(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第6236443号
(P6236443)

(45)発行日　平成29年11月22日(2017.11.22)　(24)登録日　平成29年11月2日(2017.11.2)

(51)Int.Cl.		F I		
G06F 17/16	(2006.01)	G06F	17/16	H
G06F 9/38	(2006.01)	G06F	9/38	310E
G06F 9/30	(2006.01)	G06F	9/38	310G
		G06F	9/30	330C

請求項の数 21　(全 24 頁)

(21)出願番号　特願2015-521057(P2015-521057)
(86)(22)出願日　平成25年6月11日(2013.6.11)
(65)公表番号　特表2015-522196(P2015-522196A)
(43)公表日　平成27年8月3日(2015.8.3)
(86)国際出願番号　PCT/GB2013/051530
(87)国際公開番号　WO2014/009689
(87)国際公開日　平成26年1月16日(2014.1.16)
審査請求日　平成28年6月3日(2016.6.3)
(31)優先権主張番号　13/546,227
(32)優先日　平成24年7月11日(2012.7.11)
(33)優先権主張国　米国(US)

(73)特許権者　594154428
エイアールエム　リミテッド
イギリス国　シービー１　９エヌジェイ　ケンブリッジ，チェリー　ヒントン，フルバーン　ロード　１１０
(74)代理人　110000855
特許業務法人浅村特許事務所
(72)発明者　リード、アラステア　デイビッド
イギリス国、ケンブリッジ、チェリー　ヒントン、フルボーン　ロード　１１０、エイアールエム　リミテッド　気付

審査官　田中　幸雄

最終頁に続く

(54)【発明の名称】ベクトル処理中のデータ要素処理のための順序制御

(57)【特許請求の範囲】

【請求項１】

ベクトルに関する演算を実行するためのベクトル命令のストリームを処理するためのデータ処理装置であって、前記ベクトルはそれぞれ複数のデータ要素を含み、前記データ処理装置は、

処理される前記ベクトルを記憶するための複数のレジスタを備えるレジスタ・バンクと、

前記ベクトル命令のストリームを処理するためのパイプライン化プロセッサとを備え、

前記パイプライン化プロセッサは、前記ベクトル命令のストリームによって処理される前記ベクトルであって前記複数のレジスタ内に記憶される前記ベクトルのためのデータの依存関係を検出し、且つレジスタ・データ・ハザードが生じないように前記ベクトル命令のための実行のタイミングに関する制約を決定するように構成され、前記レジスタ・データ・ハザードは、前記ベクトル命令のストリーム内で後に生じるアクセスが、前記ベクトル命令のストリーム内で先に生じるアクセスが完了する前に開始するように、少なくとも一方が書込みである、同じレジスタへの２つのアクセスが、前記ベクトル命令のストリームの順序と異なる順序において生じる場合に生じ、

前記パイプライン化プロセッサはデータ要素ハザード判断回路を備え、前記データ要素ハザード判断回路は、データの依存関係が識別されたベクトル内の前記データ要素の少なくとも幾つかに関して、前記データ要素の前記少なくとも幾つかの場合にそれぞれ、前記ベクトルに対して識別された前記データの依存関係があるか否かを判断し、依存性がない

場合には、前記データ要素を処理する命令のための実行のタイミングに関する前記決定された制約を緩和するように構成される、データ処理装置。

【請求項２】

所定のアクセス順序においてベクトル内のデータ要素にアクセスするように構成され、前記データ要素ハザード判断回路は、ベクトル命令に応答してベクトル内で少なくとも１つのデータ要素が更新されることになるとき、前記ベクトル内の前記所定のアクセス順序における後続のデータ要素が、変更される場合があり、破棄することができる値を現在含むことを判断し、且つ前記後続のデータ要素のうちの少なくとも１つを更新する少なくとも１つの更なるベクトル命令に応答して、少なくとも１つの演算を、前記データ要素を更新する前記演算に対して任意の順序において実行できることを実行回路に知らせるように構成される、請求項１に記載のデータ処理装置。

【請求項３】

前記ベクトルは複数の区画に分割され、各区画は少なくとも１つのデータ要素を含み、前記区画は複数のデータ要素を含み、前記複数のデータ要素は隣接するデータ要素であり、前記データ処理装置は、所定のアクセス順序においてベクトル内の前記区画にアクセスするように構成される、請求項２に記載のデータ処理装置。

【請求項４】

前記データ要素ハザード判断回路は状態機械を含み、前記状態機械は、命令によってベクトルがアクセスされるのに応じて、前記ベクトル内のデータ要素によって記憶される値の状態を指示し、前記状態は、前記データ要素が、保存されることになるデータ値を記憶しているか、破棄できるデータ値を記憶しているかを指示する、請求項１から３のいずれか一項に記載のデータ処理装置。

【請求項５】

前記データ要素ハザード判断回路は状態機械を含み、前記状態機械は、命令によってベクトルがアクセスされるのに応じて、前記ベクトル内のデータ要素によって記憶される値の状態を指示し、前記状態は、前記データ要素が、保存されることになるデータ値を記憶しているか、破棄できるデータ値を記憶しているかを指示し、

前記状態機械は３つの状態、すなわち、全てのデータ要素が保存されることになるデータ値を含むことを指示する未知状態と、前記ベクトル命令によって現在アクセスされている現在の区画に後続する全てのデータ要素を破棄することができ、他の全てのデータ要素が保存されることになることを指示するアクセス現在状態と、前記現在の区画内の全てのデータ要素と、後続の全てのデータ要素とを破棄することができ、全ての先行するデータ要素が保存されることになるデータ値を有することを指示する先行区画アクセス状態とを含む、請求項３、又は請求項３に従属するときの請求項４に記載のデータ処理装置。

【請求項６】

前記命令が前記区画のうちのいずれに関して演算することになるかを制御するための複数のベクトル制御レジスタと、前記複数のベクトル制御レジスタのうちのいずれが現在の区画を指示しているかを指示する値を記憶するための記憶装置とを備え、前記記憶された値から、次の区画を指示する前記ベクトル制御レジスタのうちの１つが、先行する区画を指示した前記ベクトル制御レジスタと異なることを検出するのに応答して、前記状態機械は未知状態に切り替えられる、請求項５に記載のデータ処理装置。

【請求項７】

前記先行区画アクセス状態は、前記データ要素ハザード判断回路に、前記現在の区画におけるベクトル演算及び先行する区画における前記ベクトル演算が任意の順序において実行できることを指示する、請求項５又は６に記載のデータ処理装置。

【請求項８】

前記ベクトル命令のストリームによって処理される前記ベクトルのためのデータの依存関係を検出するように構成される前記データ要素ハザード判断回路は、前記データの依存関係を判断するときに、データの依存関係がないことを指示する、前記ベクトル命令に関連付けられる注釈に応答する、請求項１から７のいずれか一項に記載のデータ処理装置。

【請求項９】
前記ベクトル命令内のレジスタ識別子によって識別されたレジスタを前記レジスタ・バンク内の前記複数のレジスタのうちの１つとマッピングするためのレジスタ・リネーミング回路を更に備え、
前記データ要素ハザード判断回路が、ベクトル命令に応答してベクトル内の少なくとも１つのデータ要素が更新されることになると判断し、且つ前記ベクトル内の前記所定のアクセス順序における後続のデータ要素が、変更される場合があり、破棄することができる値を現在含むと判断するのに応答して、前記パイプライン化プロセッサは、前記ベクトル命令内の前記レジスタ識別子によって識別された前記レジスタに現在マッピングされている前記レジスタを更新するように構成される、請求項２、又は請求項２に従属するときの請求項３から８のいずれか一項に記載のデータ処理装置。
【請求項１０】
前記パイプライン化プロセッサは、アクセスの順序においてベクトルの第１のデータ要素が更新されることになる時点を判断し、前記ベクトルを任意の現在未使用のレジスタに書き込むことができることを指示する信号を送信するための付加回路を備える、請求項９に記載のデータ処理装置。
【請求項１１】
前記付加回路は、未知状態を指示する第１の状態と、アクセスされることになる前記ベクトルの第１の要素が更新されることを指示する少なくとも１つの更なる状態とを有する第１の要素状態機械を含む、請求項１０に記載のデータ処理装置。
【請求項１２】
ベクトル内のデータ要素ごとに、前記データ要素が現在、保存されるべきであるデータ値を記憶しているか、破棄できるデータ値を記憶しているかを指示する指示を与える指示子を記憶するためのデータ記憶装置を更に備え、前記データ要素ハザード判断回路は前記指示子記憶装置を更新するように構成される、請求項１から１１のいずれか一項に記載のデータ処理装置。
【請求項１３】
ベクトル・レジスタ要素を読み出す命令に応答して、破棄することができるデータ値を記憶するとマークされた区画内のデータ要素ごとに、前記データ要素内に現在記憶されている値の代わりに、所定の値を記憶するように構成される、請求項１２に記載のデータ処理装置。
【請求項１４】
前記パイプライン化プロセッサは、前記ベクトル命令を発行し、実行するための発行回路及び実行回路を備え、前記発行回路はデータ要素ハザード検出回路を備える、請求項１から１３のいずれか一項に記載のデータ処理装置。
【請求項１５】
同期回路を備え、前記同期回路は前記実行回路によって現在処理されているデータ要素の状態を定期的に判断し、前記データ要素の状態の情報を前記発行回路に送信するように構成される、請求項１４に記載のデータ処理装置。
【請求項１６】
前記ベクトルはベクトル演算によって演算されることになるデータ要素を識別するように構成されるベクトル・マスクを備え、前記データ要素ハザード判断回路は、前記ベクトル・マスクから、前記データ要素の前記少なくとも幾つかの場合にそれぞれ、前記ベクトルに関して識別された前記データの依存関係があるか否かを判断し、依存性がない場合には、前記データ要素を処理する命令のための実行のタイミングに関する前記決定された制約を緩和するように構成される、請求項１から１５のいずれか一項に記載のデータ処理装置。
【請求項１７】
前記ベクトルは、ベクトル演算によって演算されることになるベクトル区画内のデータ要素を識別するように構成される、前記ベクトルに関連付けられるベクトル・マスクを有

