(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3915019号
(P3915019)
(45)発行日　平成19年5月16日(2007.5.16)　(24)登録日　平成19年2月16日(2007.2.16)

(51)Int.Cl.　F I
G06F 9/38 (2006.01)　G06F 9/38 310F
G06F 9/38 310H
G06F 9/38 370X

請求項の数 2　(全 25 頁)

(21)出願番号　特願平10-167875
(22)出願日　平成10年6月16日(1998.6.16)
(65)公開番号　特開2000-3279(P2000-3279A)
(43)公開日　平成12年1月7日(2000.1.7)
審査請求日　平成17年6月16日(2005.6.16)

前置審査

(73)特許権者　000005821
松下電器産業株式会社
大阪府門真市大字門真１００６番地
(74)代理人　100097445
弁理士　岩橋　文雄
(74)代理人　100109667
弁理士　内藤　浩樹
(74)代理人　100109151
弁理士　永野　大介
(72)発明者　宮地　信哉
大阪府門真市大字門真１００６番地　松下電器産業株式会社内
(72)発明者　檜垣　信生
大阪府門真市大字門真１００６番地　松下電器産業株式会社内

最終頁に続く

(54)【発明の名称】ＶＬＩＷプロセッサ、プログラム生成装置、および記録媒体

(57)【特許請求の範囲】

【請求項１】

固定長又は可変長の複数の命令を並列に実行できるＶＬＩＷプロセッサにおいて、並列に実行できる命令の総ビット数よりも小さい単位で命令フェッチを行い命令レジスタに格納する命令供給発行部と、

いずれの命令解読器に命令が供給されているかを判断する命令発行器と、

前記命令発行器に基づいて、命令が格納されていない命令レジスタに対応する解読結果としてＮＯＰを出力し、命令が格納されている命令レジスタに対応する解読結果はそのまま出力するＮＯＰ生成部とを有し、

並列に実行できる前記複数の命令が、命令フェッチされた命令と命令フェッチされていない命令とを含むとき、前記命令フェッチされた命令のみを実行することを特徴とするＶＬＩＷプロセッサの実行するプログラムを生成するプログラム生成装置であって、

一語が複数の単位命令からなるＶＬＩＷプロセッサのソースコードを格納するソースコード格納手段と、

前記ソースコード格納手段に格納された前記ソースコード中で一語内の単位命令を同時実行した場合と一語内の単位命令を逐次実行した場合で実行結果が異なる問題コードを検出する回避対象コード検出手段と、

前記回避対象コード検出手段により検出された問題コードを一語内の単位命令を同時実行した場合と一語内の単位命令を逐次実行した場合で実行結果が異ならないコードに置き換える逐次実行保証コード生成手段と、

前記逐次実行保証コード生成手段が生成した生成コードを格納する生成コード格納手段とを備えることを特徴とするプログラム生成装置。

【請求項２】

前記ソースコード格納手段に格納されたソースコード中の命令フェッチ境界を検出し命令フェッチ境界情報を出力する命令フェッチ境界検出手段とを備え、

前記回避対象コード検出手段は前記ソースコード格納手段に格納された前記ソースコードの中で一語内の単位命令を同時実行した場合と一語内の単位命令を命令フェッチ境界を単位に逐次実行した場合で実行結果が異なる問題コードを検出し、

逐次実行保証コード生成手段は一語内の単位命令を同時実行した場合と一語内の単位命令を命令フェッチ境界を単位に逐次実行した場合で実行結果が異ならないコードに置き換えることを特徴とする請求項１記載のプログラム生成装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、命令供給が十分に行えない環境で使用されても供給されたものから事項する事により、性能劣化を抑制するＶＬＩＷプロセッサ、プログラム生成装置および記録媒体に関するものである。

【０００２】

【従来の技術】

近年のマイクロプロセッサ応用製品の高機能化および高速化に伴い、高い処理能力を持つマイクロプロセッサ（以下、単に「プロセッサ」という。）が望まれている。このため、最近では、１サイクルに複数の命令を同時に実行することが行われている。

【０００３】

命令レベルの並列処理を実現する方法として、ダイナミックスケジューリングによるものとスタティックスケジューリングによるものがある。

【０００４】

ダイナミックスケジューリングによるものの代表例としてスーパースカラ方式がある。この方式では、実行時に命令コードを解読後、ハードウェアにて動的に命令間の依存関係を解析して並列実行可能か否かを判定し、適切な組み合わせの命令を並列実行する。スタティックスケジューリングによるものの代表例としてＶＬＩＷ（Ｖｅｒｙ　Ｌｏｎｇ　Ｉｎｓｔｒｕｃｔｉｏｎ　Ｗｏｒｄ）方式がある。この方式は、実行コード生成時にコンパイラ等により静的に命令間の依存関係を解析し、命令コードの移動を行って実行効率の良い命令ストリームを生成する。一般のＶＬＩＷ方式では、同時実行可能な複数の命令（ここでは「単位命令」と呼ぶ。）を一つの固定長命令供給単位（ここでは「一語」と呼ぶ。）に記述する。この方式を採ると、ハードウェアで命令間の依存解析を行う必要が無いため、ハードウェアを単純化できるというメリットがある。

【０００５】

以下、従来技術におけるＶＬＩＷプロセッサの動作を図１３を用いて説明する。

【０００６】

図１３は、従来技術におけるＶＬＩＷプロセッサの構成図であり、１０はデータ、命令等が格納されているメモリ、２０はメモリ１０から命令等を取り出す命令供給発行部、３０は命令供給発行部２０で取り出された命令を解読し解読結果を命令実行部へ与える命令解読部である。命令供給発行部２０は、メモリ１０からの命令等の取り出しを制御する命令フェッチ制御部２１とメモリ１０から取り出した命令等を格納する命令レジスタ２２からなる。また、命令解読部３０は、命令の発行を制御する命令発行制御部３１とデコーダ３２と解読結果を格納するレジスタ３３からなる。このプロセッサは、３２ビットの単位命令４つから構成される一語を同時に実行することが可能なＶＬＩＷプロセッサで、１２８ビット単位で命令フェッチされる。

【０００７】

まず、命令供給発行部２０内の命令フェッチ制御部２１は、ＰＣ２（プログラムカウンタ

）、クロック１に基づいて実行する命令のアドレスをアドレスバス１１からメモリ１０に与える。これにより、メモリ１０は指定されたアドレスに対応する命令を１２８ビットのデータバスによって、命令レジスタ２２内の４つの命令レジスタに３２ビットずつ命令を供給する。命令レジスタ２２は、クロック１に基づいてメモリ１０から供給されたデータを格納する。これとともに、命令フェッチが完了したことを意味する命令フェッチフラグ２３を”１”とする。このとき、４つの命令レジスタ２２には、常に命令が格納される。なお、命令フェッチを開始したとき（ジャンプ命令や割り込みが生じた場合等）、誤った命令の解読を防止するため命令フェッチフラグ２３は”０”とされ、キャンセル信号３４によりデコーダからＮＯＰ（Ｎｏ　Ｏｐｅｒａｔｉｏｎ）が出力される。

【０００８】

次に、命令解読部３０におけるデコーダ３２は、命令フェッチフラグ２３により命令レジスタ２２に命令が格納されたという情報を得て、命令を解読した結果を出力する。そして、レジスタ３３はクロック１によって解読した結果を格納する。

【０００９】

最後に、レジスタ３３に格納された解読結果は、命令実行部に供給され（図示せず）、命令が実行されることとなる。

【００１０】

【発明が解決しようとする課題】

しかしながら、上記従来のＶＬＩＷプロセッサでは、命令フェッチを一語長よりも小さい単位で行った場合や命令を可変長とした場合、命令レジスタに命令が供給されるタイミングに差異が生じるため性能が劣化してしまうことがあった。

【００１１】

すなわち、従来のＶＬＩＷプロセッサは一語長と命令フェッチ単位とが一致しているが、ＶＬＩＷを組み込みマイコンに適応するとコストの理由から命令フェッチ幅が一語の幅よりも小さくせざるを得ない場合がある。

【００１２】

また、たとえ最大語長と命令フェッチ単位とが一致していても可変長命令の場合、２回の命令フェッチによって初めて１つの命令を取り込むことができる場合もある。

【００１３】

以下、具体的に図面を用いて説明する。

（１）命令フェッチを一語長よりも小さい単位で行った場合

図１４はプログラム例であり、図１５は同プログラムを実行した場合のパイプラインの流れを説明したものである。

【００１４】

図１４では、$(１０００００００)_{16}$番地に、メモリから読み込んだ結果をｒ０レジスタに格納させる命令”ｍｏｖ　（ｍｅｍ）、ｒ０”が、$(１００００００４)_{16}$番地にはｒ１レジスタの値を１つ増加させる命令”ａｄｄ　＃１、ｒ１、ｒ１”が、以下同様に$(１０００００１Ｆ)_{16}$番地まで命令が配置されている。

【００１５】

この場合、図１５に示すように、タイミングｔ１で$(１０００００００)_{16}$番地の３２ビット長の２つの命令が、タイミングｔ２で$(１０００００００８)_{16}$番地の３２ビット長の２つの命令が命令フェッチされ、タイミングｔ３で４つの命令が同時にデコード、ｔ４で実行される。しかし、$(１０００００００)_{16}$番地の命令”ｍｏｖ　（ｍｅｍ）、ｒ０”は、ＭＥＭステージでメモリを読み込んだ結果をレジスタｒ０に書き込むものであるのに対して、後続する命令である$(１０００００００Ｃ)_{16}$番地の命令”ａｄｄ　＃１、ｒ０、ｒ０”はレジスタｒ０の内容を使用するものであるためＷＢステージでレジスタの書込を行うまで内容を参照出来ない。このため、レジスタ干渉が発生し、$(１０００００００Ｃ)_{16}$番地の命令”ａｄｄ　＃１、ｒ０、ｒ０”はタイミングｔ６で実行できず、タイミングｔ７で実行されることになる。

