(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3745819号
(P3745819)
(45)発行日　平成18年2月15日(2006.2.15)　(24)登録日　平成17年12月2日(2005.12.2)

(51)Int.Cl.　F I
G06F 9/32 (2006.01)　G06F 9/32 310A
G06F 12/10 (2006.01)　G06F 12/10 501A

請求項の数 22　(全 16 頁)

(21)出願番号　特願平8-33669
(22)出願日　平成8年2月21日(1996.2.21)
(65)公開番号　特開平8-292886
(43)公開日　平成8年11月5日(1996.11.5)
審査請求日　平成15年2月19日(2003.2.19)
(31)優先権主張番号　08/395579
(32)優先日　平成7年2月27日(1995.2.27)
(33)優先権主張国　米国(US)

(73)特許権者　595067970
サン　マイクロシステムズ　インコーポレイテッド
ＳＵＮ　ＭＩＣＲＯＳＹＳＴＥＭＳ，ＩＮＣ．
アメリカ合衆国　９５０５４　カリフォルニア州　サンタ　クララ　エムエス　ユーエスシーエイ１２－２０３　ネットワーク　サークル　４１２０
(74)代理人　100068755
弁理士　恩田　博宣
(72)発明者　レスリー　コーン
アメリカ合衆国　９４５３９　カリフォルニア州　フレモント　ローズメアー　ドライブ　４３９６７

最終頁に続く

(54)【発明の名称】ノン・フォルティング・ロード命令をインプリメントする方法及び装置

(57)【特許請求の範囲】

【請求項１】

変換テーブル・エントリを適用する仮想アドレス空間を特定する仮想アドレス部分と、

前記仮想アドレス空間に対応する物理アドレス空間を特定する物理アドレス部分と、

ノン・フォルティング・ロード命令、あるいはノーマル・フォルティング・ロード命令のどちらが前記仮想アドレス空間へのアクセスを行っているのかを表示するノン・フォルト・オンリ・ビットと

を含む変換テーブルを備えたプロセッサであって、

ノン・フォルティング・ロード命令が仮想アドレス空間へのアクセスを行っていることを前記ノン・フォルト・オンリ・ビットが表示している場合、当該プロセッサがノーマル・フォルティング・ロード命令によるアクセスを試みた際にフォルトを形成するフォルト・ユニットを備えたことを特徴とするプロセッサ。

【請求項２】

前記仮想アドレス部分が仮想ページ番号である請求項１に記載のプロセッサ。

【請求項３】

前記物理アドレス部分が物理ページ番号である請求項１に記載のプロセッサ。

【請求項４】

前記変換テーブルは複数の変換テーブル・エントリを含む請求項１に記載のプロセッサ。

【請求項５】

主張が取り消されたノン・フォルト・オンリ・ビットを備えた変換テーブル・エントリは合法仮想アドレスに対応している請求項４に記載のプロセッサ。
【請求項６】
主張が取り消されたノン・フォルト・オンリ・ビットを備えた変換テーブル・エントリは、前記ノーマル・フォルティング・ロード命令のメモリ・アクセス及び前記ノン・フォルティング・ロード命令のメモリ・アクセスの両方がアクセスし得る仮想アドレスに対応している請求項５に記載のプロセッサ。
【請求項７】
仮想ソース・ページ・アドレスに対する変換テーブル・エントリを使用して、ロード命令を処理する方法であって、
前記ロード命令がノン・フォルティング・ロード命令、あるいはノーマル・フォルティング・ロード命令のどちらであるのかを判定する工程と、
前記ロード命令がノーマル・フォルティング・ロード命令であるという判定に応答して、
前記変換テーブル・エントリ内のノン・フォルト・オンリ・ビットが主張されているか否かをテストする工程と、
前記ノン・フォルト・オンリ・ビットが主張されていないことが前記テスト工程で確定された場合、仮想ソース・ページ・アドレスを物理ソース・ページ・アドレスへ翻訳する工程と、
前記ノン・フォルト・オンリ・ビットが主張されていることが前記テスト工程で確定された場合、フォルトを形成する工程と
を含む方法。
【請求項８】
前記ノン・フォルト・オンリ・ビットが主張されていないことが前記テスト工程で確定した場合、前記ロード命令を前記翻訳工程後に実行する工程を含む請求項７に記載の方法。
【請求項９】
前記フォルトを形成する工程はオペレーティング・システム・ソフトウェア・ルーチンへトラップすることによって実現される請求項８に記載の方法。
【請求項１０】
仮想ソース・ページ・アドレスに対する変換テーブル・エントリを使用してロード命令を処理する方法であって、
前記ロード命令がノン・フォルティング・ロード命令であるか否かをテストする工程と、
前記ロード命令がノン・フォルティング・ロード命令である場合、仮想ソース・ページ・アドレスを物理ソース・ページ・アドレスへ翻訳する工程と、
前記ロード命令がノン・フォルティング・ロード命令でない場合、変換テーブル・エントリ内のノン・フォルト・オンリ・ビットが主張されているか否かをテストする工程と、
前記ノン・フォルト・オンリ・ビットが主張されておらず、かつ前記ロード命令がノン・フォルティング・ロード命令でない場合、仮想ソース・ページ・アドレスを物理ソース・ページ・アドレスへ翻訳する工程と、
前記ノン・フォルト・オンリ・ビットが主張され、かつ前記ロード命令がノン・フォルティング・ロード命令でない場合、フォルトを形成する工程と
を含む方法。
【請求項１１】
前記仮想ソース・ページ・アドレスを物理ソース・ページ・アドレスへ翻訳した後で、ロード命令を実行する工程を含む請求項１０に記載の方法。
【請求項１２】
前記フォルトを形成する工程はオペレーティング・システム・ソフトウェア・ルーチンへトラップすることによって実現される請求項１１に記載の方法。

