(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2014-10832
(P2014-10832A)
(43)公開日　平成26年1月20日(2014.1.20)

(51)Int.Cl.			F I			テーマコード（参考）
G06F	9/32	(2006.01)	G06F	9/32	330A	5B033
G06F	9/45	(2006.01)	G06F	9/44	322G	5B081

審査請求 未請求　請求項の数 16　ＯＬ　外国語出願　（全 27 頁）

(21)出願番号　特願2013-124943 (P2013-124943)
(22)出願日　平成25年6月13日 (2013.6.13)
(31)優先権主張番号　13/537,731
(32)優先日　平成24年6月29日 (2012.6.29)
(33)優先権主張国　米国 (US)

(71)出願人　501144003
アナログ・デバイシズ・インコーポレーテッド
アメリカ合衆国マサチューセッツ州ノーウッド，ワン・テクノロジー・ウェイ（番地なし）
(74)代理人　100102842
弁理士　葛和　清司
(72)発明者　パーキンス，マイケル　ジー.
イギリス国　イーエイチ１１　１ティーキュー　エディンバラ、ニュートン　ストリート　ゴーギー　３　３エフ２

最終頁に続く

(54)【発明の名称】ステージド・ループ命令

(57)【要約】
【課題】
ハードウェア並列性の効率的な使用
【解決手段】ループ命令が分析されて、それらの間の依存性および利用可能なマシンリソースに基づいて、ステージ番号が割り当てられる。ループ命令は、そのステージ番号に基づいて選択的に実行され、それによって明示的なループセットアップ命令およびループテアダウン命令を不要にする。単一命令多重データ（ＳＩＭＤ）マシン上で、各命令の最終インスタンスは、オリジナルループの反復数に応じて、処理要素またはベクトル要素のサブセット上で実行してもよい。
【選択図】図１

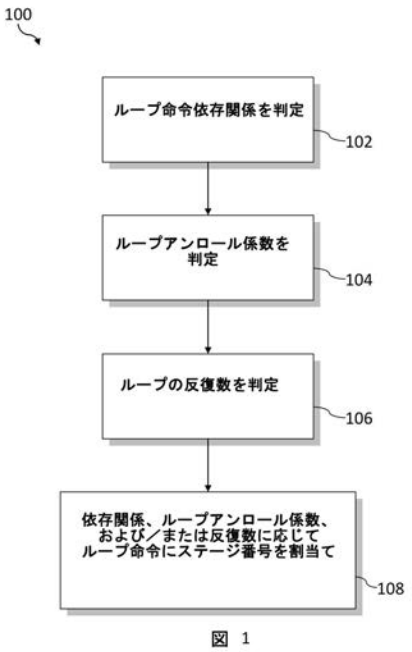

図 1

【特許請求の範囲】
【請求項１】
コンピュータプロセッサによってループを実行する方法であって、
前記ループ用のループカーネル命令であって、それぞれがそれに関連するステージ番号を有するループカーネル命令を、前記コンピュータプロセッサのメモリにロードすること；
前記ループの各反復中に、ステージ閾値を判定すること；
前記ステージ閾値およびステージ番号に基づいて、前記ループカーネル命令のサブセットを実行することによってループプロログを実行すること；
前記ループカーネル命令を繰り返し実行することによってループカーネルを実行すること；
前記ステージ閾値およびステージ番号に基づいて、前記ループカーネル命令のサブセットの内の第２のセットを実行することによって、ループエピログを実行すること
を含む、前記方法。
【請求項２】
コンピュータプロセッサがスカラープロセッサであって、ループカーネル命令が、命令レベル並列性またはマルチサイクルレーテンシを有する命令を含み、前記ループカーネル命令がソフトウェアパイプライン化されている、請求項１に記載の方法。
【請求項３】
ステージ番号が、アンロールされたループカーネルの各コピーに応じて調節される、請求項１に記載の方法。
【請求項４】
ループ命令の最終的な実行は、ループの初期の反復におけるよりも、より少ない処理要素またはベクトル要素上で行われる、請求項１に記載の方法。
【請求項５】
より少ない処理要素またはベクトル要素が、各処理要素またはベクトル要素を制御するマスクビットに応じて選択される、請求項４に記載の方法。
【請求項６】
ステージ閾値に関する情報を含む、ループセットアップ命令を受け取ることをさらに含む、請求項１に記載の方法。
【請求項７】
ループセットアップ命令が、合計ループ数、ループ命令を実行するのに使用する処理要素の数、ステージ数、またはループアンロール係数を含む、請求項６に記載の方法。
【請求項８】
ループエピログ、ループカーネル、およびループプロログを実行することが、命令ステージ番号をコンピュータプロセッサ内のステージ閾値と比較することを含む、請求項１に記載の方法。
【請求項９】
コンピュータ実行可能なループ命令を実行するシステムであって、
ループ情報を含むループセットアップ命令を受け取るためのシーケンサ；
ステージ閾値に関係するデータであって、前記ループ情報から導出されるデータを記憶する、１つまたは２つ以上のレジスタ；および
（ｉ）前記ループカーネル命令に関連する前記ステージ閾値およびステージ番号に基づいて、受け取ったループカーネル命令のサブセットを実行することによって、ループプロログを実行し、（ｉｉ）前記ループカーネル命令を繰り返し実行することによって、ループカーネルを、また（ｉｉｉ）前記ステージ閾値およびステージ番号に基づいて、前記ループカーネル命令のサブセットの第２のセットを実行することによって、ループエピログを実行する、処理要素
を含む、前記システム。
【請求項１０】

レジスタがステージマスクレジスタまたはステージ閾値レジスタを含む、請求項９に記載のシステム。
【請求項１１】
ループカーネル命令を実行する少なくとも１つの追加の処理要素、または複数データ要素を並列に処理するためのベクトル命令をさらに含む、請求項９に記載のシステム。
【請求項１２】
命令の最終実行中に、処理要素またはベクトル要素のサブセットだけが、ループカーネル命令を実行する、請求項１１に記載のシステム。
【請求項１３】
マスクレジスタが、最終反復においてループカーネル命令を実行することに関係した情報を記憶する、請求項１２に記載のシステム。
【請求項１４】
ループ情報が、合計ループ数、ループ命令を実行するのに使用する処理要素の数、ステージ数、またはループアンロール係数を含む、請求項９に記載のシステム。
【請求項１５】
ループ数を記憶するループ数レジスタをさらに含む、請求項９に記載のシステム。
【請求項１６】
ループ数が、合計ループ数を、ループ命令を実行するのに使用する処理要素の数で割ることによって導出されて、ステージ数に応じて調節されるとともに、ループアンロール係数に応じて丸められる、請求項１５に記載のシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明の態様は、一般的にはハードウェア並列性の効率的な使用に関し、具体的にはループアンローリング（loop unrolling）およびソフトウェアパイプライン化に関する。
【背景技術】
【０００２】
コンパイラは、高レベル汎用言語（例えば、Ｃ、Ｃ＋＋、またはＪａｖａ（登録商標））で書かれたコンピュータソースコードを取り込み、それを低レベルの特定マシン向けオブジェクトコードに変換する。簡単な、シングルコアプロセッサ用のコンパイルコードは、比較的に単純明快な高レベル命令の低レベル命令への一対一の変換で構成することができる。例えば、Ｃ＋＋クラスにおけるデータへのアクセスは、アドレス計算およびメモリアクセスのマシンレベル命令にコンパイルしてもよい。
【０００３】
しかしながら、複数の機能ユニット、またはベクトル処理に対するサポートを有するプロセッサ用のコンパイルコードは、ずっと複雑になることがある。典型的な目標は、各コア（および／または所与のデバイスの特定のハードウェアに応じて、各プロセッサ、実行ユニット、およびパイプライン）をできるだけビジーに保つことによって、できる限り迅速にコンパイルされたプログラムを走らせることである。しかしながら、この目標には、最初に直列的／逐次的に書かれた命令を、並列に走るようにコンパイルすることが必要であるが、すべての命令が同時に実行可能であるわけではない。例えば、第２の命令に対する入力が、第１の命令の結果に依存する場合には、第１および第２の命令を並列に走らせることは不可能であり、第２の命令は第１の命令が完了するのを待たなければならない。
【０００４】
「スマート」コンパイラは、並列に走らせることのできる命令を認識して、（超長命令語（ＶＬＩＷ：very-long-instruction-word）プロセッサ用に生成されたコードのように明示的に、またはスーパースカラプロセッサ用に生成されたコードのように暗示的に）それを行わせるように適合されたマシンコードを作成する。並列化可能な状況の２つの広範なカテゴリとして、（ｉ）命令レベル並列性を発揮する命令、および（ｉｉ）データレベル並列性を発揮する命令があげられる。命令レベル並列性とは、互いの出力に対して依存

性がなく、したがって並列に演算することのできる、２つ以上の命令を意味する。データレベル並列性とは、セットのメンバに対する個々のオペレーションが、他のメンバに関係するオペレーションに依存しない、データのセット（すなわち、ベクトル）に対して、オペレーションを実行することを意味している。例えば、２つのマトリックスを互いに加算するためには、要素レベルの加算オペレーションは独立であるので、マトリックス内の要素のデータレベル並列性を利用して、要素加算命令の一部または全部を並列に走らせてもよい。

【０００５】

コンパイラが、命令レベルおよびデータレベルの並列性を達成する１つの方法は、ソースコード内に書かれたループ（例えば、ｆｏｒループまたはｗｈｉｌｅループを）を利用することによる。ループの２回以上の反復を、並列に実行（すなわち、データレベル並列性を利用する、「ベクトル化」）してもよく、かつ／またはループの連続的反復を部分的に重複化（すなわち、命令レベル並列性を利用する、「ソフトウェアパイプライン化」）を行ってもよい。ソフトウェアパイプライン化の強力なアルゴリズムの１つが、「モジュロスケジューリング（modulo scheduling）」として知られている。ベクトル化に関しては、次のような場合には、（例えば）ｆｏｒループを呼び出して１０回反復させてもよい；すなわち各反復において実行される命令が、その他の反復の命令と独立であり、かつコンパイラが（例えば）５つの処理要素にアクセスできる場合には、コンパイラは、５つの処理要素のそれぞれにおいて、当該ループの２回の反復を並列に実行させる、アセンブリコードを作成してもよい。ソフトウェアパイプライン化に関しては、例えば、あるループが２つの命令を含むが、第１の命令が、前回反復の第２の命令の結果に依存しない場合には、ループの次回反復の第１の命令を、ループの現行反復のまだ実行中の第２の命令と、並列に走るようにスケジュール化してもよい。

【０００６】

ベクトル化およびソフトウェアパイプライン化の１つの短所は、実行コードのサイズを増大させることである。ベクトル化には、ループの奇数サイズの最終反復に対応するコードが必要となる（例えば、ループが１１回の反復を必要とし、５つの処理要素が利用可能である場合には、最後の反復は、処理要素の１つだけを使用する）。この「部分的に充填された」最終反復は、単に不効率であるだけにとどまらないことがある。つまり、多くの大規模プロセッサ配列は、正当なデータの安定したストリームを期待して調節されており、個々の処理要素をオフにすることは、それほど容易でないこともある。ソフトウェアパイプライン化には、効率的なコア命令のセット（「ループカーネル（loop kernel）」）を走らせる前に、ハードウェア環境を準備するために、セットアップ命令（「ループプロログ（loop prolog）」）を必要とし、その後には、ループをテアダウン（tear down）し、さらに別の命令のためのハードウェア環境をクリーンアップするオーバヘッド命令（「ループエピログ（loop epilog）」）がさらに必要である。多くの場合に、この追加の、オーバヘッドコードは、ループカーネルコードそれ自体よりも大きくなることもあり、制限された命令キャッシュまたはバッファキャパシティを有するプロセッサ上で、性能を低下させることがある。もう一つの短所は、可変の反復数（ループの「トリップ数」と呼ばれる）を有するループの扱いのまずさであり、この理由は、トリップ数はコンパイル時には知ることはできず、トリップ数の様々なテストが実行時には必要となり、それによって（特に、トリップ数が小さいときに）プログラムのランタイムが増大するためである。