【００１６】

結果として、命令供給不足とレジスタ干渉の為に、すべての命令を実行するまでに９サイクル必要となる。
（２）命令を可変長とした場合
図１６はプログラム例であり、図１７は同プログラムを実行した場合のパイプラインの流れを説明したものである。
【００１７】
図１６では、（１０００００００）$_{16}$番地に、メモリから読み込んだ結果をｒ０レジスタに格納させる命令”ｍｏｖ　（ｍｅｍ）、ｒ０”が、（１０００００００４）$_{16}$番地にはレジスタｒ１の値を１つ増加させる命令”ａｄｄ　＃１、ｒ１、ｒ１”が、以下、同様に（１０００００１Ｆ）$_{16}$番地まで命令が配置されている。なお、本命令中で、”ａｄｄ　＃１２３４５６７８、ｒ３、ｒ３”命令は６４ビット単位命令であり、他は３２ビット単位命令である。
【００１８】
この場合、図１７に示すように、（１０００００００Ｃ）$_{16}$番地の命令は６４ビット長の命令であるため、タイミングｔ１、ｔ２の２回の命令フェッチによって初めて４つの命令が揃い、タイミングｔ３で４つの命令が同時にデコードされ、ｔ４で実行される。しかし、（１０００００００）$_{16}$番地の命令”ｍｏｖ　（ｍｅｍ）、ｒ０”は、ＭＥＭステージでメモリを読み込んだ結果を書き込んだものであるのに対して、後続する（１０００００１０）$_{16}$番地の命令”ａｄｄ　＃１、ｒ０、ｒ０”はレジスタｒ０の内容を使用するものであるため、ＷＢステージでレジスタの書込を行うまで、内容を参照出来ない。このため、レジスタ干渉が発生し、（１０００００１０）$_{16}$番地の命令”ａｄｄ　＃１、ｒ０、ｒ０”はタイミングｔ６で実行できず、タイミングｔ７で実行されることになる。
【００１９】
結果として、命令供給不足とレジスタ干渉の為に、すべての命令を実行するまでに９サイクル必要となる。
【００２０】
このように、上記従来のＶＬＩＷプロセッサは、なるべくハードウェアを簡略化することにより高速化を図るものであるため、並列処理できる全ての命令が揃った段階でこれらの命令を同時に実行するものであり、この前提が成り立たない場合には十分な性能を発揮できないという問題点があった。
【００２１】
本願発明は、上記従来の課題を解決するもので、命令フェッチを一語長よりも小さい単位で行った場合や命令を可変長とした場合であっても十分な性能を発揮することができるプロセッサを提供するものである。
【００２２】
【課題を解決するための手段】
本願発明は、並列実行できる全ての命令が命令フェッチされなくても、命令フェッチされた命令から先に実行することを特徴とするＶＬＩＷプロセッサである。
【００２３】
【発明の実施の形態】
以下、本発明について、図面を用いて詳細に説明する。
【００２４】
（第１の実施の形態）
本実施の形態は、一語長よりも小さい単位で命令フェッチをした場合でも、効率よく命令を実行可能とするプロセッサ等に関するものである。すなわち、４つの命令を同時に実行できるＶＬＩＷプロセッサであっても、２つの命令が揃った段階で、デコード、実行を開始することにより、極力レジスタ干渉によるパイプラインインタロックを軽減するものである。また、先行的に実行した命令がＩ／Ｏに関する命令である場合、より早くデータを得ることができる。
（１）プロセッサ

図１は本発明の第１の実施の形態におけるプロセッサのブロック図である。図１３に示した従来のＶＬＩＷプロセッサと比較すると、（ａ）データバス１１２が一語長よりも小さい６４ビットである点、（ｂ）４つの命令レジスタのうち左側の２つの命令レジスタに命令が格納されたか、右側の２つの命令レジスタに命令が格納されたかを示す位置情報１２４を持つ点、（ｃ）ＮＯＰを出力させるためのキャンセル信号１３４、１３５がある点で異なる。

【００２５】

このプロセッサは、位置情報１２４により命令レジスタ１２２のどこに命令が格納されたかを認識し、この情報を元にキャンセル信号１３４、１３５を生成しＮＯＰを出力することにより、命令レジスタ１２２に命令が格納されたものから順に解読・実行することを実現している。

【００２６】

まず、命令供給発行部１２０内の命令フェッチ制御部１２１は、ＰＣ１０２、クロック１０１に基づいて実行する命令のアドレスをアドレスバス１１１からメモリ１１０に与える。これにより、メモリ１１０は６４ビットのデータバス１１２を介して、命令レジスタ１２２内の左側の２つの命令レジスタに３２ビットずつ命令を供給する。命令レジスタ１２２は、クロック１０１に基づいてメモリ１１０から供給されたデータを格納する。これとともに、命令フェッチが完了したことを表すため命令フェッチフラグ１２３を”１”、さらに命令レジスタ１２２内の左側の２つに命令が格納されたことを表すため位置情報１２４を”０”とする。このとき、４つの命令レジスタ１２２のうち、左側の２つめの命令レジスタには命令が格納されているが、右側の２つの命令レジスタには命令が格納されていないことになる。なお、従来と同様に命令フェッチが完了していない場合、命令フェッチフラグ１２３は”０”であり、このためキャンセル信号１３４、１３５は”０”となり、ＮＯＰ信号生成器１３７はＮＯＰを出力する。

【００２７】

次に、命令解読部１３０におけるデコーダ１３２は、命令フェッチフラグ１２３により命令レジスタ１２２に命令が格納されたという情報を得て、命令を解読した結果を出力する。このとき、位置情報１２４が”０”であり命令レジスタ１２２のうち左側の２つの命令レジスタにしか命令が格納されていないことを表しているので、キャンセル信号生成器１３１はキャンセル信号１３４を”１”に、キャンセル信号１３５を”０”にする。これにより、デコーダ１３２におけるＮＯＰ生成器１３７のうち左側の２つからは命令レジスタ１２２に格納された命令の解読結果が出力され、右側の２つからはＮＯＰが出力される。そして、レジスタ１３３はクロック１０１によって解読した結果を格納する。なお、ＮＯＰ生成器１３７は、命令解読器１３６の出力とキャンセル信号との論理積を演算するＡＮＤ回路である。すなわち、キャンセル信号１３４、１３５が”０”となっているときは、解読器１３６の出力に関わらず、ＮＯＰを意味する”０”を出力する。

【００２８】

最後に、レジスタ１３３に格納された解読結果は、命令実行部に供給され（図示せず）、命令が実行されることとなる。

【００２９】

なお、次の命令フェッチの際には、フェッチされた命令等は命令レジスタ１２２の右側の２つに格納され、位置情報１２４もこれに対応して更新され、そしてキャンセル信号１３４は”０”、キャンセル信号１３５は”１”となる。

【００３０】

次に、図１４に示すプログラムを実行した場合のパイプラインの流れについて、図２を用いて説明する。

【００３１】

本プロセッサのパイプラインは、命令供給発行部１２０によって命令フェッチを行うステージ（ＩＦステージ）、命令解読部１３０によって命令フェッチした命令を解読するステージ（ＤＥＣステージ）、解読した命令を演算器を使って実行する実行ステージ（以下Ｅ

Ｘステージ）、解読した命令がメモリアクセス命令であった場合にメモリアクセスを行うメモリステージ（ＭＥＭステージ）、演算やメモリアクセス結果をレジスタに反映させる書込ステージ（以下ＷＢステージ）の５段パイプラインとなっている。さらに、レジスタ間演算の様なＥＸステージで演算した実行結果を書き込んだレジスタの値は、ＷＢステージでレジスタで実際の書込を行わなくともＥＸステージ、或いはＭＥＭステージから後続する命令のＥＸステージへバイパスする事によって、直後に配置した命令でも参照可能である。

【００３２】

図１４では、（１００００００）$_{16}$番地に、メモリから読み込んだ結果をｒ０レジスタに格納させる命令”ｍｏｖ　（ｍｅｍ）、ｒ０”が、（１００００００４）$_{16}$番地にはｒ１レジスタの値を１つ増加させる命令”ａｄｄ　＃１、ｒ１、ｒ１”が、以下、同様に（１０００００１Ｆ）$_{16}$番地まで命令が配置されている。

【００３３】

この場合、図２に示すように、タイミングｔ１で（１００００００）$_{16}$番地の３２ビット長の２つの命令が命令フェッチされ、タイミングｔ２で２つの命令が同時にデコード、ｔ３で実行される。そして、タイミングｔ６ではＷＢステージを終え、レジスタｒ０の内容は使用できる状態になっている。

【００３４】

一方、タイミングｔ４で（１００００００８）$_{16}$番地の命令”ａｄｄ　＃１、ｒ０、ｒ０”の命令フェッチが行われ、タイミングｔ６でＥＸステージに入る。このとき、レジスタｒ０は使用できる状態になっているため、レジスタ干渉によるパイプラインインタロックは生じない。結果として、すべての命令を実行するまでに８サイクル必要となる。

【００３５】

図１６に示すパイプラインの流れと図２に示すパイプラインの流れとを比較すると、（１００００００８）$_{16}$番地の命令”ａｄｄ　＃１、ｒ０、ｒ０”がＥＸステージに入るのはタイミングｔ６で同一である。しかし、（１００００００）$_{16}$番地の命令”ｍｏｖ　（ｍｅｍ）、ｒ０”がＷＢステージを完了するのが、図１６ではタイミングｔ６であるのに対し、図２ではタイミングｔ５である点で異なる。これは、図１５では６４ビットの命令フェッチが２回行われ、１２８ビットの命令フェッチが完了した段階でデコード、実行されているのに対し、図２では６４ビットの命令フェッチが行われると次の６４ビットの命令フェッチを待たずにデコード、実行を行っているからである。このため、図１６ではすべての命令を実行するまでに９サイクル必要であるのに対し、図２では８サイクルで実行が完了している。

【００３６】

なお、本実施の形態では、命令の一語長が１２８ビットであるのに対して、データバスが６４ビットである場合を例としているがこれに限られるものではない。例えば、命令の一語長は６４ビットでも２５６ビットでも良く、データバスは３２ビット、１６ビット等２のべき乗であれば足りる。すなわち、命令の一語長よりもデータバスの幅が小さく、一回の命令フェッチで命令の一語長をフェッチできないケースであれば足りる。この場合、命令の一語長を何回の命令フェッチでフェッチできるかによって、位置情報１２４、キャンセル信号１３４、１３５の数が変わる。本実施の形態では、２回の命令フェッチによって命令の一語長をフェッチしているので、位置情報１２４は１ビット（１ビットで２つの情報を表すことができる）で、キャンセル信号は２種類設けている。また、４つの命令を同時に実行するＶＬＩＷを前提としているがこれに限られない。

【００３７】

また、本実施の形態では、メモリ１１０のみが接続されている場合について説明したが、さらに１２８ビットで命令フェッチされるメモリが接続されている場合であっても良い。例えば、内蔵メモリは速度重視で１２８ビットで命令フェッチされるものとし、外部メモリはコストの関係で６４ビットで命令フェッチされるものとし、データバス１１２を介して同列にメモリを接続しメモリ領域によっていずれのメモリを使用するかを切り換えても

よい。この場合、１２８ビットで命令フェッチされるメモリから読み出された場合はもちろんのこと、６４ビット単位で命令フェッチされるメモリから読み出された場合も性能の劣化をなるべく起こさないようにできる。

（２）プログラム生成装置

以上、第１の実施の形態のプロセッサについて述べたが、従来のＶＬＩＷプロセッサ用のプログラム生成装置を本第１の実施の形態のプロセッサに適応しようとすると、例えば、一語中に、命令”ａｄｄ　＃１、ｒ０、ｒ０”が４つ連続した命令を実行する場合、命令供給が十分で一語中の命令を同時に実行した場合にはｒ０レジスタの値が”１”増加するのに対して、命令供給が不十分で一語中の命令を１単位命令毎に逐次実行した場合にはｒ０レジスタの値が”４”増加し、命令供給の状態によって実行結果が異なってしまうという問題点が発生する。

【００３８】

（第１のプログラム生成装置の構成）

図６は本発明の第１の実施の形態における第１のプログラム生成装置のブロック図である。

【００３９】

３００は命令列を格納しているメモリ、３２０は一語内の単位命令を同時実行した場合と一語内の単位命令を逐次実行した場合で実行結果が異なる命令列を抽出する回避対象コード検出手段、３３０は問題となる命令列を回避する命令列を生成する逐次実行保証コード生成手段、３４０は逐次実行保証コード生成手段が生成したプログラムを格納する命令列格納手段である。

【００４０】

以上の様に構成された本発明の第１の実施の形態の第１のプログラム生成装置について、以下、その動作を説明する。

【００４１】

回避対象コード検出手段３２０はソースコード格納手段３００に格納された命令列を入力すると、その命令列中で、一語内の単位命令を同時実行した場合と、一語内の単位命令を逐次実行した場合で実行結果が異なる命令列を回避対象命令列として抽出する。実行結果が異なる命令列とは、具体的には、一語中の任意の単位命令が出力する結果を後続する単位命令が参照する場合の出力命令と参照命令の組み合わせであり、例えば、一語中に含まれる命令”ａｄｄ　ｒ０、ｒ１、ｒ１”と後続する命令”ａｄｄ　ｒ１、ｒ２、ｒ３”の組み合わせである。

