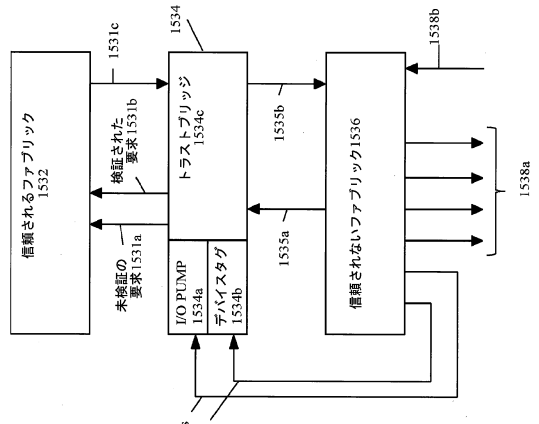
【図７５】

1520

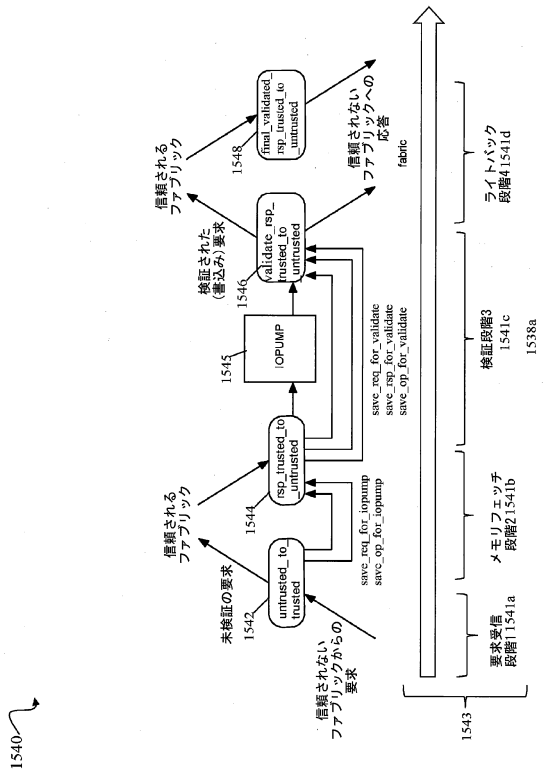
アドレス	名前	説明
0x00	transaction id	このアドレスへの書込みはトランザクション(プリフェッチのための)をインクリメントする;現在のtransaction idのリターンを読み出す
0x01	opgrp	ルールミスのためのオペグループ
0x02	byteenable	ルールミスのためのバイトイネーブル
0x03	pctag	ルールミスのためのPCタグ
0x04	citag	ルールミスのためのCI(現在の命令)タグ
0x07	mtag	ルールミスのためのメモリタグ
0x08	newpctag	命令の完了に際してPCに付けるべきタグ
0x09	rtag	命令からのメモリ結果に付けるべきタグ
0x0A	commit	書込み値が現在のtransaction idと一致するとき、IOPUMPへのルールの書込みをトリガする
0x0E	status	IOPUMPの有効/フォルトステータス
0x0F	flush	このアドレスへの書込みはIOPUMPのフラッシュをトリガする