【０００７】

これらの欠点に対処しようとする既存のシステムは、さらなる短所を生成するだけである場合がある。例えば、プロセッサによっては（例えば、ベクトルプロセッサは）、ループの最終反復において、個々の処理要素を選択的に動作不能にする方法を実装するものもある。しかしながら、ループの最終反復において処理エレメントを動作不能にすることは、ループカーネルにおける様々な反復からの命令と重複する、ソフトウェアパイプライン化とうまく相互作用しない。その他のシステムは、ループ命令を固定サイズのバッファに記憶して、特殊なループ命令を発行することによって、ループのセットアップ、テアダウ

ンおよび定常状態を表わすが、これらのシステムは、（固定バッファのサイズに基づいて）ループカーネルのサイズを制限するだけでなく、より複雑なループ（レジスタリネーミング（register renaming）を必要とするものなど）を取り扱うこともできない。さらに別のシステムは、精密な回転ハードウェアレジスタ(rotating hardware resistor）を使用して、複雑なループに対処するが、これらのレジスタは、プロセッサの他の部分から貴重なリアルエステートを取り上げる。したがって、任意のサイズと複雑度のループカーネルを効率的に実行する方法に対するニーズがある。

【発明の概要】

【０００８】

一般的には、本明細書に記載されるシステムおよび方法の様々な観点は、ソフトウェアプロログおよびエピログを必要とすることなく、ソフトウェアパイプライン化された、ベクトル化ループのセットアップおよびテアダウンを可能にする、コンパイラとハードウェアメカニズムを提供する。本発明の態様は、コードサイズを低減して、未知のトリップ数の場合には、結果として得られるコードの性能を向上させる。ループコードが分析されて、各命令に、実行の全体順序を反映するステージ番号が割り当てられ、具体的には、各命令を、時間的に順序づけられた複数の「ステージ」の１つに割り当ててもよく、これらのステージのそれぞれは、実行順序内でその位置を反映する「ステージ番号」を有する。このループが実行されると、その中の命令が、そのステージ番号に基づいて、選択的に実行される。

【０００９】

一観点において、コンピュータプロセッサによってループを実行する方法は、このループ用のループカーネル命令を、コンピュータプロセッサのメモリ中にロードすることを含む（この場合に、各ループカーネル命令は、それに関連するステージ番号を有する）。ステージ閾値がループの各反復中に判定されて、このステージ閾値およびステージ番号に基づいて、ループカーネル命令のサブセットを実行することによって、ループプロログが実行される。ループカーネルは、ループカーネル命令を繰り返し実行することによって実行され、ループエピログは、ステージ閾値とステージ番号に基づいてループカーネル命令のサブセットの内の第２のセットを実行することによって実行される。

【００１０】

コンピュータプロセッサは、スカラープロセッサとしてもよく、この場合に、ループカーネル命令は、命令レベル並列性またはマルチサイクルレーテンシを有する命令を含み、かつ／またはループカーネル命令がソフトウェアパイプライン化されている。ステージ番号は、アンロールされたループカーネルの各コピーに応じて調節してもよい。ループ命令の最終的な実行は、ループの初期の反復におけるよりも、より少ない処理要素またはベクトル要素上で行ってもよい。このより少ない処理要素またはベクトル要素は、各処理要素またはベクトル要素を制御するマスクビットに応じて、選択してもよい。ステージ閾値に関する情報を含むループセットアップ命令を受け取ってもよく、このループセットアップ命令には、合計ループ数、ループ命令を実行するのに使用する処理要素の数、ステージ数（stage count）、またはループアンロール係数（loop-unroll factor）を含めてもよい。ループエピログ、ループカーネル、およびループプロログを実行することには、命令ステージ番号をコンピュータプロセッサ内のステージ閾値と比較することを含めてもよい。

【００１１】

別の観点においては、コンピュータ実行可能なループ命令を実行するシステムは、ループ情報を含むループセットアップ命令を受け取るためのシーケンサを含む。１つまたは２つ以上のレジスタが、ステージ閾値に関係するデータを記憶する（このデータは、前記ループ情報から導出される）。処理要素は、（ｉ）ループカーネル命令に関連するステージ閾値およびステージ番号に基づいて、受け取ったループカーネル命令のサブセットを実行することによって、ループプロログを実行し、（ｉｉ）ループカーネル命令を繰り返し実行することによって、ループカーネルを実行し、また（ｉｉｉ）ステージ閾値およびステージ番号に基づいて、ループカーネル命令のサブセットの第２のセットを実行することに

よって、ループエピログを実行する。

【００１２】

レジスタは、ステージマスクレジスタまたはステージ閾値レジスタを含めてもよい。少なくとも１つの追加の処理要素がループカーネル命令を実行するか、またはベクトル命令が、複数のデータ要素を並列に処理してもよい。命令の最終実行中に、処理要素またはベクトル要素のサブセットだけが、ループカーネル命令を実行してもよい。マスクレジスタは、最終反復においてループカーネル命令を実行することに関係した情報を記憶してもよい。ループ情報には、合計ループ数、ループ命令を実行するのに使用する処理要素の数、ステージ数、またはループアンロール係数を含めてもよい。ループ数レジスタはループ数を記憶してもよく、このループ数は、合計ループ数を、ループ命令を実行するのに使用する処理要素の数で割ることによって導出して、ステージ数に応じて調節するとともに、ループアンロール係数に応じて丸めてもよい。

【００１３】

本明細書において開示する本発明の長所と特徴と共に、これらおよびその他の目的は、以下の説明、添付の図面および特許請求の範囲を参照すれば、より明白になるであろう。さらに、本明細書に記載する様々な態様の特徴は、互いに排他的ではなく、様々な組合せと順列で存在することができることを理解されたい。

【図面の簡単な説明】

【００１４】

図面において、同一の参照符号は、異なる図を通して、全体的に同一の部品を指している。以下の説明においては、本発明の様々な態様を、以下の図面を参照して説明する。

【００１５】

【図１】図１は、本発明の一態様による、ループ命令にステージ情報の注釈をつける方法を示すフローチャートである。

【図２】図２は、本発明の一態様による、ステージド・ループの実行を示すブロック図である。

【図３】図３は、本発明の一態様による、部分的にアンロールされたステージド・ループの実行を示すブロック図である。

【図４】図４は、本発明の一態様による、遅延開始を必要とする部分的にアンロールされたステージド・ループの実行を示すブロック図である。

【図５】図５は、本発明の一態様による、複数処理要素上で走るステージド・ループの実行を示すブロック図である。

【図６】図６は、本発明の一態様による、ループ命令にステージを用いて注釈をつけるシステムのブロック図である。

【図７】図７は、本発明の一態様による、ステージド・ループ命令を実行するシステムのブロック図である。

【図８】図８は、本発明の一態様による、ステージド・ループ命令を実行する方法のフローチャート図である。

【００１６】

詳細な説明

本明細書においては、効率的にループカーネルをセットアップおよびテアダウンするために、どの命令を、どの順序で、実行しなければならないかに関する追加の情報を、ループカーネル命令、またはループ命令のグループにタグ付けすることによって、複数の処理要素または機能ユニットを有する、コンピュータシステム上でループを走らせる方法およびシステムの様々な態様を記載する。プロセッサが命令を受け取ると、タグ情報を分析し、それに基づいて、ループカーネルを実行する前後に、ループカーネル情報のあるものを実行し、それによって明示的なループセットアップ／テアダウン命令を不要にする。例えば、コンパイラは、ループ内のいくつかの命令に［０］タグを、その他の命令に［１］タグを、さらに他の命令に［２］タグをタグ付けしてもよく、非常におおざっぱには、以下でより詳細に説明するように、タグ付けされたグループ内の命令は、命令レベル並列性を

示し、より大きい番号でタグ付けされたグループにおける命令は、より小さい番号のグループにおける命令の出力に依存することがある。ループの最初の反復において、プロセッサは、［０］をタグ付けされた命令を（並列に）実行し、第２の反復において、［０］または［１］をタグ付けされた命令を、そして第３および後続の反復において、すべての命令（すなわち、［０］、［１］または［２］のいずれかをタグ付けされた命令）を実行してもよい。ループの最後から２番目の反復において、プロセッサは、［１］または［２］をタグ付けされた命令だけを実行し、ループの最終反復において、［２］をタグ付けされた命令だけを実行してもよい。以下でより詳細に説明するように、この簡単な例を、より複雑なループおよび／またはプロセッサアーキテクチャに拡張することができる。

【００１７】

説明の目的で、ループについてのいくつかの一般属性または特性を最初に定義する。従来型のループ用のコードは、ループセットアップ命令（すなわち、ループプロログ）、ループカーネル、およびループテアダウン命令（すなわち、ループエピログ）が含まれる。本発明の一態様によって実行されるループは、ループ命令のすべてが実行されるわけではない、その実行の発端での１回または２回以上の反復、ループ命令のすべてが実行される、１回または２回以上の反復（すなわち、ループの実行の「定常状態」部分）、および、最終的に、ループ命令のすべてが実行されるわけではない、ループの実行の終端における、１回また「は２回以上の反復を含む。一態様において、ループ実行の発端および終端での反復によって、ループの定常状態部分に対するセットアップおよびテアダウン機能が、それを行うための明示的な追加命令を必要とすることなく、得られる。

【００１８】

パイプライン化されたループにおける、それぞれの連続するループ反復の始点間の合計サイクル数は、開始間隔（initiation interval）と呼ばれており、これは、定常状態ループカーネル内の命令行数でもある。この定常状態において全（開始間隔）サイクルを実行するときに、新規の反復が始まり、同時に、先に始まった反復がその実行を終了する。開始間隔は、非パイプライン化ループの反復を完了するのに要する合計サイクル数よりも短く、その理由は、最初のループの複数反復が重複して、ループの後からの反復を、先の反復が完了する前に始めることを可能にするからである。例えば、開始間隔が２であり、反復を完了するのに必要な合計サイクル数が６であるとすると、反復ｎ、ｎ＋１、およびｎ＋２は、すべて同時にインフライト（in flight）となる。

【００１９】

ループ内の命令は、ループ反復が始まってから経過した開始間隔の倍数によって決まる「ステージ番号」が与えられる。上記の例において、非パイプライン化ループにおける最初の２つの命令行は第１のステージに属し、命令行３および４は第２のステージに属するのに対して、命令行５および６は、第３のステージに属する。ステージの合計数は、ステージ数と呼ばれ、この例においては、ステージ数は３である。ループカーネルの定常状態においては、反復ｎは、最終ステージ（この場合には、第３のステージ）における命令を実行しており、反復ｎ＋１は、最後から２番目のステージ（この場合には、第２のステージ）を実行しており、以下同様である。我々の例においては、反復ｎ＋２は、第１のステージにおいて命令を実行することによって、実行を開始している。

【００２０】

本発明の一態様においては、ステージ番号は、ループカーネルにおける命令にタグ付けするのに使用される。第１のステージに属する命令は［０］をタグ付けされ、第２のステージに属する命令は［１］をタグ付けされ、以下同様である。

【００２１】

開始間隔が小さくなるほど、より効率的なループカーネルが得られる。開始間隔を減少させるのを助ける要因としては、プロセッサ上で利用可能な機能ユニット数が大きいこと、および命令間のデータ依存性の数が小さいことがあげられる。また、小さい開始間隔は、一般に、ループのためのコードを発生させるのに必要なステージ数を増大させ、これによって、明示的なループプロログおよびエピログを発生させなければならない場合には、

