(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2004-518183
(P2004-518183A)
(43)公表日　平成16年6月17日(2004.6.17)

(51)Int.Cl.[7]	F I			テーマコード（参考）
G06F 9/38	G06F	9/38	310X	5B013
G06F 9/46	G06F	9/38	330A	5B098
	G06F	9/46	340Z	

審査請求 未請求　予備審査請求 有　（全 47 頁）

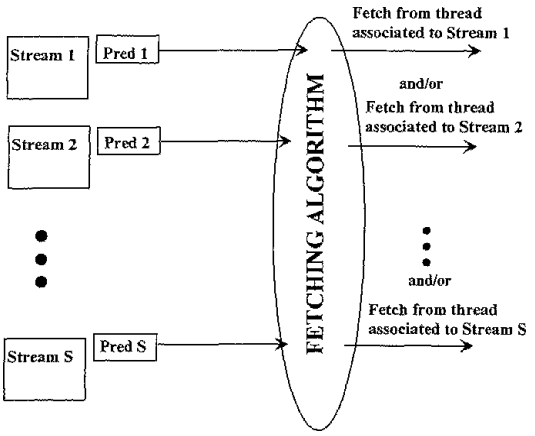
(21)出願番号	特願2002-512805(P2002-512805)
(86)(22)出願日	平成13年7月5日(2001.7.5)
(85)翻訳文提出日	平成15年1月10日(2003.1.10)
(86)国際出願番号	PCT/US2001/021372
(87)国際公開番号	WO2002/006959
(87)国際公開日	平成14年1月24日(2002.1.24)
(31)優先権主張番号	09/616,385
(32)優先日	平成12年7月14日(2000.7.14)
(33)優先権主張国	米国(US)

(71)出願人　501241449
クリアウオーター・ネツトワークス・インコーポレイテツド
アメリカ合衆国、カリフオルニア・９５０３２、ロス・ガトス、ノウレス・ドライブ・１６０
(74)代理人　100062007
弁理士　川口　義雄
(74)代理人　100105131
弁理士　井上　満
(74)代理人　100113332
弁理士　一入　章夫
(74)代理人　100114188
弁理士　小野　誠

最終頁に続く

(54)【発明の名称】マルチスレッド・システムにおける命令のフェッチとディスパッチ

(57)【要約】
マルチストリーミング・プロセッサにおいて、多数のストリーム（ストリーム１、ストリーム２、ストリームｓ）の中の個々のストリームから、命令を命令パイプラインへフェッチするシステムが提供される。このシステムは、どのストリーム（ストリーム１、ストリーム２、ストリームｓ）から命令をフェッチするかを選択するフェッチ・アルゴリズム、及びロード命令がキャッシュでヒット又はミスするかどうか、又は分岐が取られるかどうかを予測する１つ又は複数の予測手段を含む。予測は、どのストリームからフェッチするかを決定する時にフェッチ・アルゴリズムによって使用され、或る場合には、確率が決定され、意思決定においても使用される。予測手段は、フェッチ段階及びディスパッチ段階のいずれか又は双方で使用される。



【特許請求の範囲】
【請求項１】
マルチストリーミング・プロセッサにおいて、多数のストリームの中の個々のストリームから命令をパイプラインへフェッチするシステムであって、
どのストリームから命令をフェッチするかを選択するフェッチ・アルゴリズムと、
分岐命令の分岐選択が取られるかどうかを予測する分岐予測手段と、
を含み、
分岐予測手段による予測が、どのストリームからフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される、システム。
【請求項２】
分岐が取られないだろうという予測が、フェッチ・プロセスに変化を引き起こさない、請求項１に記載のシステム。
【請求項３】
分岐が取られるだろうという予測が、目標アドレスが予測手段によって与えられない場合に、異なったストリームへフェッチの切り替えを生じる、請求項１に記載のシステム。
【請求項４】
前記分岐予測手段が、分岐選択が取られる確率を決定し、該確率が、どこから次の命令をフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される、請求項１に記載のシステム。
【請求項５】
前記分岐予測手段の予測が、更に、パイプラインから機能ユニットへディスパッチする命令を選択する時にディスパッチ・アルゴリズムによって使用される、請求項１に記載のシステム。
【請求項６】
マルチストリーミング・プロセッサにおいて、多数のストリームの中の個々のストリームから命令をパイプラインへフェッチするシステムであって、
どのストリームから命令をフェッチするかを選択するフェッチ・アルゴリズムと、
分岐命令の分岐選択が取られるかどうかを予測する分岐予測手段、又は命令がデータ・キャッシュでヒットするかミスするかを予測するヒット／ミス予測手段の一方又は双方と、
を含み、
予測手段のいずれか又は双方による予測が、どのストリームからフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される、システム。
【請求項７】
分岐が取られないだろうという予測、又は命令がデータ・キャッシュでヒットするだろうという予測が、フェッチ・プロセスに変化を引き起こさない、請求項６に記載のシステム。
【請求項８】
分岐が取られるだろうという予測、又は命令がデータ・キャッシュでミスするだろうという予測が、目標アドレスが予測手段によって与えられない場合に、異なったストリームへフェッチの切り替えを生じる、請求項６に記載のシステム。
【請求項９】
前記分岐予測手段の一方又は双方が、分岐選択が取られる確率又は命令がキャッシュでミスする確率を決定し、該確率が、どこから次の命令をフェッチするかを決定する時に前記フェッチ・アルゴリズムによって使用される、請求項６に記載のシステム。
【請求項１０】
一方又は双方の予測手段の予測が、更に、パイプラインから機能ユニットへディスパッチする命令を選択する時にディスパッチ・アルゴリズムによって使用される、請求項６に記載のシステム。
【請求項１１】
マルチストリーミング・プロセッサであって、

どのストリームから命令をフェッチするかを選択するフェッチ・アルゴリズムと、
分岐命令によって提案されたジャンプが取られるか否かを予測する分岐予測手段と、
を含み、
前記分岐予測手段による予測が、どのストリームからフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される、プロセッサ。
【請求項１２】
分岐が取られないだろうという予測が、フェッチ・プロセスに変化を引き起こさない、請求項１１に記載のプロセッサ。
【請求項１３】
分岐が取られるだろうという予測が、目標アドレスが前記予測手段によって与えられない場合に、異なったストリームへフェッチの切り替えを生じる、請求項１１に記載のプロセッサ。
【請求項１４】
前記分岐予測手段が、分岐が取られる確率を決定し、該確率が、どこから次の命令をフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される、請求項１１に記載のプロセッサ。
【請求項１５】
前記分岐予測手段の予測が、更に、パイプラインから機能ユニットへディスパッチする命令を選択する時にディスパッチ・アルゴリズムによって使用される、請求項１１に記載のプロセッサ。
【請求項１６】
マルチストリーミング・プロセッサであって、
個々のスレッドを走らせる多数の物理ストリームと、
データ・キャッシュと、
どのストリームから命令をフェッチするかを選択するフェッチ・アルゴリズムと、
分岐命令の分岐選択が取られるかどうかを予測する分岐予測手段、又は命令がデータ・キャッシュでヒットするかミスするかを予測するヒット／ミス予測手段の一方又は双方と、
を含み、
前記予測手段のいずれか又は双方による予測が、どのストリームからフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される、プロセッサ。
【請求項１７】
分岐が取られないだろうという予測、又は命令がデータ・キャッシュでヒットするだろうという予測が、フェッチ・プロセスに変化を引き起こさない、請求項１６に記載のプロセッサ。
【請求項１８】
分岐が取られるだろうという予測、又は命令がデータ・キャッシュでミスするだろうという予測が、目標アドレスが前記予測手段によって与えられない場合に、異なったストリームへフェッチの切り替えを生じる、請求項１６に記載のプロセッサ。
【請求項１９】
前記分岐予測手段の一方又は双方が、分岐選択が取られる確率又は命令がキャッシュでミスする確率を決定し、該確率が、どこから次の命令をフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される、請求項１６に記載のプロセッサ。
【請求項２０】
一方又は双方の予測手段の予測が、更に、パイプラインから機能ユニットへディスパッチする命令を選択する時にディスパッチ・アルゴリズムによって使用される、請求項１６に記載のプロセッサ。
【請求項２１】
マルチストリーミング・プロセッサにおいて、命令源としての多数のストリームの中の個々のストリームから命令をパイプラインへフェッチする方法であって、
（ａ）分岐命令をロードする時に、分岐が取られるか否かを分岐予測手段によって予測し