し、前記データ処理装置は、ベクトル・マスクの制御下でベクトル区画に関する１つの演算を実行し、前記ベクトル・マスクの反転の制御下で前記ベクトル区画に関する更なる演算を実行するように構成され、前記状態機械は、前記ベクトル命令によって現在アクセスされている前記現在の区画に後続する全てのデータ要素と、前記データ区画内の前記マスクされたデータ要素とを破棄することができ、全ての他のデータ要素が保存されることを指示する、マスクアクセス現在状態を含む少なくとも１つの更なる状態を含む、請求項５、又は請求項５に従属するときの請求項６から１５のいずれか一項に記載のデータ処理装置。

【請求項１８】

データ処理装置内のベクトルに関する演算を実行するためのベクトル命令のストリームを処理する方法であって、前記ベクトルはそれぞれ複数のデータ要素を含み、前記データ処理装置内にレジスタに記憶される方法において、

前記ベクトル命令のストリームによって処理され、前記複数のレジスタ内に記憶される前記ベクトルのためのデータの依存関係を検出することであって、レジスタ・データ・ハザードが生じないように、前記ベクトル命令のための実行のタイミングに関する制約を決定し、前記レジスタ・データ・ハザードは、前記ベクトル命令のストリーム内で後に生じるアクセスが、前記ベクトル命令のストリーム内で先に生じるアクセスが完了する前に開始するように、少なくとも一方が書込みである、同じレジスタへの２つのアクセスが、前記ベクトル命令のストリームの順序と異なる順序において生じる場合に生じる、検出することと、

データの依存関係が識別されたベクトル内の前記データ要素の少なくとも幾つかに関して、前記データ要素の前記少なくとも幾つかの場合にそれぞれ、前記ベクトルに対して識別された前記データの依存関係があるか否かを判断することと、依存性がない場合には、

前記データ要素を処理する命令のための実行のタイミングに関する前記決定された制約を緩和することとを含む、方法。

【請求項１９】

所定のアクセス順序においてベクトル内のデータ要素にアクセスするように構成され、実行のタイミングに関する前記決定された制約を緩和することは、

ベクトル命令に応答してベクトル内の少なくとも１つのデータ要素が更新されることになるときに、前記ベクトル内の前記所定のアクセス順序における後続のデータ要素が、変更される場合があり、破棄できるデータ値を現在含むことを判断することと、

前記後続のデータ要素のうちの少なくとも１つを更新する少なくとも１つの更なるベクトル命令に応答して、少なくとも１つの演算を、前記データ要素を更新する前記演算に対して任意の順序において実行できることを実行回路に知らせることとを含む、請求項１８に記載の方法。

【請求項２０】

コンピュータ・プログラムを変換する方法であって、前記コンピュータ・プログラムはベクトルに関する演算を実行するための複数のベクトル命令を含み、前記ベクトルはそれぞれ複数のデータ要素を含み、前記変換する方法は、

前記複数のベクトル命令を解析することと、

前記複数のベクトル命令によって処理される前記ベクトルのためのデータの依存関係を検出することであって、レジスタ・データ・ハザードが生じないように、前記ベクトル命令のための実行のタイミングに関する制約を決定し、前記レジスタ・データ・ハザードは、前記ベクトル命令のストリーム内で後に生じるアクセスが、前記ベクトル命令のストリーム内で先に生じるアクセスが完了する前に開始するように、少なくとも一方が書込みである、同じレジスタへの２つのアクセスが、前記ベクトル命令のストリームの順序と異なる順序において生じる場合に生じる、検出することと、

データの依存関係が識別されたベクトル内の前記データ要素の少なくとも幾つかに関して、前記データ要素の前記少なくとも幾つかの場合にそれぞれ、前記ベクトルに対して識別された前記データの依存関係があるか否かを判断することと、依存性がない場合には、

前記データ要素を処理する命令のための実行のタイミングに関する前記決定された制約を緩和することと、
前記コンピュータ・プログラムをデータ処理システム上で実行するのに適したコードに変換することとを含む、コンピュータ・プログラムを変換する方法。
【請求項２１】
コンピュータ上で実行されるときに、請求項２０に記載の方法のステップを実行するように前記コンピュータを制御するコンピュータ・プログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明はデータ処理の分野に関し、詳細には、ベクトル演算の処理に関する。
【背景技術】
【０００２】
ベクトル処理は、数多くのデータ要素を含むベクトルの処理を伴う。ベクトル内の全てのデータ要素に関して、又は選択されたデータ要素に関して演算が実行される場合があり、これらの演算は、互いに並列に同時に実行される。ベクトルを１つ又は複数の要素からなる区画に分割し、その後、特定の演算によって処理するための区画を選択することによって、異なる要素を選択することができる。代替的には、演算によって処理するために、ベクトル内の幾つかの要素を選択し、他の要素を選択しないマスクを用いることができる。
【０００３】
ベクトル分割及び分割されたデータの処理は、演算を並列に実行できるようにすることによって、性能を改善することができるが、著しいデータハザードを導入する可能性がある。これらのデータハザードは回避されなければならないが、データを並列に処理できるようにしながら、これらの演算のデータ整合性及び一貫性を確保するには、更なるハードウェア、待ち時間及び電力の観点から非常に費用がかかる可能性がある。
【発明の概要】
【発明が解決しようとする課題】
【０００４】
並列処理に関連付けられる性能改善から依然として恩恵を受けながら、ベクトルを処理するときの演算の一貫性及び整合性に関するコストを削減できることが望ましい。
【課題を解決するための手段】
【０００５】
本発明の第１の態様は、ベクトルに関する演算を実行するためのベクトル命令のストリームを処理するためのデータ処理装置を提供し、そのベクトルはそれぞれ複数のデータ要素を含み、そのデータ処理装置は、
処理されているベクトルを記憶するための複数のレジスタを備えるレジスタ・バンクと、
ベクトル命令のストリームを処理するためのパイプライン化プロセッサとを備え、
パイプライン化プロセッサは、ベクトル命令のストリームによって処理され、複数のレジスタに記憶されるベクトルのためのデータ依存性を検出し、且つレジスタ・データ・ハザードが生じないようにベクトル命令のための実行のタイミングに関する制約を決定するように構成される回路を備え、レジスタ・データ・ハザードは、命令ストリーム内で後に生じるアクセスが、その命令ストリームにおいて先に生じているアクセスが完了する前に開始するように、そのうちの少なくとも一方が書込みである、同じレジスタへの２つのアクセスが、命令ストリームの順序と異なる順序において生じる場合に生じ、
パイプライン化プロセッサは、データ依存性が識別されたベクトル内のデータ要素のうちの少なくとも幾つかに関して、そのデータ要素の少なくとも幾つかの場合にそれぞれ、そのベクトルに対して識別されたデータ依存性があるか否かを判断し、データ依存性がない場合には、そのデータ要素を処理する命令のための実行のタイミングに関する決定され

た制約を緩和するように構成されるデータ要素ハザード判断回路を備える。

【０００６】

本発明は、分割されたベクトル演算が数多くのデータハザードの可能性をもたらすことを認識している。これらのデータハザードを正しく扱うには、著しいコストがかかる可能性がある。また、本発明は、通常の使用パターンにおいて、これらのデータハザードはベクトル間に存在する場合があるが、それらのハザードをベクトル内の個々のデータ要素間のハザードと見なせるなら、これらのハザードのうちの多くが消失する場合があることも認識している。これを認識することによって、ベクトルを考慮するときに生成され、且つ最適化が行われた場合に、それらの最適化が安全であり、ベクトル内のデータを破損しないのを確実にするタイミング制約のうちの幾つかを緩和できるようになる。したがって、本発明は、ベクトル内のデータ要素のうちの少なくとも幾つかの間のデータハザードを判断し、ベクトル間のデータ整合性を確保するために必要とされるが、実際のデータ要素整合性のためには必要とされないタイミング制約を緩和する。この関連で、タイミングにする決定された制約を緩和することは、特定の演算が順番に実行されることを必要とするのではなく、それらの演算が互いに重複することを許される場合があるか、又はそれらの演算が異なる順序で実行されることを許される場合があることを意味することができる。

【０００７】

先に言及されたように、そのうちの少なくとも一方が書込みである、同じレジスタへの２つのアクセスが命令ストリームの順序と異なる順序において生じる場合に、データハザードが生じる。この関連で、後続のアクセスが先行するアクセスより遅く完了する場合であっても、先行するアクセスが完了する前に命令ストリーム内の後続のアクセスが開始する場合には、命令ストリームの順序と異なる順序においてアクセスが生じると見なされ、先行するアクセスが完了する前に開始することは、誤ったデータがアクセスされる場合があるという点でデータハザードにつながる恐れがある。

【０００８】

幾つかの実施例では、データ処理装置は、所定のアクセス順序においてベクトル内のデータ要素にアクセスするように構成され、データ要素ハザード判断回路は、ベクトル命令に応答してベクトル内で少なくとも１つのデータ要素が更新されることになるとき、そのベクトル内の所定のアクセス順序における後続のデータ要素が、変更される場合があり、破棄することができる値を現在含むことを判断し、且つ後続のデータ要素のうちの少なくとも１つを更新する少なくとも１つの更なるベクトル命令に応答して、少なくとも１つの演算を、そのデータ要素を更新するその演算に対して任意の順序において実行できること実行回路に知らせるように構成される。

【０００９】

本発明の実施例は、多くの場合に、第１のデータ要素が更新され、その後、後続のデータ要素が更新されるというように、レジスタを更新するために用いられるコードが一般的に特定の順序において更新を行うことを認識している。したがって、異なるベクトル演算が実行されている場合であっても、それらのベクトル演算は同じベクトルに関して実行されている場合があり、最初の演算が最初の３つのデータ要素を満たす場合があり、一方、次の演算が次の３つのデータ要素を満たす場合がある。これを念頭に置いて、データ処理装置は、データ要素の更新を検出し、その際、ベクトル内の後続のデータ要素内に記憶される値が後に更新され、したがって、後に不要になり、破棄することができることを認識するように構成される。これを認識することによって、これらの値を上書きすることになるベクトル演算を、先行するデータ要素の更新を待つことなく、直ちに実行できるようになる。したがって、特定のベクトルを更新する後続の演算を互いに同時に、又は命令ストリームの順序と異なる順序において互いに並列に実行することができる。

【００１０】

幾つかの実施例では、ベクトルは複数の区画に分割され、各区画は少なくとも１つのデータ要素を含み、区画は複数のデータ要素を含み、複数のデータ要素は隣接するデータ要素であり、データ処理装置は、所定のアクセス順序においてベクトル内の区画にアクセス