より多くのコード拡張につながる。ループが定常状態に達する前に走らせる必要のあるステージの数は、ステージ数より１つ少ない数（すなわち、ステージ数－１）であり、ループが定常状態に達する前に走らせる必要のあるサイクルの数は、（ステージ数－１）×（開始間隔）である。本発明の一態様においては、このオーバヘッドに対するコード拡張は除外される。

【００２２】

ループをアンロールすることは、ループ命令の一部または全部を、より多数の非ループ命令でリライトすることを意味する。６回反復する、２命令ｆｏｒループは、例えば、１２の通常（非ループ）命令としてリライトすることによって完全にアンロールすることができる。別の例として、そのｆｏｒループは、反復毎の命令は多いが、合計反復は少なくなるようにリライトすることによって、部分的にアンロールすることができる（例えば、３回反復する４命令ｆｏｒループ）。ループが最初に、奇数の反復数を有するように書かれている場合（例えば、５回反復する２命令ｆｏｒループ）には、この部分アンロール技法は、ループ外に初期（または最終）命令セットを含めることができる（例えば、２命令＋２回反復する４命令ｆｏｒループ）。

【００２３】

ループアンローリングは、ループが、その結果が長時間、ライブである（すなわち、利用可能にしておく必要がある）命令を包含するときに、ソフトウェアパイプライン化を容易にするのに使用してもよい。パイプライン化されたカーネル内で、毎（開始間隔）サイクル、新規の反復が始まるので、命令の結果は、普通、その定義後に（開始間隔）サイクルを超えて使用することができず、結果として、この使用に達する前の後続の反復における命令のコピーによって上書きまたは「クロバリング（clobbering）」される。この問題は、ループカーネルをアンロールすることによって解決してもよい。ループをアンロールすることは、勿論のこと、長寿命命令の別のコピーをループカーネル中に導入し、この命令が長寿命値を上書きすることを避けるために、（先の命令の結果をクロバリングするのを回避するように、）その値をその中に記憶する、リネームされたレジスタが与えられる。上記で定義された用語の１つを使用すると、ループカーネルは、ループの開始間隔（すなわち、ループの「長さ」）が、ループ内の命令の１つ（すなわち、「長寿命」命令）の寿命よりも短い場合に、アンロールされる。必要なアンローリングのレベルは、開始間隔をどの程度、超えているかの関数であり、ループアンロール係数と呼ばれている。アンローリングが必要でない場合には、ループアンロール係数は１であり、ループカーネルの２つのコピーが必要である場合には、ループアンロール係数は２であり、以下同様である。

【００２４】

ループの別の属性は、その反復回数が固定であるか、可変であるかである。固定であれば、コンパイラは、（例えば）反復回数が奇数であるか、または偶数であるか（これは、上記で考察したように、部分的にアンロールされたループに影響を与える）を試験して、それに応じてコードを発生させることによって、その実行を容易にスケジューリングすることができる。しかしながら、反復回数が可変である場合には（例えば、１からｘまで反復するｆｏｒループ）、コンパイラは、反復回数が奇数であるか、偶数であるかを知ることができない。

【００２５】

次に、本発明の一態様に注目すると、図１は、ループに対するコードをコンパイルする方法１００を示している。第１ステップ１０２において、命令依存関係（例えば、命令レベルまたはデータレベルの並列性に対する障害）が、ループ命令において判定される。第２ステップ１０４において、マシン上で利用可能な並列リソースを超えることなく、（開始間隔）サイクル毎に、新規の反復を開始させて、現在実行中の反復と重複させるように、開始間隔と共に、これらの依存関係に従う、ループの非パイプライン化版が生成される。第３ステップ１０６において、ループに対するループアンロール係数が（例えば、ループの開始間隔およびループ命令の寿命に基づいて）判定される。第４ステップ１０８において、ループの反復数（例えば、未知か既知か、既知であれば、奇数か偶数か）が判定さ

れる。第５ステップ１１０において、依存関係およびハードウェア上で利用可能な並列性に少なくとも部分的に基づいて、ステージ番号がループ内の命令に対して割り当てられる。

【００２６】

一態様においては、所与のループに対する命令は、それらの間の依存関係を有するが、ループに対するアンロール係数は１であり（すなわち、アンローリングを必要としない）、ループの反復数は既知である。この態様においては、コンパイラは、各ループ命令に、パイプライン情報、すなわちステージ番号をタグ付けする。この情報は、３ビットまたは４ビットのバイナリ数の形態にすることができるが、本発明は、この情報のいかなる特定の表現にも限定されるものではない。ステージ番号は、命令コード化自体の範囲に保持するか、または並列データ構造などの、別のメカニズムで規定してもよい。

【００２７】

実行中に、ループの走行は最終的に定常状態に達し、この状態では、各反復においてすべてのループ命令が実行されるが、定常状態に達する前には、各反復においてループ命令のサブセットだけが実行される。独立した命令が最初に実行され、場合によっては、それらの最初の命令に依存する可能性のある中間の命令が続き、最初の命令および中間の命令に依存する可能性のある命令が続き、すべての命令が実行されるまで、以下同様である。ループの最後の数回の反復において、最初に実行を始めた命令が実行を停止し、後に追加された、依存性を有する命令が、すべての命令がその指示された回数実行されるまで、実行される。

【００２８】

意味論的（semantically）には、命令は、そのステージ番号を、各反復において変わる可能性のあるステージ閾値に対して試験することによって、実行される。ループセットアップ部分において、命令は、そのステージ番号がステージ閾値以下である場合に実行され、ループテアダウン部分においては、命令は、そのステージ番号がステージ閾値以上である場合に実行される。ステージ閾値は、ループセットアップ部分およびテアダウン部分の間には、各反復毎に増分され、ステージ閾値は、ループセットアップ部分においてその最も低い値に初期化され、ループテアダウン部分において２番目に低い値に初期化される。ステージ閾値は、ループの定常状態部分においては一定のままである。一態様においては、ステージ閾値は、以下により詳細に説明するように、ループアンロール係数と等しい量だけ増分される。ハードウェアにおけるそのような意味論を達成するのには多くの方法があり、本発明はステージ閾値を使用することに制限されるものではないことを当業者は理解するであろう。同一の結果を得るためのその他の方法として、ループカーネルの次の反復を始めるに当たり、シフトさせることのできるマスクレジスタを使用すること、またはループトリップ数、ステージ数およびアンロール係数の値から導出される、その他の状態の使用することがある。ステージ閾値の考察は、本明細書においては、説明を分かり易くする目的だけで使用される。

【００２９】

図２に示された、例２００は、本発明のこの態様を説明する助けとなる。例２００は、特定の命令セットおよび依存関係を含むが、本発明は、この構成だけに限定されるものではなく、例２００は、説明の目的だけを意味するものである。この例において、ループは、３回の反復を実行し、５つの命令：ａ、ｂ、ｃ、ｄ、およびｅを含む。命令ａとｃの間でコンフリクトが存在する（すなわち、命令ｃは命令ａの出力に依存し、かつ／またはハードウェアはそれらの同時実行を可能にしない）。同様に、命令ｄとｃは、別のコンフリクトのために、命令ａ、ｂ、またはｃと同時には走らせることはできない。これらの依存関係に基づいて、コンパイラは、命令ａおよびｂをステージ［０］に、命令ｃをステージ［１］に、命令ｄおよびｅをステージ［２］に割り当てる。

【００３０】

ループカーネルの開始間隔も判定してもよい。上記で考察したように、開始間隔は、パイプライン化されたループカーネルの１回の反復を実行するのに要するサイクル数である

。この例においては、ハードウェアは、第１サイクルにおいて命令ａ、ｃおよびｄを、第２サイクルにおいて命令ｂおよびｅを、実行することが可能であると仮定する。すなわち、すべてのループ命令を実行するのに２サイクルを要するので、開始間隔は２である。例えば、以下のようになる。

```
for（i＝1；i＜＝3；i＋＋{
    a［0］；c［1］；d［2］；／／第１サイクルで実行
    b［0］；　　e［2］；／／第２サイクルで実行
}
```

【００３１】

この情報を使用して、命令ａ、ｃ、およびｄを第１サイクルに、命令ｂおよびｅを第２サイクルに示して、ループカーネル２０２を構築することができる。ループカーネル２０２の１回の反復だけが示されているが、ループカーネル２０２は、（オリジナルコードで指定された反復数に応じて）任意の回数、実行してもよい。命令には、それらのステージ番号［０］、［１］、および［２］が注釈されており、この情報を使用して、システムは、ステージ閾値に対してステージ番号を試験することによって、ループセットアップ命令およびテアダウン命令を導出してもよい。例えば、第１の反復２０４において、システムは、［０］のステージ閾値以下のステージ番号を有する命令（すなわち、命令ａ［０］およびｂ［０］）を実行する。説明の目的で、命令には、それらが実行される回数に対応する添え字がさらに注釈されている（例えば、a_1およびb_1）が、この情報は、システムが必要とするものではない。第２の反復２０６において、システムは、増分されたステージ閾値［１］以下のステージ番号を有する命令（すなわち、命令a_2［０］、b_2［０］、およびc_1［１］）を実行する。なお、第２の反復２０６における命令c_1の第１の実行は、第１の反復２０４における命令a_1およびb_1の第１の実行の後に行われ、それによって命令間の依存関係／コンフリクトが満足されることに留意されたい。

【００３２】

この例２００においては、ループは、第３の反復２０２において定常状態に入る。ステージ閾値は、再び、［２］へ増分されて、全部の命令が実行される。その他のループは、ループの性質に応じて、より少ない、またはより多いセットアップ反復を必要とすることがあり、本発明は、いかなる特定のセットアップ（またはテアダウン）反復の回数にも限定はされない。

【００３３】

ループが最後から２番目の反復に到達すると、そのループは、ループカーネル２０２を出て、第１のテアダウン反復２０８へと入る。ステージ閾値は、２番目に低い値（すなわち［１］）に初期化されて、テアダウン部分において、ステージ閾値以上のステージ番号を有する命令（すなわち、命令c_3［１］、d_2［２］、およびe_2［２］）だけが実行される。言い換えると、この例では、命令ａおよびｂは、すでに、それらが必要とされる３回の回数、実行されており、それらの実行は、第１のテアダウン反復２０８において停止される。ステージ閾値は、再び［２］に増分されて、最終テアダウン反復２１０においては、命令d_3［２］およびe_3［２］だけが実行されて、各命令が３回実行されるという要件を満足し、同時に命令間の依存関係を尊重する。

【００３４】

本発明の別の態様において、再び図１のステップ１０４を参照すると、ループは部分的なアンローリングを必要とする。上記で考察したように、この要件は、ループ内の命令の結果の寿命が、ループの開始間隔を超えるときに行われる。上記のように、命令には、それらの依存関係に応じたステージ番号が割り当てられるとともに、命令は、その命令を走らせるハードウェアの制限に応じて、ループの各反復においてスケジューリングされる。さらに、この態様においては、長寿命命令結果がそれを使用するまで保存されることを可能にする必要があるので、ループカーネルはアンロールされる（すなわち、ループ命令は、ループカーネルにおいて複製される）。一態様において、オリジナルループカーネルの、それぞれの連続的複製において、（ｉ）命令のステージ番号が、１だけ減分され、（ｉ