【請求項１３】
仮想ソース・ページ・アドレスに対する変換テーブル・エントリを備え、かつロード命令を処理する装置であって、
前記ロード命令がノン・フォルティング・ロード命令、あるいはノーマル・フォルティング・ロード命令のどちらであるのかを判定するロード・タイプ判定回路と、
ノーマル・フォルティング・ロード命令である前記ロード命令に応答して前記変換テーブル・エントリ内のノン・フォルト・オンリ・ビットが主張されているか否かをテストするテスト回路と、
前記ノン・フォルト・オンリ・ビットが主張されていないことが前記テスト回路によって確定された場合、仮想ソース・ページ・アドレスを物理ソース・ページ・アドレスへ翻訳する翻訳回路と、
前記ノン・フォルト・オンリ・ビットが主張されていることが前記テスト回路によって確定された場合、フォルトを形成するフォルト・ユニットと
を含む装置。
【請求項１４】
前記ノン・フォルト・オンリ・ビットが主張されていないことが前記テスト回路によって確定された場合、翻訳回路が仮想ソース・ページ・アドレスを物理ソース・ページ・アドレスへ翻訳した後で、前記ロード命令を実行する実行ユニットを含む請求項１３に記載の装置。
【請求項１５】
前記フォルト・ユニットはオペレーティング・システム・ソフトウェア・ルーチンへのトラップを形成する請求項１４に記載の装置。
【請求項１６】
仮想ソース・ページ・アドレスに対する変換テーブル・エントリを備え、かつロード命令を処理する装置であって、
前記ロード命令がノン・フォルティング・ロード命令であるか否かをテストする第１のテスト回路と、
前記ロード命令がノン・フォルティング・ロード命令であることと、前記ノン・フォルト・オンリ・ビットが主張されておらず、かつ前記ロード命令がノン・フォルティング・ロード命令でないことのうちのいずれか一方が生じた際に、仮想ソース・ページ・アドレスを物理ソース・ページ・アドレスへ翻訳する翻訳回路と、
前記ロード命令がノン・フォルティング・ロード命令でない場合、変換テーブル・エントリ内のノン・フォルト・オンリ・ビットが主張されているか否かをテストする第２のテスト回路と、
前記ノン・フォルト・オンリ・ビットが主張され、かつ前記ロード命令がノン・フォルティング・ロード命令でない場合、フォルトを形成するフォルト・ユニットと
を含む装置。
【請求項１７】
前記仮想ソース・ページ・アドレスを物理ソース・ページ・アドレスへ翻訳した後で、前記ロード命令を実行する実行ユニットを含む請求項１６に記載の装置。
【請求項１８】
前記フォルト・ユニットはオペレーティング・システム・ソフトウェア・ルーチンへのトラップを形成する請求項１７に記載の装置。
【請求項１９】
変換テーブルをロード命令の処理中に維持する方法であって、
前記ロード命令が要求する仮想アドレスが違法であるか否かをテストする工程と、
前記ロード命令がノン・フォルティング・ロード命令であるか否かをテストする工程と
、
前記仮想アドレスが違法であって、かつ前記ロード命令がノン・フォルティング・ロード命令である場合、主張されたノン・フォルト・オンリ・ビットを備えた仮想アドレスに

対する変換テーブル・エントリを形成する工程と
を含む方法。
【請求項２０】
前記変換テーブル・エントリは仮想アドレスがゼロを含む物理アドレスをマップするように形成されている請求項１９に記載の方法。
【請求項２１】
変換索引バッファを前記変換テーブル・エントリで満たす工程を含む請求項２０に記載の方法。
【請求項２２】
前記変換テーブル・エントリは仮想アドレスのページに対して適用される請求項１９に記載の方法。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
本発明は命令を推論に基づいて実行するパイプライン・プロセッサ、より詳細には、ノン・フォルティング・ロード命令（Non-faulting load instruction）及びノーマル（即ち、フォルティング）ロード命令（Normal (faulting) load instruction）をサポートすることに関する。
【０００２】
【従来の技術及び発明が解決しようとする課題】
初期の一般的なマイクロプロセッサは中央演算処理装置（ＣＰＵ）を含む。ＣＰＵはメモリとの間のインターフェースを提供し、かつ命令を連続的に順次実行し得る装置を実現する。図１は最近のプロセッサの簡単なブロック図である。一般的に、命令実行は少なくとも４つの主なステージ、即ち、命令及びオペランドのフェッチ１、命令の復号２、実行３、並びに結果の書き戻し４に分割される。最大クロック周波数は４つの全てのステージにおける最も長い伝搬遅延によって決定される。
【０００３】
パイプラインの概念は各クロック・サイクルにおいて実施されるロジックの総数を削減することにより最大クロック周波数を増加させる。４つのステージをそれぞれ別のクロック・サイクルにおいて実行し、これによりスループットを増加すべくパイプラインの概念を拡大できる。また、スーパーパイプラインのデザインでは、命令実行を４個を越す数のパイプライン・ステージに分割して、任意の１つのステージを通じて形成される最大伝搬遅延を短縮し、これにより動作周波数及びスループットを増加させ得る。これを実現すべく、前記の４個のステージの全てまたは一部に含まれるロジックが分割される。プロセッサ・ロジックを同一の待ち時間をそれぞれに有するＮ個のステージへ分割することにより、理論的にはスループットをＮ倍に増やせる。
【０００４】
スーパースカラ・マイクロプロセッサは複数の実行ユニットを有する。２個または３個の命令を同時に実行するスーパースカラ・プロセッサのデザインが一般に使用されている。更に４個の命令を同時に実行するデザインが既に存在する。将来的には、コンピュータ設計者が８～１６個の命令を同時に実行し得るチップを設計するであろう。殆どのスーパースカラ・マイクロプロセッサは特別な複数のパラレル実行分岐５を提供している。しかし、個々のパラレル実行分岐は命令セット全体をサポートできない。その代わり、各パラレル実行分岐は命令セットのサブセットを処理すべく設計されている。
【０００５】
パイプライン・プロセッサ及びスーパースカラ・プロセッサに付随する基本的な問題は実行シーケンスに含まれる条件付き分岐の処理に関連する。殆どのプログラムは多くの条件付き分岐を含む。このようなプログラムをパイプラインで実行する場合、最終的に実行する分岐を正確に確定する方法がない。また、分岐命令の判断結果は同命令が実行されない限り得られない。この結果、分岐命令実行前に、同分岐命令直後に実行する命令を確定す

る方法は存在しない。フェッチ・ステージ１は分岐命令後に次の命令をパイプライン内へ既にフェッチしているため、分岐後の一連の命令の中止が必要となり得る。この問題は分岐実行ステージ以前のステージ数の増加に伴って更に悪化する。従って、プロセッサの処理速度を向上すべくパイプラインが更に長くなった場合、分岐の問題は更に複雑になる。

【０００６】

スーパースカラ・プロセッサはパイプライン化されているか否かに拘わらず同様の問題を有し得る。スーパースカラ・プロセッサは任意の時点において複数の命令を実行できる。従って、条件付き分岐命令を一方の実行ユニットで実行する間に、同条件付き分岐命令の条件が満たされた場合にのみ実行される別の命令を他方の実行ユニットで実行できる。条件付き分岐命令の条件が満たされないことにより、命令を他方の実行ユニット内で実行すべきでない事態が発生した際、アーキテクチャはこのコンティンジェント命令（Contingent instruction）を中止またはリバースする方法を提供する必要がある。コンティンジェント命令がレジスタ位置またはメモリ位置への書き込みを行い、かつ以前の内容を破壊した場合、システムは同内容を回復する必要がある。これに代えて、中間記憶位置を使用することにより、潜在的に重要なレジスタ位置またはメモリ位置に対する上書きを、全てのコンティンジェンシ（Contingencies）を解決するまで保留すべくスーパースカラ・プロセッサのコンパイラを設計し得る。

【０００７】

しかし、コンティンジェンシを確定するまでメモリ内容を破壊しないようにコンパイラを形成したとしても、別の問題が存在する。例えば、以下のＣプログラム・フラグメントは概念上少なくとも２つの部分に分割できる。