するように構成される。

【００１１】

ベクトル演算は単一のデータ要素に関して演算することができるが、幾つかの実施例では、ベクトルは分割され、各演算は特定の区画に関して演算する。１つの区画は単一のデータ要素を含む場合があるか、又は複数の隣接するデータ要素を含む場合がある。

【００１２】

幾つかの実施例では、データ要素ハザード判断回路は状態機械を含み、その状態機械は、命令によってベクトルがアクセスされるのに応じて、ベクトル内のデータ要素によって記憶される値の状態を指示し、その状態は、そのデータ要素が保存されることになるデータ値を記憶しているか、破棄することができるデータ値を記憶しているかを指示する。

【００１３】

データ要素ハザードを判断する１つの方法は、ベクトルが命令によってアクセスされた時点でベクトルのデータ要素内に記憶される値の状態を指示する状態機械を用いることによる。この関連で、実際のアクセスが行われるのに先立って、どの状態が存在することになるかをこの時点で判断する。アクセスに先立って判断を行うことによって、収集された情報を用いて、他の演算のタイミングに影響を及ぼすことができ、最適化を実行できるようになる。その状態は、記憶されるデータ要素が保存される必要があるか、破棄することができるかを指示する。アクセス順序制約を用いて、ベクトル内のどの要素が依然としてアクセスされることになるか、それゆえ、破棄することができるかを判断することによってこれを行うことができ、一方、既に更新されている要素は保存される必要がある。

【００１４】

幾つかの実施例では、ベクトル命令のストリームによって処理されるベクトルのためのデータ依存性を検出するように構成される回路は、データ依存性を判断するときに、データ依存性がないことを指示する、ベクトル命令に関連する注釈に応答する。

【００１５】

本発明の実施例は、データ依存性が予想される場合がある命令に関して、実際にはデータ依存性でないことを指示する、データ依存性に関連付けられる注釈を有する命令を与える。そのような場合に、予想されるデータ依存性に関連する命令の実行のスケジューリングに関する制約を緩和することができる。この関連で、ベクトル内の個々の要素を更新する命令が存在し、そのベクトルを一見すると、同じベクトルを更新する後続の命令が順番に実行されるべきであるが、それらの命令が異なる個々の要素をそれぞれ更新する場合には、これは必要ではなく、命令内の注釈はこのことを指示することができ、それにより、タイミング制約を緩和することができる。

【００１６】

幾つかの実施例では、状態機械は３つの状態機械、すなわち、全てのデータ要素が保存されることになるデータ値を含むことを指示する未知状態（ｕｎｋｎｏｗｎ　ｓｔａｔｅ）と、ベクトル命令によって現在アクセスされている現在の区画の後の全てのデータ要素を破棄することができ、他の全てのデータが保存されることになることを指示するアクセス現在状態（ａｃｃｅｓｓ　ｃｕｒｒｅｎｔ　ｓｔａｔｅ）と、現在の区画内の全てのデータ要素と、全ての後続のデータ要素とを破棄することができ、全ての先行するデータ要素が保存されることになるデータ値を有することを指示する先行区画アクセス状態（ａｃｃｅｓｓｅｄ　ｂｙ　ｐｒｅｖｉｏｕｓ　ｐａｒｔｉｔｉｏｎ　ｓｔａｔｅ）とを含む。

【００１７】

状態機械は複数の状態を含む場合があるが、幾つかの実施例では、必要とされる情報を与えるのに３つの状態で十分な場合がある。未知状態が存在する場合があり、安全性の理由から、未知状態では、全てのデータ要素が保存されるべきであるとマークされる。これは、許容できない場合に、これらの要素が上書きされるのを防ぐ。その後、命令がベクトル内の特定の要素又は区画にアクセスしているときに生じるアクセス現在状態が存在し、この時点で、現在の要素又は区画の後の全てのデータ要素を破棄することができ、他の全てのデータ要素が更新されたか、又は更新されつつあるので保存される必要があることが

わかる。更なる状態が存在し、その状態は先行区画アクセス状態であり、その状態は、次の区画が更新されることになるが、まだ更新されていないことを指示する。したがって、この新たな現在の区画内の全てのデータ要素及び後続の全てのデータ要素を破棄することができ、一方、先行する全てのデータ要素は、保存されることになるデータ値を有する。これらの状態は、最適化を判断する際に有用である。未知状態では、最適化は不可能である。先行区画状態では、現在の区画に関するベクトル演算及び先行する区画に関するベクトル演算は任意の順序において実行することができ、それゆえ、先行する区画に関する演算を最初に実行する必要があった任意のタイミング制約は緩和することができる。

【００１８】

幾つかの実施例では、データ処理装置は、その命令が区画のうちのいずれに関して演算することになるかを制御するための複数のベクトル制御レジスタと、複数のベクトル制御レジスタのうちのどのベクトル制御レジスタが現在の区画を指示しているかを指示する値を記憶するための記憶装置とを備え、記憶された値から、次の位置を指示するベクトル制御レジスタのうちの１つが、先行する区画を指示したベクトル制御レジスタと異なることを検出するのに応答して、その状態機械は未知状態に切り替えられる。

【００１９】

どの区画がアクセスされることになるかを制御することができる２つ以上のベクトル制御レジスタが存在することがある。これは独自の問題を提起し、そのような場合に、現在の区画を指示するベクトル制御レジスタを示す値を記憶するデータ記憶装置が必要とされる。このようにして、異なるベクトル制御レジスタの制御下でレジスタの次の区画がアクセスされる場合には、記憶されたデータ値からこれを判断することができ、状態機械内の値は新たなベクトル制御レジスタにとってもはや適切ではなく、それゆえ、状態機械は未知に設定されるべきである。

【００２０】

幾つかの実施例では、データ処理装置は、ベクトル命令内のレジスタ識別子によって識別されるレジスタをレジスタ・バンク内の複数のレジスタのうちの１つとマッピングするためのレジスタ・リネーミング回路を更に備え、データ要素ハザード判定回路が、ベクトル命令に応答してベクトル内の少なくとも１つのデータ要素が更新されることになること、且つそのベクトル内の所定のアクセス順序における後続のデータ要素が、変更される場合があり、破棄することができる値を現在含むことを判断するのに応答して、パイプライン化プロセッサは、ベクトル命令内のレジスタ識別子によって識別されたレジスタに現在マッピングされているレジスタを更新するように構成される。

【００２１】

データ処理装置がレジスタ・リネーミング回路を有する場合に、ベクトル演算を用いてレジスタを更新するとき、個々の要素が異なる時点で更新されるのに応じて、このベクトル演算のために新たなレジスタがリネームされる必要があり、以前にリネームされたレジスタからのデータは、演算が実行されるときに、このレジスタの中に併合される必要がある。これは非常にコストがかかる演算である。本発明の実施例は、特定のデータ要素を破棄できるという判断は、以前にリマッピングされたレジスタ内に記憶される値が実際には必要でない場合があり、それゆえ、それらの値を併合するのに意味はなく、それらの値を新たな値で単に上書きできることを意味することを認識している。この結果、著しい節約になる。

【００２２】

したがって、ベクトル演算に応答して後続のデータ要素を更新するときに、リネーミング回路によって現在マッピングされている物理レジスタは単に書き込むことができ、データを更にリネームしたり、コピーしたりする必要はないことがある。実際には、これらの演算のためのレジスタ・リネーミングを抑止し、大量の時間及び電力を節約する。

【００２３】

幾つかの実施例では、パイプライン化プロセッサは、アクセス順序におけるベクトルの第１のデータ要素が更新されることになる時点を判断し、そのベクトルを現在使用されて

いない任意のレジスタに書き込むことができることを指示する信号を送信するための付加回路を備える。

【００２４】

データ処理装置に、ベクトルの第１のデータ要素が更新されることになる時点を判断する付加回路を設けることになれば、レジスタ・リネーミングが存在する場合、この第１の値を新たなレジスタに書き込み、この新たなレジスタをリネームことができる。その後、その中の値がアクセスされたとき、以前にマッピングされたレジスタは破棄することができる。以前にマッピングされたレジスタ内に記憶される値が新たなプロセスによって上書きされず、これらの値が通常どおりに終了したときに、レジスタは割当てを解除されることになるので、これはタイミング制約を改善することができる。したがって、新たな書込みは、他の値が使用されるのを待つ必要はない。実際には、本発明の実施例は、第１のデータ要素の更新のためにリネーミングをトリガし、後続の更新の場合にリネーミングを抑止する。

【００２５】

幾つかの実施例では、その付加回路は、未知状態を指示する第１の状態と、アクセスされることになるベクトルの第１の要素が更新されることを指示する少なくとも１つの更なる状態とを有する状態機械を含む。

【００２６】

第１の要素が更新されている時点を判断するために、少なくとも２つの状態を有する更なる状態機械を用いることができ、一方の状態は第１の未知状態であり、他方の状態は、ベクトルの第１の要素がアクセスされることを指示する状態である。幾つかの実施例では、実施態様に応じて、更なる状態を有する場合もある。

【００２７】

幾つかの実施例では、データ処理装置は、ベクトル内のデータ要素ごとに、そのデータ要素が現在、保存されるべきデータ値を記憶しているか、破棄することができるデータ値を記憶しているかを指示する指示を与える指示子を記憶するためのデータ記憶装置を更に備え、データ要素ハザード判断回路が、この指示子記憶装置を更新するように構成される。

【００２８】

場合によっては、どのデータ要素が保存されることになり、どのデータ要素を破棄することができるかを記録しておくことが有利な場合がある。この記録は、レジスタ・リネーミングが存在する場合、レジスタ・リネーミング回路内に保持することができるか、又はレジスタ・リネーミングが存在しない場合、ベクトル・レジスタに関連して保持することができる。その指示は、データ要素ごとに、有効なデータが記憶されるか、有効でないデータが記憶されるかを、すなわち、データを保存することができるか、破棄することができるかを指示する指示子の形をとることができるか、又は３つまでの要素が保存されるべきであり、３つを超える要素を破棄することができるなどの別の形において行うことができる。

【００２９】

幾つかの実施例では、そのデータ処理装置は、ベクトルをメモリに記憶する命令に応答して、破棄することができるデータ値を記憶するとマークされた区画内のデータ要素ごとに、そのデータ要素内に現在記憶されている値の代わりに所定の値を記憶するように構成される。