ｉ）複製の（すなわち、リネームされた）レジスタが、（それによってそれらの値が互いにクロバリングするのを防止するために）ループカーネルでの長寿命命令の複製（複数を含む）に対して定義される。最終的に、（ループアンロール係数が１である）上記の例においては、命令は、ステージ閾値を１だけ増分させることによって実行するために選択されるが、ステージ閾値は、（ループの２つ以上のアンロールされた反復が、ループカーネルにおいて存在することを反映するために）ループアンロール係数だけ増分される。また、この機能を提供するその他の方法が、それに制約はされないが、マスクレジスタの使用を含む、ハードウェアにおいて可能である。本明細書における、ステージ閾値による説明は、分かり易くするために示したものである。

【００３５】

一般に、ステージ番号の範囲は、ゼロから必要な最大値（例えば、１、２、またはそれ以上）までにわたるように割り当てることができる。しかしながら、ループアンロール係数が、１よりも大きいときには、ステージ番号範囲の下限は、負の数としてもよい。一態様において、下限は（１－ループアンロール係数）に等しく、上限は（ステージ数－１）に等しい。任意の便利な範囲を使用してもよいが、本発明は、いかなる特定の範囲にも限定されない。例えば、（１－ループアンロール係数－ステージ数）から－１までの全負の範囲によって、ハードウェアの設計を簡略化してもよい。

【００３６】

前例のように、説明のための例３００によって、本発明のこの態様の説明を助けることができる。この例においては、ループは７つの反復と３つの命令ａ、ｂ、およびｃを有し、命令ｂは命令ａの出力に依存し、命令ｃは命令ｂの出力に依存する。コードを走らせるシステムは、３つの命令を同時に走らせて、ループの開始間隔を１に等しくすることができる。例えば、次のようになる。

```
for（ｉ＝１；ｉ＜＝７；ｉ＋＋）｛
    ａ；ｂ；ｃ；
｝
```

すなわち、直截的な場合においては、３つの命令ａ、ｂ、ｃは、それぞれ３つのステージ番号［０］、［１］、［３］を割り当てて、図２に示した上記の例２００と同様の方法で、ループを走らせてもよい。

【００３７】

しかしながら、この場合には、命令ｂは完了するのに２サイクルを要し、その実行時間が、ループの開始間隔（すなわち１）よりも大きくなる。したがって、ループは、この命令に対応するためにアンロールしてもよく、１つのアンロールレベル（ループアンロール係数＝２）は、より効率的な動作をもたらす（例えば、命令ｂのより長い実行時間による、パイプラインストールがないか、または少い動作）。さらに、命令ｂの出力に依存する、命令ｃには、この依存性を反映するために遅いステージ番号を割り当ててもよい。一態様においては、命令ａにはステージ［０］を割当て、命令ｂにはステージ［１］を、命令ｃには［３］を割り当てる（すなわち、命令ｂの実行時間が長いために、ステージ［２］は空である）。アンロールに追加された、ループカーネル命令の第２のコピーは、１ユニットだけ減分されたステージ番号を有する。例えば、以下のようになる。

```
for（ｉ＝１；ｉ＜＝７；ｉ＋＝２）｛
    ａ［０］；ｂ［１］；ｃ［３］；
    ａ［-１］；ｂ［０］；ｃ［２］；
｝
```

【００３８】

再び図３を参照すると、ループカーネル３０２が、ループカーネル命令ａ［０］、ｂ［１］、およびｃ［３］の第１の反復を含む、第１の部分３０４、およびステージ番号を１だけ減分された（例えば、ａ［-１］，ｂ［０］，ｃ［２］）、同一の命令を含む、第２の部分３０６と共に、示されている。上記の例のように、このループを実行するシステムは、この情報（すなわち、ループカーネル命令およびそのタグ情報）だけを渡されて、そ

れらだけに基づいて、ループセットアップ命令およびループテアアダン命令を導出してもよい。

【００３９】

すなわち、ループの第１の反復３０８において、ステージ閾値はステージ番号［０］に設定され、０以下のステージ番号を有する、ループカーネル３０２における命令が走らされる（すなわち、命令a_1［０］、a_2［－１］およびb_1［０］）。第２の反復３１０において、ステージ閾値は、ループアンロール係数２に応じて、［２］に増分され、［２］以下のステージ番号を有する命令が走らされる（すなわち、a_3［０］、b_2［１］、a_4［０］、b_3［０］およびc_1［２］）。すなわち、第２の反復３１０において、命令a_3［０］およびb_2［１］は、ループの第１のアンロールインスタンスにおいて走らされて、命令a_4［-１］、b_3［０］およびc_1［２］は、ループの第２のアンロールインスタンスにおいて走らされる。命令ｂは、２サイクルの寿命を有するので、命令b_3［０］の結果は、先に実行された命令b_2［１］の結果に上書きするのを避けるために、リネームされたレジスタに記憶される。

【００４０】

上記のように、第２の反復３１０の命令が実行されて、（かつ、命令b_3［０］の場合には、まだ実行中であると）、ループカーネル３０２の命令の実行が始まる。一態様においては、ステージ閾値は、再び、２だけ（すなわち、ループアンロール係数に等しい量だけ）増分して［４］にされて、［４］は最大ステージ番号よりも大きいので、ループ内のすべての命令が実行される。ここでも、ループカーネル３０２の１つのコピーだけが図示されているが、これは、ループの定常状態部分であり、（オリジナルソースコードに指定されたループパラメータに応じて、）任意の回数、実行してもよい。

【００４１】

ループの最後から２番目の反復に到達すると、第１のテアダウン反復３１２が始まる。この反復においては、ステージ閾値は、２番目に低いステージ番号（すなわち、［０］）に初期化され、ゼロ以上のステージ番号を有する命令（すなわち、a_6［０］、b_6［１］、c_4［３］、b_7［０］およびc_5［２］）が実行される。最終的に、最終テアダウン反復３１４において、再び増分されたステージ閾値（すなわち［２］）以上のステージ番号を有する命令が、実行される。上記の例と同様に、各命令ａ、ｂ、ｃは、オリジナルループ反復回数に応じて７回、実行され、それぞれの依存関係が尊重される（すなわち、命令ｂの各インスタンスは、対応する命令ａの実行より１クロックサイクル後に実行され、同様に、命令ｃは、命令ｂの２サイクル後に実行される）。

【００４２】

本発明の別の態様においては、図１のステップ１０８を参照すると、ループの反復数は、コンパイル時には既知ではないことがある。図２に示された例２００におけるように、ループアンロール係数が１である場合には、反復回数における不確定性は問題とはならず、ループの定常状態部分は、実行時反復回数に応じて、必要な回数だけ単に繰り返えされる。しかしながら、ループアンロール係数が１より大きい場合には、必要な反復数を生成するのはより困難である。ループカーネルにおける部分的にアンロールされたループは、２つ以上の反復を内蔵しているので、これらの２つ以上のアンロール反復の倍数ではない、合計ループ反復回数は、構築するのが困難であろう。

【００４３】

例として、図３に示されたループ３００は、（ループアンロール係数２を有する、）部分的にアンロールされたカーネル３０２を有する。その簡単な例においては、カーネル３０２は、１回だけ実行され、ループの合計反復数は７（すなわち、各命令は７回、実行される）である。合計反復数が９、１１、または１３などである場合には、ループカーネル３０２は、２、３、または４回繰り返されて、必要な合計反復数を生成する。しかしながら、所望の反復の合計数が８である場合に、図３のループ構造は、修正なしにその数を生成することはできない（すなわち、ループ３０２の１回の反復は少なすぎるが、２回の反復は多すぎる）。

【００４４】
一態様において、ループセットアップの第１の反復を構成する命令は、所望の合計反復数を生成するために、必要に応じて、１回または２回以上の開始間隔だけ遅延される。ソースコードにおいてオリジナルループによってコールされる反復数はＮであり、本発明の態様により構築されるループの反復数は（Ｎ＋［ステージ数－１］）であり、この場合に、（ステージ数－１）の追加反復は、ループセットアップおよびテアダウンの反復に使用することができる。（Ｎ＋［ステージ数－１］）がループアンロール係数の整数倍数でない場合には、そうなるまで切り上げられる（すなわち、ループを、少なくともう１反復、走らせるように設定される）。この場合には、ステージ閾値の初期値は、ゼロより低く低減されて、その結果として、パイプライン化されたループに入ると、アンロールされたループの初期部分はオフにされる。

【００４５】
ここで再び、例をあげるのが役立つ。７回の合計反復を走らせるように構成された、図３のループ３００が、図４の図解４００に示されるように、８回の反復を走らせるように修正されている。普通は、ループの合計９回の反復を生成することになる、ループカーネル４０２が、２回実行されるように複製されている。しかしながら、この場合に、ステージ閾値の初期値が［－１］に設定されている。このラインにおけるすべての命令が、初期ステージ閾値よりも大きいステージ番号を有するので、したがって、ループプロログの第１のサイクル４０４は動作不能にされる。結果として、命令ａ、ｂ、ｃのそれぞれは、（９回ではなく）８回実行される。

【００４６】
４０４において指示される遅延の別の利点には、ループアンローリングにおいて必要なレジスタリネーミングが含まれる。パイプライン化されたループから外に出ると、ループ中に演算されて、その外部で使用される値は、正しい値を見出すことを可能にするために、既知のレジスタに記憶させなくてはならない。しかしながら、ループアンローリング中に、長寿命を有する結果のレジスタリネーミングの行為は、２つ以上のレジスタを生成し、その中では、ループの間中、その値はライブ状態である。ループカーネル４０２が、その実行の途中で出ることができる場合には（または、その実行が完全に終わる以外のある点において、２より大きいループアンローイング係数を有するループの場合には）、ループの外部で使用される結果の最終値は、これらのリネーミングされたレジスタのいずれかに保持してもよく、すなわち、ループトリップ数がコンパイル時に未知の場合には、どのレジスタがその最終結果を保持するかは分からない。ループプロログ４０４の始めに遅延４０４を設置することによって、カーネル４０２は、その実行を完了した後に出ることによって、どのレジスタがループにおける結果の最終値を保持しているかが常に確実にわかるようになる。カーネル４０２に入ると、必要であれば、リネーミングされたレジスタのすべてのコピーが、正しい初期値に初期化されることを確実にするのは容易である。

【００４７】
一態様においては、２つ以上の処理要素がコードの実行に利用可能であり、コンパイラは、データ独立性命令を各処理要素に送る。ループの反復数が、処理要素の数の倍数でない場合には、処理要素の一部は、ループの最後の反復においてアイドル状態になる（すなわち、その動作がマスキングにより無効にされる）ことがある。２処理要素システム上で実行される７反復ループの例示態様５００が、図５に示されている。このコードは、以下の依存関係を示すであろう：

```
for（i＝1；i＜＝7；i＋＋）{
    a［1］；c［0］；
    b［0］；d［1］；
}
```

ここで７は、２の偶数倍数ではないので、ループの４回の反復が、第１の処理要素５０２に送られて、３つの命令が第２の処理要素５０４に送られる。このステージ［０］命令は、ステージ閾値に応じて、第１のステージ５０６において実行を始め、ループカーネル５

０８は、後続のステージにおいて実行される。カーネル５０８は、最初に第２のプロセッサ５０４上に出て、その間に、追加の反復を第１のプロセッサ５０２上で走らせる。両方のプロセッサ５０２、５０４は、最終反復５１０においてステージ［１］を実行する。

【００４８】

ループ命令にステージ番号を注釈付けする、システム６００の一態様が、図６に示される。ソースコード分析モジュール６０２は、命令レベルまたはデータレベルの並列性を示す、候補命令（例えば、ループ命令）に対するソースコードを分析する。（モジュールは、通常、ソフトウェア形態で、すなわちプロセッサによる実行のために非揮発性メモリに記憶されたコンピュータ実行可能な命令のセットとして実現される。プログラムモジュールとしては、特定のタスクを実行するか、特定の抽象データタイプを実装する、ルーチン、プログラム、オブジェクト、構成要素、および／またはデータ構造、その他があげられる。しかしながら、当該技術において周知のように、機能モジュールも、特定のアプリケーションの要件に応じて、ハードウェアに実装するか、またはソフトウェア／ハードウェア混合構成要素として実装してもよい。）