、
（ｂ）もし分岐が取られるだろうと予測されるなら、目標アドレスが予測手段によって与えられない場合に、フェッチ源を変更する
ステップを含む、方法。
【請求項２２】
前記予測手段が確率を決定し、該確率が、フェッチ源を決定する時に使用される、請求項２１に記載の方法。
【請求項２３】
データ・キャッシュを有するマルチストリーミング・プロセッサにおいて、命令源としての多数のストリームの中の個々のストリームから命令をパイプラインへフェッチする方法であって、
（ａ）命令をロードする時に、命令が分岐命令である時には分岐が取られるかどうかについての分岐予測手段、又は命令がデータ・キャッシュでヒットするかどうかについてのヒット／ミス予測手段の一方又は双方によって予測し、
（ｂ）なされた予測に従って、どのストリームからフェッチを継続するかを弁別する
ステップを含む、方法。
【請求項２４】
前記１つ又は複数の予測手段が確率を決定し、該確率がフェッチ源を決定する時に使用される、請求項２３に記載の方法。
【発明の詳細な説明】
【０００１】
（技術分野）
本発明は、マイクロプロセッサの分野に関し、より具体的には、同時マルチスレッド・プロセッサの構造及び機能に関する。
【０００２】
（関連書類への相互参照）
本願は、２０００年６月１６日に出願された先行同時係属特許出願第０９／５９５，７７６号の一部継続出願（ＣＩＰ）であり、前記出願第０９／５９５，７７６号は、１９９８年１２月１６日に出願された先行同時係属特許出願第０９／２１６，０１７号、１９９９年１月２７日に出願された第０９／２４０，０１２号、１９９９年３月２２日に出願された第０９／２７３，８１０号、及び１９９９年５月１４日に出願された第０９／３１２，３０２号のＣＩＰである。これらの５つの出願は、全て、参照して全体をここに組み込まれる。
【０００３】
（発明の背景）
多数のスレッドを処理することのできるマルチストリーミング・プロセッサは技術分野で知られており、多くの研究開発の主題であった。本発明は、この分野における先行研究に注意し、その研究を土台として、装置及び方法の自明でない新規な改善を技術分野にもたらすものである。発明者は、この特許出願と共に、マルチストリーミング・プロセッサの技術分野における多数の既出版文献をリストした情報開示ステートメントを提出した。それらは、ここに開示された本発明の幾つかの態様について、追加の背景及び状況を提供する。
【０００４】
定義を目的として、この明細書では、処理システムに関連したストリームとは、命令のスレッドをサポート及び処理するための、プロセッサのハードウェア能力を意味する。スレッドとは、ストリームの中で走行する実際のソフトウェアである。例えば、デスクトップ・コンピュータを作動するＣＰＵとして実現されるマルチストリーミング・プロセッサは、例えば、ワード処理プログラム及びオブジェクト指向描画プログラムなどの２つ以上のアプリケーションからのスレッドを同時に処理することができる。他の例として、マルチストリーミング可能なプロセッサは、例えば、パケット交換ネットワークにおけるルータ

などのように、定常的な人間による命令がなくともマシンを作動させることができる。例えば、ルータにおいて、データ・パケットを処理してネットワークへ転送する１つ又は複数のスレッドが存在し、またネットワークへ接続された他のルータ及びサーバとサービス品質（ＱｏＳ）について交渉する他のスレッドが存在し、またルーティング・テーブルなどを保守する他のスレッドが存在しうる。多数の並列スレッドを処理するマルチストリーミング・プロセッサの最大能力は、プロセッサがサポートするハードウェア・ストリームの数に固定される。

【０００５】

単一のスレッドを作動するマルチストリーミング・プロセッサは、シングルストリーム・プロセッサとして動作し、使用されないストリームはアイドルになる。説明の目的のために、ストリームは、スレッドをサポートする全ての時点でアクティブであり、そうでなければイナクティブであると考えられる。相互参照セクションでリストされた様々な関連ケース、及び相互参照特許出願の少なくとも１つに含められたＩＤＳの文献から分かるように、スーパースカラ・プロセッサも技術分野で知られている。スーパースカラ・プロセッサとは、１つ又は複数のタイプの機能ユニットを複数個含み、並列命令を多数の機能ユニットへ出すことができるプロセッサを意味する。今日構築される大部分の中央処理ユニット（ＣＰＵ）は、各々のタイプの機能ユニットを複数個有し、前記の定義によれば、スーパースカラ・プロセッサである。幾つかのＣＰＵは、例えば、多数の浮動小数点ユニット、整数ユニット、論理ユニット、ロード／ストアユニットなどを含む多くのユニットを有する。マルチストリーミング・スーパースカラ・プロセッサも、技術分野で知られている。

【０００６】

従来技術のプロセッサは、シングルストリーミング・プロセッサであれ、ダイナミックなマルチストリーミング・プロセッサであれ、通常、パイプラインを使用する。技術分野で知られるように、パイプラインとは、多数の命令が、実行へ進むステップのキューの中に入れられ、命令の実行をスピードアップする技法である。大部分のプロセッサは、命令の実行をパイプラインで処理し、従って、命令は、実行されるまで数ステップを取る。ＲＩＳＣアーキテクチャにおける典型的な段階の簡単な説明を、以下に記す。

（ａ）フェッチ段階：　命令はメモリからフェッチされる。

（ｂ）デコード段階：　命令がデコードされる。

（ｃ）読み出し／ディスパッチ段階：　ソース・オペランドがレジスタ・ファイルから読み出される。

（ｄ）実行段階：　操作が実行され、アドレスが計算されるか、分岐が解決される。

（ｅ）アクセス段階：　データがアクセスされる。

（ｆ）書き込み段階：　結果がレジスタへ書き込まれる。

【０００７】

パイプラインの段階は、１つのクロック・サイクルを取り、従って、サイクルは最も遅い操作が可能であるように十分長くなければならない。本発明は、命令を実行することができない状況がパイプラインの中に存在することと関連している。そのような事象は、技術分野でハザードと呼ばれる。通常、３つのタイプのハザードが存在する。

（ａ）構造的ハザード

（ｂ）データ・ハザード

（ｃ）制御ハザード

【０００８】

構造的ハザードとは、同じクロック・サイクルで実行される命令の組み合わせをサポートする適切なリソース（例えば、機能ユニット）が存在しないことを意味する。データ・ハザードは、解決されていない１つ又は複数の先行命令の結果に命令が依存する時に生じる。データ・ハザードの影響を軽減するためには、通常、転送手法又はバイパス手法が使用される。制御ハザードは、プログラム・カウンタ（ＰＣ）を変更する分岐及び他の命令のパイプラインから生じる。この場合、パイプラインは、分岐が解決されるまで停止される

。
【０００９】
分岐における停止は、プロセッサの性能（１サイクル当たりに実行される命令の数、すなわちＩＰＣによって測定される）に劇的な影響を与える。パイプラインが長く、スーパースカラが広ければ、それだけ負の影響が大きくなる。停止のコストは非常に高いので、分岐の結果を予測することが、技術分野で普通に行なわれる。分岐予測手段は、分岐が「取られる」か「取られない」か、及び目標アドレスを予測する。分岐予測手段は静的であっても動的であってよい。動的分岐予測手段は、プログラム実行の間に、所与の分岐の予測を変更することができる。
【００１０】
分岐予測の典型的なアプローチは、各々の分岐について履歴を保存し、過去を使用して将来を予測することである。例えば、もし所与の分岐が過去で常に取られたのであれば、同じ分岐が将来で再び取られる高い確率が存在する。他方、もし分岐が２回取られ、５回取られず、再び１回とられた、のように続くなら、予測は低い信頼レベルを有するであろう。予測が違うと、パイプラインはフラッシュされなければならず、パイプラインの制御によって、推測が間違った分岐に続く命令が確実に廃棄され、適切な目標アドレスからパイプラインを再スタートしなければならない。これは、コストのかかる操作である。
【００１１】
マルチストリーミング・プロセッサ・アーキテクチャは、微細であるか粗大であってよい。粗大マルチストリーミング・プロセッサは、通常、多数のコンテキストを有する。これらのコンテキストは、例えば、キャッシュ・ミスに起因する長い待ち時間をカバーするために使用される。所与の時間では、ただ１つのスレッドが実行される。対照的に、例えば、本発明者が関係しているＸＳｔｒｅａｍ Ｌｏｇｉｃ社の開発であるダイナミック・マルチストリーミング（ＤＭＳ）などの微細マルチストリーミング技術は、単一プロセッサでの真のマルチタスキング又はマルチストリーミングを可能にし、多数の個別のスレッド又はタスクからの命令を並列に実行する。ＤＭＳプロセッサは、ＣＰＵレジスタ又はハードウェア・コンテキストの多数の集合を実装し、このスタイルの実行をサポートする。
【００１２】
プロセッサのために命令レベルの並列（ｉｎｓｔｒｕｃｔｉｏｎ ｌｅｖｅｌ ｐａｒａｌｌｅｌｉｓｍ（ＩＬＰ））の相対的量を増加することは、データ・ハザード及び制御ハザードを減少させ、従って、アプリケーションは、並列のピークレベルの間、増加する数の機能ユニットを利用することができ、今日の汎用スーパースカラ・プロセッサにおけるダイナミック・マルチストリーミング（ＤＭＳ）のハードウェア及び手法は、ＩＬＰの量を増加してワークロード内で更に均一にＩＬＰを配分することによって、性能を著しく改善する。しかし、依然として、ＤＭＳプロセッサにおける命令のフェッチ及びディスパッチの選択がまずいために、性能を低下させる場合が存在する。
【００１３】
明らかに必要とされるものは、ダイナミック・マルチストリーミング・プロセッサにおいて、特にフェッチ及びディスパッチ操作の時点でパイプラインのヒット／ミス予測を利用する改善された方法及び装置である。
【００１４】
（発明の概要）
本発明の好ましい実施形態では、マルチストリーミング・プロセッサにおいて、多数のストリームの中の個々のストリームから命令をパイプラインへフェッチするシステムが、提供される。このシステムは、どのストリームから命令をフェッチするかを選択するフェッチ・アルゴリズム、及び分岐命令の分岐選択が取られるかどうかを予測する分岐予測手段を含む。分岐予測手段による予測は、どのストリームからフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される。
【００１５】
幾つかの実施形態において、分岐が取られないだろうという予測は、フェッチ・プロセス

に変化を引き起こさない。更に、分岐が取られるだろうという予測は、異なったストリームへフェッチの切り替えを生じる。

【００１６】

幾つかの場合には、分岐予測手段は、分岐選択が取られる確率を決定し、その確率は、どこから次の命令をフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される。他の実施形態において、分岐予測手段の予測は、更に、パイプラインから機能ユニットへディスパッチする命令を選択する時にディスパッチ・アルゴリズムによって使用される。

【００１７】

本発明の他の態様では、マルチストリーミング・プロセッサにおいて、多数のストリームの中の個々のストリームから命令をパイプラインへフェッチするシステムが、提供される。このシステムは、どのストリームから命令をフェッチするかを選択するフェッチ・アルゴリズム、及び分岐命令の分岐選択が取られるかどうかを予測する分岐予測手段、又は命令がデータ・キャッシュでヒット又はミスするかどうかを予測するヒット／ミス予測手段の一方又は双方を含む。この実施形態において、予測手段のいずれか又は双方による予測は、どのストリームからフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される。