【００４２】

図７は回避対象コード検出手段が回避対象命令列を生成するアルゴリズムを示したものである。

【００４３】

ステップ４０１はソースプログラムから１語を読み出すステップ、ステップ４０２は読み込んだ１語を先頭側から１命令単位ずつ読み出すステップ、ステップ４０３はステップ４０２で読み込んだ１命令単位中の出力レジスタ情報を登録するステップ、ステップ４０４は後続する命令単位を先頭側から１命令単位ずつ読み出すステップ、ステップ４０５はステップ４０４で読み込んだ１命令単位中の参照レジスタを登録するステップ、ステップ４０６はステップ４０２で登録した出力レジスタとステップ４０５で登録した参照レジスタが一致しているかどうかを判断するステップ、ステップ４０７はステップ４０５で一致していた場合に後続する命令単位を登録するステップ、ステップ４０８は後続する命令単位があるかを判断し存在する場合にはステップ４０４以降を実行する判断ステップ、ステップ４０９は登録された出力命令と参照命令の組み合わせが存在する場合には回避対象コードとして出力するステップ、ステップ４１０は後続する命令単位があるかを判断し存在する場合にはステップ４０２以降を実行する判断ステップ、ステップ４１１は後続する１語があるかを判断し存在する場合にはステップ４０１以降を実行する判断ステップである。

【００４４】

逐次実行保証コード生成手段３３０は、回避対象コード検出手段３２０の出力する回避対象命令列の情報を用いて、ソースコード格納手段３００に格納された命令列を、同時実行した場合と逐次実行した場合で動作が同一になる命令列への変換を行う。具体的には、命令列中で使用されていないレジスタを検索し、問題となる命令列中の問題となるレジスタを出力する命令の出力レジスタを使用されていないレジスタで置き換えると共に、後続する語で問題となるレジスタを参照する命令の参照レジスタを置き換えたレジスタに置き換える。例えば、一語中に命令”ａｄｄ　ｒ０、ｒ１、ｒ１”と後続する命令”ａｄｄ　ｒ１、ｒ２、ｒ３”が存在し、後続する語に命令”ａｄｄ　＃１、ｒ１、ｒ１”が存在する場合（以降、”ａｄｄ　ｒ０、ｒ１、ｒ１　＆　ａｄｄ　ｒ１、ｒ２、ｒ３　；　ａｄｄ　＃１、ｒ１、ｒ１”と記述する。ここで”＆”は同一語に含まれ、逐次実行の場合には左から右へ実行する事を、”；”は、後続する語との境界であることを示す）は、命令列中で使用していないレジスタをｒ４とすると、問題となる命令列中の問題となるレジスタｒ１を出力する命令”ａｄｄ　ｒ０、ｒ１、ｒ１”の出力レジスタを使用されていないレジスタで置き換え”ａｄｄ　ｒ０、ｒ１、ｒ４”にすると共に、後続する語で問題となるレジスタを参照する命令”ａｄｄ＃１、ｒ１、ｒ１”の参照レジスタを置き換えたレジスタに置き換え”ａｄｄ＃１、ｒ４、ｒ１”にする。変換された命令列は命令列格納手段３４０に出力される。

【００４５】

使用されていないレジスタの検索は、検索を全く行わずに問題となる命令語の前後にスタックへの退避復帰処理を装入することによってレジスタを確保することも可能であるし、最適化コンパイラのレジスタ割付けの要素技術を流用することによって基本ブロック内部や基本ブロックを越えた検索を行い、使用されていないレジスタが存在しない場合には問題となる命令語の前後にスタックへの退避復帰処理を装入することによってレジスタを確保するという方法も可能である。

【００４６】

（命令列生成装置の動作）

次に具体的な命令を解読実行した場合の本命令列生成装置の動作について説明する。

【００４７】

図８（ａ）は、ソースコード格納手段３００に格納された従来のＶＬＩＷプロセッサ用のプログラム生成装置が生成した命令列である。

【００４８】

まず、（１００００００）$_{16}$番地から始まる一語の処理を行う。

回避対象コード検出手段３２０はソースコード格納手段３００に格納された（１００００００）$_{16}$番地から始まる命令列一語分”ａｄｄ　＃１、ｒ０、ｒ０＆　ａｄｄ　＃１、ｒ１、ｒ１　＆　ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ＃１、ｒ３、ｒ３”を入力し、その命令列中で、一語を同時実行した場合と一語内の単位命令を逐次実行した場合で実行結果が異なる命令列がないかを検査する。この命令列中には問題となる命令列は存在しないので、回避対象コード検出手段３２０は問題となる命令列を出力しない。

【００４９】

逐次実行保証コード生成手段３３０は、回避対象コード検出手段３２０が回避対象命令列を出力しないので、ソースコード格納手段３００に格納された（１００００００）$_{16}$番地から始まる命令列一語分をそのまま命令列格納手段３４０へ出力する。

【００５０】

次に、後続する（１００００１０）$_{16}$番地から始まる一語の処理を行う。

回避対象コード検出手段３２０はソースコード格納手段３００に格納された（１００００１０）$_{16}$番地から始まる命令列一語分”ａｄｄ　ｒ０、ｒ１、ｒ０＆　ｓｕｂ　ｒ０、ｒ１、ｒ１　＆　ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ＃１、ｒ３、ｒ３”を入力し、その命令列中で、一語を同時実行した場合と一語内の単位命令を逐次実行した場合で実行結果が異なる命令列がないかを検査する。この命令列中には、”ａｄｄ　ｒ０、ｒ１、ｒ０　＆　ｓｕｂ　ｒ０、ｒ１、ｒ１”が該当する命令となる。

【００５１】
逐次実行保証コード生成手段３３０は、回避対象コード検出手段３２０の出力する回避対象命令列”ａｄｄ　ｒ０、ｒ１、ｒ０　＆　ｓｕｂ　ｒ０、ｒ１、ｒ１”の情報を用いて、ソースコード格納手段３００に格納された命令列を、同時実行した場合と逐次実行した場合で動作が同一になる命令列への変換を行う。後続する命令列を参照し、使用していないレジスタとしてｒ４レジスタを使い、回避対象命令列中の命令”ａｄｄ　ｒ０、ｒ１、ｒ０”を命令”ａｄｄ　ｒ０、ｒ１、ｒ４”に変換すると共に、後続するｒ０を参照する命令を検索し、命令”ａｄｄ　＃１、ｒ０、ｒ０を命令ａｄｄ　＃１、ｒ４、ｒ０”に変換した後、命令列格納手段３４０に出力する。
【００５２】
同様にして、$(１０００００２０)_{16}$番地から始まる命令列一語を処理する事によって、”ａｄｄ　ｒ１、ｒ２、ｒ１　＆　ｓｕｂ　ｒ１、ｒ２、ｒ２　；　ａｄｄ　＃１、ｒ１、ｒ１”を　”ａｄｄ　ｒ１、ｒ２、ｒ５　＆　ｓｕｂ　ｒ１、ｒ２、ｒ２　；　ａｄｄ　＃１、ｒ５、ｒ１”に変換する。
【００５３】
また、$(１０００００３０)_{16}$番地から始まる命令列一語を処理し、回避対象コードが存在するが使用していないレジスタが存在しない場合には、たとえばｒ６レジスタをスタックへの退避命令”ｐｕｓｈ　ｒ６”により確保し、スタックからの復帰命令”ｐｏｐ　ｒ６”により復元する事により、”ａｄｄ　ｒ２、ｒ３、ｒ２　＆　ｓｕｂ　ｒ２、ｒ３、ｒ３”を”ｐｕｓｈ　ｒ６；ａｄｄ　ｒ２、ｒ３、ｒ６　＆　ｓｕｂ　ｒ２、ｒ３、ｒ３；ｍｏｖ　ｒ６、ｒ２　＆　ｐｏｐ　ｒ６”に変換する。
【００５４】
以上の処理によって、回避対象コード検出手段３２０は、図８（ｂ）の様に、斜線部分の命令列を検出し、逐次実行保証コード生成手段３３０は、図８（ｃ）の様に、回避対象コード検出手段３２０の出力する斜線部分の命令列の出力レジスタを変更すると共に、後続する語に含まれる、濃い斜線部分の出力レジスタを参照する参照レジスタを変更した命令列や追加したスタックへのアクセス命令やＮＯＰ命令の命令列を命令列格納手段３４０へ出力する。
【００５５】
（第２のプログラム生成装置の構成）
図９は本発明の第１の実施の形態における第２のプログラム生成装置のブロック図である。
【００５６】
３００は命令列を格納しているメモリシステム、
３１０はプロセッサの命令フェッチ境界を検出する命令フェッチ境界検出手段、３２０は一語内の単位命令を同時実行した場合と一語内の単位命令を命令フェッチ境界を単位に逐次実行した場合で実行結果が異なる命令列を抽出する回避対象コード検出手段、３３０は問題となる命令列を回避する命令列を生成する逐次実行保証コード生成手段、３４０は逐次実行保証コード生成手段が生成したプログラムを格納する命令列格納手段である。
【００５７】
以上の様に構成された本発明の第１の実施の形態における第２のプログラム生成装置について、以下、その動作を説明する。
【００５８】
命令フェッチ境界検出手段３１０はソースコード格納手段３００に格納された命令列を入力すると、その命令列中で、プロセッサの命令フェッチの境界がどこに存在するかを検出する。本実施の形態ではプロセッサの命令フェッチ幅は６４ビットであるので、プロセッサの命令フェッチ境界は、$(１０００００００)_{16}$、$(１００００００８)_{16}$、$(１０００００１０)_{16}$番地という様なアドレスの下位が０または８の番地となる。
【００５９】
回避対象コード検出手段３２０はソースコード格納手段３００に格納された命令列、およ

び、命令フェッチ境界検出手段３１０から出力される命令フェッチ境界情報を入力すると、その命令列中で、一語内の単位命令を同時実行した場合と一語内の単位命令を命令フェッチ境界を単位に逐次実行した場合で実行結果が異なる命令列を抽出する。実行結果が異なる命令列とは、具体的には、一語中の任意の単位命令が出力する結果を後続する単位命令が参照する場合の出力命令と参照命令の組み合わせのうち、命令フェッチ境界を跨いでいるものであり、例えば、一語中に含まれる命令”ａｄｄ　ｒ０、ｒ１、ｒ１”と後続する命令”ａｄｄｒ１、ｒ２、ｒ３”の組み合わせで、命令フェッチ境界を跨いでいるものである。