【００４９】

ソースコード分析モジュール６０２は、ソースコード内のこれらの命令を、パターンマッチング、構文解析、部分（または全体）コンパイル、または当該技術において知られているコードを分析するその他任意の方法に基づいて識別してもよい。任意の種類のソースコードが、本発明の範囲内である。識別がされると、本発明の態様によれば、ステージ番号発生モジュール６０４が、命令間依存関係、コードを走らせるハードウェアの制限または特徴、あるいはその他任意の関係する要因に基づいて、ステージ番号を命令に割り当てる。モジュール６０２、６０４の一方または両方が、開始間隔、ステージ数、および／またはループアンロール係数などのループに関係する特性を判定して、これらの特性の一部または全部を、ステージ番号を命令に割り当てる際に使用してもよい。ステージ番号が、命令の一部または全部に対して判定されると、アセンブリコードまたはオブジェクトコード出力モジュール６０６が、１つまたは２つ以上のターゲットプロセッサ上で実行するのに適した、（ステージ番号を含む）アセンブリコードまたはオブジェクトコードを、準備、フォーマット、かつ出力してもよい。しかしながら、本発明は、これらのモジュール６０２、６０４、６０６だけを含む、実装に限定されることはなく、当業者は、モジュール６０２、６０４、６０６に実装された機能は、より多数または少数のモジュールに実装してもよいことを理解するであろう。

【００５０】

ステージ番号注釈を有するコードを実行するシステム７００が、図７に示されている。シーケンサ７０２は、状態情報を包含し、１つまたは複数の処理要素（ＰＥ）７０４に命令を発行する。シーケンサ７０２は、ループアンロール係数、ループ内に残る反復数（すなわち、「ループ数」）、ループの各反復においてどのステージ番号を実行すべきか、および複数の処理要素７０４の内のどれが、ループの最後の反復において命令を実行すべきか、などの現在実行中のループに関する情報を記憶してもよい。当業者は、シーケンサ７０２を（または、その類似の機能を別の機能ブロック中に）実装する多くの方法があり、本発明はいずれの特定の実装形態に限定されないことを認識するであろう。さらに、処理要素７０４は、シーケンサによって制御される、任意のタイプの汎用または専用コンピューティングブロックとしてもよい。さらに、本発明はまた、この意味で別個の処理要素内に存在するかのように取り扱うことのできる、単一のコンピューティングユニット内で処理されるベクトル内の独立した要素の制御にも適用されることも理解されたい。そのようなマシンは、通常、単一命令多重データ（ＳＩＭＤ）プロセッサと呼ばれる。

【００５１】

シーケンサ７０２（より具体的には、それを実装するハードウェア）は、ループ数を記憶するループ数（ＬＣ）レジスタ７０６や、（モジュロ可変拡張、または「ＭＶＥ」アンロール係数としても知られている）ループアンロール数を記憶するループアンロール係数レジスタ７０８などの、状態情報を記憶するためのレジスタを含む。ステージマスクレジ

スタ７１０には、（そのステージ番号に基づいて）どの反復においてどの命令を実行すべきかに関する情報が場所を占めており、例えば、ステージマスクレジスタ７１０の１ビットが、ステージ［０］の命令が実行されるべきことを指示し、別のビットが、ステージ［１］の命令が実行されるべきことを指示し、また以下同様にしてもよい。ステージマスクレジスタは、（各処理要素７９４が、各反復において異なるステージ番号を有する命令を実行してもよいことを説明するために）各処理要素７０４において、ローカルステージマスクレジスタ７１４内に再生してもよい。最終反復マスク（ＦＭＡＳＫ）レジスタ７１２は、ループの最終反復において、複数の処理要素７０４の内のどれを、オンまたはオフにすべきかを指示するのに使用してもよい。

【００５２】

ループ数レジスタ７０６は、サイズが８ビット、１６ビット、またはその他のビット数であってもよく、ループ数レジスタ７０６に記憶された値は、ループが反復する毎に（例えば、ループアンロール係数だけ）減少する。ループ数レジスタ７０６内の値が、ゼロに達すると、ループは、反復を停止する。ループアンロール係数レジスタ７０８は、サイズを２ビットとして、最大アンロール係数４をサポートとするか、サイズを３ビットとして最大アンロール係数８をサポートするか、またはその他のサイズとしてもよい。ステージマスクレジスタ７１０は、少なくとも、ループアンロール係数と最大ステージ数の合計に等しいサイズにしてもよい。一態様において、ステージマスクレジスタ７１０は、サイズが２３ビットである。最終反復マスクレジスタ７１２は、処理要素７０４のそれぞれに対して１ビットを有してもよい。

【００５３】

ループ命令を、シーケンサ７０２内の、状態レジスタ７０６、７０８、７１０、７１２を初期化するのに使用してもよい。このループ命令には、合計ループ数、使用する処理要素の数、ステージ数、およびループアンロール係数を指示するパラメータを含めてもよい。ループアンロール係数レジスタ７０８には、ループ命令において与えられる、ループアンロール係数の値をロードしてもよく、その他のレジスタ７０６、７１０、７１２の値は、以下に示すように、式（１）～（３）に従って、ループ命令パラメータから導出してもよい。

【数１】

$$\text{ループ数レジスタ７０６} = \left\lceil \frac{\left\lceil \frac{N}{VLEN} \right\rceil + SC - 1}{MVE} \right\rceil \times MVE \qquad (1)$$

【数２】

$$\text{ステージマスクレジスタ７１０} = \left((-1) << \left(SC + \left(LC - \left(\left\lceil \frac{N}{VLEN} \right\rceil + SC - 1 \right) - 1 \right) \right) \right) \qquad (2)$$

【数３】

$$\text{最終反復マスクレジスタ７１２} = \left(1 << \left(\left((N-1)\%VLEN \right) + 1 \right) \right) - 1 \qquad (3)$$

【００５４】

式（１）に関して、ループ数は、命令を並列に実行する処理要素７０４の数（ＶＬＥＮまたはベクトル長）で除算した、オリジナルコードにおける反復数（Ｎ）に関係する。追

加の反復（ＳＣ－１）が、ループセットアップおよびテアダウンに必要である。結果は、ループアンロール係数（ＭＶＥ）の最も近い倍数に切り上げられる。式（２）に関して、バイナリの１は、ステージ数（ＳＣ）に応じて、ステージマスクレジスタ７１０中へとシフトされ、ループ数レジスタ７０６（ＬＣ）において演算されるループアンロール係数（ＭＶＥ）によって必要となる切り上げが考慮に入れられる。式（３）に関して、処理要素７０４の数（ＶＬＥＮ）を法（modulo）とする、反復数（Ｎ）の剰余に等しい、最終反復マスクレジスタ７１２中に２値の１がシフトされて、最後の反復において不必要な処理要素７０４をマスキングで無効にする（すなわち、モジュロ演算は、最後の反復における「残存」命令の数を与える）。例えば、Ｎ＝１３かつＶＬＥＮ＝８であると、ＦＭＡＳＫ＝０００１１１１１となり、一方でＮ＝１６であれば、ＦＭＡＳＫ＝１１１１１１１１となる。最終的に、ループ数レジスタ７０６の値が、ステージマスクレジスタ７１０内のビット数以下である（すなわち、ループの最後の数回の反復が接近している）場合には、ステージマスクレジスタ７１０内の（ＬＣ－１）番目ビットの上で、かつそれを含む任意のビットがクリアされて、最終反復マスクレジスタ７１２からの値が、ステージマスクレジスタ７１０における（ＬＣ－１）番目ビット位置に再配置するのに使用される。

【００５５】

ステージマスクレジスタ７１０の再演算されたコンテンツは、次いで、各処理要素７０４におけるレジスタのローカルコピー７１４に押し出され、ここで（ＬＣ－１）番目ビット位置は、それに到達すると、最後の反復中に、一部の処理要素７０４のスイッチをオフにさせる。各処理要素７０４は、命令を受け取り、デコードして、各命令のステージ番号を検査し、その処理要素のローカルマスクレジスタ７１４における、それに対応するビットエントリが設定されている場合には、命令が実行される。一態様において、ローカルマスクレジスタ７１４内の対応するステージ番号のビット位置が、コンパイル時またアセンブリ時に演算されて、命令内にコード化され、それによって値を演算および／または記憶することからハードウェアを解放する。

【００５６】

一例において、ループ命令は、ループ数が５２３であり、処理要素の数（ＶＬＥＮ）が４であり、ステージ数が３であり、ループアンロール（ＭＶＥ）係数が２であることを規定する。この例においては、ループ数レジスタ７０６は、１３４の値を受け取り、ループアンロール係数レジスタ７０８は、２の値を受け取り、ステージマスクレジスタ７１０は、（すべてのローカルレジスタ７１４と同様に）１１…１１０００の値を受け取り、最終反復マスクレジスタ７１２は、０１１１の値を受け取る。

【００５７】

状態レジスタ７０６および７１０は、ループが反復する度毎に更新してもよい。一態様においては、ループ数レジスタ７０６は、ループアンロールレジスタ７０８の値だけ減分され、状態マスクレジスタ７１０は、ループアンロール係数レジスタ７０８の値に等しい回数だけ、右にシフトされる。ループが、その最後の数回の反復に近づいている場合（例えば、ループ数レジスタ７０６の値が、ステージマスクレジスタ７１０におけるビット数以下である場合）には、ゼロがシフトインされ、そうでない場合には、１がシフトインされる。上述のように、ループがその最後の（または最後の数回の）反復中である場合には、最終反復マスクレジスタ７１２の値が、（その（ＬＣ－１）番目ビット位置において）ローカルマスクレジスタ７１４にコピーされる。

【００５８】

例えば、最後の例において紹介された、ループの１回の反復の後に、ループ数レジスタ７０６は、１３２の値に更新されて、ステージマスクレジスタ７１０は、（すべてのローカルレジスタ７１４がそうであるように）１１…１１１１０の値に更新される。その寿命の終端に近いが、終端ではないループの別の反復において、ループ数レジスタ７０６は４の値に更新され、ステージマスクレジスタ７１０は、００…００１１１の値に更新される。ローカルレジスタ７１４は、最終反復マスクレジスタ７１２に応じて、異なる値、すなわち００…１１１１、００…１１１１、００…１１１１、および００…０１１１を有する

ように更新される。ループの最終反復において、ループ数レジスタ７０６は、２の値に更新されて、ステージマスクレジスタ７１０は、００…００１の値に更新される。ローカルレジスタ７１４は、最終反復マスクレジスタ７１２に応じて、異なる値、すなわち００…００１１、００…００１１、００…００１１、００…０００１を有するように、再び、更新される。

【００５９】

本発明の態様によるループ命令を実行する方法８００が、図８に示されている。第１のステップ８０２において、ループ命令が、走らせようとするループについての情報（例えば、ループアンロール係数および反復数）とともに、受け取られる。（例えば、シーケンサ内の）状態レジスタが、この情報（および／またはそれから導出される値）によって初期化される。第２のステップ８０４において、マスクが、１つまたは２つ以上の処理要素に対して演算されて、それに送られ、このマスク（例えば、図７を参照して上述されたステージマスク）は、命令のどのステージを実行するかについての情報を包含する。第３のステップ８０６において、ループ命令は、そのステージ番号とマスクに応じて実行される。第４のステップ８０８において、マスク（および状態情報）は折り返し時に更新される。処理要素を制御するビットを、マスクレジスタ内、またはその他の方法で編成することのできる多くの方法があり、本発明は、シーケンサ内部に保持されるグローバルマスクレジスタ、および各処理要素内部のローカルマスクレジスタとして説明した実装形態に限定されないことを、当業者は認識するであろう。例えば、すべての必要な状態をシーケンサ内部に保持して、どの命令が現在、各処理要素に対して動作可能にされているかに応じて、命令を処理要素に分散させることが同等に可能である。