【００１８】

幾つかの実施形態において、分岐が取られないだろうという予測、又は命令がデータ・キャッシュでヒットするだろうという予測は、フェッチ・プロセスに変化を引き起こさない。更に、幾つかの実施形態において、分岐が取られるだろうという予測、又は命令はデータ・キャッシュでミスするだろうという予測は、異なったストリームへフェッチの切り替えを生じる。

【００１９】

幾つかの場合において、分岐予測手段の一方又は双方は、分岐選択が取られる確率、又は命令がキャッシュでミスする確率を決定し、その確率は、どこから次の命令をフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される。更に、一方又は双方の予測手段の予測は、パイプラインから機能ユニットへディスパッチする命令を選択する時にディスパッチ・アルゴリズムによって使用される。

【００２０】

更に、本発明の他の態様において、どのストリームから命令をフェッチするかを選択するフェッチ・アルゴリズム、及び分岐命令によって提案されたジャンプが取られるかどうかを予測する分岐予測手段を含むマルチストリーミング・プロセッサが提供される。分岐予測手段による予測は、どのストリームからフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される。

【００２１】

これらの実施形態の幾つかにおいて、分岐が取られないだろうという予測は、フェッチ・プロセスに変化を引き起こさず、分岐が取られるだろうという予測は、異なったストリームへフェッチの切り替えを生じる。分岐予測手段は、分岐が取られる確率を決定することができ、その確率は、どこから次の命令をフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される。幾つかの場合には、分岐予測手段の予測は、更に、パイプラインから機能ユニットへディスパッチする命令を選択する時にディスパッチ・アルゴリズムによって使用される。

【００２２】

更に、他の実施形態において、個々のスレッドを走らせる多数の物理ストリーム、データ・キャッシュ、どのストリームから命令をフェッチするかを選択するフェッチ・アルゴリズム、及び分岐命令の分岐選択が取られるかどうかを予測する分岐予測手段、又は命令がデータ・キャッシュでヒット又はミスするかどうかを予測するヒット／ミス予測手段の一方又は双方を含むマルチストリーミング・プロセッサが提供される。予測手段のいずれか又は双方による予測は、どのストリームからフェッチするかを決定する時にフェッチ・ア

ルゴリズムによって使用される。幾つかの実施形態において、分岐が取られないだろうという予測、又は命令がデータ・キャッシュでヒットするだろうという予測は、フェッチ・プロセスに変化を引き起こさず、他の実施形態においては、分岐が取られるだろうという予測、又は命令がデータ・キャッシュでミスするだろうという予測は、異なったストリームへフェッチの切り替えを生じる。

【００２３】

幾つかの場合において、分岐予測手段の一方又は双方は、分岐選択が取られる確率、又は命令がキャッシュでミスする確率を決定し、それらの確率は、どこから次の命令をフェッチするかを決定する時にフェッチ・アルゴリズムによって使用される。一方又は双方の予測手段の予測は、パイプラインから機能ユニットへディスパッチする命令を選択する時にディスパッチ・アルゴリズムによって使用される。

【００２４】

本発明を実施する方法も教示される。また、この後で詳細に説明される様々な実施形態において、最初に、装置及び方法がマルチストリーミング・プロセッサへ応用されて、それらの性能が著しく改善される。

【００２５】

（好ましい実施の形態の説明）

図１ａは、本発明の実施形態に従ったダイナミック・マルチストリーミング（ＤＭＳ）プロセッサにおけるパイプラインの略図である。この略図において、パイプラインは７つの段階を有する。それらの段階は、フェッチ、デコード、読み出し、ディスパッチ、実行、アクセス、及び書き込みである。これらの段階は、前記の背景セクションで説明したものと同じであるが、図１ａでは、機能を示すため読み出しとディスパッチが分離されている。本発明がディスパッチへ情報を付加し、プロセッサの性能を改善している点で、ディスパッチは本発明において重要である。パイプライン内のフェッチ段階は、多数のストリームから命令をパイプラインの中へフェッチし、本発明の実施形態では、選択的フェッチの能力を有する。

【００２６】

パイプラインの各々の段階に命令が存在することは、動作しているプロセッサの要件ではないが、そのような場合があり、発明者は、説明の混乱を避けるため、各々の段階が単一の命令によって占有されている例を選択した。多くの場合、様々な段階に複数の命令が存在するか、全く存在しないことがある。

【００２７】

図１ａにおいて、パイプライン内の命令は、任意に、パイプラインの連続した段階の１時点で命令ＡからＧとして示される。図１ｂは、１サイクル経過した後の図１ａのパイプラインを示す。命令Ａはフェッチからデコードへ移動し、図１ａに示される他の命令は、同様に１段階だけ前方へ移動していることに注意されたい。更に、新しい命令Ｈが、フェッチ段階でパイプラインへ入っている。

【００２８】

図１ｃは、１サイクル経過した後の同じパイプラインを示す。全ての命令は、更に前方へ１段階移動し、新しい命令Ｉがフェッチ段階でパイプラインへ入っている。図１ｄは、更に１サイクル後の同じパイプラインを示す。その時点で、命令は更に前方へ移動しており、更に他の命令Ｊがパイプラインへ入っている。

【００２９】

４番目のサイクルの後で、命令Ａはフェッチからディスパッチへ移動したことに注意されたい。この例において、命令Ａは、キャッシュからデータ値をロードするロード命令であると仮定する。もしそうであれば、特定のデータがキャッシュの中にあるか否かについて或る確率が存在するであろう。当技術分野において、これはヒット／ミス確率として知られる。もしデータがキャッシュの中にあれば、システムはヒットを記録する。もしなければ、システムはミスを記録する。

【００３０】

ロード操作のヒット／ミス確率と、パイプライン・アーキテクチャとの組み合わせは、プロセッサの効率に重要である。なぜなら、従来の場合、パイプライン内の一般的シーケンスは、単一のスレッドからなり、典型的には、ロード命令に続く多くの命令が、ロードされたデータをどの命令が使用するかの結果に依存することに関連するからである。即ち、ロードされたデータをどの命令が使用するかが解決されるまで、多くの後続の命令は実行されることができない。例外は、幾つかの場合に、投機ベースで実行される場合である。

【００３１】

従来のプロセッサは、ロード命令がパイプラインに入ると、単純にヒットと仮定する。しかし、もしロードがミスであれば、一度、ロード命令が実行されると、キャッシュの中に存在しない必要なデータをメモリからロードするのに、多数のサイクルを取るかも知れない。都合が悪いことに、ロード命令がディスパッチされて実行されるまで、ミスは明らかにならないだろう。後続の命令は、データがロードされ、そのデータに依存する命令が実行されるまで、停止しなければならない。

【００３２】

本発明者は、マルチスレッド・アーキテクチャにおけるデータ・キャッシュ・ミスの影響を減少させる装置及び方法を提供する。この手法は、ＤＭＳの多数のストリームで走行するスレッドの各々について、データ・キャッシュへの次のアクセスがミスになるかどうかを予測することからなる。もしミスになれば、（一般的に）次のようになる。

【００３３】

フェッチ段階で、どのストリームからフェッチするかを決定する時に、ストリームに低い優先順位を与えることができる。

【００３４】

データ・キャッシュにアクセスする命令の従属命令は、ディスパッチ段階で、より効率的に機能ユニット（ＦＵ）へディスパッチされることができる。

【００３５】

この新規な装置及び手法は、命令をフェッチ及びディスパッチする時に、マルチストリーミング・プロセッサの性能を改善する。

【００３６】

（ヒット／ミスの予測によるフェッチ）

新しい手法は次の事実を利用する、すなわち、ＤＭＳプロセッサにおいて、命令はストリームの中の個々のストリームからパイプラインへフェッチされるので、どのストリームから命令をフェッチするかをサイクル・ベースで選択するフェッチ方針又はアルゴリズムの選択に自由度が存在するということである。

【００３７】

マルチストリーミング・アーキテクチャにおいて、ここで提案される手法を使用しない場合に、スレッドの切り替えを生じる典型的な事象は、データ・キャッシュ・ミスである。必要なデータは、利用可能になるまで数サイクルを取るかも知れないので（正確な数は、実際に、データがプロセッサのメモリ階層のどこに存在するかに依存する）、データ・キャッシュでミスしたスレッドは、他へ切り替えられる。なぜなら、ミスする可能性が最も高い命令の従属命令は、データへの依存性に起因して実行しないからである。従って、他のスレッドからの命令のフェッチと実行によって、より多くの作業を行なうことができる。この場合、ミスした命令の後続命令であって既にフェッチされた命令は、フラッシュにより消去される必要があり、従って、有用な命令がフェッチされる場合と比較して、プロセッサの性能を低下させるだろう。

【００３８】

命令がデータ・キャッシュでミスする事実を、プロセスの中で早期に知ることができれば、結局はフラッシュされるかも知れない命令のフェッチを避けて、データ・キャッシュでミスする命令に続く命令の代わりに、他のストリームからの命令をフェッチすることができ、これは、フェッチされた命令が素早く実行される可能性を改善する。従って、本発明の実施形態におけるフェッチ・アルゴリズムは、全てのストリームについて、次のアクセ

スがデータ・キャッシュでミスするかどうかの予測を考慮に入れ、命令の実行及びコミットの可能性が最も高いスレッドを走行させるストリームからフェッチすることができる。

【００３９】

ヒット／ミス予測を実現する様々な技術が既に存在する。しかし、目的は、常に同じであって、データ・キャッシュに対するヒットとミスを最も正確に予測することである。更に、そのような予測手段の望ましい特性は、できるだけ早くデータ・キャッシュへの次のアクセスを予測できることであり、それによって、パイプラインの中に入る命令（結局はフラッシュによって消去される）の数を少なくすることである。