【００３０】

そのデータが妥当であるか否かの指示を記憶することは、ベクトルが読み出されるときなどの場合に有用な場合がある。この場合、レジスタから破棄することができる値を読み出すのではなく、１又は０のような所定の値を単に読み出すことができる。これは、必要とされないこれらのデータ値がレジスタから読み出されるのを回避し、これは、セキュリティが問題であり、データ値が機密情報を含む場合に、現在必要とされていないデータ値が出力されるべきでない場合に重要なことがある。

【００３１】
幾つかの実施例では、パイプライン化プロセッサは、ベクトル命令を発行し、実行するための発行回路及び実行回路を備え、発行回路は、データ要素ハザード検出回路を備える。

【００３２】
パイプライン化プロセッサは、発行回路及び実行回路を備えることができる。発行回路は、プロセッサのフロントエンドにあり、命令の取込み、復号化、及びデータ冗長検査を実行する。パイプライン内でこれらのデータ冗長検査を早期に実行することは、潜在的な最適化がその手順において早期に判断され、これにより、更なる最適化を実行できることを意味する。

【００３３】
幾つかの実施例では、データ処理装置は同期回路を備え、同期回路は、実行回路によって現在処理されているデータ要素の状態を定期的に判断し、その状態情報を発行回路に送信するように構成される。

【００３４】
幾つかの実施例では、データ処理装置は同期回路を備えることができ、同期回路を用いて、現在処理されているデータ要素の状態を定期的に判断し、この情報を発行回路に返送することができる。幾つかの理由から、状態機械はデータの状態を正確に追跡せず、その情報を容易に回復できないことがある。長い符号長にわたってこれが生じる場合には、潜在的な最適化が不可能になる。したがって、定期的に起動するか、又は同期エラーを引き起こす可能性がある特定のイベントに応答して起動するか、又は最近、最適化が実行されていないことを検出するのに応答して起動する同期回路を有することが有利な場合がある。この同期回路は、実行回路からの要素の状態を検出し、これを発行回路にフィードバックし、発行回路は状態機械をリセットし、最適化を継続できるようにする。

【００３５】
幾つかの実施例では、そのベクトルは、ベクトル演算によって演算されることになるデータ要素を識別するように構成されるベクトル・マスクを含み、データ要素ハザード判断回路は、ベクトル・マスクから、データ要素のうちの少なくとも幾つかのデータ要素の場合にそれぞれ、そのベクトルに関して識別されたデータ依存性があるか否か判断し、データ依存性がない場合には、そのデータ要素を処理する命令のための実行のタイミングに関して決定された制約を緩和するように構成される。

【００３６】
ベクトル演算によって演算されることになるデータ要素及び演算されないデータ要素を識別するベクトル・マスクが存在する場合がある。そのような場合、データ要素ハザード判断回路は、これらのマスクから、演算されているベクトル内のデータ要素の一部のみに起因して、そのベクトルに関して識別されたデータ依存性が生じないことを判断できる場合がある。したがって、これらのマスクを解析することによって、タイミングのうちの幾つかを緩和できる場合がある。

【００３７】
幾つかの実施例では、そのベクトルは、ベクトル演算によって演算されることになるベクトル区画内のデータ要素を識別するように構成されるベクトル・マスクを含み、データ処理装置は、１つのベクトル・マスクの制御下で１つのベクトル区画に関する１つの演算を実行し、そのベクトル・マスクの反転の制御下でそのベクトル区画に関する更なる演算を実行するように構成され、その状態機械は、そのベクトル命令によって現在アクセスされている現在の区画の後の全てのデータ要素と、そのデータ区画内のマスクされたデータ要素とを破棄することができ、他の全てのデータ要素が保存されることを指示する、マスクアクセス現在状態（ｍａｓｋｅｄ　ａｃｃｅｓｓ　ｃｕｒｒｅｎｔ　ｓｔａｔｅ）を含む少なくとも１つの更なる状態を含む。

【００３８】
データ依存性を緩和することができる１つの実例は、分割されたベクトル内でベクトル

・マスクを用いて、その区画のサブセットを選択し、そのマスクの反転を用いて他のサブセットを選択する場合である。そのような場合に、分割されたベクトルのために３つの状態が用いられる状態機械に類似であるが、幾つかの更なる状態を有する状態機械を用いることができ、これらの更なる状態は、マスクによって制御される区画のどの部分を破棄することができ、どの部分が保存される必要があるかを指示する。このようにして、どのデータを上書きできるかを知ることによって、動作の順序付けを決定することができる。

【００３９】

本発明の第２の態様は、データ処理装置内でベクトルに関する演算を実行するためのベクトル命令のストリームを処理する方法を提供し、ベクトルはそれぞれ複数のデータ要素を含み、データ処理装置内のレジスタに記憶され、その方法は、

ベクトル命令のストリームによって処理され、複数のレジスタ内に記憶されるベクトルのためのデータ依存性を検出することであって、レジスタ・データ・ハザードが生じないように、ベクトル命令のための実行のタイミングに関する制約を決定し、レジスタ・データ・ハザードは、命令ストリーム内で後に生じるアクセスが、その命令ストリーム内で先に生じているアクセスが完了する前に開始するように、そのうちの少なくとも一方が書込みである、同じレジスタへの２つのアクセスが、命令ストリームの順序と異なる順序において生じる場合に生じる、検出することと、

データ依存性が識別されたベクトル内のデータ要素の少なくとも幾つかのデータ要素に関して、データ要素のうちの少なくとも幾つかのデータ要素の場合にそれぞれ、そのベクトルに対して識別されたデータ依存性があるか否かを判断することと、データ依存性がない場合には、

データ要素を処理する命令のための実行のタイミングに関して決定された制約を緩和することとを含む。

【００４０】

本発明の第３の態様は、コンピュータ・プログラムを変換する方法を提供し、そのコンピュータ・プログラムは、ベクトルに関する演算を実行するための複数のベクトル命令を含み、ベクトルはそれぞれ複数のデータ要素を含み、コンパイルする方法は、

複数のベクトル命令を解析することと、

複数のベクトル命令によって処理されるベクトルのためのデータ依存性を検出することであって、レジスタ・データ・ハザードが生じないように、ベクトル命令のための実行のタイミングに関する制約を決定し、レジスタ・データ・ハザードは、命令ストリーム内で後に生じるアクセスが、その命令ストリームにおいて先に生じているアクセスが完了する前に開始するように、そのうちの少なくとも一方が書込みである、同じレジスタへの２つのアクセスが、命令ストリームの順序と異なる順序において生じる場合に生じる、検出することと、

データ依存性が識別されたベクトル内のデータ要素のうちの少なくとも幾つかのデータ要素に関して、データ要素のうちの少なくとも幾つかのデータ要素の場合にそれぞれ、そのベクトルに関して識別されたデータ依存性があるか否かを判断することと、データ依存性がない場合には、

データ要素を処理する命令のための実行のタイミングに関して決定された制約を緩和することと、

そのコンピュータ・プログラムをデータ処理システム上で実行するのに適したコードに変換することとを含む。

【００４１】

本発明の第４の態様は、コンピュータ上で実行されるときに、本発明の第３の態様による方法のステップを実行するようにコンピュータを制御するコンピュータ・プログラムを提供する。

【００４２】

本発明は、一例にすぎないが、添付の図面において例示される実施例を参照しながら更に説明されることになる。

【図面の簡単な説明】
【００４３】
【図１】本発明の一実施例によるデータ処理装置を示す概略図である。
【図２】本発明の一実施例による、レジスタ・リネーミング回路を備えるデータ処理装置を示す概略図である。
【図３】本発明の一実施例によるレジスタ・リネーミング回路を示す概略図である。
【図４】本発明の一実施例によるベクトル及びデータ依存性検出回路を示す概略図である。
【図５】本発明の一実施例による状態機械を示す概略図である。
【図６】本発明の一実施例によるレジスタ・リネーミングを実行する時点を判断するための状態機械を示す概略図である。
【図７】状態機械の例示的なコード及び対応する状態を示す図である。
【図８】本発明の一実施例による区画及びマスクの両方を有するコードのための状態機械を示す概略図である。
【図９】本発明の一実施例による方法のステップを示す流れ図である。
【図１０】本発明の一実施例によるコンパイラを備えるデータ処理装置を示す図である。
【図１１】注釈付き命令の１つの実例を示す図である。
【発明を実施するための形態】
【００４４】
図１は、本発明の一実施例によるデータ処理装置を概略的に示す。このデータ処理装置は、非常に概略的に示されており、その回路と、実際に存在する更なる回路との間に更なる相互接続が存在する場合があることは当業者には明らかであろう。図示されるデータ処理装置では、パイプライン化プロセッサが存在し、パイプライン化プロセッサは、発行回路１０と、実行回路２０と、ライトバック回路３０と備える。発行回路１０は、命令キャッシュ１２から命令を検索し、これらの命令を復号化し、また、ベクトルの処理に起因して生じるデータ依存性も検出し、復号化された命令を実行回路に発行する。
【００４５】
したがって、発行回路１０はベクトルデータ依存性回路１４を有し、その回路は、ベクトル間のデータ依存性を判断し、ライトバック回路３０から、種々の演算が完了した時点を知ることができるようにする情報を受信する。また、発行回路は、データ要素依存性検出回路１６も備え、その回路は、個々の要素間のデータ依存性を検出し、データ要素自体がアクセスされる方法のために、それらの演算がベクトル間にデータ依存性を引き起こす場合があるが、これらのデータ依存性は実際には生じることなく、それゆえ、ベクトルデータ依存性に起因して特定の順序で実行される必要があるように見えるベクトル演算が、実際には異なる順序において実行することができるので、ベクトルデータ依存性検出回路１４によって課せられるタイミング制約を実際に緩和できると判断する。このデータ要素依存性検出回路１６は、タイミング制約を緩和できる場合を指示する命令に関連付けられる注釈に応答する（図１１を参照）。
【００４６】
データ処理装置５は、ベクトル制御レジスタ４０及びベクトル・レジスタ・バンク５０も備える。データを記憶するためのメモリ６０も存在する。ベクトル・レジスタ・バンク５０は、命令キャッシュ１２から検索されたベクトル命令の制御下で実行回路２０が演算を実行する複数のデータ要素を有するベクトルを含む。各ベクトル演算は、ベクトル内の０、１つ又は複数の要素に関して実行される場合がある。ベクトル制御レジスタ４０は、どのデータ要素に関してベクトル演算が実行されることになるかを制御する。これらのベクトル制御レジスタ４０は、現在の演算によってどの要素が演算されることになるかを指示するマスクの形をとることができるか、又はベクトルのどの区画が演算されることになるかを指示するベクトル区画レジスタの形をとることができる。後者の場合、ベクトルが複数の区画に分割される場合があり、これらの区画はそれぞれ後に説明される。各区画は、１つ又は複数のデータ要素を含む場合があり、これらのデータ要素は隣接する要素であ