【００６０】

逐次実行保証コード生成手段３３０は、回避対象コード検出手段３２０の出力する回避対象命令列の情報を用いて、ソースコード格納手段３００に格納された命令列を、同時実行した場合と逐次実行した場合で動作が同一になる命令列への変換を行う。具体的には、命令列中で使用されていないレジスタを検索し、問題となる命令列中の問題となるレジスタを出力する命令の出力レジスタを使用されていないレジスタで置き換えると共に、後続する語で問題となるレジスタを参照する命令の参照レジスタを置き換えたレジスタに置き換える。例えば、一語中に命令”ａｄｄ　ｒ０、ｒ１、ｒ１”と後続する命令”ａｄｄ　ｒ１、ｒ２、ｒ３”が存在し、後続する語に命令”ａｄｄ　＃１、ｒ１、ｒ１”が存在する場合（以降、”ａｄｄ　ｒ０、ｒ１、ｒ１　＆　ａｄｄ　ｒ１、ｒ２、ｒ３　；　ａｄｄ　＃１、ｒ１、ｒ１”と記述する。ここで”＆”は同一語に含まれ、逐次実行の場合には左から右へ実行する事を、”；”は、次の語との境界であることを示す）は、命令列中で使用していないレジスタをｒ４とすると、問題となる命令列中の問題となるレジスタｒ１を出力する命令”ａｄｄ　ｒ０、ｒ１、ｒ１”の出力レジスタを使用されていないレジスタで置き換え”ａｄｄ　ｒ０、ｒ１、ｒ４”にすると共に、後続する語で問題となるレジスタを参照する命令”ａｄｄ　＃１、ｒ１、ｒ１”の参照レジスタを置き換えたレジスタに置き換え”ａｄｄ　＃１、ｒ４、ｒ１”にする。変換された命令列は命令列格納手段３４０に出力される。

【００６１】

（命令列生成装置の動作）

次に具体的な命令を解読実行した場合の本命令列生成装置の動作について説明する。

【００６２】

図１０（ａ）は、ソースコード格納手段３００に格納された従来のＶＬＩＷプロセッサ用のプログラム生成装置が生成した命令列である。

【００６３】

まず、（１００００００００）$_{16}$番地から始まる一語の処理を行う。

命令境界検出手段３１０はソースコード格納手段３００に格納された（１００００００００）$_{16}$番地から始まる命令列一語分中の命令境界である、（１００００００８）$_{16}$番地を検出する。

【００６４】

回避対象コード検出手段３２０はソースコード格納手段３００に格納された（１００００００００）$_{16}$番地から始まる命令列一語分”ａｄｄ　＃１、ｒ０、ｒ０＆　ａｄｄ　＃１、ｒ１、ｒ１　＆　ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ＃１、ｒ３、ｒ３”を入力し、その命令列中で、一語を同時実行した場合と、一語内の命令境界検出手段３１０の出力する命令フェッチ境界を単位として単位命令を逐次実行した場合で実行結果が異なる命令列がないかを検査する。つまり、命令列一語分”ａｄｄ　＃１、ｒ０、ｒ０　＆　ａｄｄ　＃１、ｒ１、ｒ１　＆　ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ　＃１、ｒ３、ｒ３”を同時実行した場合と、”ａｄｄ　＃１、ｒ０、ｒ０　＆　ａｄｄ　＃１、ｒ１、ｒ１”の２つの単位命令と　”ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ　＃１、ｒ３、ｒ３”の２つの単位命令を逐次実行した場合に実行結果が異なる事はないかを検査する。この命令列中には問題となる命令列は存在しないので、回避対象コード検出手段３２０は問題となる命令列を出力しない。

【００６５】
逐次実行保証コード生成手段は３３０は、回避対象コード検出手段３２０が回避対象命令列を出力しないので、ソースコード格納手段３００に格納された（１０００００００）$_{16}$番地から始まる命令列一語分をそのまま命令列格納手段３４０へ出力する。
【００６６】
次に、後続する（１０００００１０）$_{16}$番地から始まる一語の処理を行う。
命令境界検出手段３１０はソースコード格納手段３００に格納された（１０００００１０）$_{16}$番地から始まる命令列一語分中の命令境界である、（１０００００１８）$_{16}$番地を検出する。
【００６７】
回避対象コード検出手段３２０はソースコード格納手段３００に格納された（１０００００１０）$_{16}$番地から始まる命令列一語分”ａｄｄ　ｒ０、ｒ１、ｒ０＆　ｓｕｂ　ｒ０、ｒ１、ｒ１　＆　ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ＃１、ｒ３、ｒ３”を入力し、その命令列中で、一語を同時実行した場合と、一語内の命令境界検出手段３１０の出力する命令フェッチ境界を単位として単位命令を逐次実行した場合で実行結果が異なる命令列がないかを検査する。つまり、命令列一語分”ａｄｄ　ｒ０、ｒ１、ｒ０　＆　ｓｕｂ　ｒ０、ｒ１、ｒ１　＆　ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ　＃１、ｒ３、ｒ３”を同時実行した場合と、”ａｄｄ　ｒ０、ｒ１、ｒ０　＆　ｓｕｂ　ｒ０、ｒ１、ｒ１”の２つの単位命令と”ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ　＃１、ｒ３、ｒ３”の２つの単位命令を逐次実行した場合に実行結果が異なる事はないかを検査する。この命令列中にも問題となる命令列は存在しないので、回避対象コード検出手段３２０は問題となる命令列を出力しない。
【００６８】
逐次実行保証コード生成手段３３０は、回避対象コード検出手段３２０が回避対象命令列を出力しないので、ソースコード格納手段３００に格納された（１０００００１０）$_{16}$番地から始まる命令列一語分をそのまま命令列格納手段３４０へ出力する。
【００６９】
次に、後続する（１０００００２０）$_{16}$番地から始まる一語の処理を行う。
命令境界検出手段３１０はソースコード格納手段３００に格納された（１０００００２０）$_{16}$番地から始まる命令列一語分中の命令境界である、（１０００００２８）$_{16}$番地を検出する。
【００７０】
回避対象コード検出手段３２０はソースコード格納手段３００に格納された（１０００００２０）$_{16}$番地から始まる命令列一語分”ａｄｄ　＃１、ｒ０、ｒ０＆　ａｄｄ　ｒ１、ｒ２、ｒ１　＆　ｓｕｂ　ｒ１、ｒ２、ｒ２　＆　ａｄｄ＃１、ｒ３、ｒ３”を入力し、その命令列中で、一語を同時実行した場合と、一語内の命令境界検出手段２１０の出力する命令フェッチ境界を単位として単位命令を逐次実行した場合で実行結果が異なる命令列がないかを検査する。つまり、命令列一語分”ａｄｄ　＃１、ｒ０、ｒ０　＆　ａｄｄ　ｒ１、ｒ２、ｒ１　＆　ｓｕｂ　ｒ１、ｒ２、ｒ２　＆　ａｄｄ　＃１、ｒ３、ｒ３”を同時実行した場合と、”ａｄｄ　＃１、ｒ０、ｒ０　＆　ａｄｄ　ｒ１、ｒ２、ｒ１”の２つの単位命令と”ｓｕｂ　ｒ１、ｒ２、ｒ２　＆　ａｄｄ　＃１、ｒ３、ｒ３”の２つの単位命令を逐次実行した場合に実行結果が異なる事はないかを検査する。この場合、”ａｄｄ　ｒ１、ｒ２、ｒ１　＆　ｓｕｂ　ｒ１、ｒ２、ｒ２”命令が該当する命令となる。
【００７１】
逐次実行保証コード生成手段３３０は、回避対象コード検出手段３２０の出力する回避対象命令列”ａｄｄ　ｒ１、ｒ２、ｒ１　＆　ｓｕｂ　ｒ１、ｒ２、ｒ２”の情報を用いて、ソースコード格納手段３００に格納された命令列を、同時実行した場合と逐次実行した場合で動作が同一になる命令列への変換を行う。後続する命令列を参照し、使用していないレジスタとしてｒ４レジスタを使い、回避対象命令列中の命令”ａｄｄ　ｒ１、ｒ２、

ｒ１”を命令”ａｄｄ　ｒ１、ｒ２、ｒ５”に変換すると共に、後続するｒ１を参照する命令を検索し、命令”ａｄｄ　＃１、ｒ１、ｒ１”を命令”ａｄｄ　＃１、ｒ５、ｒ１”に変換した後、命令列格納手段３４０に出力する。

【００７２】

以降、（１０００００３０）$_{16}$番地から始まる命令列一語は問題が無いのでそのまま命令列格納手段３４０に出力する。

【００７３】

以上の処理によって、命令フェッチ境界検出手段３１０は図１０（ａ）の太線で示す命令フェッチ境界情報を出力し、回避対象コード検出手段３２０は、図１０（ａ）の様に、斜線部分の命令列を検出し、逐次実行保証コード生成手段３３０は、図１０（ｂ）の様に、回避対象コード検出手段３２０の出力する斜線部分の命令列の出力レジスタを変更すると共に、後続する語に含まれる、出力レジスタを参照する濃い斜線部分の命令列の参照レジスタを変更し、命令列を命令列格納手段３４０へ出力する。

【００７４】

なお、本実施の形態では、命令フェッチ幅６４ビット、１２８ビット固定長、最大同時実行４命令のＶＬＩＷプロセッサを想定しているが、これらの値は特に限定しない。例えば、命令の一語長は６４ビットでも２５６ビットでも良く、データバスの幅は１６ビットでも３２ビットでも良く、すなわち、命令の一語長よりもデータバスの幅が小さいケースが存在すれば足りる。

【００７５】

また、逐次実行保証コード生成手段は、命令列中で使用されていないレジスタを検索し、問題となる命令列中の問題となるレジスタを出力する命令の出力レジスタを使用されていないレジスタで置き換えると共に、後続する語で問題となるレジスタを参照する命令の参照レジスタを置き換えたレジスタに置き換えるアルゴリズムで説明を行ったが、あらかじめ問題となるレジスタを使用されていないレジスタに転送し、問題となるレジスタを参照する命令の参照レジスタを置き換えたレジスタに置き換えるアルゴリズムを行っても構わない。具体的には、実施例では、”ａｄｄ　ｒ０，ｒ１，ｒ０　＆　ｓｕｂ　ｒ０，ｒ１，ｒ１　；　ａｄｄ　＃１，ｒ０，ｒ０”の命令列を”ｍｏｖ　ｒ０、ｒ４　；　ａｄｄ　ｒ０，ｒ１，ｒ０　＆　ａｄｄ　ｒ４，ｒ１，ｒ１　；　ａｄｄ　＃１，ｒ０，ｒ０”としてもよい。

【００７６】

また、回避対象コード検出手段が出力する命令列は、出力命令と参照命令の組み合わせであるので、２命令とは限らない。参照命令が複数ある場合には３命令以上の組み合わせになる場合も存在する。

【００７７】

また、命令列格納手段は、フロッピーディスクやテープやハードディスクやメモリなどの記録媒体でも構わないし、コンパイラやアセンブラオプティマイザ等の最適化プログラムへの入力ファイルであっても構わない。最適化プログラムで処理を繰り返すことにより出力ファイルの更なる最適化を図ることが可能となる。

【００７８】

また、命令フェッチ境界検出手段の認識する命令フェッチ幅は、固定である必要はなく、例えば、それぞれのメモリ領域毎に異なる値を設定しても構わない。その場合には、命令フェッチ境界検出手段は、アドレス情報で命令フェッチ幅を判断する。

【００７９】

また、命令フェッチ幅情報は、プログラム生成装置に組み込んでも構わないし、外部から情報を与えても構わない。具体的には、コンパイラやアセンブラやリンカに、定数として組み込んだ形で指定しても構わないし、引き数や環境ファイルの形で指定しても構わない。また、指定する命令フェッチ幅は一定でも構わないし、空間毎に個別に与えても構わない。