【００４０】

ここで教示される手法は、信頼レベルを予測へ関連付けることによって改善されることができる。本発明の１つの実施形態において、フェッチ段階で動作している予測手段は、予測に加えて、この信頼レベルの値を生成する。信頼レベルは、例えば、２つ以上の予測手段がデータ・キャッシュのミスを予測し、１つが切り替えで外されるように選択される場合に、フェッチ・アルゴリズムを支援する。この場合、より高い信頼レベルを有するストリームが選択される。

【００４１】

図２は、マルチストリーミング・アーキテクチャにおけるフェッチ・アルゴリズムの略図である。このアルゴリズムは、ストリームの各々に関連づけられたキャッシュ・ヒット／ミス予測手段に基づいて、どのストリームからフェッチするかを決定する。図２において、予測手段は、ストリーム１、ストリーム２、．．．、ストリームＳに関連づけられる。従って、理論的には、各々のサイクルで、Ｓ個までのストリーム（Ｓはマルチストリーミング・アーキテクチャによってサポートできるストリームの最大数である）から命令を同時にフェッチすることができる。しかし、実際には、実現上の制限から（例えば、命令キャッシュ・ポートの利用可能性）、フェッチ・アルゴリズムは、Ｐ個のストリーム（Ｐ＜Ｓ）からの命令のフェッチに制限されるかも知れない。更に、フェッチ・アルゴリズムは、他の情報に基づいて（例えば、各々のストリームに対する分岐予測の信頼度、スレッドの優先順位、パイプラインの状態など）、どのストリームからフェッチするかを選択することも考えられる。

【００４２】

これまで、データ・キャッシュに対するヒット／ミスの予測手段について説明してきた。データ・キャッシュは、性能の理由から、異なったレベル（第１のレベルＬ１は、プロセッサ・コアに最も近い）で実現されうることに注意すべきである。本発明の代替実施形態において、異なったヒット／ミス予測手段が、データ・キャッシュ・レベルの各々に対して存在することができる。

【００４３】

本発明の代替の実施形態におけるフェッチ・アルゴリズムは、フェッチされる命令の選択を、データ・キャッシュの第２のレベルＬ２に対する予測をベースとすることができる。なぜなら、大部分のプロセッサ・システムにおいて、キャッシュの第２レベルにおけるミスは、サイクル数の点で非常にコストが高いからである（それに対して、Ｌ１におけるミスの不利益は、比較的小さい）。

【００４４】

（分岐予測によるフェッチの弁別）

前記の「背景」セクションで幾分詳細に説明したように、制御ハザードは、プログラム・カウンタ（ＰＣ）を変更する分岐及び他の命令のパイプラインから生じる。この場合、パイプラインは、分岐が解決されるまで停止される。これまでの説明は、特に、パイプライン内の命令がデータ・キャッシュでヒットするかミスするかの確率、即ち、これらの命令を実行するために必要なデータがキャッシュの中に存在するか否かの確率に関連する。本題の場合には、キャッシュのヒット／ミス予測ではなく、分岐予測によって弁別が達成される。

【００４５】

分岐での停止は、プロセッサの性能（１サイクル当たりに実行される命令の数、すなわちＩＰＣによって測定される）に劇的な影響を与える。プロセッサ内のパイプラインが長く、スーパースカラが広くなれば、それだけ負の影響が大きくなる。停止のコストは非常に高いので、シングルストリーミング・プロセッサに関する技術では、分岐の結果を予測することが普通に行なわれる。分岐予測手段は、分岐命令が取られるかどうかを予測し、更に、分岐命令の信頼レベルと、もし分岐が取られるのであれば目標アドレスとを指示することができる。分岐予測手段は、静的であっても動的であってよい。動的分岐予測は、プログラム実行の間に、所与の分岐に対する予測を変更することができる。

【００４６】

分岐予測の典型的なアプローチは、各々の分岐の履歴を保存し、過去を使用して将来を予測することである。例えば、もし所与の分岐が、過去において常に取られたのであれば、同じ分岐が将来再び取られる確率は高い。他方、もし分岐が２回取られ、５回取られず、再び１回取られるように続いたのであれば、予測は低い信頼レベルを有するであろう。予測が間違った時に、パイプラインはフラッシュされなければならず、パイプラインに制御によって、間違って推測された分岐に続く命令を確実に廃棄しなければならず、適切な目標アドレスからパイプラインを再スタートさせなければならない。これはコストのかかる操作である。

【００４７】

更に、例を挙げると、図５は、特定のスレッドに対するプログラム・カウンタ（ＰＣ）シーケンスの一般的な図であり、命令０から命令９までのシーケンスを示している。命令３は分岐命令である。具体的には、もしｘが２よりも小さければ命令９へジャンプし、そうでなければ、命令４からスレッドのシーケンスを継続する。パイプライン・プロセッサにおいて、分岐命令３がフェッチされる時に、それが機能ユニットへディスパッチされて解決される前に、少なくとも数サイクルが存在するので、分岐が取られるかどうかの可能性を知ることは良いことであろう。もし、分岐命令をパイプラインの中にフェッチする時点で、分岐予測手段が使用され、分岐が取られる可能性が高いことが分かり、目標アドレスが９であれば、命令９から新しい命令のパイプラインへのフェッチを開始するように決定することができる。もし可能性が低ければ、新しい命令が順次パイプラインの中へフェッチすることができ、プロセッサの性能は分岐予測手段を使用することによって著しく改善される。

【００４８】

マルチストリーミング・プロセッサを含む本発明の好ましい実施形態において、本発明者は、分岐が取られるかどうかを最大の可能性で予測するため、分岐予測手段がプロセッサの各々のストリームに関連付けられるシステムを提供し、好ましい実施形態では、予測の信頼レベルを提供する。分岐予測手段の出力は、どのストリームから命令をパイプラインの中へフェッチするかの決定を助けるため、入力としてフェッチ・アルゴリズムへ送られる。

【００４９】

ヒット／ミス予測の場合に説明した図２は、分岐予測の例を説明するために使用することができる。再び、Ｓ個のストリームが示され、予測手段が各々のストリームと関連付けられる。この場合の予測手段は、前述したヒット／ミス予測手段ではなく、分岐予測手段である。マルチストリーミング・プロセッサにおいて、分岐命令がフェッチされ、パイプラインへ入ると、各々のストリームに関連付けられた分岐予測手段は、分岐がパイプラインへ入る確率を決定する。予測は、示されるようにフェッチ・アルゴリズムへ入力として送られ、フェッチ・アルゴリズムは、重要な決定を行なうため、この入力、及び、恐らく他の入力も使用するように構成される。この場合、分岐が取られる確率が低いので、プロセッサは、現在使用しているフェッチ情報を継続することができる。もし目標アドレスが予測されなければ、分岐が取られる高い確率を使用して、フェッチ・アルゴリズムに、分岐命令が取られたストリームではなく異なったストリームからフェッチを開始させる。もし分岐が取られる確率が高く、目標アドレスが分岐のために予測されるなら、更なる命令を

目標アドレスから開始してフェッチすることができる。
【００５０】
所与の分岐のために、分岐予測手段は、分岐が取られるか否かを予測し、更に、予測の信頼レベルを生成することができる。好ましい実施形態において、信頼レベル（確率）は、０（回数の約半分が真）から１（確実）までの数ｐによって与えられる。１に近い値は、予測が真になる可能性が非常に高いことを意味する。好ましい実施形態において、Ｎビットの信頼レベル・フィールド（ＣＬＦ）が分岐予測手段へ付加される。Ｎビットはｐをディジタル化したものである。例えば、Ｎ＝１である時に、もし信頼レベルが低ければＣＬＦ＝０であり、そうでなければ１である。Ｎ＝２である時に、確実から最低レベルまで４レベルの信頼度が存在する。フェッチ・アルゴリズムは、ＣＬＦの値に基づいて決定を行ない、例えば、最高ＣＬＦを有するストリームから分岐命令をフェッチする。低い値のＣＬＦを有する分岐が解決される時に、もしそのストリームからのフェッチが、問題の分岐に続いて起こらないなら、その分岐のＣＬＦは、より高い値へアップグレードされることができる。その間に、他のストリームからの命令が、占拠されたリソースを維持しながらフェッチされ、パイプラインの停止の危険性を防止する。
【００５１】
（ヒット／ミス予測によるディスパッチ）
データ・キャッシュ・ヒット／ミス予測手段を使用する手法は、パイプラインのディスパッチ段階で、どの命令を命令キュー（もしあれば）から抽出し、実行のために機能ユニット（ＦＵ）へ送るかを決定するプロセスでも有用である。
【００５２】
現在の技術では、命令（今後は生成手段と呼ぶ）がデータ・キャッシュへの読み出しアクセスを生成する時に、データ・キャッシュがアクセスされ、ヒット／ミスの結果が決定されるまで、結果の待ち時間を知ることはできない。生成手段によって生成されたデータの従属命令（今後は、消費手段と呼ぶ）のディスパッチは、２つの方針に従うことができる。
（ａ）データの利用可能性が保証される時にのみ、命令をディスパッチする。
（ｂ）生成手段が、データ・キャッシュの第１のレベルでヒットするものと仮定して、命令をディスパッチする。
【００５３】
従って、方針（ｂ）は、消費手段命令を投機的にディスパッチする（キャッシュ内のヒット率は通常非常に高いので、生成手段命令に対してヒットが常に仮定される）。もし消費手段命令がＦＵへ到着し、データが依然として利用可能でなければ、命令はＦＵで停止しなければならないか、後のサイクルでディスパッチへ再スケジュールされなければならない（このオプションは、他の非従属命令がＦＵへディスパッチされることを可能にする）。とにかく、双方のオプションは、プロセッサのパフォーマンスを低下させる。
【００５４】
方針（ａ）は、最低パフォーマンスを提供する。なぜなら、消費手段命令は、ディスパッチされる前に不必要に停止されるかも知れないからである。生成手段命令は、データ・キャッシュ内でヒットするや否や直ちにディスパッチされるか、ミスした場合は、ミスしたデータがメモリ階層の次のレベルから到着する時にディスパッチされる。他方、この方針は、再スケジュールが起こらないので、最も単純な実現形態を提供する。
【００５５】
本発明の実施形態において、ヒット／ミス予測手段は、生成手段がデータ・キャッシュでヒットするかどうかを予測することによって、方針（ｂ）の性能を高める。従って、データ・キャッシュでミスするものと予測される生成手段の消費手段命令は、方針（ａ）に従ってディスパッチされる。もし生成手段命令がヒットするものと予測されるなら、ディスパッチ方針は（ｂ）である。しかし、この場合、予測が不正確である場合に備えて、再スケジュール論理が必要になる。予測がヒットであり、実際の結果がミスである場合にのみ、消費手段命令はＦＵで停止されるか、再スケジュールされる必要がある。