る。ベクトル制御レジスタは、それに関連付けられる状態機械１９を有し、その状態機械は、更新されることになる第１のデータ要素を追跡するために用いられ、レジスタ・リネーミングが適している場合、及び適していない場合を判断するために用いることができ、これが図４～図６に関して説明されることになる。

【００４７】

幾つかの実施例では、分割は２つ以上のベクトル制御レジスタによって制御される場合があり、そのような場合、特定のベクトル演算に関するベクトル制御レジスタへの変更を追跡するには十分ではなく、全てのベクトル制御レジスタへの変更を追跡する必要があり、ベクトル・レジスタごとに、そのレジスタ上で最後に分割されたベクトル演算によってどのベクトル制御レジスタが用いられたかを追跡する必要がある。これを追跡するために、特定のレジスタへの書込みのために用いられた最後のベクトル制御レジスタを指示する値を記憶するデータ記憶装置が用いられる場合がある。したがって、特定のベクトル制御レジスタの制御下での特定のレジスタへの書込みに応答して、そのデータ記憶装置は、このベクトル制御レジスタがこのレジスタへの以前の書込みを制御したベクトル制御レジスタであるか否かを判断するためにチェックされ、そうである場合には、その状態機械内の現在値をチェックすることができ、そうでない場合には、そのレジスタのための状態機械は未知に移行する必要がある。

【００４８】

ベクトル・レジスタ・バンク５０に関連付けられる妥当性テーブル７０及び状態機械１７も存在し、妥当性テーブル７０は、レジスタ内のどのデータ要素が有効であり、保存される必要があるか、及びどのデータ要素を破棄することができるかを指示し、一方、状態機械１７は、現在の区画に対してどの区画が演算されているか、どの区画が保存される必要があるデータを記憶し、どの区画が破棄することができるデータを記憶するかを指示する。この状態機械の更なる詳細は、図４及び図５の説明において与えられる。妥当性テーブル７０及び状態機械１７に関する情報は、これらのデータ要素にアクセスする命令及びアクセスのパターンを解析するときに、データ要素依存性検出回路１６によって与えられる。データ要素が無効であると指示され、読み出されるときに破棄できることを指示される場合に、いずれにしても無効である記憶された実際の値ではなく、単に所定の値を使用することができるように、この情報は、ベクトルが読み出されるときに用いることができる。これは電力を節約することができ、もはや有効ではないが、以前に記憶された機密値を含む場合があるデータの出力を防ぐこともできる。したがって、これは、データのセキュリティを保護することができる。

【００４９】

命令復号化／発行を実行から独立して進行できるようにし、それゆえ、ストールを導入するのを避けるので、要素データ依存性回路のみが命令ストリームを監視する必要があることは重要である。しかしながら、これは幾つかの近似を導入し、それは、最適化が安全であるときでも、最適化が実行されないことを意味する場合がある。これらの近似は決して誤った挙動を引き起こさないことに留意されたい。

【００５０】

詳細には、状態についての情報は、ＶＰ（ベクトル区画）又はベクトル・レジスタがスタックに保存され、その後、スタックから復元されるときに発生する可能性がある。状態についての情報を失うことは、ベクトル演算を大量に使用するが、分割を使用しないコードにおいて（すなわち、制御レジスタが、ベクトルの全ての要素を変更する値を与えられ、その後、数多くのベクトル演算が実行される場合に）最も有害である。これは、情報が正確な情報で更新される機会が全く、又はほとんどないので、且つ正確さを欠くことによって不要なコストがかかるので有害である。

【００５１】

これを低減するために、実行回路２０内に同期回路２５が存在する場合があり、これは、必要に応じて、発行回路１０に同期信号を返送する。実行回路から発行回路に戻るこの更なる経路を用いて、２つを再同期させるのを助ける。再同期が生じる間、いずれかのパ

イプラインがストールされなければならないので再同期はコストがかかるか、又はパイプラインがアイドルであるか、少なくとも安定している（すなわち、演算がベクトル区画ＶＰレジスタを変更していない）ときにのみ再同期を実行することができる。

【００５２】

この再同期は、定期的に、又は特定の命令及び／若しくはデータ値に応答して（例えば、データ処理パイプラインが、ベクトル制御レジスタが最適化できるようにする値を有していたが、最適化が適用されなかったことを検出するとき）、又は状況に応じて（すなわち、再同期のコストに応じて）実行することができる。

【００５３】

図２は、図１のデータ処理装置に類似であるが、レジスタ・リネーミング回路７５を更に有する更なるデータ処理装置を示す。本発明の実施例は、インオーダ処理及びアウト・オブ・オーダ処理の両方において用いることができることに留意されたい。この関連で、インオーダ・プロセッサと見なされる処理装置は一般的に、命令ストリームの順序において命令を実行する。しかしながら、それらの装置は、実行速度を高めるために、これらの命令を重複し、並列に実行できる場合もある。本発明の実施例を用いて、そのような並列処理を改善することができ、ベクトル間の潜在的なデータ依存性が実際には生じないようにして、ベクトル内のデータ要素がアクセスされる。

【００５４】

アウト・オブ・オーダ・プロセッサ内にレジスタ・リネーミング回路７５が存在する場合があり、レジスタ・バンク内に存在する物理レジスタの数が、命令セットが命名することができるレジスタ数より多い場合に用いられる。したがって、レジスタ・バンク内の特定のレジスタが、レジスタ・リネーミング回路７５によって、命令内で指定されたレジスタにマッピングされ、そのマッピングはリネーミング・テーブル７２内に記憶される。この実施例では、レジスタ・リネーミング回路は、ベクトル内のどの要素が未定義であるかについての情報を記憶する妥当性テーブル７０も含む。

【００５５】

レジスタ・リネーミング回路７５が図３において更に詳細に示される。レジスタ・リネーミング回路は、現在使用されているレジスタの命令識別子をレジスタ・バンク５０内の物理的位置にマッピングするリネーミング・テーブル７２を含む。この実施例では、レジスタ・リネーミング回路７５は妥当性テーブル７０も含み、妥当性テーブルは、レジスタ内のデータ要素のそれぞれの妥当性を指示する。これは図１の妥当性テーブルに類似であるが、レジスタ・リネーミング回路が存在する場合、レジスタ・リネーミングが生じるのとほぼ同時に妥当性情報が得られるので、ここで妥当性情報を記憶するのに好都合である。

【００５６】

本発明の実施例は、リネーミングを用いるデータ処理装置の効率を大きく改善することができる。レジスタ内に記憶されるベクトル内のデータ要素に関する演算は一般的に、リネームされることになる新たなレジスタと、この演算によって演算されず、新たにリネームされたレジスタと併合される以前にリネームされたレジスタ内に記憶されるデータ要素とを必要とするので、ベクトル処理中のレジスタ・リネーミングは非常に費用がかかる。後続のデータ要素を破棄できることを認識することによって、データのこのリネーミング及びコピーを抑止することができ、電力及び時間を大きく節約することができる。実際には、幾つかの演算が特定のレジスタを更新することになる場合には、一度しかリネームされる必要はなく、後続のベクトル演算は単に同じレジスタにアクセスし、破棄できると認識される要素を更新することができる。更に、その後、ベクトル内の第１の要素がアクセスされる場合、リネーミングを助長することができ、第１の要素のこの新たな更新のために、新たなレジスタをマッピングすることができる。このレジスタは、その後、更なる要素の更新において用いられ、以前にマッピングされたレジスタは上書きされず、これは、この以前のレジスタが読出し演算において使用されている場合にタイミング制約を改善することができる。

【００５７】
図４は、ベクトルデータ依存性回路及び要素依存性回路を更に詳細に示す。ベクトルデータ依存性回路１４は、発行ステージ及びライトバックステージから情報を受信し、これらの情報からベクトルデータ依存性を判断する。その後、これらの情報は、データ要素依存性回路１６によって判断された依存性を踏まえて修正され、データ要素依存性回路は、この実施例では、ベクトル・レジスタ・バンク５０のベクトルに関連付けられる状態機械１７と、ベクトル制御レジスタ４０に関連付けられる状態機械１９との形をとる。状態機械１７は、本実施例では３つの状態を有し、未知状態ＵＮ、現在区画書込み状態（ｗｒｉｔｅ ｔｏ ｃｕｒｒｅｎｔ ｐａｒｔｉｔｉｏｎ ｓｔａｔｅ）ＷＣ及び先行区画書込み状態（ｗｒｉｔｅ ｔｏ ｐｒｅｖｉｏｕｓ ｐａｒｔｉｔｉｏｎ ｓｔａｔｅ）ＷＰが存在する。これらは図５に関して更に詳細に説明される。
【００５８】
動作時に、ベクトル演算が解析されるとき、状態機械は、現在の演算に関連する状態で更新され、これを用いて、判断されたベクトルデータ依存性に関する最適化が可能であるか否かを判断することができる。したがって、状態機械内の値は、現在の区画に対してどの区画が演算されているか、そして、どの区画が保存される必要があるデータを記憶し、どの区画が破棄することができるデータを記憶するかを指示する。個々の要素を考慮するときにデータハザードが生じないので、この情報を用いて、ベクトル演算間のこれらのハザードを回避するために生じるタイミング制約が不要である場合を判断することができる。
【００５９】
この実施例では、ベクトル制御レジスタ４０に関連付けられる状態機械１９も存在する。これは、更新されることになる第１のデータ要素を追跡するために用いることができ、レジスタ・リネーミングが適切である場合、及び適切でない場合を判断することができ、これが図６に関して説明される。
【００６０】
ベクトルとともに区画を用いるとき、ベクトル・レジスタＡ内の幾つかの値を計算するために用いられる初期区画Ｐ０を生成し、その後、Ｐ０を用いて、Ｐ０と重複せず、後にベクトル・レジスタＡに併合される幾つかの更なる値を計算するために用いられる後継区画Ｐ１を生成し、更に、十分な要素が処理されるまで、任意の先行する区画と重複しない更なる区画を生成し続けるのが一般的である。
【００６１】
分割は幾つかの方法において実行することができるが、１つの方法は「ベクトル区画」レジスタ（ＶＰ）が現在の区画を識別する一対の要素インデックス値を保持することによる。例えば、ＶＰが対［０，３］を含む場合には、要素位置０、１、２及び３が現在の区画である。分割するときに用いられる２つの重要な演算は、ＶＰＣＬＲ（ＶＰレジスタを「クリアする」）及びＶＰＮＸＴ（ＶＰを次の区画に変更する）である。
【００６２】
ＶＰＣＬＲは、ＶＰレジスタを、有効要素を含まない初期区画［０，０］に設定する（これは、一例にすぎず、代替の設計はＶＰを［０，７］に設定することができる）。
【００６３】
ＶＰＮＸＴ演算は、現在の区画後に生じる要素のシーケンスである次の区画を識別するようにＶＰを変更する。例えば、［０，０］のＶＰの初期値を与えると、ＶＰＮＸＴを一度だけ用いて、ＶＰを［０，２］に変更することができ、ＶＰＮＸＴを二度目に用いて、ＶＰを［３，５］に変更することができ、ＶＰＮＸＴを三度目に用いて、ＶＰを［６，７］に変更することができる。明らかであるように、区画は小さな数字の要素から大きな数字の要素まで進み、ＶＰＮＸＴによって生成される一連の区画は互いに重複しない（区画の厳密な数及びシーケンスは多くの場合にデータによる）。
【００６４】
ＶＰレジスタの値は、有効要素のみが変更される場合があるという点で、他のベクトル