【００８０】

（第２の実施の形態）
本実施の形態は、可変長命令についても効率よく命令を実行できるプロセッサ等に関するものである。
（１）プロセッサ
図３は本発明第２の実施の形態におけるＶＬＩＷプロセッサのブロック図である。このプロセッサは、３２ビットと６４ビットの２通りの単位命令を持ち、最大４つの単位命令から構成される可変長の一語を同時に実行可能なＶＬＩＷプロセッサである。
【００８１】
基本的な構造は図１のＶＬＩＷプロセッサと同じであるが、可変長命令を扱うために、（ａ）命令供給発行部２２０において、メモリ１１０から１２８バイト単位で命令フェッチした命令を命令バッファ２２５を用いて命令バッファ中に３２ビットを１単位とし最大８個のレジスタに格納している点、（ｂ）３２ビット命令または６４ビット命令を切り換えるためにセレクタ２２９を有している点で異なる。
【００８２】
このＶＬＩＷプロセッサは同時に実行できる４つの命令が２回の命令フェッチによって初めて供給されるものであっても、４つの命令の命令フェッチを待たずにデコード、実行するものである。なお、同時に実行できる最大の命令数は４つであるが、命令中に埋め込まれた同時実行できる命令の境界情報により、４以下の同時実行できる命令の数を指定できるが、この機構については図面を省略している。
【００８３】
以上の様に構成された本発明の第２の実施の形態のプロセッサについて、以下、その動作を説明する。
（命令供給部２２０）
まず、命令供給発行部２２０内の命令フェッチ制御部２２１は、ＰＣ２０２、クロック２０１に基づいて実行する命令のアドレスをアドレスバス２１１からメモリ２１０に与える。これにより、メモリ２１０は命令を１２８ビットのデータバス２１２を介して、命令レジスタ２２２内の４つの命令レジスタに３２ビットづつ命令を供給する。命令レジスタ２２２は、クロック２０１に基づいてメモリ２１０から供給されたデータを格納する。これとともに、４つの命令レジスタに命令を格納したことを表すため、格納フラグ２２３を（００００１１１１）$_2$とする。なお、命令バッファ２２５は１２８バイトで命令フェッチされた命令を一旦格納しておくことにより、命令レジスタ２２２に最大２５６ビットの命令を格納するためのものである。
（命令解読部２３０）
次に、命令解読部２３０におけるデコーダ２３２のうち第１命令解読器は一番左端のセレクタ２２９の出力をデコードする。デコードの際には、命令が３２ビット命令である６４ビット命令かを認識し命令長情報２４１とデコード結果２４２とを出力する。具体的には、図４に示すように３２ビットを１単位する先頭に３２ビット命令か６４ビット命令かを示すフォーマット情報が割り当てられているので、この情報をそのまま命令長情報２４１として出力する。なお、セレクタ２２９はそれぞれ、命令が３２ビット命令であるか６４ビット命令であるかに関係なく常に６４ビットのデータを出力する。
【００８４】
デコーダ２３２のうち第１命令発行器は、格納フラグ２２３の値（００００１１１１）$_2$を用いて命令が供給されているか否かを判断する。具体的には、命令が３２ビット命令であった場合には、使用フラグ更新部２４０が（０００００００）$_2$を命令長情報２４１に基づいて左から”１”を入れつつ右に１ビットシフトし（１０００００００）$_2$を得る。そして、これと格納フラグ２２３の値（００００１１１１）$_2$とについてそれぞれのビット単位で論理積を演算し、（０００００００）$_2$となった場合（すべてのビットが”０”）には命令が供給されていると判断し”１”をキャンセル信号２３４として出力する。なお、６４ビット命令の場合、使用フラグ更新部２４０は左から”１”を入れつつ右に２ビットシフトし（（１１０００００）$_2$を得て、格納フラグ２２３の値（００００１

１１１）$_2$ついてそれぞれのビットの論理積を演算し、（０００００００００）$_2$を得て命令が供給されていると判断し”１”をキャンセル信号２３４として出力する。なお、使用フラグ更新部２４０は、キャンセル信号２３４が”０”すなわち命令供給不足であった場合、シフトはしない。

【００８５】

一番左端の格納フラグシフタ２３９は、命令長情報２４１に基づいて、右から”１”を入れつつ格納フラグ２２３を左シフトする。具体的には、第１命令解読部で３２ビット命令を解読した場合は格納フラグ２２３（０００００１１１１）$_2$を１ビット左にシフトして（０００１１１１１）$_2$を得てこれを第２命令発行器に渡す。６４ビット命令であった場合は、２ビット左にシフトして（００１１１１１１）$_2$を得てこれを第２命令発行器に渡す。例えば、格納フラグ２２３が（０００００１１１１）$_2$であるにも関わらず、第１、２命令解読部でそれぞれ６４ビット命令が解読された場合、第３命令発行器は格納フラグシフタ２３９から（１１１１１１１１）$_2$を受け取り、命令供給不足と判断する。これとともに、第２命令解読器に対応したセレクタ２３９で選択すべき命令レジスタ２２２を切り換える。なお、第１～第４命令解読器で使用したビット数は使用フラグ更新部２４０で計算され、使用フラグ２２４として格納される。

【００８６】

そして、ＮＯＰ生成器２３７はデコード結果を出力する。ＮＯＰ生成器２３７は図１のＮＯＰ生成器１３７と同じで、解読器２３６の出力とキャンセル信号２３４との論理積を演算するＡＮＤ回路である。すなわち、キャンセル信号２３４が”０”となっているときは、解読器２３６の出力に関わらず、ＮＯＰを意味する”０”を出力する。

【００８７】

次に、図１６のプログラムを実行した場合のパイプラインの流れについて、図５を用いて説明する。

【００８８】

図１６では、（１００００００００）$_{16}$番地に、メモリから読み込んだ結果をｒ０レジスタに格納させる命令”ｍｏｖ　（ｍｅｍ）、ｒ０”が、（１００００００４）$_{16}$番地にはレジスタｒ１の値を１つ増加させる命令”ａｄｄ　＃１、ｒ１、ｒ１”が、以下、同様に（１０００００１Ｆ）$_{16}$番地まで命令が配置されている。なお、本命令中で、”ａｄｄ　＃１２３４５６７８、ｒ３、ｒ３”命令は６４ビット単位命令であり、他は３２ビット単位命令である。

【００８９】

この場合、図５に示すように、（１０００００１０）$_{16}$番地の命令は６４ビット長の命令であるため、タイミングｔ１、ｔ２の２回の命令フェッチによって初めて４つの命令が揃うが、このプロセッサでは図５に示すように２回目の命令フェッチをまたずに（１０００００００）$_{16}$番地の命令”ｍｏｖ　（ｍｅｍ）、ｒ０”を含む３つの命令をデコード、実行する。そして、タイミングｔ６でレジスタｒ０が使用できる状態になる。

【００９０】

一方、タイミングｔ３で（１０００００２９）$_{16}$番地の命令”ａｄｄ　＃１、ｒ０、ｒ０”の命令フェッチが行われ、タイミングｔ５でＥＸステージに入るが、レジスタｒ０が使用できる状態にまだなっていないためレジスタ干渉によるパイプラインインタロックが発生する。そして、タイミングｔ６でレジスタｒ０は使用できる状態になっているため、”ａｄｄ　＃１、ｒ０、ｒ０”が実行される。結果として、すべての命令を実行するまでに８サイクル必要となる。

【００９１】

図１７に示すパイプラインの流れと図５に示すパイプラインの流れとを比較すると、（１０００００２０）$_{16}$番地の命令”ａｄｄ　＃１、ｒ０、ｒ０”がＥＸステージに入るのはタイミングｔ５で同一である。しかし、（１００００００００）$_{16}$番地の命令”ｍｏｖ　（ｍｅｍ）、ｒ０”がＷＢステージを完了するのが、図１７ではタイミングｔ７であるのに対し、図５ではタイミングｔ６である点で異なる。これは、図１７では並列実行する４つ

の命令全てがそろった段階でデコード、実行されているのに対し、図５では２回目の命令フェッチを待たず（４つ目の命令が命令フェッチされるのを待たずに）にデコード、実行を行っているからである。このため、図１７ではすべての命令を実行するまでに９サイクル必要（タイミングｔ５、ｔ６でパイプラインインタロックが発生）であるのに対し、図５では８サイクルで実行が完了（タイミングｔ５でのみパイプラインインタロックが発生）している。

【００９２】

なお、タイミングｔ２で、（１０００００１０）$_{16}$番地の命令”ａｄｄ　＃１２３４５６７８、ｒ３、ｒ３”命令がフェッチされると同時に、（１０００００２０）$_{16}$番地までの命令もフェッチされるが、”ａｄｄ　＃１２３４５６７８、ｒ３、ｒ３”命令が同時に実行できる命令の境界であるため、この命令のみをタイミングｔ３で実行する。

【００９３】

また、本実施の形態では、４つの命令を同時に実行できるハードウェアを持つＶＬＩＷプロセッサに対し、常に４つの命令を供給することを前提としているが、同じハードウェアに対して、同時実行できる命令の境界を示す技術を用いて４つ未満の命令を供給するものとしても良い。この場合であっても、同時実行できる命令の数に満たない場合であっても、１回の命令フェッチごとにデコード、実行を行う。

（プログラム生成装置）

（第１のプログラム生成装置の構成）

図６は本発明の第２の実施の形態における第１のプログラム生成装置のブロック図である。

【００９４】

基本的な構造は第１の実施の形態の第１のプログラム生成装置と同じであるが、単位命令や一語のビット幅が可変であることに起因して、回避対象コード検出手段３２０、および、逐次実行保証コード生成手段３３０が、単位命令中の並列実行境界情報３０１、および、フォーマット情報３０２を認識する点が異なる。

【００９５】

（命令列生成装置の動作）

以上の様に構成された本発明の第２の実施の形態の第１のプログラム生成装置について、以下、具体的な命令を解読実行した場合の動作を説明する。

【００９６】

図１１（ａ）は、ソースコード格納手段３００に格納された従来のＶＬＩＷプロセッサ用のプログラム生成装置が生成した命令列である。

【００９７】

まず、（１０００００００）$_{16}$番地から始まる一語の処理を行う。

回避対象コード検出手段３２０はソースコード格納手段３００に格納された（１０００００００）$_{16}$番地から始まる命令列一語分”ａｄｄ　＃１、ｒ０、ｒ０＆　ａｄｄ　＃１、ｒ１、ｒ１　＆　ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ＃１２３４５６７８、ｒ３、ｒ３”を入力し、その命令列中で、一語を同時実行した場合と一語内の単位命令を逐次実行した場合で実行結果が異なる命令列がないかを検査する。この命令列中には問題となる命令列は存在しないので、回避対象コード検出手段３２０は問題となる命令列を出力しない。

【００９８】

逐次実行保証コード生成手段３３０は、回避対象コード検出手段３２０が回避対象命令列を出力しないので、ソースコード格納手段３００に格納された（１０００００００）$_{16}$番地から始まる命令列一語分をそのまま命令列格納手段３４０へ出力する。

【００９９】

次に、後続する（１０００００１４）$_{16}$番地から始まる一語の処理を行う。

回避対象コード検出手段３２０はソースコード格納手段３００に格納された（１０００００１４）$_{16}$番地から始まる命令列一語分”ａｄｄ　ｒ０、ｒ１、ｒ０＆　ｓｕｂ　＃１２

３４５６７８、ｒ０、ｒ１　＆　ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ　＃１、ｒ３、ｒ３”を入力し、その命令列中で一語を同時実行した場合と一語内の単位命令を逐次実行した場合で実行結果が異なる命令列がないかを検査する。この命令列中には、”ａｄｄ　ｒ０、ｒ１、ｒ０　＆　ｓｕｂ＃１２３４５６７８、ｒ０、ｒ１”が該当する命令となる。