【００６０】

なお、本発明の態様は、１つまたは２つ以上の製造物上、またはその中に具現化される、１つまたは２つ以上のコンピュータ可読プログラムとして、提供することもできることに留意されたい。製造物は、例えば、フレキシブルディスク、ハードディスク、ＣＤＲＯＭ、ＣＤ－ＲＷ、ＣＤ－Ｒ、ＤＶＤＲＯＭ、ＤＶＤ－ＲＷ、ＤＶＤ－Ｒ、フラッシュメモリカード、ＰＲＯＭ、ＲＡＭ、ＲＯＭ、または磁気テープなどの、任意好適なハードウェア装置としてもよい。一般に、コンピュータ可読プログラムは、任意のプログラム言語で実装してもよい。使用することのできる言語の例としては、Ｃ、Ｃ＋＋、またはＪＡＶＡ（登録商標）があげられる。ソフトウェアプログラムは、機械言語または仮想マシン命令にさらに変換して、その形態でプログラムファイルに記憶させてもよい。次いで、このプログラムファイルを、１つまたは２つ以上の製造物上、またはその中に記憶させてもよい。

【００６１】

本発明のいくつかの態様を上記で説明した。しかしながら、本発明はそれらの態様に限定されることはなく、むしろ本明細書において明白に記載された内容に対する追加および修正も、本発明の範囲に含めることを意図するものであることに、特に留意されたい。さらに、本明細書に記載された様々な態様の特徴は、互いに排他的ではなく、様々な組合せおよび順列で存在することが、たとえそのような組合せまたは順列が本明細書において明白にされていなくても、本発明の趣旨と範囲から逸脱することなく、可能であることを理解されたい。実際に、本明細書において記載されたものの変形形態、修正形態、およびその他の実装形態は、本発明の趣旨と範囲から逸脱することなく、当業者は思い付くであろう。したがって、本発明は、前述の例証のための説明だけによって定義されるものではない。

【図１】

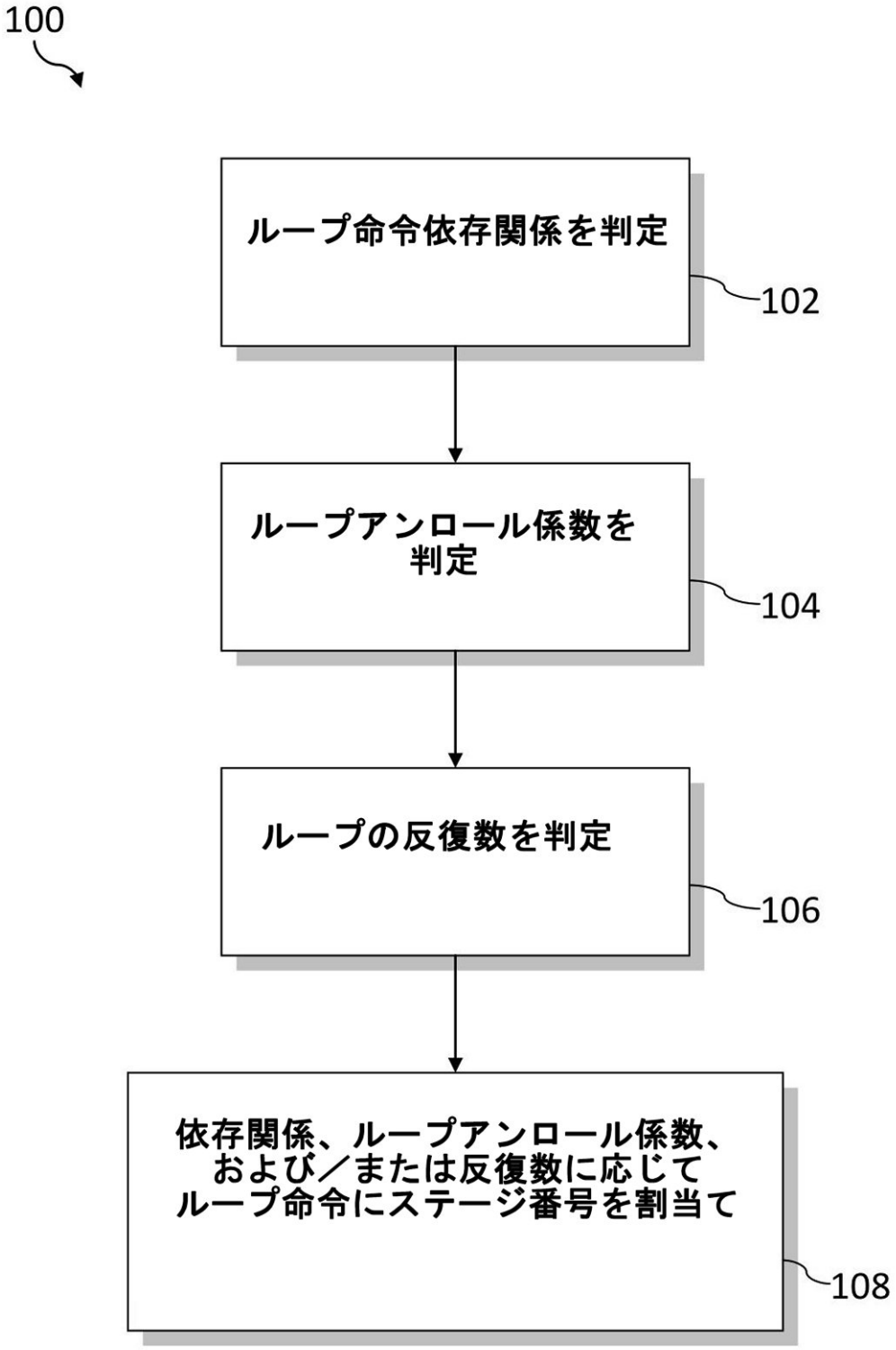
100
ループ命令依存関係を判定
102
ループアンロール係数を
判定
104
ループの反復数を判定
106
依存関係、ループアンロール係数、
および／または反復数に応じて
ループ命令にステージ番号を割当て
108

図 1

【図２】

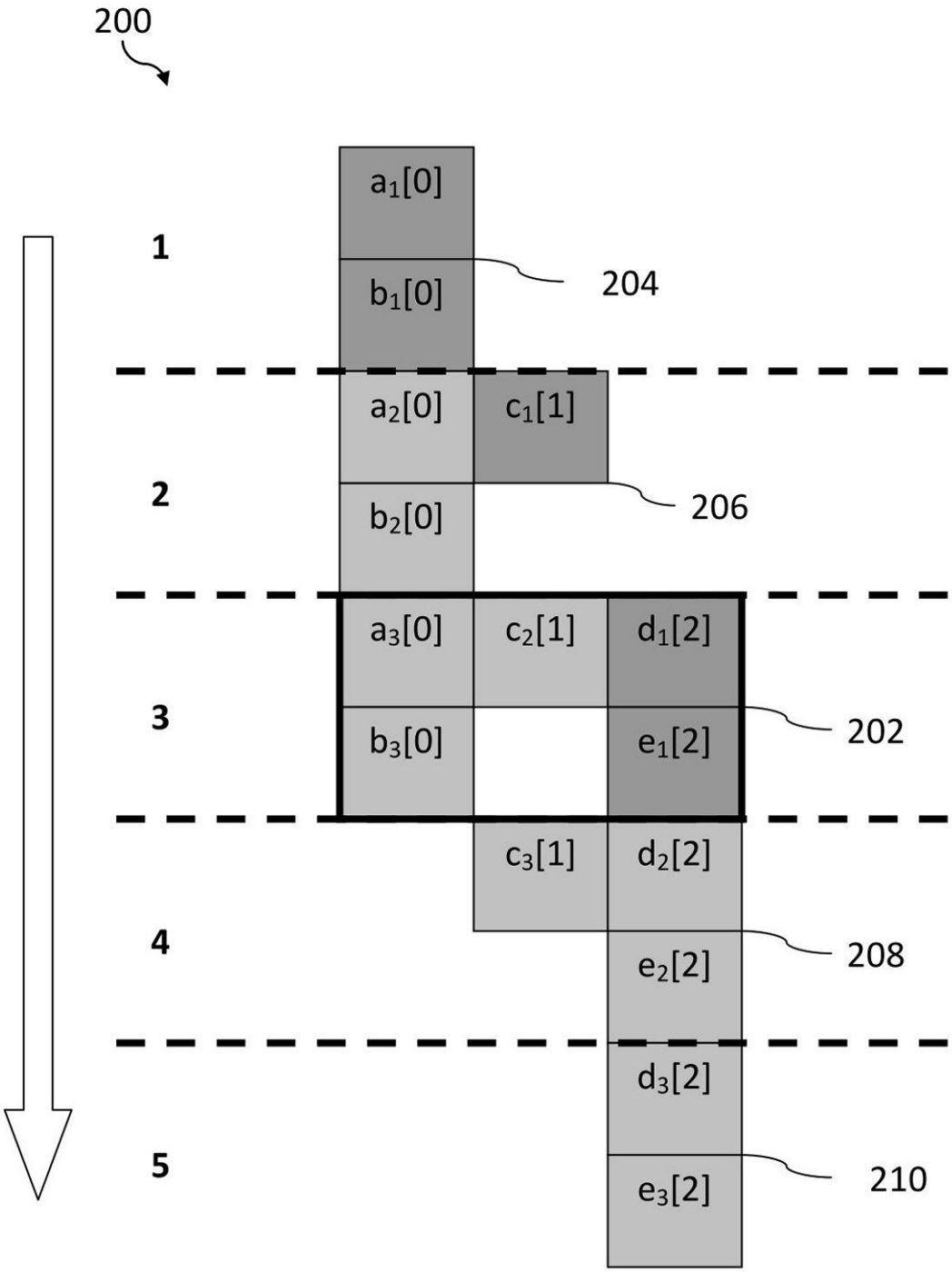
200
1
2
3
4
5
$a_1[0]$
$b_1[0]$
204
$a_2[0]$
$c_1[1]$
$b_2[0]$
206
$a_3[0]$
$c_2[1]$
$d_1[2]$
$b_3[0]$
$e_1[2]$
202
$c_3[1]$
$d_2[2]$
$e_2[2]$
208
$d_3[2]$
$e_3[2]$
210