演算（ＶＡＤＤ、ＶＭＵＬ、ＶＬＯＡＤなど）が適用される要素に影響を及ぼす。更に、小さな数字の要素の値は変更されず、大きな数字の要素の値は、後に変更することができるという点で未定義であり、それらの値は上書きすることができ、保存される必要がない。

【００６５】

あるコードの一例が以下に与えられる。

ＶＰＣＬＲ
；ＶＰ＝［０．．０］
ＶＰＮＸＴ　Ｖ５；Ｖ５内の情報を用いて、次の区画を決定する（例示のために０．．２が用いられる）。
；ＶＰ＝［０．．２］
ＶＡＤＤ　Ｖ０，Ｖ１，Ｖ２；Ｖ１［０．．２］の要素をＶ２［０．．２］に加算し、結果としてＶ０［０．．２］が生成される。

【００６６】

【表１】

3	5	7	?	?	?	?	?

【００６７】

ＶＰＮＸＴ　Ｖ５；Ｖ５内の情報を用いて、次の区画を決定する（例示のために３．．５が用いられる）。
；ＶＰ＝［３．．５］
ＶＳＵＢ　Ｖ０，Ｖ４，Ｖ６；Ｖ４［３．．５］の要素をＶ６［３．．５］から減算し、結果としてＶ０［３．．５］が生成され、Ｖ０［０．．２］を保存する。

【００６８】

【表２】

3	5	7	2	1	0	?	?

【００６９】

ＶＰＮＸＴ　Ｖ５；Ｖ５内の情報を用いて、次の区画を決定する（例示のために６．．７が用いられる）。
；ＶＰ＝［６．．７］
ＶＡＤＤ　Ｖ０，Ｖ８，Ｖ９；Ｖ８［６．．７］の要素をＶ９［６．．７］に加算し、結果としてＶ０［６．．７］が生成される。

【００７０】

【表３】

3	5	7	2	1	0	9	11

【００７１】

上記のコード例において留意すべき２つの重要なことは以下のとおりである。

１）各ベクトル演算は、以前の演算とは異なる（重複しない）１組の要素への書込みを行う。この特性は区画の場所とは無関係であり、１つのベクトル演算が実行されるときにはいつでも当てはまり、その後、ＶＰＮＸＴを用いて、次の区画に移動し、その後、別のベクトル演算が実行される。この結果は、レジスタレベルにおいてＲＡＷハザードが存在するが、要素レベルではＲＡＷハザードは存在しないことを意味する。

２）現在の区画を超える要素は未定義であり（？で示される）、次のＶＰＮＸＴ及び後続のベクトル演算後まで、プログラムによって任意の特定の値を有するように信頼することはできない。これは要素レベルにおける誤ったＷＡＲハザードを排除する。

【００７２】
その結果、幾つかの最適化を適用することができる：
１）Ｖ０への書込みを行う３つ全てのベクトル演算は、その間に実際のハザード又は依存性が存在しないので、原理的には、並列に、又は異なる順序において実行することができる。必要とされるのは、それらの結果を単一の結果レジスタに併合することだけである。
２）同じ宛先レジスタに関与する演算を並列に、又はアウト・オブ・オーダで実行するとき、正確な例外をサポートするために、又はＷＡＷハザードを取り扱うストールを導入するために、通常、レジスタ・リネーミングを用いる必要がある。区画内の早期の書込みは、後の要素を未定義のままにするので、この例では、レジスタ・リネーミング又はストールを使用する必要はない。全ての演算が単一の物理レジスタへの書込みを行うことができる。
【００７３】
そのような最適化をいかに実施できるかを説明する前に、これらの最適化の可能性を検出する必要がある。これは、例えば、図４において示されるような、状態機械１７を用いて命令ストリームを監視することによって行うことができる。
【００７４】
そのような状態機械の状態が図５に示される。これらの状態は、先に図示されたようなコードを処理するときに、プロセッサが有することになる検出された状態に従う。最初に、プロセッサは未知状態にあり、この状態では念のために、最適化を適用することはできない。先に示されたＶＡＤＤのような分割ベクトル演算が実行されるとき、宛先レジスタＶｄは、新たな「ＷＣ」状態への書込みが行われ、プロセッサはその状態に、すなわち、現在区画書込み状態に移行する。この状態では、現在の区画及び先行する区画の値は保存されるべきであるが、その後の区画は未定義であり、上書きされる場合がある。任意の更なる分割ベクトル演算が実行されても、プロセッサはこの状態のままであるが、ＶＰＮＸＴ命令は、プロセッサを次の区画に移動し、プロセッサをＷＣ状態に戻すことになる分割ベクトル演算が実行される前に、プロセッサは「ＷＰ」状態になり、この状態は、現在の区画及び後続の区画内の全ての要素が未定義である先行区画書込み状態（ｗｒｉｔｔｅｎ　ｂｙ　ｐｒｅｖｉｏｕｓ　ｐａｒｔｉｔｉｏｎ　ｓｔａｔｅ）である。未定義値のみ上書きしており、それゆえ、データハザードが生じないので、以前の書込みと組み合わせることによってこの状態を最適化することができる。したがって、この状態を認識することは、最適化を実行することができるというトリガである。
【００７５】
新たなベクトル区画が受信されるとき、未知状態に切り替わる。
【００７６】
この状態機械に加えて、更なる状態機械（図１４の１９）が幾つかの最適化にとって役に立つ場合がある。この更なる状態機械を用いて、更新されることになる第１のデータ要素を追跡することができ、この状態機械を用いて、レジスタ・リネーミングが適している場合、及び適していない場合を判断することができる。したがって、図６から明らかであるように、その状態は未知ＶＰＵＮ状態として開始する。ＶＰＣＬＲ演算を受信すると、状態がＶＰＺＥＲＯに移動し、それは初期区画の前であり、要素は有効ではない。ＶＰＮＸＴは状態を第１の区画に移動し、これは、レジスタ・リネーミングが実行されるべきであるか否かを判断する際に重要である。本発明の実施例では、状態がＶＰＦＩＲＳＴと識別されるときに、宛先レジスタに対してレジスタ・リネーミングが実施される。同じ状態機械１９及びＷＰ状態によって識別される同じ宛先レジスタに関する任意の後続の演算では、レジスタ・リネーミングを抑止されるべきである。

【００７７】
区画を変更する任意の更なる演算は、その状態を未知に移行させ、ＶＰＣＬＲと、その後のＶＰＦＩＲＳＴに応答してのみ、ＶＰＦＩＲＳＴ状態が再び達成される。

【００７８】
図７は、図４の状態機械１７及び１９の例示的なコードシーケンス及び対応する状態を示す。したがって、この場合、いずれの状態機械も未知状態において開始し、状態機械１９は、ＶＰＣＬＲ命令に応答してＶＰＺＥＲＯ状態に移行し、ＶＰＮＸＴ命令に応答してＶＰＦＩＲＳＴに移行し、この時点で、宛先レジスタはレジスタ・リネーミング回路を用いてリネームされる。状態機械１７は、第１の区画においてベクトル演算ＶＡＤＤが実行されるまで、未知状態にとどまり、ＶＡＤＤが実行されると、現在の区画への書込み状態又はＷＣ状態に入る。この時点で、状態機械１９は、未知状態に入る。次の区画を指示するＶＰＮＸＴＶ５命令に応答して、状態機械１７は、ＷＰ状態に入り、次の演算を最適化し、先行する演算と並列に実行することができ、この演算の場合のレジスタ・リネーミングは抑止され、同じレジスタに書き込まれるようになる。

【００７９】
要するに、本発明の実施例は、ＷＡＷ要素ハザードが存在しない場合（すなわち、宛先レジスタ上の状態機械が、ＷＡＷハザードがないこと、すなわち、ＷＰ状態を指示する場合）を判断し、この時点で、未使用の物理レジスタの割当て（レジスタ・リネーミング）を抑止し、新たな分割ベクトル演算が現在の物理レジスタに書込みを行うことができるようにするのが安全である。これは、１つの物理レジスタから別の物理レジスタに現在の区画の外部にある要素を繰り返しコピーしなければならないのを回避し、同じレジスタ上で継続的な分割演算をシリアル化するという問題を回避する。