【０１００】

逐次実行保証コード生成手段３３０は、回避対象コード検出手段３２０の出力する回避対象命令列”ａｄｄ　ｒ０、ｒ１、ｒ０　＆　ｓｕｂ　＃１２３４５６７８、ｒ０、ｒ１”の情報を用いて、ソースコード格納手段３００に格納された命令列を、同時実行した場合と逐次実行した場合で動作が同一になる命令列への変換を行う。後続する命令列を参照し、使用していないレジスタとしてｒ４レジスタを使い、回避対象命令列中の命令”ａｄｄ　ｒ０、ｒ１、ｒ０”を命令”ａｄｄ　ｒ０、ｒ１、ｒ４”に変換すると共に、後続するｒ０を参照する命令を検索し、命令”ａｄｄ　＃１、ｒ０、ｒ０”を命令”ａｄｄ　＃１、ｒ４、ｒ０”に変換した後、命令列格納手段３４０に出力する。

【０１０１】

以降、$(１０００００２８)_{16}$番地から始まる命令列一語を処理する事によって、”ａｄｄ　ｒ１、ｒ２、ｒ１　＆　ｓｕｂ　＃１２３４５６７８、ｒ１、ｒ２　；　ａｄｄ　＃１、ｒ１、ｒ１　を　ａｄｄ　ｒ１、ｒ２、ｒ５　＆　ｓｕｂ　＃１２３４５６７８、ｒ１、ｒ２　；　ａｄｄ　＃１、ｒ５、ｒ１”に、$(１０００００３ｃ)_{16}$番地から始まる命令列一語を処理することによって、”ａｄｄ　ｒ２、ｒ３、ｒ２　＆　ｓｕｂ　＃１２３４５６７８、ｒ２、ｒ３”を”ａｄｄ　ｒ２、ｒ３、ｒ６　＆　ｓｕｂ　＃１２３４５６７８、ｒ２、ｒ３”に変換する。

【０１０２】

以上の処理によって、回避対象コード検出手段３２０は、図１１（ｂ）の様に、網かけ部分の命令列を検出し、逐次実行保証コード生成手段３３０は、図１１（ｃ）の様に、回避対象コード検出手段３２０の出力する網かけ部分の命令列の出力レジスタを変更すると共に、後続する語に含まれる、出力レジスタを参照する濃い網かけ部分の命令列の参照レジスタを変更し、命令列を命令列格納手段３４０へ出力する。

【０１０３】

（第２のプログラム生成装置の構成）

図９は本発明の第２の実施の形態における第２のプログラム生成装置のブロック図である。

【０１０４】

基本的な構造は第１の実施の形態の第２のプログラム生成装置と同じであるが、単位命令や一語のビット幅が可変であることに起因して、回避対象コード検出手段３２０、および、逐次実行保証コード生成手段３３０が、フォーマット情報３０２を認識する点、及び、回避対象コード検出手段３２０において、命令フェッチ境界が単位命令中にあった場合には、命令フェッチ境界が該当する単位命令の先頭に存在すると見なして評価する点、及び、命令フェッチ境界検出手段の検出する命令フェッチ幅が目的とするプロセッサの命令フェッチ幅である１２８ビットとなっている点が異なる。

【０１０５】

（命令列生成装置の動作）

次に具体的な命令を解読実行した場合の本命令列生成装置の動作について説明する。

【０１０６】

図１２（ａ）は、ソースコード格納手段３００に格納された従来のＶＬＩＷプロセッサ用のプログラム生成装置が生成した命令列である。

【０１０７】

まず、$(１０００００００)_{16}$番地から始まる一語の処理を行う。

命令境界検出手段３１０はソースコード格納手段３００に格納された$(１０００００００)_{16}$番地から始まる命令列一語分中の命令境界である、$(１０００００１０)_{16}$番地を検

出する。
【０１０８】
回避対象コード検出手段３２０はソースコード格納手段３００に格納された（１００００００）$_{16}$番地から始まる命令列一語分”ａｄｄ　＃１、ｒ０、ｒ０＆　ａｄｄ　＃１、ｒ１、ｒ１　＆　ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ＃１２３４５６７８、ｒ３、ｒ３”を入力し、その命令列中で、一語を同時実行した場合と一語内の命令境界検出手段３１０の出力する命令フェッチ境界を単位として単位命令を逐次実行した場合で実行結果が異なる命令列がないかを検査する。この命令列中には問題となる命令列は存在しないので、回避対象コード検出手段３２０は問題となる命令列を出力しない。
【０１０９】
逐次実行保証コード生成手段３３０は、回避対象コード検出手段３２０が回避対象命令列を出力しないので、ソースコード格納手段３００に格納された（１０００００００）$_{16}$番地から始まる命令列一語分をそのまま命令列格納手段３４０へ出力する。
【０１１０】
次に、後続する（１０００００１４）$_{16}$番地から始まる一語の処理を行う。
命令境界検出手段３１０はソースコード格納手段３００に格納された（１０００００１４）$_{16}$番地から始まる命令列一語分中の命令境界である、（１０００００２０）$_{16}$番地を検出する。
【０１１１】
回避対象コード検出手段３２０はソースコード格納手段３００に格納された（１０００００１４）$_{16}$番地から始まる命令列一語分”ａｄｄ　ｒ０、ｒ１、ｒ０＆　ｓｕｂ　＃１２３４５６７８、ｒ０、ｒ１　＆　ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ　＃１、ｒ３、ｒ３”を入力し、その命令列中で、一語を同時実行した場合と、一語内の命令境界検出手段３１０の出力する命令フェッチ境界を単位として単位命令を逐次実行した場合で実行結果が異なる命令列がないかを検査する。つまり、命令列一語分”ａｄｄ　ｒ０、ｒ１、ｒ０　＆　ｓｕｂ　＃１２３４５６７８、ｒ０、ｒ１　＆　ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ　＃１、ｒ３、ｒ３”を同時実行した場合と、”ａｄｄ　ｒ０、ｒ１、ｒ０　＆　ｓｕｂ　＃１２３４５６７８、ｒ０、ｒ１”の２つの単位命令と”ａｄｄ　＃１、ｒ２、ｒ２　＆　ａｄｄ　＃１、ｒ３、ｒ３”の２つの単位命令を逐次実行した場合に実行結果が異なる事はないかを検査する。この命令列中にも問題となる命令列は存在しないので、回避対象コード検出手段３２０は問題となる命令列を出力しない。
【０１１２】
逐次実行保証コード生成手段３３０は、回避対象コード検出手段３２０が回避対象命令列を出力しないので、ソースコード格納手段３００に格納された（１０００００１４）$_{16}$番地から始まる命令列一語分をそのまま命令列格納手段３４０へ出力する。
【０１１３】
次に、後続する（１０００００２８）$_{16}$番地から始まる一語の処理を行う。
命令境界検出手段３１０はソースコード格納手段３００に格納された（１０００００２８）$_{16}$番地から始まる命令列一語分中の命令境界である、（１０００００３０）$_{16}$番地を検出する。
【０１１４】
回避対象コード検出手段３２０はソースコード格納手段３００に格納された（１０００００２８）$_{16}$番地から始まる命令列一語分”ａｄｄ　＃１、ｒ０、ｒ０＆　ａｄｄ　ｒ１、ｒ２、ｒ１　＆　ｓｕｂ　＃１２３４５６７８、ｒ１、ｒ２　＆　ａｄｄ　＃１、ｒ３、ｒ３”を入力し、その命令列中で、一語を同時実行した場合と、一語内の命令境界検出手段３１０の出力する命令フェッチ境界を単位として単位命令を逐次実行した場合で実行結果が異なる命令列がないかを検査する。つまり、命令列一語分”ａｄｄ　＃１、ｒ０、ｒ０　＆　ａｄｄ　ｒ１、ｒ２、ｒ１　＆　ｓｕｂ　＃１２３４５６７８、ｒ１、ｒ２　＆　ａｄｄ　＃１、ｒ３、ｒ３”を同時実行した場合と、”ａｄｄ　＃１、ｒ０、ｒ０　＆　ａｄｄ　ｒ１、ｒ２、ｒ１”の２つの単位命令と”ｓｕｂ　＃１２３４５６７８、ｒ１

、ｒ２　＆　ａｄｄ　＃１、ｒ３、ｒ３”の２つの単位命令を逐次実行した場合に実行結果が異なる事はないかを検査する。この場合、”ａｄｄ　ｒ１、ｒ２、ｒ１　＆　ｓｕｂ　＃１２３４５６７８、ｒ１、ｒ２”命令が該当する命令となる。

【０１１５】

逐次実行保証コード生成手段３３０は、回避対象コード検出手段３２０の出力する回避対象命令列”ａｄｄ　ｒ１、ｒ２、ｒ１　＆　ｓｕｂ　＃１２３４５６７８、ｒ１、ｒ２”の情報を用いて、ソースコード格納手段３００に格納された命令列を、同時実行した場合と逐次実行した場合で動作が同一になる命令列への変換を行う。後続する命令列を参照し、使用していないレジスタとしてｒ４レジスタを使い、回避対象命令列中の命令”ａｄｄ　ｒ１、ｒ２、ｒ１”を命令”ａｄｄ　ｒ１、ｒ２、ｒ５”に変換すると共に、後続するｒ１を参照する命令を検索し、命令”ａｄｄ　＃１、ｒ１、ｒ１”を命令”ａｄｄ　＃１、ｒ５、ｒ１”に変換した後、命令列格納手段３４０に出力する。

【０１１６】

以降、$(１０００００３０)_{16}$番地から始まる命令列一語は問題が無いのでそのまま命令列格納手段３４０に出力する。

【０１１７】

以上の処理によって、命令フェッチ境界検出手段３１０は図１２（ａ）の太線で示す命令フェッチ境界情報を出力し、回避対象コード検出手段３２０は、図１２（ａ）の様に、網かけ部分の命令列を検出し、逐次実行保証コード生成手段３３０は、図１２（ｂ）の様に、回避対象コード検出手段３２０の出力する網かけ部分の命令列の出力レジスタを変更すると共に、後続する語に含まれる、出力レジスタを参照する濃い網かけ部分の命令列の参照レジスタを変更し、命令列を命令列格納手段３４０へ出力する。

【０１１８】

なお、本実施の形態では、命令フェッチ幅１２８ビット、３２ビットと６４ビットの可変長、最大同時実行４命令のＶＬＩＷプロセッサを想定しているが、これらの値は特に限定しない。

【０１１９】

また、逐次実行保証コード生成手段は、命令列中で使用されていないレジスタを検索し、問題となる命令列中の問題となるレジスタを出力する命令の出力レジスタを使用されていないレジスタで置き換えると共に、後続する語で問題となるレジスタを参照する命令の参照レジスタを置き換えたレジスタに置き換えるアルゴリズムで説明を行ったが、第１の実施例における第２のプログラム生成装置と同じく、あらかじめ問題となるレジスタを使用されていないレジスタに転送し、問題となるレジスタを参照する命令の参照レジスタを置き換えたレジスタに置き換えるアルゴリズムを行っても構わない。

【０１２０】

また、回避対象コード検出手段が出力する命令列は、出力命令と参照命令の組み合わせであるので、２命令とは限らない。参照命令が複数ある場合には３命令以上の組み合わせになる場合も存在する。

【０１２１】

また、命令列格納手段は、フロッピーディスクやテープやハードディスクやメモリなどの記録媒体でも構わないし、コンパイラやアセンブラオプティマイザ等の最適化プログラムへの入力ファイルであっても構わない。最適化プログラムで処理を繰り返すことにより出力ファイルの更なる最適化を図ることが可能となる。