【０００８】

ｉｆ（ｐ ！＝ＮＵＬＬ）ｘ＝*ｐ＋ｙ；

Ｃプログラム言語において、*ｐはｐがポインタであって、さらには*ｐがｐによって示された位置の内容であることを表す。第１の部分は変数ｐを格納するメモリ位置がＮＵＬＬ記号で表されたコードを格納していないことを確認する条件付きテストである。条件が真である場合、第２の部分は変数ｙと、メモリ位置ｐに格納されたアドレスを有するメモリ位置に格納された数値との合計を、変数ｘを保持するメモリ位置に代入する。前記したように、条件付き分岐命令の結果を確定するまでは、変数ｘの現在の数値を上書きしないことによって同数値の破壊を防止すべくコンパイラは命令の順番付を行う。これにより、ｘの現在の数値が維持される。従って、前記の条件が偽である場合、回復は全く必要ない。この結果、コンパイラは前記のフラグメントを以下のシーケンスに分解し得る。

【０００９】

ｔｅｍｐ＿ｒｅｇｉｓｔｅｒ＝*ｐ；

ｉｆ（ｐ！＝ＮＵＬＬ）ｘ＝ｔｅｍｐ＿ｒｅｇｉｓｔｅｒ＋ｙ；

ｔｅｍｐ＿ｒｅｇｉｓｔｅｒへの代入と、ｐがＮＵＬＬでないことを確認するテストとを異なる実行ユニットにおいて同時に実施すべく、前記の命令をスケジュールし得る。しかし、前記のシーケンスを使用しても、以下の問題が依然発生し得る。即ち、ｐがＮＵＬＬの場合、同ｐは何も指していない。このため、ｐが指す位置の内容を読み出す試みが無意味になる。そして、このｐの内容に対するノーマル・アクセスは多くのプロセッサにおいてプログラムの中止を引き起こす。

【００１０】

この問題を解決すべく、ノー・フォルト・ローディング（No-fault loading）の概念が導入された。ノー・フォルト・ローディングでは、ｐがＮＵＬＬであって、かつ命令が*ｐのロードを試みる場合、ロードはフォルト、即ちエラーを形成する代わりにゼロなどの所定の数値を返す。違法アドレスからのノン・フォルティング・ロードによって返された数値を後から使用することは殆どない。このため同ノン・フォルティング・ロードによって返された数値は重要ではない。誤って予測され、かつノン・フォルティング・ロードを引き起こした分岐は同ロードによって返されたデータを使用する前に解決される。従って、

同データが使用されることは殆どない。また、返されたデータが使用されたとしても、次の命令の実行結果は放棄される。この結果、違法アドレスのノン・フォルティング・ロードによって返された数値は、同数値が後から使用された場合にも例外を形成しないよう選択すべきである。

【００１１】

ノン・フォルティング・ロードはその使用を防止する条件付き制御構造（Conditional control structures）の先へロードを移行させる最適化を可能にする。この結果、ノン・フォルティング・ロードは命令のスケジューリングを改善することによりロードの待ち時間が及ぼす影響を最小限に抑制し得る。ノン・フォルティング・ロードの意味は、回復不能なフォルト（例：アドレス・アウト・オブ・レンジ・エラー）が生じた場合を除いて他の全てのロードの意味と同一である。回復不能なフォルトが生じた場合、同フォルトは無視され、ハードウェア及びシステム・ソフトウェアは互いに協働してゼロを返すことによりロードが正常に完了したように見せかける。コンパイラのオプティマイザはノン・フォルティング・ロードと断定したロードに対してロード変更命令を形成する。これはクリティカル・コード・パスからアドレス確認を排除することにより特定のアルゴリズムにおける性能の改善を可能にする。ポインタの“間接参照（Dereferencing）”は同ポインタが指す位置の内容に対するアクセスを試みる。リンクされたリストに従う場合、ノン・フォルティング・ロードは、同リストの末端を指すヌル・ポインタを推論に基づいて先読みした際に、同ポインタを安全に間接参照することを可能にする。これにより仮想アドレス０ｘ０に存在するページに対してペナルティを受けることなく安全にアクセスできる。

【００１２】

最近のプロセッサは物理アドレス空間とは異なる仮想アドレス空間をサポートしている。仮想アドレスはプロセッサがメモリ位置を特定するために使用するラベルである。プロセッサは仮想アドレスを用いてメモリ位置へアクセスできる限り、物理メモリ内におけるメモリ位置の実際の場所を問題にしない。プロセッサ・アーキテクチャの仕様は、サポートを要する特定の仮想アドレス空間を限定する。コンピュータ・システムを管理するオペレーティング・システムは、仮想アドレス空間を物理メモリにマップする方法に関して柔軟性を有する。従って、仮想アドレスから物理アドレスへの翻訳が必要になる。

【００１３】

オペレーティング・システムは翻訳情報を“ソフトウェア変換テーブル（Software translation table）”と称される任意のデータ構造内に維持している。１つ以上の変換索引バッファ（以下、ＴＬＢと称する）がソフトウェア変換テーブルの独立したキャッシュとして使用されている。変換索引バッファの使用により、頻繁にアクセスする仮想ページを１サイクルで翻訳できる。“ＴＬＢヒット”は所望の翻訳がオンチップＴＬＢ内に存在することを意味する。“ＴＬＢミス”は所望の翻訳がオンチップＴＬＢ内に存在しないことを意味する。ＴＬＢは小型、かつ高速である。ソフトウェア変換テーブルは大きく、かつ複雑であり得る。直接マップされたキャッシュ（Direct-mapped cache）として機能する変換格納バッファ（Translation storage buffer、略してＴＳＢ）はＴＬＢ及びソフトウェア変換テーブル間のインターフェースとして機能する。

【００１４】

ＴＬＢミスが生じた場合、プロセッサはＴＬＢミス処理を実施すべくオペレーティング・システム・ソフトウェアへ即座にトラップする。ＴＬＢミス・ハンドラは利用可能な任意の手段を用いてＴＬＢを満たすオプションを有している。ＴＬＢが所望のアドレスの翻訳で満たされた後、ＴＬＢミスを初めに引き起こした命令が再実行される。

【００１５】

一般的なＴＬＢミス・ソフトウェアはＴＳＢへのアクセスに必要な複数の特定のポインタを最初にロードする。ハンドラはこれらのポインタを使用して、所望の翻訳がＴＳＢ内に存在するか否かを確認すべくＴＳＢ内の適切な位置をチェックする。所望の翻訳がＴＳＢ内に存在する場合、同翻訳は置換アルゴリズムに基づいて選択されたＴＬＢエントリ内へロードされる。翻訳がＴＳＢ内に存在しない場合、ＴＬＢミス・ハンドラは更に精巧、か