【００８０】
復号化及び発行ロジックにおいて変更を加えることに加えて、この最適化は、ベクトル・レジスタファイル内の変更も必要とする。ベクトル・レジスタの全ての要素への書込みを行う代わりに、演算は、現在の区画内の要素にのみ書込みを行うことになる。これを成し遂げるために、要素３、４及び５（例えば）への書込み行うことのみを選択できるように、ベクトル・レジスタファイルはベクトル・レジスタの要素ごとに「書込み選択」信号を用いることができる。また、これにより、どの要素に書込みを行うかを選択する制御信号がライトバックステージに返送されなければならない。

【００８１】
ＲＡＷ要素ハザードが存在しない場合（すなわち、ＶＰレジスタ上の状態機械が、現在の区画が第１の要素を含むことを指示する場合）、未使用の物理レジスタを割り当てる（レジスタ・リネーミングをトリガする）必要があるが、先行する物理レジスタからのいかなる要素もコピーする必要はない。これは、ベクトル演算間に更なる依存性を有するという問題の一部を解消する。

【００８２】
これらの最適化は、新規の物理レジスタを割り当てるプロセス、レジスタ名を物理レジスタに変換するプロセス、及びこれらのプロセスを実行する命令復号化及び発行ロジックにおいて最良に実行されるように、レジスタの先行する値を読み出すように準備するプロセスに影響を及ぼす。

【００８３】
この技法は、全てのコピー及び全ての依存性を解消するとは限らない。コピー又は依存性が必要とされる場合もあり、その技法は、コピー又は依存性が必要とされないことを検出できない場合もある。コピーが必要とされるとき、２つの方法のうちの１つにおいて行うことができる。

【００８４】
１つの手法は、実行ロジックが、分割ベクトル演算の２つの変形：併合を実行する変形と、併合を実行しない最適化された形とをサポートすることによる。命令発行ロジックは、状態機械の値に基づいて、適切な変形を選択することができる。

【００８５】
代替の手法は、実行ロジックが、分割ベクトル演算の１つの変形だけをサポートするが、２つのベクトルの要素を１つのベクトルに併合することができる更なる「併合」演算を

含むことによる。その技法が、ＷＡＷハザードが存在しないこと、又はＲＡＷハザードが存在しないことを検出するとき、分割ベクトル演算のみが発行される必要がある。他の全ての場合には、復号化／発行ロジックは、この「併合」演算を更に発行し、レジスタの先行する値を読み出し、その値を新たな結果と併合し、その結果を新たな物理レジスタファイルにライトバックする必要がある。

【００８６】

図８は、マスクを有するプロセッサの状態が区画とともにいかに用いられるかを示す。特定の区画に関する演算を実行することに加えて、プロセッサは、演算が実行されるべき区画内の要素を識別するマスクの制御下で演算を行うことがある。例えば、ベクトル比較演算を用いて、ベクトル・マスクを設定し、そのマスクを用いて、ベクトル加算を実行し、ベクトル・レジスタＶ０への書込みを行い、そのマスクの反転を用いて、ベクトル減算を実行し、同じくベクトル・レジスタＶ０への書込みを行う。マスキングがいかに規定されるかによって、これは、ベクトル加算及びベクトル減算の結果を単一のベクトルに併合することができる。ベクトル分割と同様に、ベクトル・マスキングは、単一の状態機械を用いて監視することができる特定の共通コードシーケンスを伴う。詳細には、ベクトル分割と組み合わせられるときに、ＷＣ状態（上記）を幾つかのサブ状態に分割することができる。

【００８７】

図８は、マスクを使用することに起因して、更なる２つの状態ＷＣ＋及びＷＣ－が生じるときに生じる図５の状態機械に対する拡張を示す。この場合、最初の演算はマスクによって選択された要素に関して実行され、これはＷＣ＋の場合であり、現在の区画の要素のうちの幾つかが未定義であり、その後、ＶＭＮＯＴ演算を実行して、他の要素を選択することができ、その際、これらが演算のために選択されるが、演算が実行される前に、それらの要素は有効である（選択される）が、まだ定義されていない（更新されていない）。演算が実行されると、プロセッサはＷＣ状態に入り、その後、ＶＰＮＸＴに応答して、ＷＰ状態に入り、その状態は、先行する演算と並列に実行されることによって、次の演算を最適化できることを指示する。

【００８８】

図９は、本発明の一実施例による方法のステップを示す流れ図を示す。この方法では、ベクトル命令のストリームによって処理されることになるベクトル内のデータ依存性が検出され、ベクトル命令の実行のタイミング関する制約が決定される。その後、そのデータ要素に関して、識別されたデータ依存性があるか否か、及びデータ依存性がない場合には、これらのデータ要素の処理に関してそのタイミング制約を緩和できるか否かが判断される。この方法は、コンピュータ・プログラムに応答してコンピュータによって実行することができ、ベクトル命令のストリームをコンパイルしているコンパイラにおいて実行することができる。この関連で、これらの命令の処理中に最適化を判断するのではなく、最適化はコンパイル時間に判断される。コンパイラは、システムの外部に存在する場合があるか、又は図１０に示されるように処理装置内に存在する場合がある。

【００８９】

図１０は、データ処理装置を示しており、ランタイムコンパイラ９０を用いて、命令キャッシュ１２から受信されたコードをコンパイルする。コンパイラはコードを解析し、ベクトルデータ依存性及び命令の実行タイミングに関する関連する制約を判断し、データハザードが生じないのを確実にする。その後、個々のデータ要素データ依存性を調べ、以前に決定されたタイミング制約を緩和できるか否かを判断する。このようにして最適化されたコンパイル済みのコードは、その後、発行ステージ１０に送信され、パイプラインを通って進む。ここでは、コンパイラはランタイムコンパイラとして示されるが、処理前にコードをコンパイルするために用いられるリモートコンパイラとすることもできることに留意されたい。

【００９０】

要するに、本発明の実施例は、ハザードが生じない場合を識別し、ハザードが生じると

きに動作を抑止することに基づいて最適化できるようにする。アウト・オブ・オーダ・プロセッサでは、不必要なレジスタ・リネーミング、その結果、不必要なレジスタ読出しが抑止される。更に、パイプラインストールが抑止される。

【００９１】

部分的に順次にベクトル演算が実行される（例えば、全ての要素が処理されるまで、一度に２つの要素が処理される）プロセッサでは、同じベクトルに書込みを行う複数の演算が並列に処理されるのを許される。

【００９２】

本発明の実施例は、命令デコーダ内で監視回路を使用し、監視回路は、ベクトル区画を計算する命令と、ベクトル区画によって変更される命令とを探す命令ストリームを監視する。この監視回路は、各ベクトル・レジスタ及びベクトル制御レジスタの状態を追跡し、最適化が可能であり、且つ安全である場合を識別する小さく簡単な状態機械を使用し、異なる動きをする復号化／発行ロジックをトリガする。

【００９３】

アウト・オブ・オーダ・プロセッサは多くの場合に、レジスタ・リネーミングを用いて、誤ったレジスタ依存性を排除し、正確な例外をサポートする。レジスタ・リネーミングは、命令内で用いられたレジスタ名から物理レジスタ名への変換を確立する。各命令が復号化され、発行できる準備ができたとき、命令の結果を保持するために未使用の物理レジスタが割り当てられ、入力レジスタ名が、その現在の物理マッピングに変換される。これはスカラー命令の場合に十分に機能するが、ベクトル分割ベクトル演算のために用いられるとき、２つのタイプの非効率を導入する。

１．分割ベクトル演算は宛先レジスタの先行値を現在の区画のための新たに計算された値と併合するので、このレジスタに以前にマッピングされた物理レジスタから先行値を読み出す必要がある。これは、更なる読出しポートを必要とし、ベクトル演算が部分的に順次の実施態様を有する場合には、著しい時間を要する場合がある。このコピープロセスは多くの場合に、要素ごとに何度も繰り返される。ベクトルがＮ個の別々の区画に分割される場合には、第１の区画の要素がＮ－１回コピーされることになる。

２．それは同じレジスタ上の演算間に更なる依存性を生み出す。それらが互いに素な要素集合に関して演算する場合であっても、同じレジスタ上の第１の演算がその結果を生成するまで、第２の演算は完了することができない。このシリアル化は、分割ベクトル演算を用いるプログラムにおける潜在的な並列処理を低下させる。

【００９４】

これらの問題に対する本発明の解決策は、上記の技法を用いて、要素ハザードがない状況を識別し、それを用いて、これらの非効率をそれぞれ抑制することである。

【００９５】

パイプライン化インオーダ・プロセッサは、独立した命令の実行を重複させることを試み、依存する命令を検出するハザード検出ロジック（インターロックともいう）に頼り、第１の命令の結果が入手できるまで、第２の依存する命令の発行をストールする。

【００９６】

上記のように、分割ベクトル演算は多くの場合にレジスタ・ハザードをかかえており、それにより、本技法を用いない場合、命令発行がストールする。その技法をパイプライン化インオーダ・プロセッサに追加することによって、ハザード検出ロジックが、要素ハザードがない状況を識別し、それにより、不必要なストールの導入を回避できるようになる。

【００９７】

アウト・オブ・オーダ・プロセッサと同様に、更なる要件は、書込み選択信号を用いてベクトル・レジスタファイルを変更し、どの要素が書き込まれるかを制御し、ＶＰレジスタからライトバックステージに信号を返送するように準備することである。

【００９８】

図１１は、異なる命令間にデータ依存性がないことを指示するように注釈を付された例

示的な命令を示す。この実例は、それぞれ‘＿ｆｉｒｓｔ’、‘＿ｐａｒｔｉａｌ’及び‘＿ｌａｓｔ’を付された、８つの命令のうちの最初の２つの命令及び最後の２つの命令を示しており、注釈なしの変形の場合に存在することになる幾つかのデータ依存性がないことを指示する。注釈なしの変形は、ベクトル・レジスタＶ０の要素番号０をスカラーレジスタＲ０の内容に設定し、ベクトル・レジスタＶ０の要素番号１をスカラーレジスタＲ１の内容に設定し、．．．そして、Ｖ０の要素番号７をスカラーレジスタＲ７の内容に設定する。その注釈なしの変形は、命令が順次に実行することになるライト・アフター・ライト（ＷＡＷ）ハザードにもつながることになる。注釈付きの変形も、対応する要素番号をスカラーレジスタの内容に設定するが、これらの命令はデータ要素ハザードを生み出さないので、命令を並列に実行できるか、又はアウト・オブ・オーダ実行できることを指示する。