【０１２２】

また、命令フェッチ境界検出手段の認識する命令フェッチ幅は、固定である必要はなく、例えば、それぞれのメモリ領域毎に異なる値を設定しても構わない。その場合には、命令フェッチ境界検出手段は、アドレス情報で命令フェッチ幅を判断する。

【０１２３】

また、命令フェッチ幅情報は、プログラム生成装置に組み込んでも構わないし、外部から情報を与えても構わない。具体的には、コンパイラやアセンブラやリンカに、定数として

組み込んだ形で指定しても構わないし、引き数や環境ファイルの形で指定しても構わない。また、指定する命令フェッチ幅は一定でも構わないし、空間毎に個別に与えても構わない。

【０１２４】

【発明の効果】

以上のように、本願発明によれば、命令供給が十分に行えない環境で使用されても供給されたものから実行する事により、性能劣化を抑制することができる。

【図面の簡単な説明】

【図１】本発明の第１の実施の形態におけるプロセッサのブロック構成図

【図２】本発明の第１の実施の形態における第１のプログラム例及びパイプライン図

【図３】本発明の第１、第２の実施の形態における第２のプログラム例及びパイプライン図

【図４】本発明の第１、第２の実施の形態における第１のプログラム生成装置のブロック図

【図５】本発明の第１、第２の実施の形態における第１のプログラム生成装置におけるプログラム図

【図６】本発明の第１、第２の実施の形態における第１のプログラム生成装置のブロック図

【図７】本発明の第１の実施の形態における第１のプログラム生成装置における回避対象コード検出手段の検出アルゴリズムを示す図

【図８】本発明の第１の実施の形態における第１のプログラム生成装置のプログラム図

【図９】本発明の第１、第２の実施の形態における第２のプログラム生成装置のブロック図

【図１０】本発明の第１の実施の形態における第２のプログラム生成装置のプログラム図

【図１１】本発明の第２の実施の形態における第１のプログラム生成装置のプログラム図

【図１２】本発明の第２の実施の形態における第２のプログラム生成装置のプログラム図

【図１３】第１の従来例におけるプロセッサのブロック構成図

【図１４】第１のプログラム例を示す図

【図１５】従来例における第１のプログラム例のパイプライン図

【図１６】第２のプログラム例を示す図

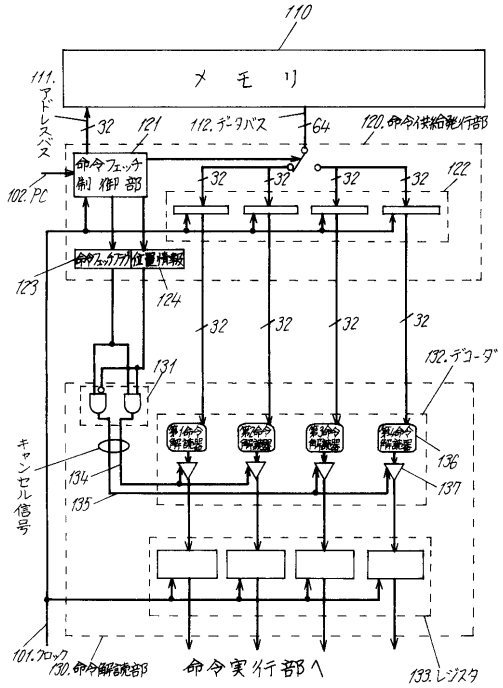
【図１７】従来例における第２のプログラム例のパイプライン図

【符号の説明】

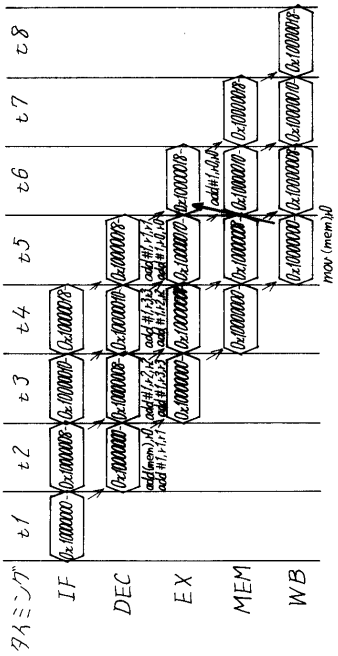
１０１、２０１　クロック
１０２、２０２　ＰＣ
１１０、２１０　メモリ
１１１、２１１　アドレスバス
１１２、２１２　データバス
１２０、２２０　命令供給発行部
１２１、２２１　命令フェッチ制御部
１２２、２２２　命令レジスタ
１２３　命令フェッチフラグ
１２４　位置情報
１３０、２３０　命令解読部
１３１　キャンセル信号生成部
１３２、２３２　デコーダ
１３３、２３３　レジスタ
１３４、１３５、２３４　キャンセル信号
１３６、２３６　解読器
１３７、２３７　ＮＯＰ信号生成器
２２３　格納フラグ

２２４　使用フラグ

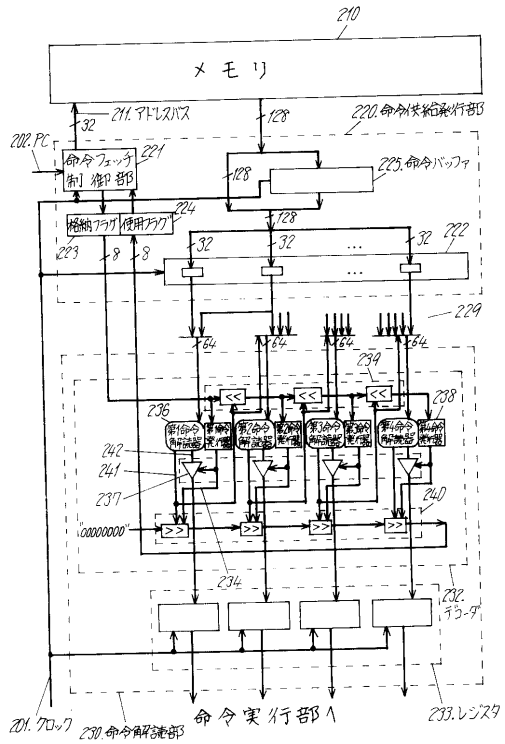
【図１】



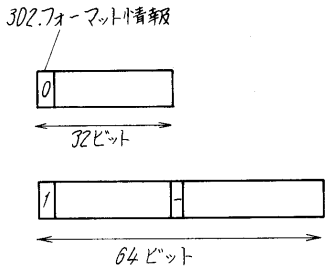
【図２】



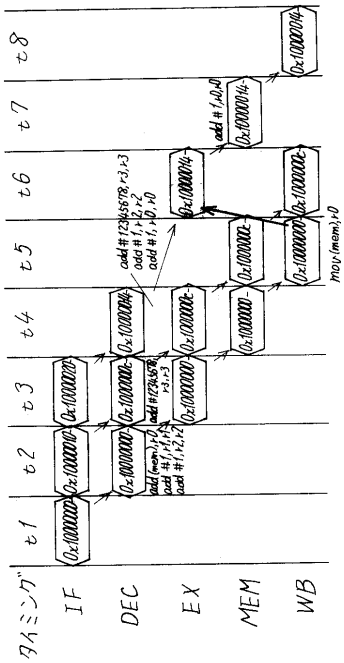
【図３】


210
メモリ
211.アドレスバス
220.命令供給発行部
202.PC
221
命令フェッチ制御部
225.命令バッファ
224
格納フラグ
使用フラグ
223
222
229
239
238
236
242
241
237
240
234
232.デコーダ
201.プロック
230.命令解読部
命令実行部
233.レジスタ

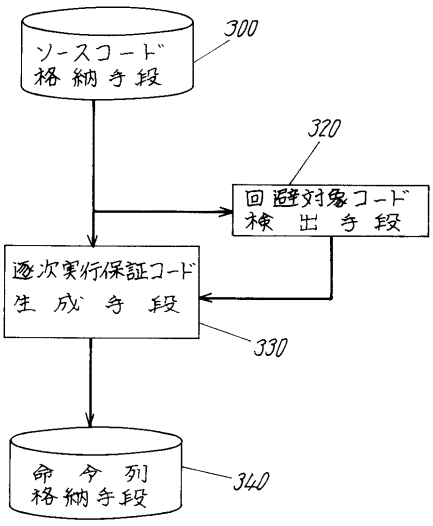
【図４】


302.フォーマット情報
0
32ビット
1
64ビット

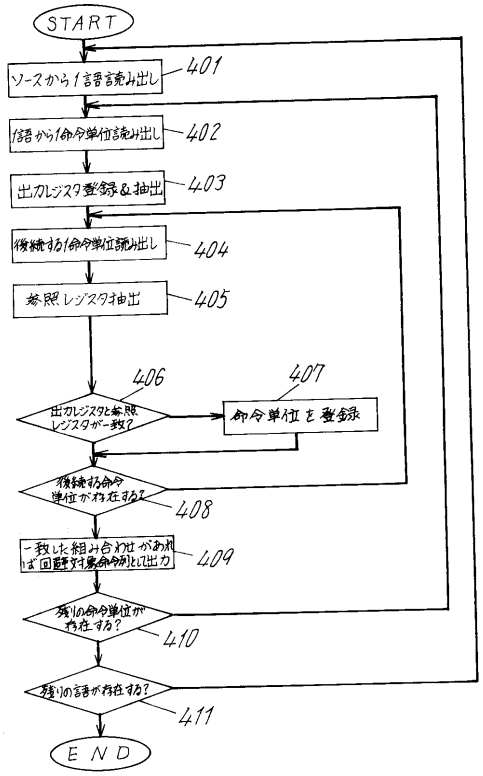
【図５】


t8
t7
t6
t5
t4
t3
t2
t1
add #1,r0,r0
add #123456789,r3,r3
add #1,r2,r2
add #1,r0,r0
mov(mem),r0
0x10000004
タイミング
IF
DEC
EX
MEM
WB

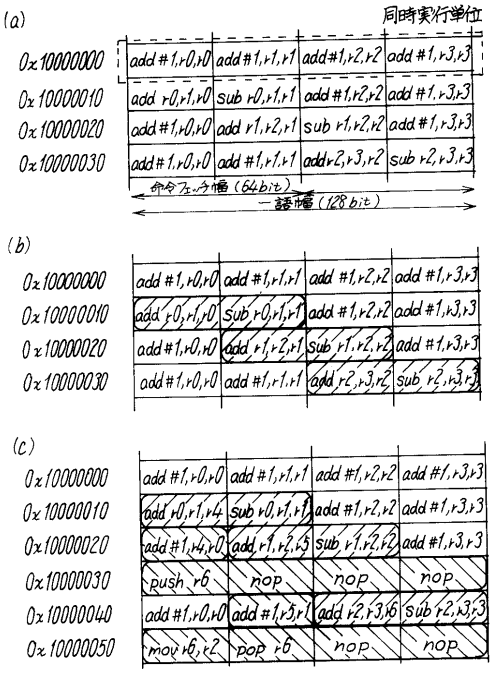
【図６】


ソースコード格納手段
300
320
回避対象コード検出手段
逐次実行保証コード生成手段
330
命令列格納手段
340

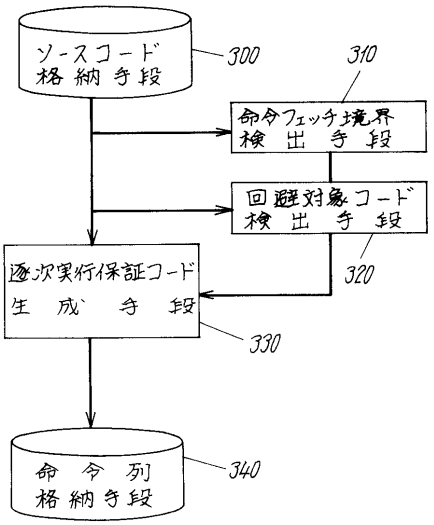
【図７】


START
ソースから1言語読み出し
401
1語から1命令単位読み出し
402
出力レジスタ登録&抽出
403
後続する1命令単位読み出し
404
参照レジスタ抽出
405
406
出力レジスタと参照レジスタが一致?
407
命令単位を登録
後続する命令単位が存在する?
408
一致した組み合わせがあれば回避対象命令列として出力
409
残りの命令単位が存在する?
410
残りの語が存在する?
411
END