つ処理速度の遅いＴＳＢミス・ハンドラへジャンプする。ＴＳＢミス・ハンドラは大きく、かつ複雑なソフトウェア変換テーブル内において所望の翻訳を検索する。所望の翻訳がソフトウェア変換テーブル内に存在しない場合、所望のメモリ位置はメイン・メモリ内にマップされない。この場合、アドレスは違法であり得る。これに代えて、特定された仮想アドレスによって参照されたデータが合法であり、かつディスク等の別の記憶装置内に存在することがある。アドレスが合法であって、かつメイン・メモリ内に存在しない場合、要求されたデータをメイン・メモリ内にマップし、さらにはＴＬＢミスを引き起こした命令が実行可能となる前に、該当する翻訳をＴＬＢ内へ格納する必要がある。

【００１６】

ノン・フォルティング・ロードがＴＬＢミスに直面した場合、オペレーティング・システムはページの翻訳を試みる必要がある。翻訳がエラー（例：アドレス・アウト・オブ・レンジ）を形成した場合、ゼロが返され、ロードは静的に完了する。従来の幾つかの方式では、所望のアドレスが違法な場合、同アドレスへアクセスするノン・フォルティング・ロード命令では、違法ページに対するマッピングは存在しない。従って、ＴＬＢミスが生じる。次いで、ソフトウェア・トラップはＴＳＢミスを形成する。そして、アドレスが違法であることを最終的に確定するには比較的長い時間が必要である。この結果、プロセッサによる他の有用な作業の実施が阻害される。プロセッサが合法なページのマッピングのみを許容する場合、違法なアドレスからのノン・フォルティング・ロードは返しに長い時間を必要とするトラップを常に形成する。従って、ヌル・ポインタに対する間接参照は最終的にゼロを返す前に長い時間を要するソフトウェア・トラップを招来する。

【００１７】

図２はロード命令を処理する従来の方法の流れ図である。一般的に、破線９によって囲まれた全てのステップはハードウェア内に実装されている。ロード命令はステップ１０において最初にディスパッチされる。テスト１１では、比較オペレーションが対応するハードウェアを用いて全ての変換テーブル・エントリに対して同時に実施される。現在のロードが要求する仮想ページ番号がＴＬＢ内において探索される。ＴＬＢヒットが存在する場合、ＴＬＢは適切な物理ページをステップ１２において提供する。次いで、ロード命令がステップ１３において実行される。そして、同ロード命令はステップ１４において完了する。

【００１８】

該当する仮想アドレスがＴＬＢ内に存在しない場合、ＴＬＢミス・ハンドラがステップ１５において呼び出される。テスト１６において、ソフトウェアは変換テーブル内において翻訳を探索する。該当する翻訳が変換テーブル内に存在する場合、ソフトウェアはＴＬＢを適切な翻訳で満たし、ロードが実行される。また、同翻訳が変換テーブル・データ構造の全てのレベルに含まれていない場合、ソフトウェアはアドレスが合法であるか否かをテスト１８で確定する。アドレスが合法である場合、回復可能なページ・フォルトがステップ１９で発生する。ページ・フォルトが発生した場合、要求されたページはディスクまたは別の外部記憶装置内に存在し得る。そして、ページ・フォルトから回復すべく同ページをマップし、かつメイン・メモリ内へ転送する必要がある。アドレスが違法である場合、テスト２０は現在の命令がノン・フォルティング・ロードであるか否かを確定する。現在の命令がノーマル（フォルティング）ロードである場合、違法アドレスは回復不能なフォルトをステップ１９で形成する。しかし、現在の命令がノン・フォルティング・ロードである場合、ソフトウェアはステップ２１においてゼロを返す。そして、ロード命令はステップ１４において円滑に完了する。

【００１９】

ゼロを返す処理速度を向上する１つの方法としては、全てがゼロであるページに対してアドレス・ゼロをマッピングすることが挙げられる。この方式を使用することにより、デザインはソフトウェア変換テーブル内に存在する全てのマッピングが合法ページに対するものとする標準的概念から離れる。このアプローチを使用した場合、違法ページはＴＬＢ内に翻訳を有し得る。ヌル・ポインタに対する間接参照が行われた場合、違法ページの翻訳

はＴＬＢ内に存在するため、ＴＬＢヒットが生じる。そして、ソフトウェアへのトラップを伴うことなく即座にゼロが返される。実質的に、このアプローチは頻繁にアクセスする特定の違法アドレスを合法アドレスへ単に変換する。

【００２０】

ノン・フォルティング・ロードによって頻繁にアクセスされない他の違法アドレスをノーマル・ロードがアクセスした際にフォルトを形成するように同違法アドレスをマップする必要はない。これらの違法アドレスはＴＬＢミス及びソフトウェア・トラップを招来し、最終的にはゼロが返される。ノン・フォルティング・ロードが最も頻繁にアクセスするページ（例：０ｘ０）のみをマップする必要がある。これにより、ノン・フォルティング・ロードがもたらす更に短い待ち時間の恩恵を享受できる一方、ノーマル・ロードが不正確に作用するリスクを最小限に抑制できる。

【００２１】

ノン・フォルティング・ロード命令をサポートするプロセッサでは、ノーマル（フォルティング）ロードも併せてサポートする必要がある。しかし、全てがゼロであるページをマップした場合、ノーマル・ロードは他のメカニズムが存在しない限り同ページへアクセスし得る。技術的に、これは誤りである。このアプローチに付随する問題点としては、ノーマル・ロードも違法ページへアクセスできる点が挙げられる。ノーマル（フォルティング）ロードは同ロードが違法ページへのアクセスを試みた際、ゼロを返すよりも、寧ろフォルトを形成する必要がある。この方式を用いた場合、プログラマは違法ページからのノン・フォルティング・ロードにおいて迅速にゼロを返すことと、違法ページへアクセスした際にゼロを返すより、寧ろフォルトを形成して正確に作用するノーマル（フォルティング）ロードを有することのうちのいずれか一方をページ単位で選択する必要がある。

【００２２】

多くのプログラマはコードをデバッグするためにアドレス０ｘ０へアクセスした際に生じる例外を考慮する。仮想アドレス０ｘ０は“ヌル”であり、常には違法なアドレスである。一般的なプログラムに多く発生するバグは、ヌル・ポインタに対する間接参照を試みる。ソフトウェア開発のデバッグ段階において、プログラマはヌル・ポインタを間接参照した際にフォルトを形成すべくノーマル（フォルティング）ロードをコード内に使用する。プログラマはフォルトの発生後、誤りの原因を確定すべくマシーンの状態を検査できる。

【００２３】

プログラマはプログラムを完全にデバッグした後、ロードを条件付き分岐の先へ移行することにより実行速度を向上すべく、ノン・フォルティング・ロードを機械コード内に使用できることをコンパイラへ通知する。ノン・フォルティング・ロードの使用はデバッグ済みプログラム内において大きな問題となる回復不能なフォルトを防止する。しかし、ノン・フォルティング・ロードはアドレス０ｘ０（または他の任意の違法アドレス）へアクセスする際、ページがメモリ位置０ｘ０に対してマップされていない限りソフトウェア・トラップを引き起こす。これらの反復して実施されるトラップは非常に長い時間を要し得る。フォルト発生時における不必要な処理を最小限に抑制すべく、全てがゼロのページ（Page of all zeros）に対して小さい数値のアドレス（例、特にアドレス・ゼロ）をマップすることが好ましい。この結果、ヌル・ポインタを介した参照は不必要なトラップを形成しない。従って、ソフトウェア・トラップは全てがゼロのページをメモリ位置０ｘ０へマップすることによって回避できる。