1522a 1522b 1522c 1522d 1522e 1522f 1522g 1522h 1522i 1522j 1522k
1524 1526 1528

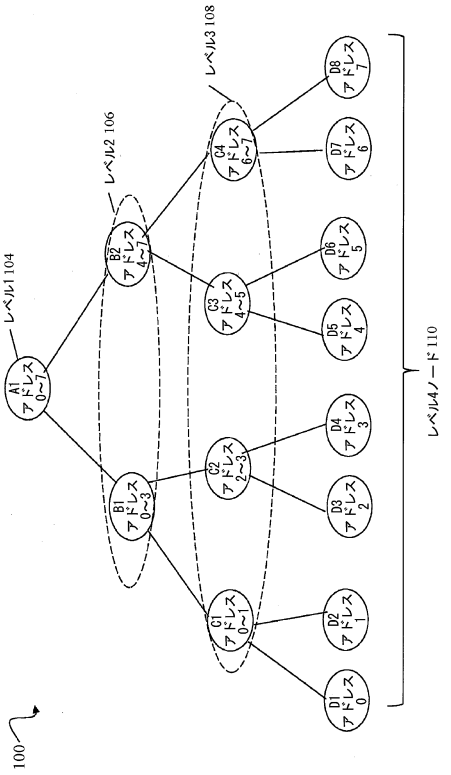
【図７６】



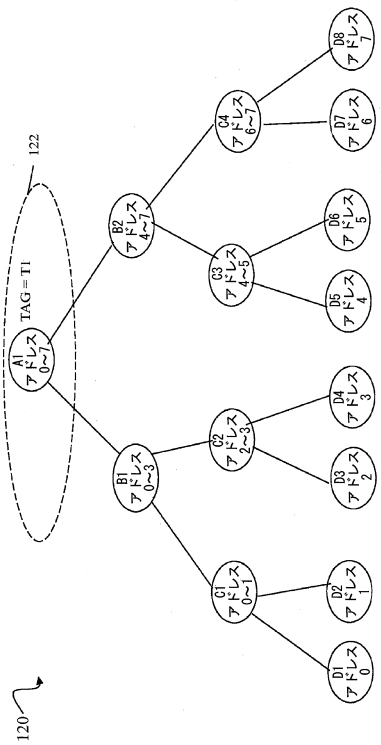
【図７７】



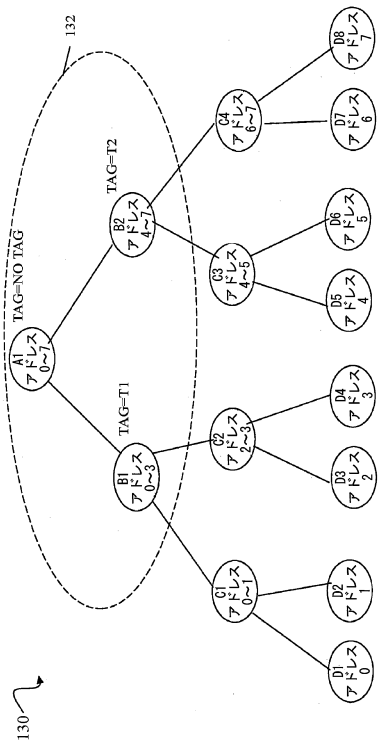
【図７８】



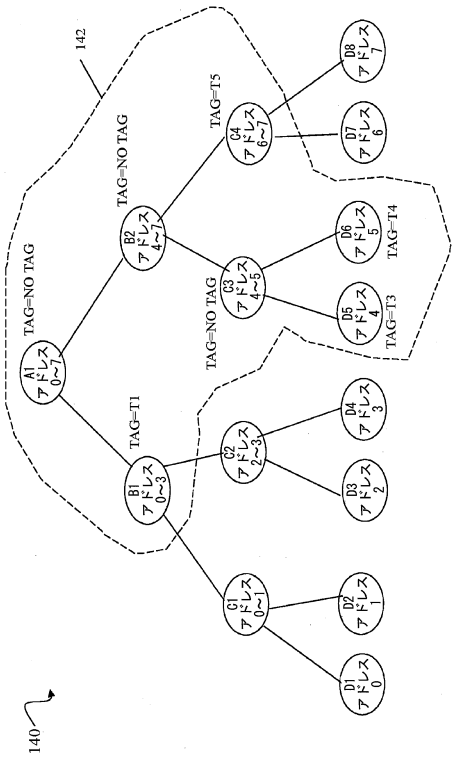
【図７９】



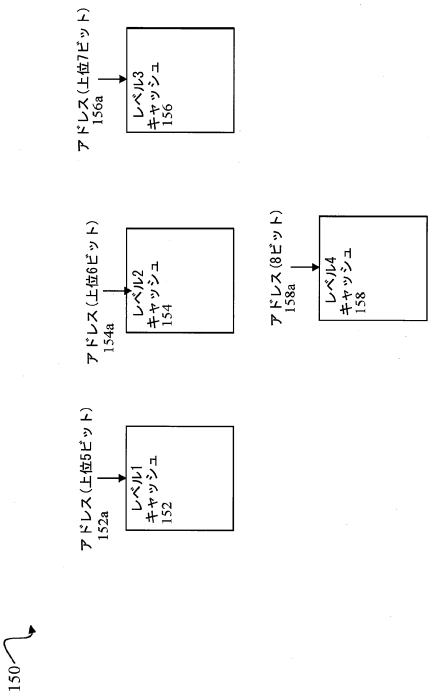
【図８０】



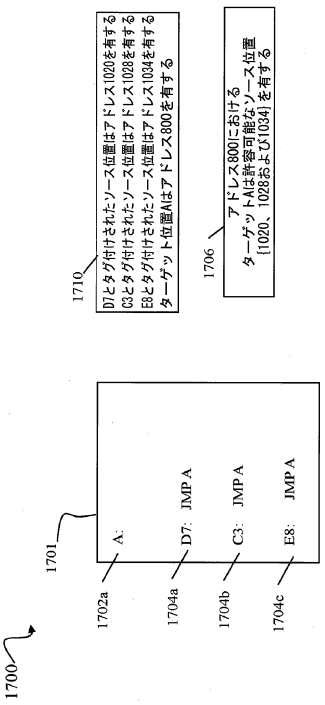
【図８１】



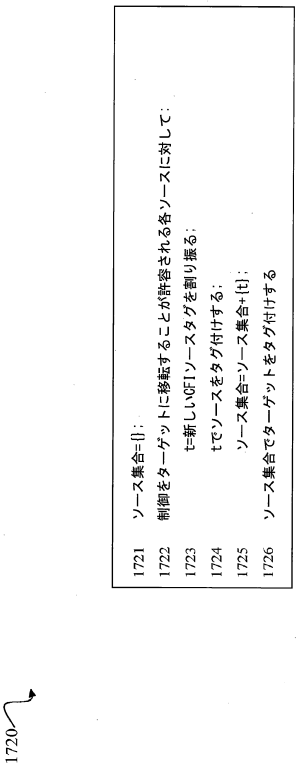
【図８２】



【図８３】



【図８４】



フロントページの続き

(51)国際特許分類　　ＦＩ
Ｇ０６Ｆ　21/52　(2013.01)　　Ｇ０６Ｆ　21/52

(33)優先権主張国・地域又は機関
米国(US)
(31)優先権主張番号　62/270,187
(32)優先日　平成27年12月21日(2015.12.21)
(33)優先権主張国・地域又は機関
米国(US)
ストリート・４７４３
(72)発明者　イーライ・ボリング
アメリカ合衆国・マサチューセッツ・０１９４４・マンチェスター・プレザント・ストリート・９９
審査官　金沢 史明
(56)参考文献　英国特許出願公開第２５１９６０８（ＧＢ，Ａ）
特開２０１３－２４２６３３（ＪＰ，Ａ）
DHAWAN, Udit et al.，PUMP: a programmable unit for metadata processing，Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy (HASP '14)，ACM，2016年06月，Article No. 8，pp. 1-8，＜DOI: 10.1145/2611765.2611773＞
DHAWAN, Udit et al.，Architectural Support for Software-Defined Metadata Processing，Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15)，ACM，2015年03月，pp. 487-502，＜DOI: 10.1145/2694344.2694383＞
(58)調査した分野　(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　２１／００－２１／８８
Ｇ０６Ｆ　１２／１４
Ｇ０６Ｆ　９／３０
Ｇ０６Ｆ　１２／０８７５