図 2

【図３】

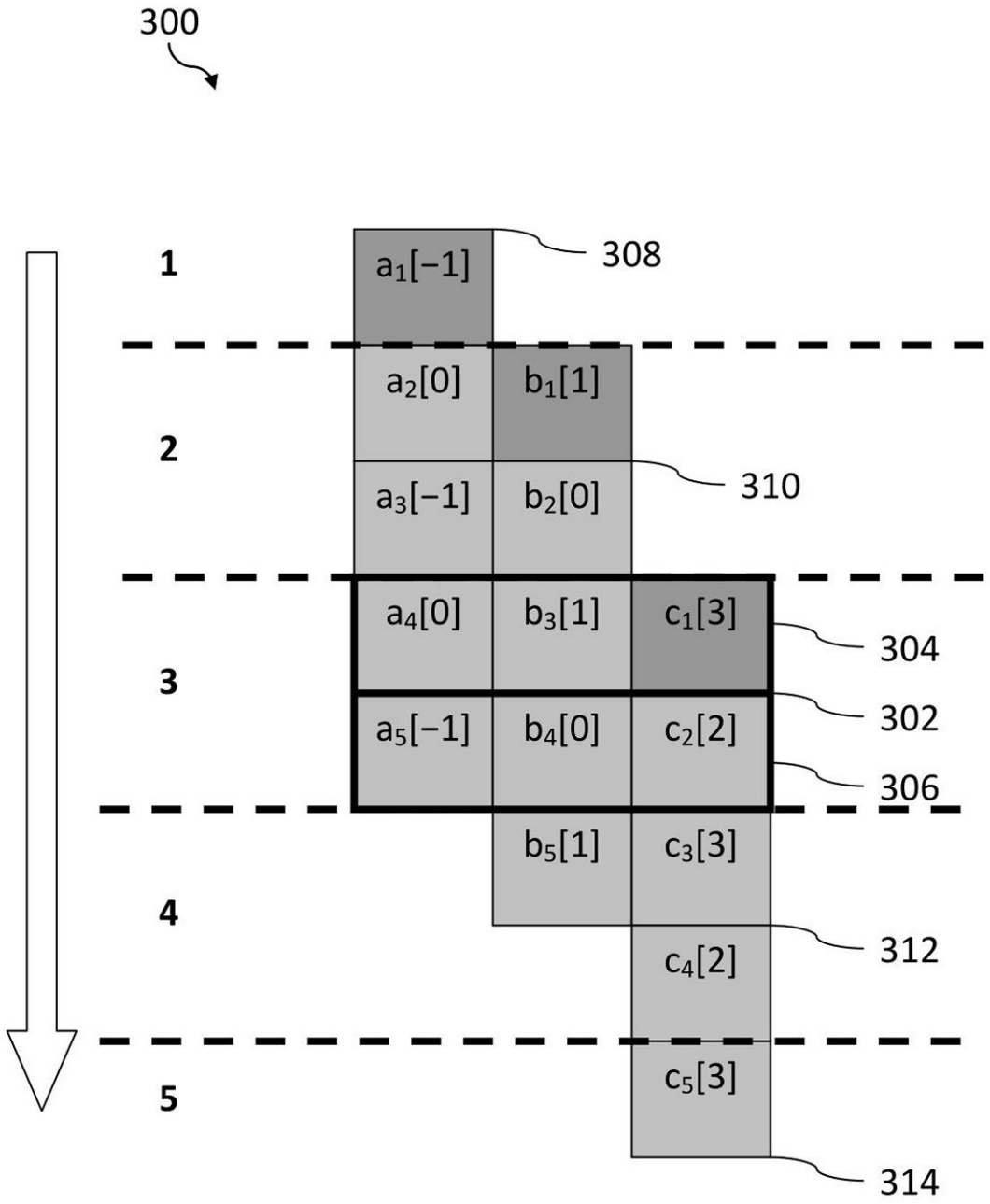
300
1
2
3
4
5
$a_1[-1]$
$a_2[0]$
$b_1[1]$
$a_3[-1]$
$b_2[0]$
$a_4[0]$
$b_3[1]$
$c_1[3]$
$a_5[-1]$
$b_4[0]$
$c_2[2]$
$b_5[1]$
$c_3[3]$
$c_4[2]$
$c_5[3]$
308
310
304
302
306
312
314

図 3

【図４】

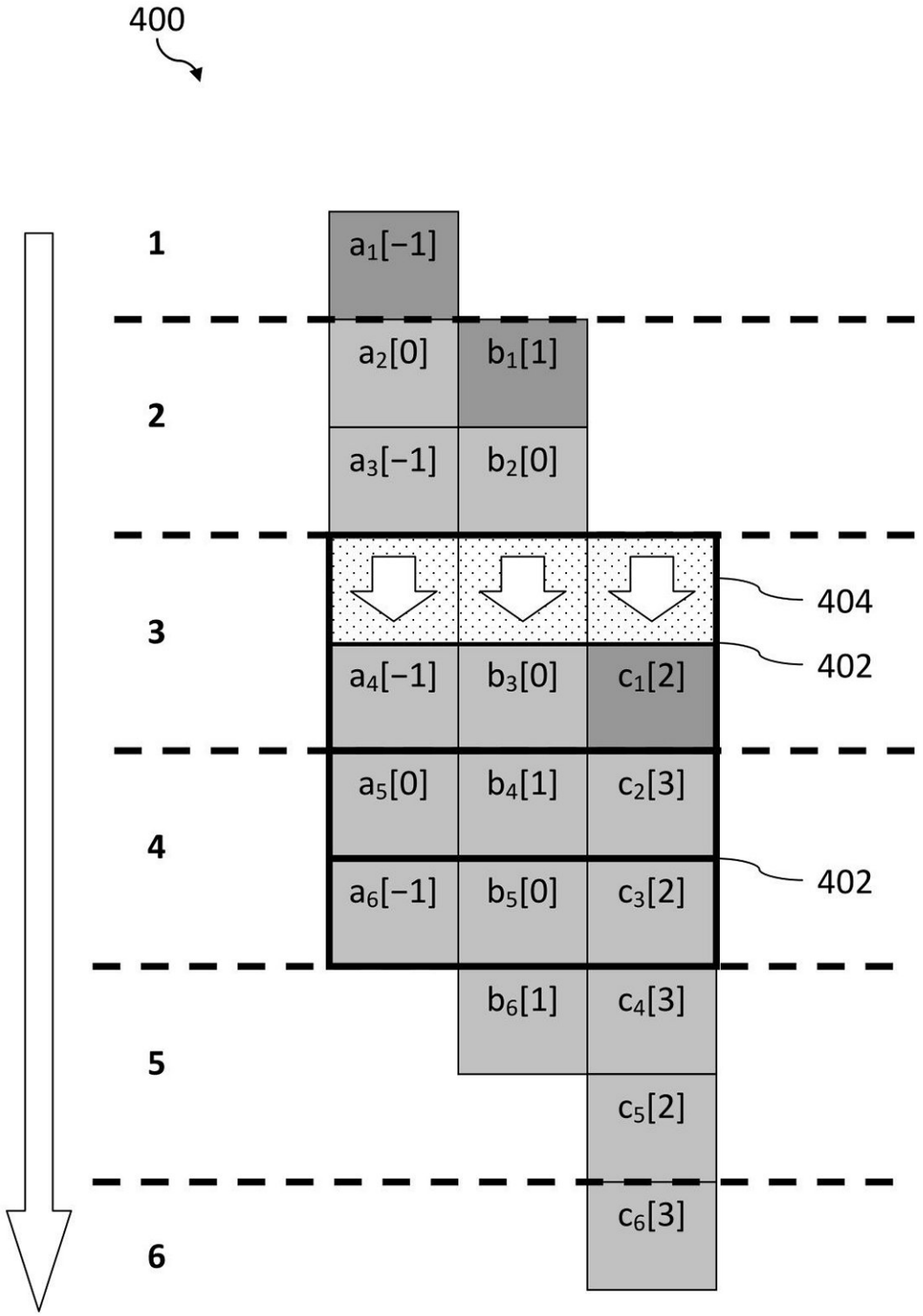
400
1
2
3
4
5
6
$a_1[-1]$
$a_2[0]$
$b_1[1]$
$a_3[-1]$
$b_2[0]$
$a_4[-1]$
$b_3[0]$
$c_1[2]$
$a_5[0]$
$b_4[1]$
$c_2[3]$
$a_6[-1]$
$b_5[0]$
$c_3[2]$
$b_6[1]$
$c_4[3]$
$c_5[2]$
$c_6[3]$
404
402
402

図 4

【図５】

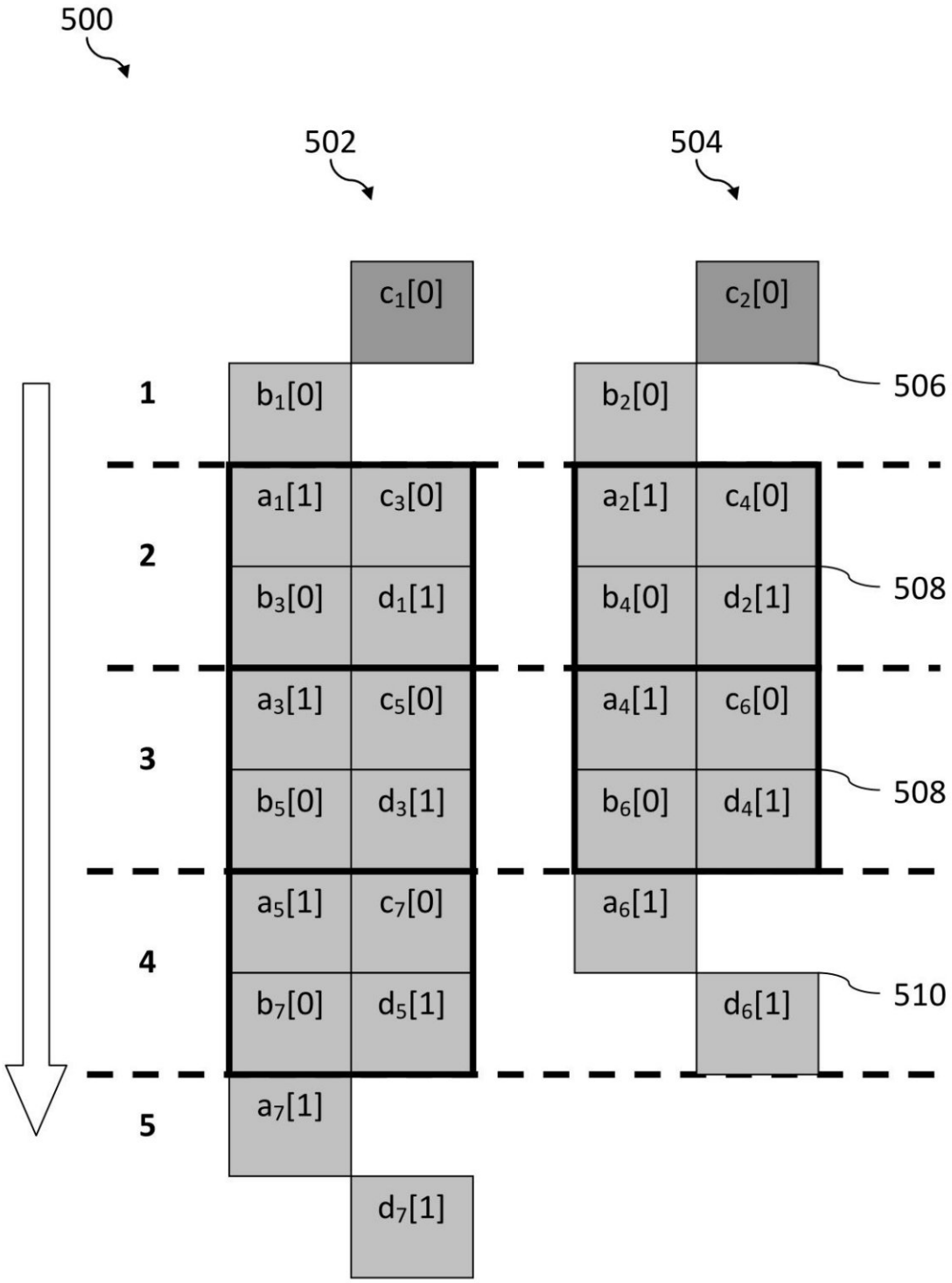
500
502
504
$c_1[0]$
$c_2[0]$
1
$b_1[0]$
$b_2[0]$
506
$a_1[1]$
$c_3[0]$
$a_2[1]$
$c_4[0]$
2
$b_3[0]$
$d_1[1]$
$b_4[0]$
$d_2[1]$
508
$a_3[1]$
$c_5[0]$
$a_4[1]$
$c_6[0]$
3
$b_5[0]$
$d_3[1]$
$b_6[0]$
$d_4[1]$
508
$a_5[1]$
$c_7[0]$
$a_6[1]$
4
$b_7[0]$
$d_5[1]$
$d_6[1]$
510
5
$a_7[1]$
$d_7[1]$