【００２４】

最近のプログラム開発はステージ毎に実施される。特定のルーチンまたはプロシジャは他の関連するソフトウェアが完了し、かつデバッグされる前にデバッグされる。多くの場合、デバッグしたプロシジャはプログラム内にバグが含まれた状態でコンパイルする必要がある。全てがゼロであるページがメモリ位置０ｘ０に対してマップされている場合、ヌル・ポインタを間接参照した際に、フォルトを形成する代わりにゼロが返される。このため、開発中のプログラムまたはプロシジャがフォルトを形成することの利点が失われる。アドレス０ｘ０は違法であり、かつノーマル・ロードに対してゼロを返してはならない。こ

のため、このゼロを返す動作は誤っている。他のメカニズムが存在しない限り、フォルティング・ロード及びノン・フォルティング・ロードを含む全てのロードは、全てがゼロであるページをメモリ位置０ｘ０へマップした場合、ヌル・ポインタを間接参照した際にゼロを返す。

【００２５】

本発明は前述した事情に鑑みてなされたものであって、その目的は、ノン・フォルティング・ロード命令及びノーマル（フォルティング）ロード命令の迅速かつ正確な処理を可能にする装置及び方法を提供することにある。

【００２６】

【課題を解決するための手段】

本発明の変換テーブル・エントリは、同変換テーブル・エントリを適用する仮想アドレス空間を特定する仮想アドレス部分と、仮想アドレス空間に対応する物理アドレス空間を特定する物理アドレス部分と、仮想アドレス空間が違法であることを表示するノン・フォルト・オンリ・ビット（Non-fault-only bit）とを含む。そして、本発明の変換テーブルは変換テーブル・エントリを複数有する。仮想アドレス部分を仮想ページ番号とし、物理アドレス部分を物理ページ番号とすることができる。

【００２７】

本発明の１つの態様では、ノン・フォルト・オンリ・ビットはページ毎に提供されており、同ノン・フォルト・オンリ・ビットをセットした場合、そのページに対してアクセスするノン・フォルティング・メモリ・アクセスとしてのノン・フォルティング・ロードは翻訳を招来する。ノン・フォルト・オンリ・ページに対する他の全てのアクセスはエラーであり、プロセッサはオペレーティング・システム・ソフトウェア・トラップを介してフォルトを形成する。ノン・フォルティング・ロードは、ノン・フォルト・オンリ・ビットがセットされたページに対してアクセスした際にもフォルトを形成しない点を除けば、ノーマル・ロードのように作用する。変換テーブル・エントリ内のノン・フォルト・オンリ・ビットは、ノン・フォルティング・ロードによる安全なアクセスを提供すべくマップされたページをマークする。しかし、これは他のノーマル・アクセスが行われた際に依然フォルトを形成し得る。

【００２８】

ノーマル・メモリ・アクセスとしてのノーマル（フォルティング）ロードは、主張されたノン・フォルト・オンリ・ビット（Asserted non-fault-only bit）によってマークされた違法ページに対するアクセスを試みた際にフォルトを形成する。

【００２９】

ノン・フォルト・オンリ・ビットは違法なページを表示する。変換索引バッファ・ヒットが発生した後、全ての結果はハードウェア・サポートを有する。このため、全てのノーマル・ロード及びノン・フォルティング・ロードはハードウェア内において迅速に処理され、かつ正確に実行される。仮想ページ０ｘ０等の選択されたページを変換テーブル内でマップできる。アクセスが一度行われると、仮想ページ０ｘ０に対する変換テーブル・エントリは変換索引バッファ内に存在する。この結果、ノン・フォルティング・ロードがヌル・ポインタを間接参照した場合、ＴＬＢヒットが生じる。このため、要求されたページを発見すべくソフトウェアへトラップすることなくゼロが即座に返される。

【００３０】

本発明の別の態様では、ＴＬＢミスによって呼び出されたオペレーティング・システム・ソフトウェア・ルーチンを使用することにより、過去に変換テーブル内へ翻訳されたことがない違法な仮想ページに対するアクセスをノン・フォルティング・ロードが試みたことが発見された場合、オペレーティング・システムは同仮想ページに対する変換テーブル・エントリを形成する。変換テーブル・エントリは前記の仮想ページを全てがゼロである物理ページに対してマップし、さらには同仮想ページに対するノン・フォルト・オンリ・ビットを主張、即ちセットする。その後実行される同仮想ページからのノン・フォルティング・ロードは、ＴＬＢヒットを形成し、かつＴＬＢミス・ハンドラを回避することと、前

記の違法な仮想ページに対応する変換テーブル・エントリを探索すべくソフトウェア変換テーブル全体に対して実施される高い費用及び長い時間を要する不成功なソフトウェア・サーチを少なくとも回避することのうちの一方を招来する。

【００３１】

【発明の実施の形態】

本発明に基づくノン・フォルト・オンリ・ビットはページ毎に形成された変換テーブル・エントリ内に存在する。ノン・フォルト・オンリ・ビットをセットした場合、該当するページにアクセスするノン・フォルティング・ロードは翻訳を招来する。ノン・フォルト・オンリ・ページに対する他の全てのアクセスはエラーであり、プロセッサはオペレーティング・システム・ソフトウェア・トラップを介してフォルトを形成する。ノン・フォルティング・ロードは、ノン・フォルト・オンリ・ビットがセットされたページに対してアクセスした際にもフォルトを形成しない点を除けば、ノーマル・ロードのように作用する。

【００３２】

図３は本発明に基づく変換テーブル・エントリ２２を示す。各エントリは仮想アドレス部分２３を有している。変換テーブル内のエントリは仮想アドレスに基づいて編成されるとともに、同仮想アドレスに対する物理アドレス・マッピングを有する。一般的に、メモリは複数のページに分割されている。このため、仮想アドレス２３及び対応する物理アドレス２４の各ビットのうちの最も重要なビットのみが変換テーブル・エントリに格納されている。前記の仮想アドレス２３及び物理アドレス２４の各ビットのうち、変換テーブル・エントリに格納されなかった最も重要度の低いビットは、メモリ・ページ内にインデックスまたはオフセットを提供する。ノン・フォルト・オンリ・ビット２５はエントリ毎に存在する。別の情報２６も変換テーブル・エントリ内に存在し得る。