【００５６】
一般的に、ディスパッチ・レベルで動作しているヒット／ミス予測手段は、データの待ち時間を予測することによって、消費手段命令のディスパッチを最適化する。もしＬ１におけるヒットが予測されるなら、データの待ち時間は、Ｌ１キャッシュの待ち時間であると予測される。もしミスが予測されるなら、データの予測待ち時間は、キャッシュの更なるレベルの存在およびヒット／ミス予測手段のこれらレベルの各々での存在に依存する。例えば、もし２レベルのキャッシュが存在し、Ｌ２のヒット／ミス結果が予測されるなら、データの予測待ち時間は、図３で示されるようにして計算される（注意：キャッシュの出力から、消費手段が実行される機能ユニットの入力まで、もしあれば、データを送るのに必要なサイクルは、データの予測された待ち時間へ加算される必要がある）。

【００５７】
ディスパッチ論理のためにヒット／ミス予測手段が有する利点は、マルチストリーミング・プロセッサのみに限定されないが、この手法は、通常の（シングルストリーミング）プロセッサ・アーキテクチャにおけるよりも、マルチストリーミング・プロセッサにおいて大きな利点を有する。データ・ヒット／ミス予測手段を有する通常のプロセッサにおいて、データ・キャッシュのミスが予測される場合には、命令を実行することはできないか（順序内ディスパッチ・エンジンの場合）、ミスするデータに依存しない命令のみを実行することができる（順序外ディスパッチ・エンジンの場合）。いずれにせよ、プロセッサのリソースは、ミスするデータが利用可能になるまで、数サイクルの間アイドルになるかも知れない。マルチストリーミング・プロセッサにおいては、それらのアイドル・サイクルは、他のスレッドからの他の命令を実行するために使用されることができる。なぜなら、それらの命令はミスするデータに依存しないからである。従って、マルチストリーミング・プロセッサの場合、データ・キャッシュ・ヒット／ミス予測手段の利点は、図３で示されるように２倍になる。

【００５８】
（分岐予測によるディスパッチでの弁別）
これまで、ヒット／ミス予測を使用してマルチストリーミング・プロセッサのディスパッチ段階で弁別する方法が説明された。プロセッサ・パフォーマンスを改善するため、同様に分岐予測をディスパッチ段階で使用することができる。前述したように、フェッチ・アルゴリズムへの入力として分岐予測がフェッチ段階で使用される好ましい実施形態では、パイプラインへ入る全ての分岐について、おそらく確率を付加された予測が分岐命令のために存在する。この情報は保持されて、フェッチ・アルゴリズムからディスパッチ・アルゴリズムへ渡され、分岐命令の直後にフェッチされた命令の選択的ディスパッチで使用することができる。例えば、１つの簡単な例では、高い確率の分岐命令に続く命令に、ディスパッチにおいて他の命令よりも優先順位を与えることができる。

【００５９】
フェッチの弁別が用いられない代替の実施形態において、ディスパッチ段階での弁別が依然として使用される。一度、ここでの教示が与えられると、パイプライン・プロセッサのフェッチ段階及びディスパッチ段階のいずれか又は双方で、ヒット／ミス及び分岐の予測が単独又は前後して行われうることが当業者に明らかであろう。

【００６０】
本発明の代替実施形態において、予測は、フェッチ段階及びディスパッチ段階で異なるように行なわれることができる（即ち、異なった情報を予測の基礎として使用し、および／または、異なった予測アルゴリズムを使用して）。１つの例として、ディスパッチ段階におけるヒット／ミス予測は、消費手段命令のプログラム・カウンタ（ＰＣ）アドレスを使用することができ（なぜなら、命令は既にデコードされ、そのＰＣは知られているから）、分岐予測で使用された予測スキームに類似したアルゴリズムに従うことができる。フェッチ段階における予測は、他のタイプのアドレス（例えば、キャッシュ・ライン）又は他の非アドレス情報を使用してよい。

【００６１】

異なった実施形態における予測アルゴリズムは、プロセッサが効率的にサポートしなければならないワークロードに依存して変わってよい。Ｗｉｎｄｏｗｓ（登録商標）プログラム又はＳＰＥＣベンチマークのような伝統的アプリケーションのためには、分岐予測で使用されたアルゴリズムに類似したアルゴリズムが、ヒット／ミスの場合に、ヒット及びミスの双方で所望の予測精度を生成する。ネットワーク・プロセッサ内のパケット処理アプリケーションのような他のタイプのワークロードについては、予測手段は、処理されているパケットが所属するフロー番号のような追加的情報を利用することができる（新しいフローの最初のパケットの処理によって実行されたデータ・キャッシュ・アクセスは、ミスの可能性が最も高い）。

【００６２】

ここで教示された本発明の実施形態において、本発明の趣旨及び範囲から逸脱することなく、多くの変更がなされることは、当業者に明らかであろう。例えば、予測手段は様々な方法で実現され、割り当てられた確率に基づいて、異なったアクションが取られる。更に、予測は、パイプライン内の異なったレベルで使用される。例えば、予測手段は、デコード段階から入力を得て、フェッチ・アルゴリズムへ出力する。更に、本発明の異なった実施形態を達成するメカニズムは、通常、ハードウェア又はソフトウェアのいずれかで実現される。同様に、本発明の趣旨及び範囲内で行なわれる多くの他の変更が存在する。本発明は、添付のクレイムの範囲に対して許容されるべきである。