図 5

【図６】

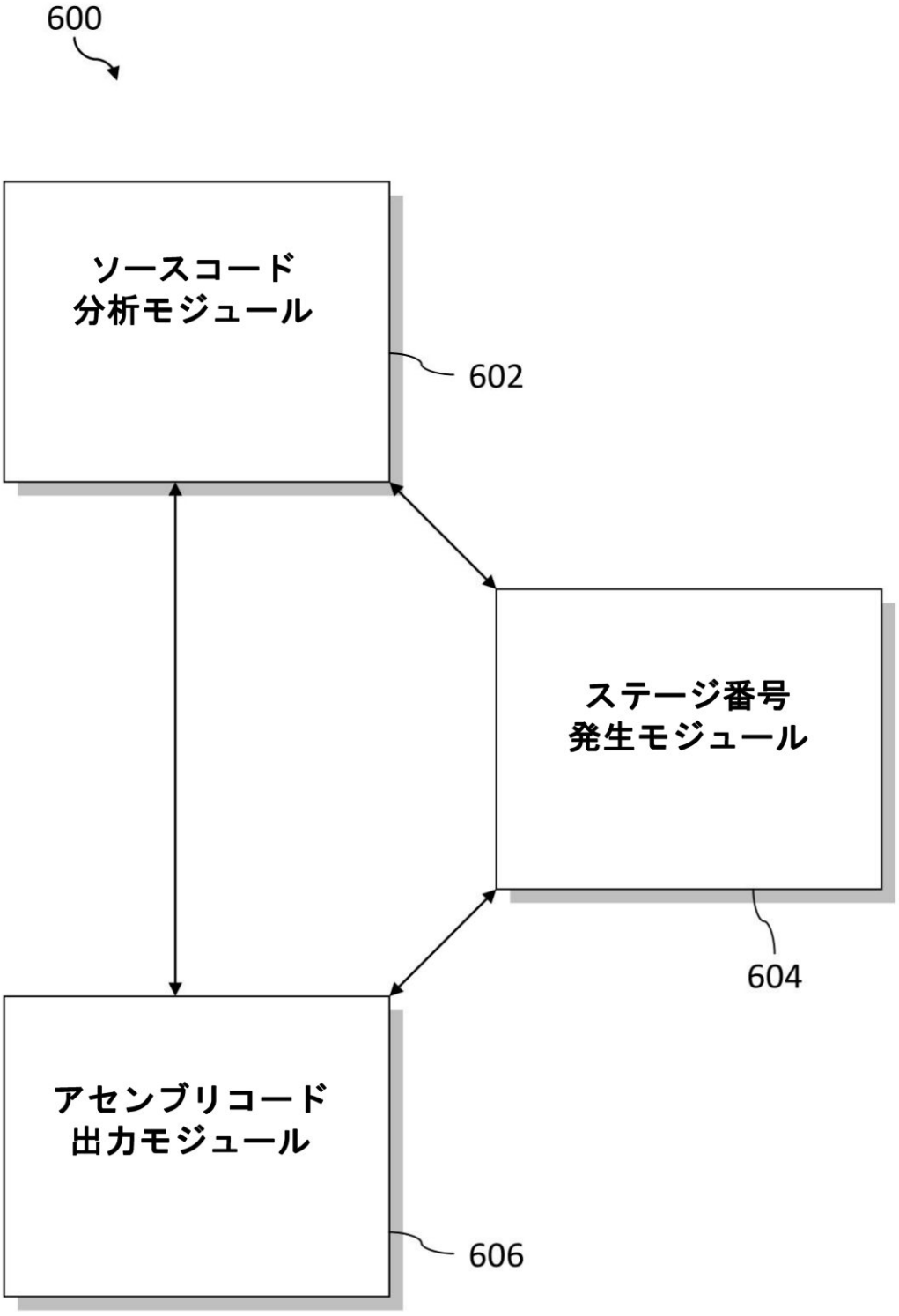
600
ソースコード
分析モジュール
602
ステージ番号
発生モジュール
604
アセンブリコード
出力モジュール
606

図 6

【図７】

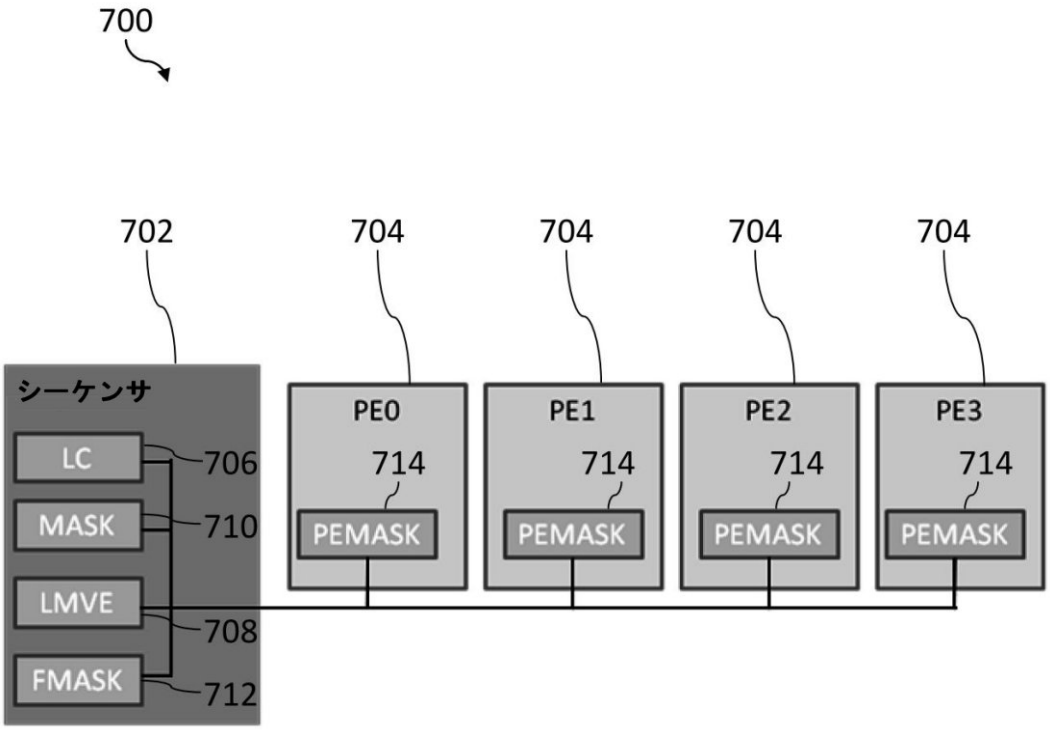
700
702
704
704
704
704
シーケンサ
LC
MASK
LMVE
FMASK
706
710
708
712
PE0
PE1
PE2
PE3
714
714
714
714
PEMASK
PEMASK
PEMASK
PEMASK

図 7

【図８】

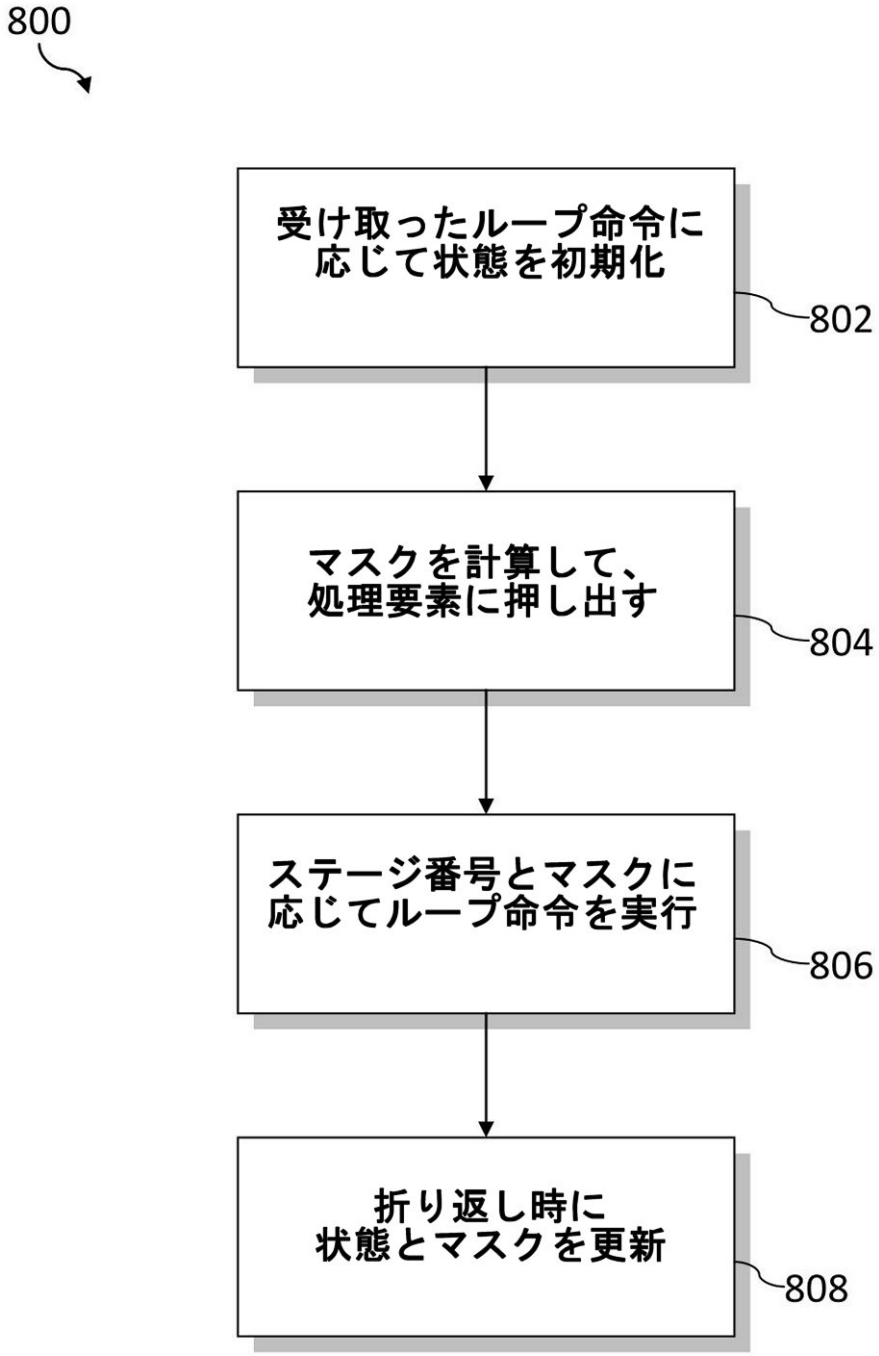
800
受け取ったループ命令に
応じて状態を初期化
802
マスクを計算して、
処理要素に押し出す
804
ステージ番号とマスクに
応じてループ命令を実行
806
折り返し時に
状態とマスクを更新
808

図 8

フロントページの続き

(72)発明者　ハイアム，アンドリュー　ジェイ．
イギリス国　イーエイチ９　１イーエヌ　エディンバラ、ウォレンダー　パーク　ロード　１０９
フラット　３　エフ１

Ｆターム(参考) 5B033 CA11
5B081 CC21

【外国語明細書】
2014010832000001.pdf