【００３３】

主張されたノン・フォルト・オンリ・ビットを備えたページに対して実施されるノーマル・アクセス、即ち同ページからのロードはエラー・ハンドリング・ソフトウェア・トラップを招来し、同トラップはエラーを形成する。従って、変換テーブル・エントリ内のノン・フォルト・オンリ・ビットは特定のページをマークする。同ページはノン・フォルティング・ロードによる安全なアクセスを提供すべくマップされるとともに、同ページに対する他のノーマル・アクセスが行われた際にフォルトを引き起こし得る。これにより、デバッグ中のソフトウェアが使用するデバッグ済みライブラリ・ルーチンへのノン・フォルティング・アクセスを加速することによって得られる効果が維持される一方で、プログラマはデバッグ段階においてワイルド・ポインタ・リファレンスへのトラップが可能である。ノーマル（フォルティング）ロードは、主張されたノン・フォルト・オンリ・ビットによってマークされた違法ページに対するアクセスを試みた際にフォルトを形成する。このため、正確なプログラムの実行が保証される。

【００３４】

ノン・フォルト・オンリ・ビットは違法なページを表示する。従って、全てがゼロである物理ページ（例えば、物理アドレス０ｘＦに位置する）が仮想メモリ位置０ｘ０に対してマップされた場合、仮想ページ０ｘ０に対応する変換テーブル・エントリ内のノン・フォルト・オンリ・ビットが主張される。ノン・フォルティング・ロードがヌル・ポインタを間接参照した場合、仮想アドレス０ｘ０は適切な物理ページ（０ｘＦ）へ翻訳される。そして、同ページに対応するメモリ位置の内容、即ちゼロが返される。しかし、ノーマル・フォルティング・ロードがヌル・ポインタを間接参照した場合、仮想アドレス０ｘ０に対する変換テーブル・エントリ内に存在する主張されたノン・フォルト・オンリ・ビットにより、ページ０ｘ０は違法である旨マークされている。このため、仮想アドレス０ｘ０は翻訳されない。この違法ページに対するノーマル（フォルティング）ロード・アクセスはトラップを招来し、かつフォルトを形成する。従って、プログラマはノーマル（フォルティング）ロードの正確な作用に影響を及ぼすことなく、ノン・フォルティング・ロードに対して迅速にゼロを返すためにハードウェア・サポートがもたらす効果を享受できる。

【００３５】

図４はヌル・ポインタを全てがゼロであるページに対してマッピングするエントリ３１を有する小さな変換索引バッファ３０を示している。そして、エントリ３１に対するノン・フォルト・オンリ・ビット３２は主張された状態にある。更に、図４は主張が取り消されたノン・フォルト・オンリ・ビット（Deasserted non-fault-only bit）３４を有する合法な仮想アドレスに対するエントリ３３を示す。図５は合法なアドレス（“Ｙ”で表す）と、ヌル・ポインタの間接参照値（“Ｘ”で表す）とをそれぞれノーマル（フォルティング）ロード及びノン・フォルティング・ロードした際の結果を示す。ここで、ノーマル（フォルティング）ロードは“ＬＯＡＤ”を用いて示し、ノン・フォルティング・ロードは“ＮＦ＿ＬＯＡＤ”で示す。

【００３６】

ヌル・ポインタに対するＴＬＢエントリ３１に示すように、ハードウェア・サポートは違法ページへアクセスするノン・フォルティング・ロードに対して迅速にゼロを返すべく提供されている。ノン・フォルト・オンリ・ビットは仮想ページが合法であるか否かを実質的に表示する。ノン・フォルト・オンリ・ビットが主張された場合、仮想ページは違法である。主張されたノン・フォルト・オンリ・ビットを有する仮想ページに対してノーマル・ロードがアクセスを試みた場合、プロセッサは前記のソフトウェアへトラップし、かつフォルトを形成する。

【００３７】

図６（ａ）はアクセスしたページに対する変換テーブル・エントリが存在する場合に、命令の実行及びフォルトの形成のうちのいずれを実施するか確定すべく使用するロジックの真理値表を示す。更に、図６（ｂ）は前記のロジックを実施する簡単なロジック回路５５を示す。ロジック回路５５は３つの論理素子から構成され、２つの入力（ＮＦＬ，ＮＦＯ）及び２つの出力（ＴＲＡＮＳＬＡＴＥ，ＦＡＵＬＴ）を備えている。出力ＴＲＡＮＳＬＡＴＥが論理レベル“１”の時は翻訳を行い、同出力が論理レベル“０”の時は翻訳を行わない。また、出力ＦＡＵＬＴが論理レベル“１”の時はフォルトを形成し、同出力が論理レベル“０”の時はフォルトを形成しない。

図７はプロセッサ内においてロード命令を実行する際の本発明に基づく事象のシーケンスを示す。本発明のロード命令処理はステップ１００において開始される。ステップ１０１は、ロードするメモリ位置についての仮想から物理ページへの翻訳に対するノン・フォルト・オンリ（ＮＦＯ）ビットを含む変換テーブル・エントリが、変換索引バッファ（ＴＬＢ）内に存在するか否かを確認する。該当するエントリが変換索引バッファ内に存在しない場合、プロセッサはオペレーティング・システム・ソフトウェア・ルーチンであるＴＬＢミス・ハンドラ１０２へトラップする。ＴＬＢミス・ハンドラはロード命令が要求する仮想ページに対する変換テーブル・エントリを探索すべくソフトウェア変換テーブルの別のレベルを調べる。ソフトウェア変換テーブル内に翻訳が存在しない場合、テスト１０３はテスト１１４を呼び出す。テスト１１４はアドレスが真に違法であるか否かを確定する。アドレスが合法であって、かつ単にマップされていない場合、回復可能なフォルト１０４が形成される。例えば、これはメモリ位置がメイン・メモリより寧ろディスク上に存在する場合に発生し得る。この結果、ページ・フォルトが発生する。ページ・フォルトが発生した場合、適切なページがディスクから読み込まれ、同ページは物理メモリに対してマップされる。そして、その仮想アドレスに対する変換テーブル・エントリが形成され、同エントリは変換索引バッファ内に格納される。ロード命令を再実行した後、プロセッサはフォルトから回復する。要求された仮想アドレスが真に違法である場合、テスト１１１は現在のロードがノン・フォルティング・ロードであるか否かを確定する。現在のロードがノン・フォルティング・ロードである場合、ソフトウェアはステップ１１２においてゼロを返し、命令の実行はフォルティングを伴うことなく継続される。ソフトウェアがゼロを返した場合、ロード命令はステップ１１０において完了する。テスト１１１によって、現在のロードがノーマル（フォルティング）ロードであることが確定された場合、違法アドレスは回復不能なフォルト１０４を形成する。

【００３８】

前記の説明から推測し得るように、フォルト・バブル（Fault bubble）１０４はソフトウェア・ルーチンへのトラップを示す。ページ・フォルト等の幾つかのフォルトは回復可能である。違法アドレスへのアクセスするノーマル・ロード等の他のフォルトは回復不能である。