【図面の簡単な説明】

【図１ａ】

本発明の実施形態におけるパイプラインの略図である。

【図１ｂ】

１サイクルを経過した後の図１ａのパイプラインを示す図である。

【図１ｃ】

他の１サイクルを経過した後の図１ａ及び図１ｂのパイプラインを示す図である。

【図１ｄ】

更に、他の１サイクルを経過した後の図１ａ、図１ｂ、及び図１ｃのパイプラインを示す図である。

【図２】

本発明の実施形態において、予測手段をストリームに関連付ける略図である。

【図３】

キャッシュ内の異なったレベルに対する予測手段を示す略図である。

【図４】

本発明の実施形態における手法の利点を示す略図である。

【図５】

プログラム・カウンタのシーケンスを示す図である。

【図１ａ】

フェッチ	命令Ａ
デコード	命令Ｂ
読み出し	命令Ｃ
ディスパッチ	命令Ｄ
実　行	命令Ｅ
アクセス	命令Ｆ
書き込み	命令Ｇ

Fig.1a

【図１ｂ】

フェッチ	命令Ｈ
デコード	命令Ａ
読み出し	命令Ｂ
ディスパッチ	命令Ｃ
実　行	命令Ｄ
アクセス	命令Ｅ
書き込み	命令Ｆ

Fig.1b

【図１ｃ】

フェッチ	命令Ｉ
デコード	命令Ｈ
読み出し	命令Ａ
ディスパッチ	命令Ｂ
実　行	命令Ｃ
アクセス	命令Ｄ
書き込み	命令Ｅ

Fig.1c

【図１ｄ】

フェッチ	命令Ｊ
デコード	命令Ｉ
読み出し	命令Ｈ
ディスパッチ	命令Ａ
実　行	命令Ｂ
アクセス	命令Ｃ
書き込み	命令Ｄ

Fig.1d

【図２】

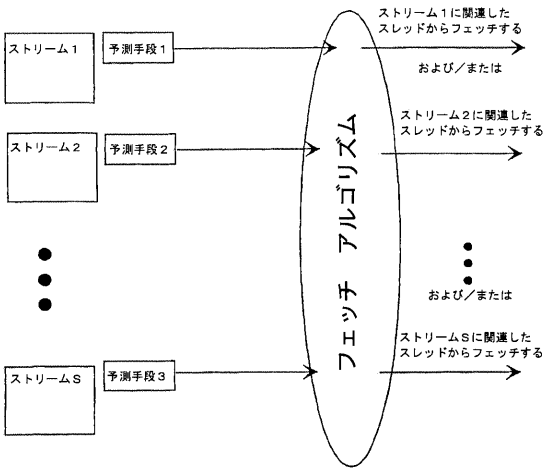


Fig.2

【図３】

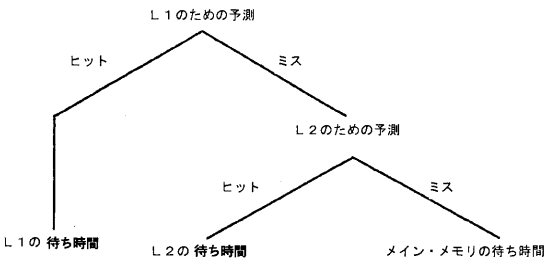


Fig.3

【図４】

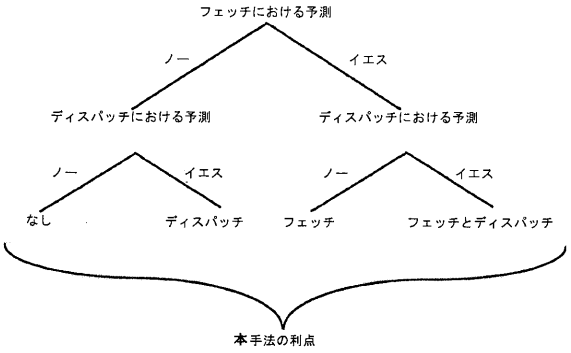


Fig.4

【図５】

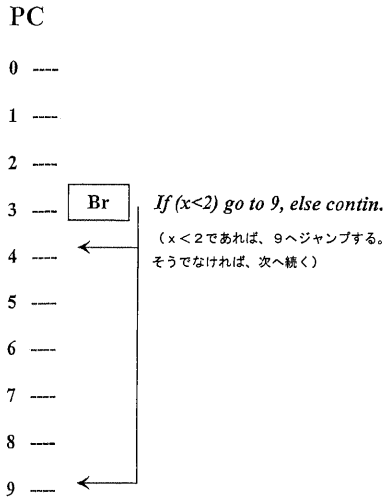

PC
0 ----
1 ----
2 ----
3 ----
4 ----
5 ----
6 ----
7 ----
8 ----
9 ----
Br
If (x<2) go to 9, else contin.
（ｘ＜２であれば、９へジャンプする。
そうでなければ、次へ続く）

Fig.5

【国際公開パンフレット】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 January 2002 (24.01.2002)

PCT

(10) International Publication Number
WO 02/06959 A1

(51) International Patent Classification[7]: G06F 9/48

(21) International Application Number: PCT/US01/21372

(22) International Filing Date: 5 July 2001 (05.07.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/616,385 14 July 2000 (14.07.2000) US

(71) Applicant: CLEARWATER NETWORKS, INC. [US/US]; 160 Knowles Drive, Los Gatos, CA 95032 (US).

(72) Inventors: MUSOLL, Enric; 7210 Via Romera, San Jose, CA 95139 (US). NEMIROVSKY, Mario, D.; 19750 Northampton Drive, Saratoga, CA 95070 (US).

(74) Agent: BOYS, Donald, R.; P.O. Box 187, Aromas, CA 95004 (US).

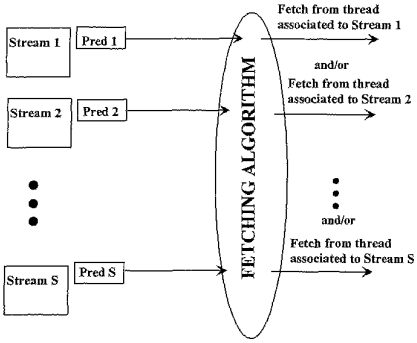
(81) Designated States (national): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States (regional): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— with international search report

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: INSTRUCTION FETCH AND DISPATCH IN MULTITHREADED SYSTEM



(57) Abstract: In a multi-streaming processor, a system for fetching instructions from individual ones of multiple streams (stream 1, stream 2, stream s) to an instruction pipeline is provided, comprising: a fetch algorithm for selecting from which stream (stream 1, stream 2, stream s) to fetch an instruction, and one or more predictors for forecasting whether a load instruction will hit or miss the cache or a branch will be taken. The prediction or predictions are used by the fetch algorithm in determining from which stream to fetch, in some cases probabilities are determined and also used in decisions, and predictors may be used at either or both of fetch and dispatch stages.

WO 02/06959 A1

INSTRUCTION FETCH AND DISPATCH IN MULTITHREADED SYSTEM

Field of the Invention

The present invention is in the area of microprocessors, and pertains more particularly to structure and function of simultaneous multithreaded processors.

Cross Reference to Related Documents

The present application is a continuation-in-part (CIP) of prior copending patent application S/N 09/595,776, filed on 6/16/200, which is a CIP of prior co-pending patent applications 09/216,017, filed 12/16/98, 09/240,012, filed 1/27/99, 09/273,810, filed 3/22/99 and 09/312,302 filed 5/14/99 all five of which are incorporated herein in their entirety by reference.

Background of the Invention

Multi-streaming processors capable of processing multiple threads are known in the art, and have been the subject of considerable research and development. The present invention takes notice of the prior work in this field, and builds upon that work, bringing new and non-obvious improvements in apparatus and methods to the art. The inventors have provided with this patent application an Information Disclosure Statement listing a number of published papers in the technical field of multi-streaming processors, which together provide additional background and context for the several aspects of the present invention disclosed herein.

For purposes of definition, this specification regards a *stream* in reference to a processing system as a *hardware* capability of the processor for supporting and

processing an instruction thread. A *thread* is the actual software running within a stream. For example, a multi-streaming processor implemented as a CPU for operating a desktop computer may simultaneously process threads from two or more applications, such as a word processing program and an object-oriented drawing program. As another example, a multi-streaming-capable processor may operate a machine without regular human direction, such as a router in a packet switched network. In a router, for example, there may be one or more threads for processing and forwarding data packets on the network, another for quality-of-service (QoS) negotiation with other routers and servers connected to the network and another for maintaining routing tables and the like. The maximum capability of any multi-streaming processor to process multiple concurrent *threads* remains fixed at the number of hardware *streams* the processor supports.

A multi-streaming processor operating a single thread runs as a single-stream processor with unused streams idle. For purposes of discussion, a stream is considered an *active* stream at all times the stream supports a thread, and otherwise inactive. As in various related cases listed under the cross-reference section, and in papers provided by IDS, which were included with at least one of the cross-referenced applications, superscalar processors are also known in the art. This term refers to processors that have multiples of one or more types of functional units, and an ability to issue concurrent instructions to multiple functional units. Most central processing units (CPUs) built today have more than a single functional unit of each type, and are thus superscalar processors by this definition. Some have many such units, including, for example, multiple floating point units, integer units, logic units, load/store units and so forth. Multi-streaming superscalar processors are known in the art as well.

State-of-the-art processors typically employ pipelining, whether the processor is a single streaming processor, or a dynamic multi-streaming processor. As is known in the art, pipelining is a technique in which multiple instructions are queued in steps leading to execution, thus speeding up instruction execution. Most processors pipeline instruction execution, so instructions take several steps until they are

executed. A brief description of typical stages in a RISC architecture is listed immediately below:

a) Fetch stage: instructions are fetched from memory
b) Decode stage: instructions are decoded
c) Read/Dispatch stage: source operands are read from register file
d) Execute stage: operations are executed, an address is calculated or a branch is resolved
e) Access stage: data is accessed
f) Write stage: the result is written in a register

Pipeline stages take a single clock cycle, so the cycle must be long enough to allow for the slowest operation. The present invention is related the fact that there are situations in pipelining when instructions cannot be executed. Such events are called hazards in the art. Commonly, there are three types of hazards:

a) Structural
b) Data
c) Control

A structural hazard means that there are not adequate resources (e.g., functional units) to support the combination of instructions to be executed in the same clock cycle. A data hazard arises when an instruction depends on the result of one or more previous instructions not resolved. Forwarding or bypassing techniques are commonly used to reduce the impact of data hazards. A control hazard arises from the pipelining of branches and other instructions that change the program counter (PC). In this case the pipeline may be stalled until the branch is resolved.

Stalling on branches has a dramatic impact on processor performance (measured in instructions executed per cycle or IPC). The longer the pipelines and the wider the superscalar, the more substantial is the negative impact. Since the cost of stalls is quite high, it is common in the art to predict the outcome of branches. Branch predictors predict branches as either *taken* or *untaken* and the target address. Branch predictors may be either static or dynamic. Dynamic branch predictors may change prediction for a given branch during program execution.

A typical approach to branch prediction is to keep a history for each branch, and then to use the past to predict the future. For example, if a given branch has always been taken in the past, there is a high probability that the same branch will be taken again in the future. On the other hand, if the branch was taken 2 times, not taken 5 times, taken again once, and so forth, the prediction made will have a low confidence level. When the prediction is wrong, the pipeline must be flushed, and the pipeline control must ensure that the instructions following the wrongly guessed branch are discarded, and must restart the pipeline from the proper target address. This is a costly operation.

Multistreaming processor architectures may be either fine-grained or coarse-grained. Coarse-grained multistreaming processors typically have multiple contexts, which are used to cover long latencies arising, for example, due to cache misses. Only a single thread is executing at a given time. In contrast, fine-grained multistreaming technologies such as Dynamic Multi-Streaming (DMS), which is a development of XStream Logic, Inc., with which the present inventors are associated, allow true multi-tasking or multistreaming in a single processor, concurrently executing instructions from multiple distinct threads or tasks. DMS processors implement multiple sets of CPU registers or hardware contexts to support this style of execution.

Increasing the relative amount of instruction level parallelism (ILP) for a processor reduces data and control hazards, so applications can exploit increasing number of functional units during peak levels of parallelism, and Dynamic Multi-Streaming (DMS) hardware and techniques within today's general-purpose superscalar processors significantly improves performance by increasing the amount of ILP, and more evenly distributing it within workload. There are still occasions, however, for degraded performance due to poor selection in fetching and dispatching instructions in a DMS processor.

What is clearly needed is improved methods and apparatus for utilizing hit/miss prediction in pipelines in dynamic multi-streaming processors, particularly at the point of fetch and dispatch operations.

Summary of the Invention

In a preferred embodiment of the present invention, in a multi-streaming processor, a system for fetching instructions from individual ones of the multiple streams to a pipeline is provided, comprising a fetch algorithm for selecting from which stream to fetch instructions, and a branch predictor for forecasting whether a branch alternative of a branch instruction will be taken. The prediction by the branch predictor is used by the fetch algorithm in determining from which stream to fetch.