【００９９】

レジスタ・リネーミングを用いるプロセッサでは：

－　Ｖｓｅｔ＿ｆｉｒｓｔ命令によって、ベクトル・レジスタＶ０にマッピングされる未使用の物理レジスタＰが割り当てられるが、Ｖ０に以前にマッピングされた物理レジスタの内容がＰにコピーされる必要がないことが指示される。

－　Ｖｓｅｔ＿ｐａｒｔｉａｌ命令は、Ｖ０にマッピングされた物理レジスタＰの適切な要素を更新する（その命令は、未使用の物理レジスタを割り当て、レジスタの以前の内容を新たな要素値と併合する必要はない）

－　Ｖｓｅｔ＿ｌａｓｔ命令は、Ｖ０にマッピングされた物理レジスタＰの最後の要素を更新し（再び、その命令は、未使用の物理レジスタを割り当てる必要はないか、又は古い値を併合する必要はなく）、ここでレジスタが最終的に更新されたことを指示する。

【０１００】

インオーダ・プロセッサでは、Ｖｓｅｔ＿ｆｉｒｓｔ命令は、Ｖ０を用いる先行する命令後に実行しなければならず、Ｖｓｅｔ＿ｌａｓｔ命令は、Ｖ０を用いる後続の命令前に実行しなければならないが、Ｖｓｅｔ＿ｐａｒｔｉａｌ命令は、このシーケンス内のＶｓｅｔ＿ｆｉｒｓｔ命令、他のＶｓｅｔ＿ｐａｒｔｉａｌ命令及びＶｓｅｔ＿ｌａｓｔ命令のいずれかと並列に実行することができる。

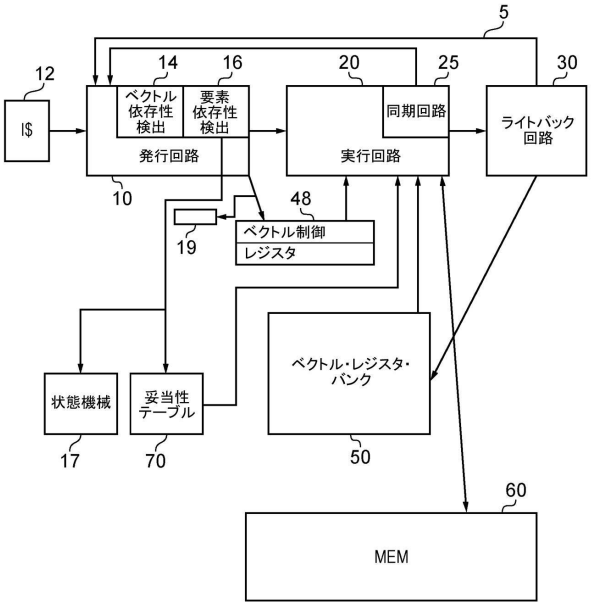
【０１０１】

‘＿ｆｉｒｓｔ’注釈は、要素番号０を設定するときに用いられ、‘＿ｐａｒｔｉａｌ’注釈は、ベクトル内の最も小さい番号、最も大きい番号のいずれでもない要素である要素１～６の場合に用いられ、‘＿ｌａｓｔ’注釈は要素７の場合に用いられる。これにより、＿ｆｉｒｓｔ、＿ｐａｒｔｉａｌ、＿ｌａｓｔ間の違いが要素番号によって決定されてもされなくても、効率的に命令を符号化できるようになる。

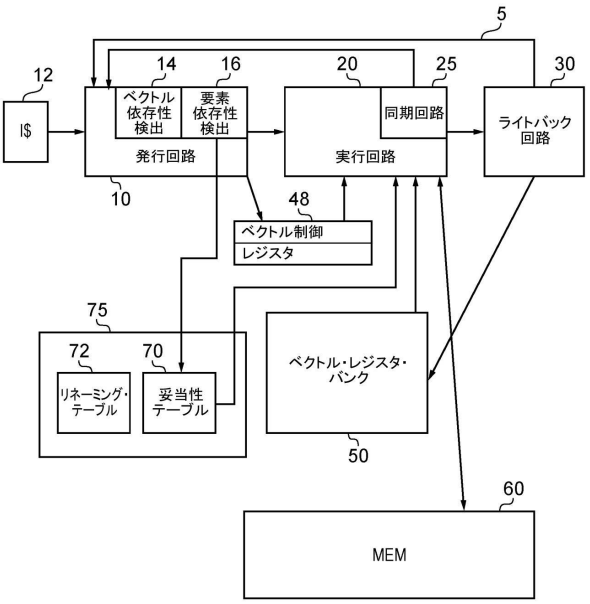
【０１０２】

本発明の種々の更なる態様及び特徴が添付の特許請求の範囲において規定される。本発明の範囲から逸脱することなく、先に説明された本明細書における実施例に対して種々の変更を加えることができる。

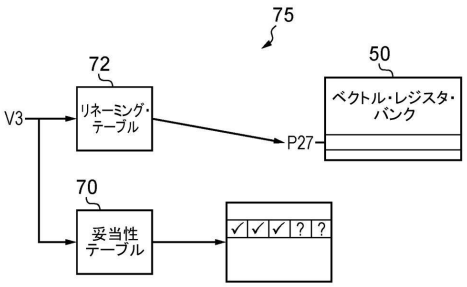
【図１】



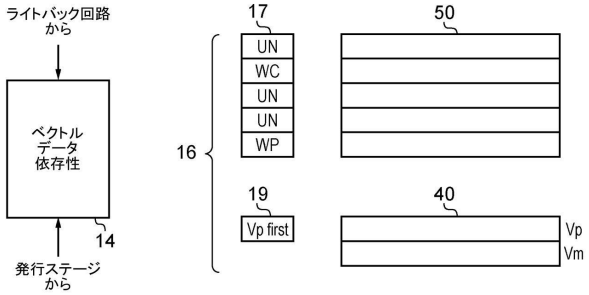
【図２】



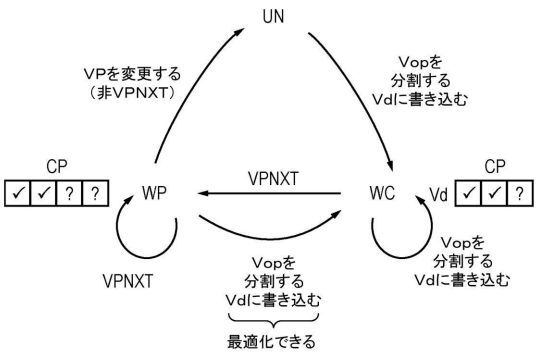
【図３】



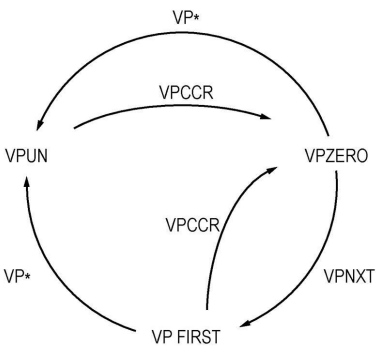
【図４】



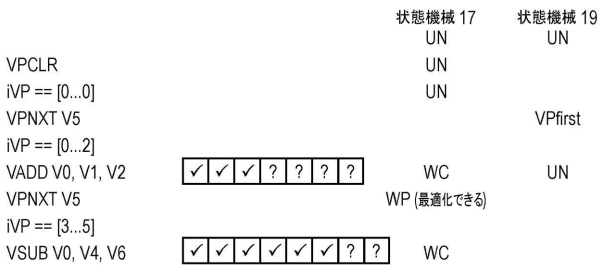
【図５】



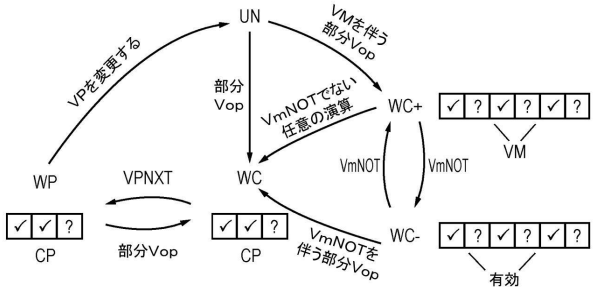
【図６】



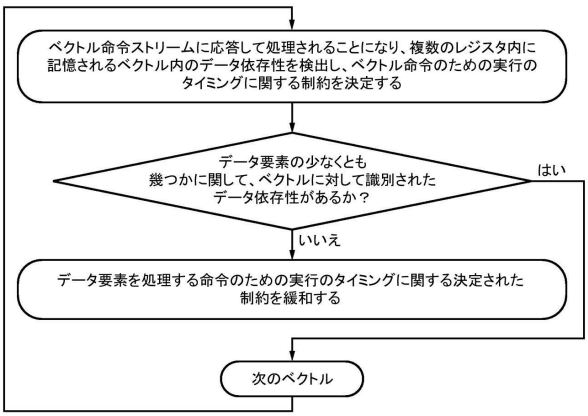
【図７】



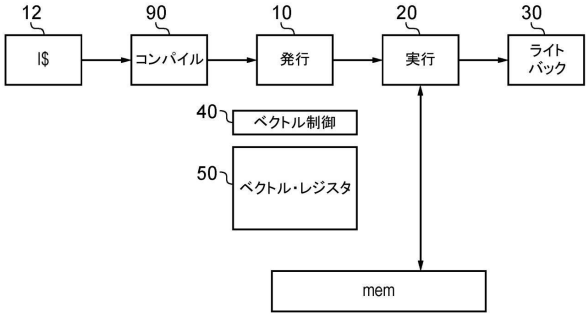
【図８】



【図９】



【図１０】



【図１１】

```
Vset_first      V0, R0, #0
Vset_partial    V0, R1, #1
.
.
.
Vset_partial    V0, R6, #6
Vset_last       V0, R7, #7
```

フロントページの続き

(56)参考文献 特開２０１２－１０３９５９（ＪＰ，Ａ）
特開昭６１－１６３６２（ＪＰ，Ａ）
特開２００５－２３４９６８（ＪＰ，Ａ）
特開平７－２１１５４（ＪＰ，Ａ）
特開２０１０－２１８０７６（ＪＰ，Ａ）
米国特許出願公開第２０１２／０１２４３３２（ＵＳ，Ａ１）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ １７／１６
Ｇ０６Ｆ ９／３０
Ｇ０６Ｆ ９／３８