【００３９】

要求されたページに対する変換テーブル・エントリがソフトウェア・テーブル内で発見された場合、テスト１０３は、変換索引バッファを要求されたテーブル・エントリで満たすフィリングをステップ１０５において呼び出す。変換テーブル・エントリが検索された後、テスト１０６はロードがノン・フォルティング・ロードであるか否かを確認すべく現在のマシンの状態を確認する。現在のロードがノーマル（フォルティング）ロードである場合、テスト１０７はロードが要求するページに対してノン・フォルト・オンリ（略して、ＮＦＯ）ビットが主張されているか否かを確認する。要求されたページに対してノン・フォルト・オンリ・ビットが主張されていない場合、要求されたページはノーマル・ページであり、ノン・フォルティング・ロード及びノーマル・ロードの両方によるアクセスに対して安全である。しかし、要求されたページに対してノン・フォルト・オンリ・ビットが主張されている場合、要求されたページは違法ページであり、ノーマル（フォルティング）ロードによるアクセスは回復不能なフォルト１０４を形成する。ステップ１０８において、仮想から物理アドレスへの翻訳が行われる。次いで、ロードはステップ１０９において実行され、ロード命令はステップ１１０において完了する。ステップ１１０に到達した後、次のロード命令の開始がステップ１００で行われ得る。

【００４０】

図７に示す破線内に存在する各ステップはハードウェア内に実装することが好ましい。現在の命令がノン・フォルティング・ロードであり、かつ要求されたページに対応する変換テーブル・エントリが発見された場合、テスト１０６は翻訳１０８及びロード実行１０９を招来する。ステップ１０１においてＴＬＢヒットが存在する場合、本発明の効果が実現される。変換索引バッファ・ヒットが生じた後、全ての結果はハードウェア・サポートを有する。全てのノーマル・ロード及びノン・フォルティング・ロードはハードウェア内において迅速に処理され、かつ正確に実行される。仮想ページ０ｘ０等の選択されたページは変換テーブル内でマップできる。アクセスが行われた後、仮想ページ０ｘ０に対する変換テーブル・エントリは変換索引バッファ内に存在する。従って、ノン・フォルティング・ロードがヌル・ポインタを間接参照した場合、ＴＬＢヒットが生じ、要求されたページを発見すべくソフトウェアに対してトラップすることなくゼロが即座に返される。

【００４１】

仮想ページ０ｘ０等、ノン・フォルティング・ロードがアクセスする可能性の高いページのみが変換テーブル内にマップされる。このため、他の違法なページは変換テーブル内に存在しない。この場合、ＴＬＢミスによって引き起こされたオペレーティング・システム・ソフトウェア・トラップは、要求された仮想ページに対応する変換テーブル・エントリが存在しないことを最終的に発見し、ゼロを返す。

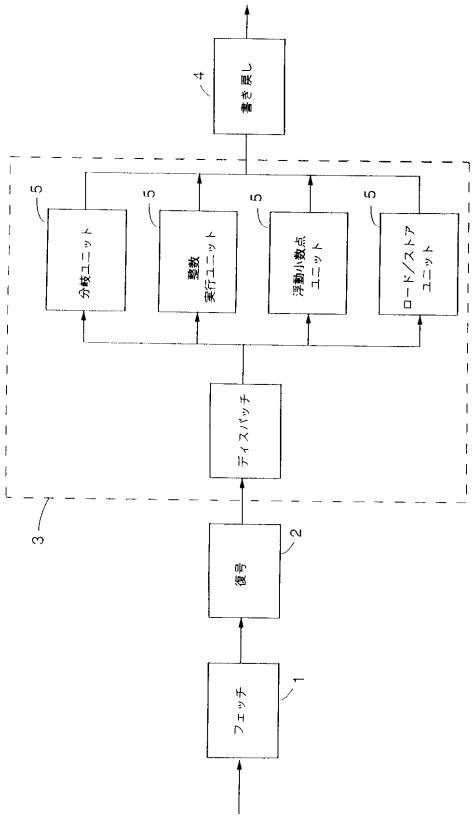
【００４２】

本発明の第２の実施の形態では、ＴＬＢミスによって呼び出されたオペレーティング・システム・ソフトウェア・ルーチンを使用することにより、過去に変換テーブル内へ翻訳されたことがない違法な仮想ページに対するアクセスをノン・フォルティング・ロードが試みたことが発見された場合、オペレーティング・システムは同仮想ページに対する変換テーブル・エントリを形成する（図７のステップ１１３において）。変換テーブル・エントリは前記の仮想ページを全てがゼロである物理ページに対してマップし、さらには同仮想ページに対するノン・フォルト・オンリ・ビットを主張する。この結果、その後実行される同仮想ページからのノン・フォルティング・ロードは、ＴＬＢヒットを形成し、かつＴＬＢミス・ハンドラを回避することと、前記の違法な仮想ページに対応する変換テーブル・エントリを探索すべくソフトウェア変換テーブル全体に対して実施される高い費用及び長い時間を要する不成功なソフトウェア・サーチを少なくとも回避することのうちの一方