In some embodiments a prediction that a branch will not be taken precipitates no change in the fetching process. Also, a prediction that a branch will be taken results in switching fetching to a different stream.

In some cases the branch predictor determines a probability that a branch alternative will be taken, and the probability is used by the fetch algorithm in determining from where to fetch next instructions. In other embodiments the forecast of the branch predictor is also used by a dispatch algorithm in selecting instructions from the pipeline to dispatch to functional units.

In another aspect of the invention, in a multi-streaming processor, a system for fetching instructions from individual ones of the multiple streams to a pipeline is provided, comprising a fetch algorithm for selecting from which stream to fetch instructions, and one or both of a branch predictor for forecasting whether a branch alternative of a branch instruction will be taken, or a hit-miss predictor for forecasting whether instructions will hit or miss a data cache. In this embodiment the prediction by either or both of the predictors is used by the fetch algorithm in determining from which stream to fetch.

In some embodiments a prediction that a branch will not be taken or that an instruction will hit the data cache precipitates no change in the fetching process. Also in some embodiments a prediction that a branch will be taken or that an instruction will miss a data cache results in switching fetching to a different stream.

In some cases one or both of the branch predictors determine a probability that a branch alternative will be taken or that an instruction will miss the cache, and the

probability is used by the fetch algorithm in determining from where to fetch next instructions. Also, the forecast of one or both predictors may be also used by a dispatch algorithm in selecting instructions from the pipeline to dispatch to functional units.

In yet another aspect of the invention a multi-streaming processor is provided, comprising a fetch algorithm for selecting from which stream to fetch instructions, and a branch predictor for predicting whether jumps proposed by branch instructions will be taken or not. A prediction by the branch predictor is used by the fetch algorithm in determining from where stream to fetch.

In some of these embodiments a prediction that a branch will not be taken precipitates no change in the fetching process, and a prediction that a branch will be taken results in switching fetching to a different stream. The branch predictor may determine a probability for whether a branch will be taken, and the probability is used by the fetch algorithm in determining from where to fetch next instructions. In some cases the forecast of the branch predictor is also used by a dispatch algorithm in selecting instructions from the pipeline to dispatch to functional units.

In still another embodiment a multistreaming processor is provided, comprising multiple physical streams for running individual threads, a data cache, a fetch algorithm for selecting from which stream to fetch instructions, and one or both of a branch predictor for forecasting whether a branch alternative of a branch instructions will be taken, or a hit-miss predictor for forecasting whether instructions will hit or miss a data cache. The prediction by either or both of the predictors is used by the fetch algorithm in determining from which stream to fetch. In some embodiments a prediction that a branch will not be taken or that an instruction will hit the data cache precipitates no change in the fetching process, while in others a prediction that a branch will be taken or that an instruction will miss a data cache results in switching fetching to a different stream.

In some cases one or both of the branch predictors determine a probability that a branch alternative will be taken or that an instruction will miss the cache, and the probability is used by the fetch algorithm in determining from where to fetch next

instructions, and the forecast of one or both predictors may be used by a dispatch algorithm in selecting instructions from the pipeline to dispatch to functional units.

Methods for practicing the invention are taught as well, and, in the various embodiments described in enabling detail below, for the first time apparatus and methods are applied to multistreaming processors to significantly improve their performance.

Brief Description of the Drawing Figures

Fig. 1a is a simplified diagram of a pipeline in an embodiment of the present invention.

Fig. 1b shows the pipeline of Fig. 1a after a cycle.

Fig. 1c shows the pipeline of Fig. 1a and 1b after another cycle.

Fig. 1d shows the pipeline of Fig. 1a, 1b and 1c after yet another cycle.

Fig. 2 is a schematic diagram associating predictors with streams in an embodiment of the present invention.

Fig. 3 is a schematic showing predictors for different levels in cache.

Fig. 4 is a schematic illustrating benefits of the technique in embodiments of the invention.

Fig. 5 is a depiction of a program counter sequence.

Description of the Preferred Embodiments

Fig. 1a is a simplified diagram of a pipeline in a dynamic, multi-streaming (DMS) processor according to an embodiment of the present invention. In this simplified view the pipeline has seven stages, which are fetch, decode, read, dispatch, execute, access and write. These are the same as described in the background section above, except for the separation of read and dispatch in Fig. 1a to illustrate the

functions. Dispatch is important in the present invention in that the present invention adds intelligence to Dispatch, improving the performance of the processor. The fetch stage in the pipeline fetches instructions into the pipeline from the multiple streams, and in an embodiment of the present invention is capable of selective fetching.

Although there is no requirement in operating processors that there be instructions at each stage of a pipeline, it is often true that this is the case, and the inventors choose to illustrate each stage as occupied by a single instruction to avoid confusion in description. In many cases there will a plurality of instructions at various stages, or none at all.

In Fig. 1a the instructions in the pipeline are arbitrarily indicated as instructions A through G, at successive stages in the pipeline at one point in time. Fig. 1b shows the pipeline of Fig. 1a one cycle later. Note that instruction A has moved from fetch to decode, and the other instructions shown in Fig. 1a have moved one stage forward as well. Also, a new instruction, H, has entered the pipeline at the fetch stage.

Fig. 1c shows the same pipeline one cycle later. All instructions have moved forward one further stage, and a new instruction I has entered the pipeline at the fetch stage. Fig. 1d shows the same pipeline after yet another cycle, at which point in time the instructions have moved forward yet again, and yet another instruction J has entered the pipeline.

Note that after the fourth cycle, instruction A has moved from fetch to dispatch. Assume for the sake of this example that instruction A is a load instruction for loading a data value from cache. If this is the case, there will be some probability as to whether the particular data is in cache or not. In the art this is known as the hit/miss probability. If the data is in the cache, the system scores a hit. If not, the system scores a miss.

The combination of hit/miss probability for load operations with pipelined architecture has significance for processor efficiency, because, in the conventional case the general sequence of instructions in the pipeline will be from a single thread, and will typically be related in that many instructions following a load instruction may

depend upon the result of whatever instruction is to use the data loaded. That is, until the resolution of whatever instruction is to use the data loaded, many following instructions cannot be executed, except in some cases, on a speculative basis.

Conventional processors simply assume a hit when a load instruction enters a pipeline. If the load is a *miss*, however, once the load instruction is executed, then it may take a number of cycles for the needed data, not in cache, to be loaded from memory. And, unfortunately, the miss will not be apparent until the load instruction is dispatched and executed. The following instructions have to stall until the data is loaded and the instruction(s) depending on the data are executed.

The present inventors provide apparatus and methods for reducing the impact of data cache misses in multithreaded architectures. The technique consists of predicting, for each of the threads running in the multiple streams of the DMS, whether the next access to the data cache will result in a miss. If this is the case, then (generally):

- The stream can be given a lower priority when deciding, in the fetch stage, from which stream to fetch, and
- The dependent instructions of the instruction that accesses the data cache can be more efficiently dispatched to the functional units (FU's) in the dispatch stage.

This new apparatus and technique improves the performance of a multistreaming processor in the fetching and dispatching of instructions.

Fetching With Hit-Miss Prediction

The new technique takes advantage of the fact that, in DMS processor, as instructions are fetched to the pipeline from individual ones of the streams, there is freedom in choosing a fetching policy or algorithm that will select, on a cycle-by-cycle basis, from which stream instructions are to be fetched.

In a multistreaming architecture, without the technique proposed here, a typical event that causes a thread switch is a data cache miss. Since the required data may take several cycles to be available (the exact number depending on where the data really resides in the memory hierarchy of the processor), the thread that missed the data cache may be switched out since the dependent instructions of the instruction that missed most likely will not execute due to the dependencies on the data. Thus, more work can be done by fetching and executing instructions from another thread. In this case, the instructions following the one that missed, and that have already been fetched, will need to be flushed out, thus degrading the performance of the processor with respect to the case in which useful instructions had been fetched.

If the fact that an instruction will miss the data cache could be known early in the process the fetching of instructions that might eventually be flushed may be avoided by fetching, instead of the instructions following the instruction that missed the data cache, instructions from another stream, improving the likelihood that the fetched instructions may be quickly executed. Thus, a fetching algorithm, in an embodiment of the present invention, may take into account, for all the streams, the predictions on whether the next access will miss the data cache, and fetch from the stream running a thread that is most likely to have its instructions executed and committed.

There already exist in the art a variety of implementations for hit-miss predictors. The goal, however, is always the same: to predict with the highest accuracy both the hits and misses to the data cache. Moreover, a desirable property of such a predictor is to be able to predict the next access to the data cache as soon as possible so that fewer instructions (that would eventually be flushed out) will enter the pipeline.

The technique taught herein can be improved by associating a confidence level to the prediction. The predictor, in one embodiment of the invention, operating at the fetch stage, in addition to predicting also generates this confidence level value. The confidence level helps the fetching algorithm, for example, in cases in which two or

more predictors predicted a miss in the data cache and one is selected to be switched out. In this case, the stream with higher confidence level will be selected.

Fig. 2 is a schematic diagram of a fetching algorithm in a multistreaming architecture. The algorithm decides from which stream(s) to fetch based on cache hit/miss predictors associated to each of the streams. In Fig. 2 a predictor is associated with streams 1, 2, and so on through stream S. Thus, theoretically, instructions from up to S streams (S being the maximum number of streams supported by the multistreaming architecture) can be simultaneously fetched every cycle. In reality, however, the fetching algorithm might be restricted to fetch instructions from P streams ($P < S$) due to implementation restrictions (for example, availability of instruction cache ports). Moreover, the fetching algorithm might select from which streams to fetch based on other information (for example, confidence on the branch prediction for each stream, thread priorities, state of the pipeline, etc.)