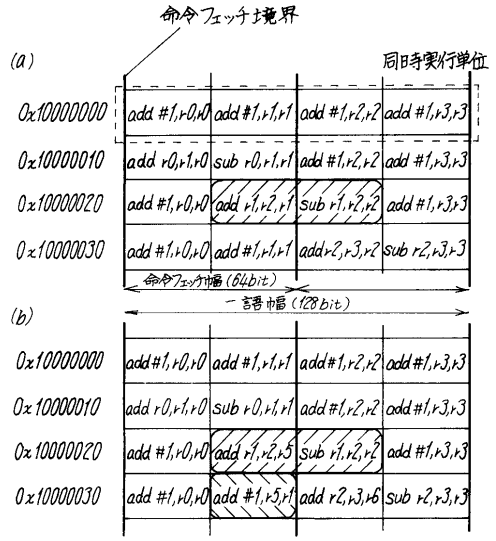
【図８】


(a)
同時実行単位
0x10000000
0x10000010
0x10000020
0x10000030
add #1,r0,r0 | add #1,r1,r1 | add #1,r2,r2 | add #1,r3,r3
add r0,r1,r0 | sub r0,r1,r1 | add #1,r2,r2 | add #1,r3,r3
add #1,r0,r0 | add r1,r2,r1 | sub r1,r2,r2 | add #1,r3,r3
add #1,r0,r0 | add #1,r1,r1 | add r2,r3,r2 | sub r2,r3,r3
命令フェッチ幅(64bit)
一語幅(128bit)
(b)
0x10000000
0x10000010
0x10000020
0x10000030
add #1,r0,r0 | add #1,r1,r1 | add #1,r2,r2 | add #1,r3,r3
add r0,r1,r0 | sub r0,r1,r1 | add #1,r2,r2 | add #1,r3,r3
add #1,r0,r0 | add r1,r2,r1 | sub r1,r2,r2 | add #1,r3,r3
add #1,r0,r0 | add #1,r1,r1 | add r2,r3,r2 | sub r2,r3,r3
(c)
0x10000000
0x10000010
0x10000020
0x10000030
0x10000040
0x10000050
add #1,r0,r0 | add #1,r1,r1 | add #1,r2,r2 | add #1,r3,r3
add r0,r1,r4 | sub r0,r1,r1 | add #1,r2,r2 | add #1,r3,r3
add #1,r4,r0 | add r1,r2,r5 | sub r1,r2,r2 | add #1,r3,r3
push r6 | nop | nop | nop
add #1,r0,r0 | add #1,r3,r1 | add r2,r3,r6 | sub r2,r3,r3
mov r6,r2 | pop r6 | nop | nop

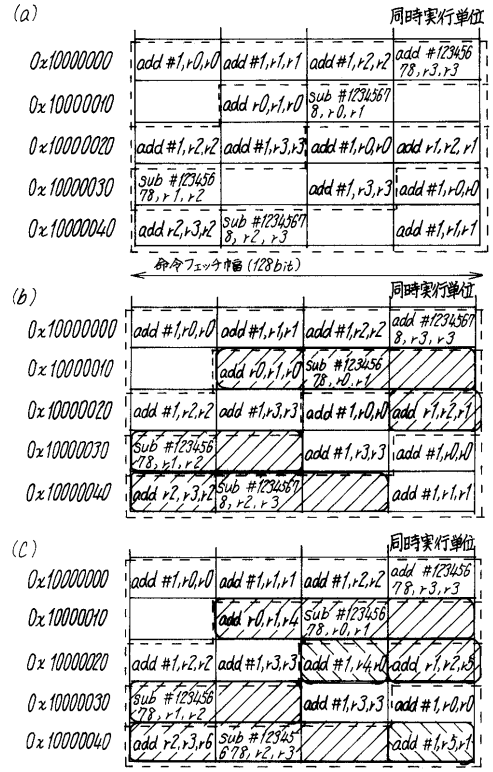
【図９】


ソースコード格納手段
300
310
命令フェッチ境界検出手段
回避対象コード検出手段
320
逐次実行保証コード生成手段
330
命令列格納手段
340

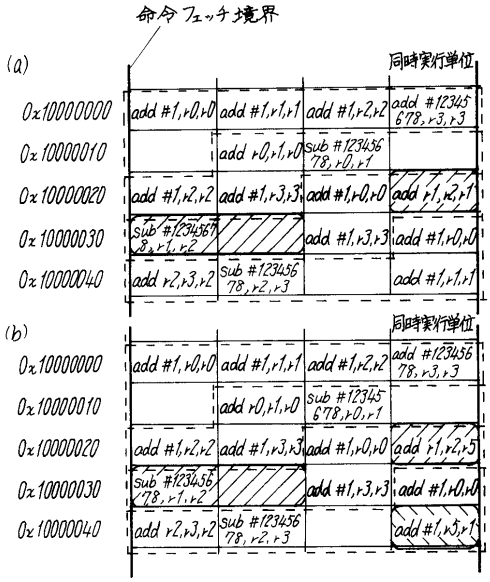
【図１０】


命令フェッチ境界
(a)
同時実行単位
0x10000000
0x10000010
0x10000020
0x10000030
add #1,r0,r0 | add #1,r1,r1 | add #1,r2,r2 | add #1,r3,r3
add r0,r1,r0 | sub r0,r1,r1 | add #1,r2,r2 | add #1,r3,r3
add #1,r0,r0 | add r1,r2,r1 | sub r1,r2,r2 | add #1,r3,r3
add #1,r0,r0 | add #1,r1,r1 | add r2,r3,r2 | sub r2,r3,r3
命令フェッチ幅(64bit)
一語幅(128bit)
(b)
0x10000000
0x10000010
0x10000020
0x10000030
add #1,r0,r0 | add #1,r1,r1 | add #1,r2,r2 | add #1,r3,r3
add r0,r1,r0 | sub r0,r1,r1 | add #1,r2,r2 | add #1,r3,r3
add #1,r0,r0 | add r1,r2,r5 | sub r1,r2,r2 | add #1,r3,r3
add #1,r0,r0 | add #1,r3,r1 | add r2,r3,r6 | sub r2,r3,r3

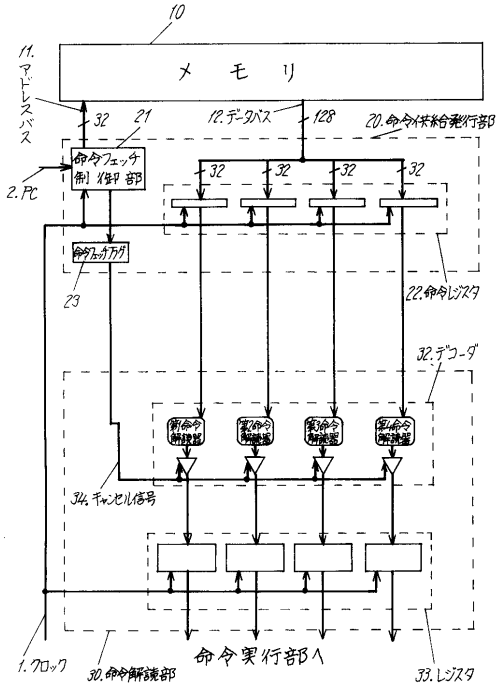
【図１１】



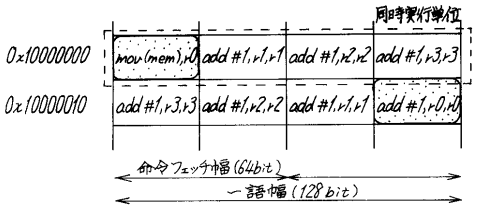
【図１２】



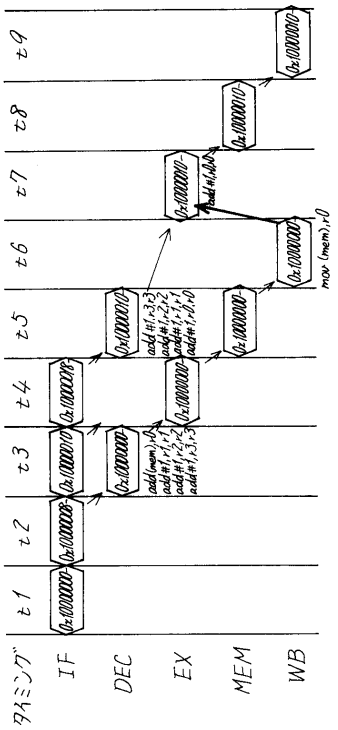
【図１３】



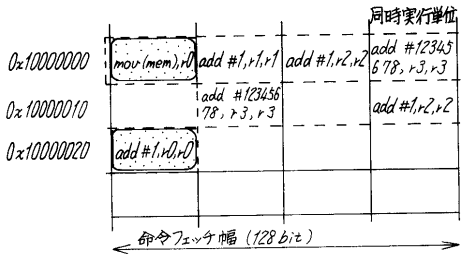
【図１４】



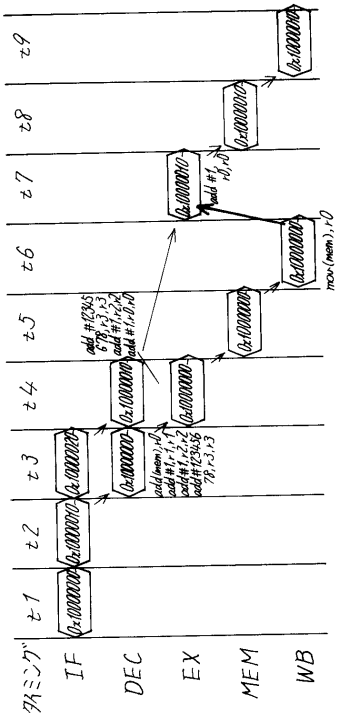
【図１５】


t1
t2
t3
t4
t5
t6
t7
t8
t9
IF
DEC
EX
MEM
WB

【図１６】


同時実行単位
0x10000000
0x10000010
0x10000020
mov (mem),r0
add #1,r1,r1
add #1,r2,r2
add #12345
678,r3,r3
add #123456
78,r3,r3
add #1,r2,r2
add #1,r0,r0
命令フェッチ幅（128bit）

【図１７】


t1
t2
t3
t4
t5
t6
t7
t8
t9
IF
DEC
EX
MEM
WB

フロントページの続き

(72)発明者　田中　哲也
大阪府門真市大字門真１００６番地　松下電器産業株式会社内

審査官　石川　正二

(56)参考文献　特開平０９－１６７０９３（ＪＰ，Ａ）
特開平０９－０２６８７８（ＪＰ，Ａ）
特開昭５９－１８０６３５（ＪＰ，Ａ）
特開平０８－１６１１６９（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F　9/38