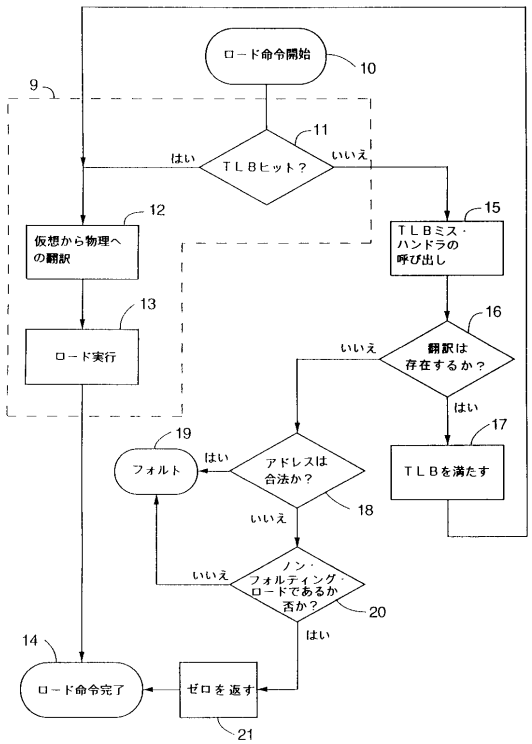
を招来する。
【００４３】
この第２の実施の形態はプログラマまたはコンパイラが、ノン・フォルティング・ロードによってアクセスされる違法な仮想アドレスを命令の実行前に予測する必要性を排除する。ノン・フォルティング・ロードが違法ページを間接参照した場合、オペレーティング・システムは適切な変換テーブル・エントリを形成する。その後実施される同ページからのノン・フォルティング・ロードは変換索引バッファをヒットし、これによりソフトウェア・トラップを回避し得る。
【００４４】
以上、本発明を少なくとも２種類のロード命令、即ち、ノン・フォルティング・ロード及びノーマル（フォルティング）ロードを処理する方法に関して詳述した。一般的に、最近のＲＩＳＣ（縮小命令セット・コンピューティング）プロセッサにおいて、ロード命令はメイン・メモリからデータを検索する唯一の命令である。しかし、これは本発明が任意のプロセッサに付随するメイン・メモリを参照する他の命令に使用できないことを意味しない。例えば、任意のプロセッサは内部レジスタよりも寧ろメイン・メモリから読み出される１つ以上のオペランドを使用する“ａｄｄ”命令をサポートし得る。この場合、加算命令に対応するオペランドをメイン・メモリから検索することは本明細書中に開示したロード命令に相当する。従って、ＣＩＳＣ（複合命令セット・コンピューティング）プロセッサはノン・フォルティング加算命令及びノーマル（フォルティング）加算命令を有し得る。実質的に、本発明はメイン・メモリからのデータの検索を形成する任意の命令に使用できる。簡単なＲＩＳＣプロセッサでは、このような命令はロード命令のみとなる場合がある。しかし、このような命令はメイン・メモリからデータを検索する全ての命令を一般的に含む。
【００４５】
本発明の方法及び装置を現在における好ましい実施の形態に基づいて詳述したが、当業者は請求項に開示する本発明の精神及び範囲内における改良及び変更が可能なことを認識し得る。従って、本明細書及び図面は本発明を限定するものではなく、寧ろ本発明の例示を目的とする。
【００４６】
以上詳述したように、ノン・フォルティング・ロードが違法ページへアクセスした際、最終的にゼロを返すまでに長い時間を要するソフトウェアへのトラップを伴うことなく、即座にゼロが返される。更に、ノン・フォルティング・ロードがアクセスする可能性の高いページのみが変換テーブル内にマップされており、同ページへのアクセスに要する時間を削減できる。また、過去に変換テーブル内へ翻訳されたことがない違法ページが発見された場合、オペレーティング・システムは同仮想ページに対する変換テーブル・エントリを形成するため、ノン・フォルティング・ロードが同ページへアクセスした際に、迅速にゼロを返し得る。この結果、ノン・フォルティング・ロード命令を迅速かつ正確に処理できる。更に、ノーマル（フォルティング）ロードが違法ページへアクセスした際に、ゼロを返すより、寧ろフォルトが形成される。このため、プログラム開発段階等におけるノーマル（フォルティング）ロード命令の迅速かつ正確な処理が保証される。
【００４７】
【発明の効果】
本発明によれば、ノン・フォルティング・ロード命令及びノーマル（フォルティング）ロード命令の迅速かつ正確な処理を行い得るという優れた効果を発揮する。
【図面の簡単な説明】
【図１】従来のパイプライン型スーパスカラ・プロセッサを示すブロック図。
【図２】ロード命令を処理する従来の方法を示す流れ図。
【図３】本発明に基づく変換テーブル・エントリを示す図。
【図４】本発明に基づく変換索引バッファを示す図。
【図５】４つのロード命令と、変換索引バッファが図４に示すエントリを有する際に、同

４つの命令が形成する結果とを示す図。
【図６】（ａ）アクセスされたページに対する変換テーブル・エントリが存在する際に、処理を指示する制御信号を実現する本発明のロジックの真理値表。（ｂ）本発明のロジック回路。
【図７】本発明の第１及び第２の実施の形態に基づくロード命令を処理する方法を示す流れ図であり、ステップ１１３は第１の実施の形態にのみ含まれる。
【符号の説明】
２２…変換テーブル・エントリ、２３…仮想アドレス部分、２４…物理アドレス部分、２５…ノン・フォルト・オンリ・ビット、３０…変換索引バッファ。

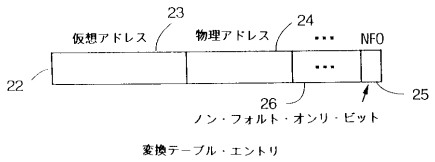
【図１】



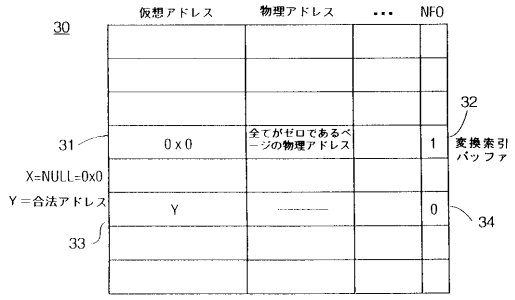
【図２】



【図３】



【図４】



【図５】

命令	結果
LOAD X	フォルト
NF_LOAD X	ゼロを返す
LOAD Y	データを返す
NF_LOAD Y	データを返す

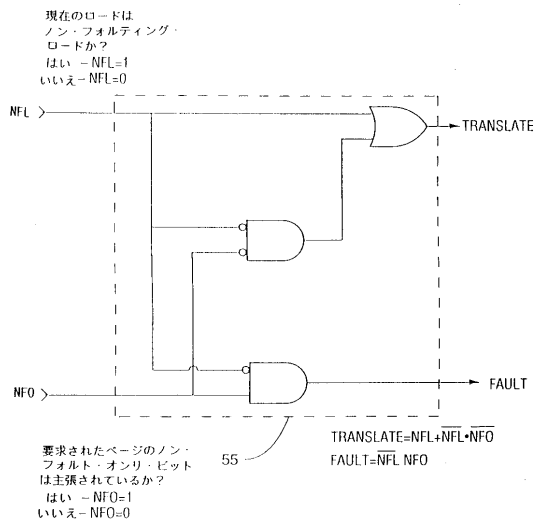
【図６】

（ａ）

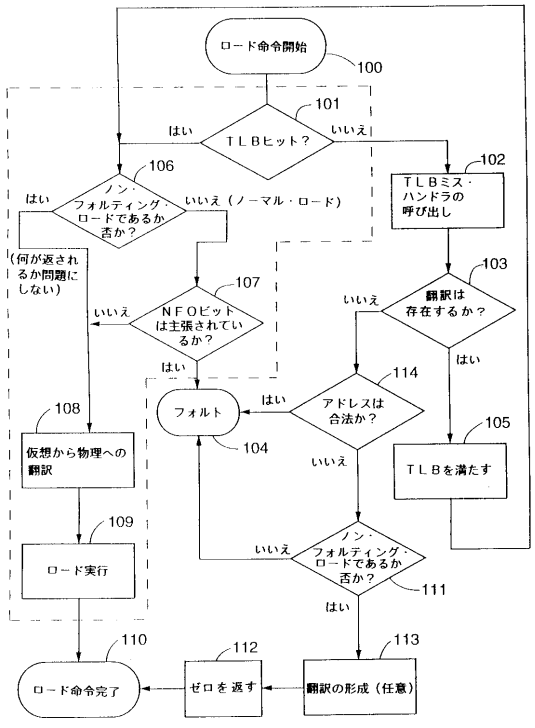
50

ノン・フォルト・ロード？	ノン・フォルト・オンリ？	TRANS-LATE	FAULT
0	0	1	0
0	1	0	1
1	0	1	0
1	1	1	0

（ｂ）



【図７】



フロントページの続き

審査官　後藤　彰

(56)参考文献　特開平４－３１４１５１（ＪＰ，Ａ）
特開平７－１２９４０９（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 9/30 - 9/355
G06F 12/08 - 12/10