So far, we have mentioned predictors of hit/miss for the data cache. Note that the data cache might be implemented for performance reasons in different levels (the first level - L1- being the closest to the processor core). In alternative embodiments of the invention different hit/miss predictors may exist for each of the data cache levels.

The fetching algorithm in alternative embodiments of the present invention may base selection of instructions to be fetched on the prediction for the second level - L2 - of data cache since, in most processor systems, a miss in the second level of cache is very costly in number of cycles (whereas the penalty of a miss in the L1 is comparatively relatively small).

Fetching Discrimination by Branch Prediction

As was described in some detail above in the "Background" section, a control hazard arises from the pipelining of branches and other instructions that change the program counter (PC). In this case the pipeline may be stalled until the branch is resolved. The description above relates in particular to the probability of whether

instructions in the pipeline will hit or miss the data cache; that is, whether the data needed to execute these instructions may or may not be in the cache. In the present case discrimination is accomplished by branch prediction, rather than cache hit-miss prediction.

Stalling on branches has a dramatic impact on processor performance (measured in instructions executed per cycle or IPC). The longer the pipelines and the wider the superscalar in a processor, the more substantial is the negative impact. Since the cost of stalls is quite high, it is common in the art in regard to single-streaming processors to predict the outcome of branches. Branch predictors predict whether a branch instruction will be taken, and may also indicate a confidence level for branch instructions and the target address if the branch is taken. Branch predictors may be either static or dynamic. Dynamic branch predictors may change prediction for a given branch during program execution.

A typical approach to branch prediction is to keep a history for each branch, and then to use the past to predict the future. For example, if a given branch has always been taken in the past, there is a high probability that the same branch will be taken again in the future. On the other hand, if the branch was taken 2 times, not taken 5 times, taken again once, and so forth, the prediction made will have a low confidence level. When the prediction is wrong, the pipeline must be flushed, and the pipeline control must ensure that the instructions following the wrongly guessed branch are discarded, and must restart the pipeline from the proper target address. This is a costly operation.

To further illustrate, Fig. 5 is a generic diagram of a program counter (PC) sequence for a specific thread, showing instructions 0 through 9 in sequence. Instruction 3 is a Branch instruction, specifically that if x is less than 2, jump to instruction 9, and if not, continue with the thread sequence at instruction 4. In a pipelined processor, when Br instruction 3 is fetched, since it will be some several cycles at least before it is dispatched to functional units and resolved, it would be good to know the likelihood as to whether the branch will be taken. If, at the time of fetching the branch instruction into the pipeline, a branch predictor is employed, and

the likelihood that the branch will be taken is found to be high, and the target address is 9, a decision can be made to begin to fetch new instructions into the pipeline at instruction 9. If the likelihood is low, then new instructions may be fetched into the pipeline sequentially, and processor performance may be considerably improved by use of the branch predictor.

The inventors have provided, in a preferred embodiment of the present invention comprising a multi-streaming processor, a system in which a branch predictor is associated with each stream of the processor to predict, to the greatest possible degree, whether a branch will be taken, and in a preferred embodiment, the confidence level of the prediction. Output from the branch predictors is fed as input to a fetching algorithm to aid in determining from which stream to fetch instructions into the pipeline.

Fig. 2 described above in the case of hit-miss prediction may also serve to illustrate the instant case for branch prediction. Again S streams are indicated, and a predictor is associated with each stream. The predictor in this case is a branch predictor, rather than the hit-miss predictor descried above. As branch instructions are fetched and enter the pipeline in the multi-streaming processor, the branch predictor associated with each stream determines the probability that the branch will enter the pipeline. The predictions are fed as input to the fetching algorithm as shown, and the fetching algorithm may be structured to use this input, and perhaps other input as well, in making important decisions. In this case, a low probability that a branch will be taken allows the processor to continue with whatever fetching intelligence is currently in use. A high probability that a branch may be taken, if no target address is predicted, may be used to cause the fetching algorithm to begin fetching from a different stream than the stream from which the branch instruction was taken. If the probability that a branch will be taken is high, and a target address is predicted for the branch, further instructions may be fetched beginning from the target address.

For a given branch, a branch predictor predicts that a branch will be taken or not taken, and also may generate a confidence level of the prediction. In a preferred embodiment the confidence level (probability) is given by a number p between 0

(about half of the time is true) to 1 (certainty). A value close to unity means it is highly likely that the prediction will become true. In a preferred embodiment a confidence-level field (CLF) of N bits is added to the branch predictor. The N bits are a digitalization of p. For example, if N = 1, CLF = 0 if the confidence level is low and one otherwise; for N = 2 there are 4 levels of confidence, say, from certainty to the lowest level. The fetching algorithm makes a decision based on the value of CLF such as to fetch branch instructions from streams with the highest CLF. When a branch with low value of CLF is resolved, if no fetching from that stream has taken place following the offending branch, the CLF for that branch could be upgraded to a higher value. Meanwhile, instructions from other streams were fetched maintaining resources occupied, and avoiding the risk of stalling the pipeline.

Dispatch With Hit-Miss Prediction

The technique of having a data cache hit/miss predictor is also useful in the process of deciding, at the dispatch stage in the pipeline, which instructions are to be extracted from the instruction queue (if any) and sent to the functional units (FUs) for execution.

In current art, when an instruction (henceforth called a producer) generates a read access to the data cache, the latency of the result is not known until the data cache is accessed and the hit/miss outcome is determined. The dispatch of a dependent instruction (henceforth termed a consumer) on the data generated by the producer can follow two policies:

a) Dispatch the instruction only when it is guaranteed that the data will be available.

b) Dispatch the instruction assuming that the producer will hit in the first level of the data cache.

Policy (b), then, dispatches the consumer instruction speculatively (a hit is always assumed for the producer instruction since the hit ratio in a cache is usually very high). If the consumer instruction arrives to the FU and the data is still not available, the instruction has to either stall at the FU or be rescheduled for dispatch in a later cycle (this option will allow other non-dependent instructions to be dispatched to the FU). In any case, both options degrade the performance of the processor.

Policy (a) provides the lowest performance since the consumer instruction might be unnecessarily stalled before it is dispatched. The producer instruction will be dispatched as soon as the producer hits in the data cache or, in case it misses, when the missing data arrives from the next level of memory hierarchy. On the other hand, this policy provides the simplest implementation, since no re-scheduling will occur.

In an embodiment of the present invention a hit/miss predictor enhances the performance of policy (b) by predicting whether the producer will hit in the data cache. Thus, the consumer instructions of a producer that is predicted to miss in the data cache will be dispatched following policy (a). If the producer instruction is predicted to hit, then the dispatch policy is (b). In this case, however, the re-scheduling logic is still needed in case the prediction is incorrect. Only in the case in which the prediction is a hit but the real outcome is a miss, the consumer instructions will need to be either stalled at the FUs or re-scheduled.

In general, the hit/miss predictor operating at the dispatch level optimizes the dispatch of consumer instructions by predicting the latency of the data. If a hit in the L1 is predicted, the latency of the data is predicted to be the latency of the L1 cache. If a miss is predicted, the predicted latency of the data depends on whether more levels of cache exist and on whether a hit/miss predictor exists for each of these levels. If, for example, two levels of cache exist and the hit/miss outcome of the L2 is also predicted, the predicted latency of the data is computed as shown in Fig. 3 (Note: the necessary cycles, if any, to bring the data from the output of the cache to the input of the functional unit where the consumer will be executed need to be added to the predicted latency of the data).

The benefits of a hit/miss predictor for dispatch logic are not restricted to multistreaming processors only, but in a multistreaming processor where the technique has larger benefits than in a conventional (single-streaming) processor architecture. In a conventional processor having a data hit/miss predictor, when a data cache miss is predicted, no instructions (in case of an in-order dispatch engine), or only those that do not depend on the missing data (in case of an out-of-order dispatch engine) can execute. In any case, the processor resources might be idle for several cycles until the missing data is available. In multistreaming processors those idle cycles can be used to execute other instructions from other threads since they do not depend on the missing data. Thus, for a multistreaming processor, the benefits of a data cache hit/miss predictor are twofold as shown in Figure 3.

Discrimination at Dispatch by Branch Prediction

Discrimination at the dispatch stage in a multi-streaming processor using hit-miss prediction is described above. Branch prediction can be used at the dispatch stage as well to improve processor performance. In a preferred embodiment, wherein branch prediction is used at the fetch stage as input to a fetch algorithm as described above, for every branch that enters the pipeline a there will be a prediction, possibly with an attached probability, for the branch instruction. This information may be retained and passed from the fetch algorithm to a dispatch algorithm, and used in selective dispatching of instructions fetched right after the branch instruction. In one simple case, for example, the instructions following a high probability branch instructions may be given preference in dispatch versus other instructions.

In an alternative embodiment, wherein fetch discrimination is not employed, discrimination at the dispatch stage may still be used. It will be apparent to the skilled artisan, once given the teachings herein, that hit-miss and branch prediction may be done singly or in tandem at either or both of fetch and dispatch stages in a pipelined processor.

In alternative embodiments of the invention the prediction can be done differently at the fetch and dispatch stages (i.e. using different information on which to base the prediction and/or using a different prediction algorithm). As an example, the hit-miss prediction at the dispatch stage could use the program counter (PC) address of the consumer instruction (since the instruction has already been decoded and its PC is known) and could follow an algorithm similar to the prediction schemes used in branch prediction. The prediction at the fetch stage may use another type of address (cache line, for example) or other non-address information.

The prediction algorithm in different embodiments may vary depending on the workload that the processor has to efficiently support. For traditional applications, like Windows programs or SPEC benchmarks, similar algorithms to those used in branch prediction may produce the desired prediction accuracy in both hits and misses for the hit-miss case. For other types of workloads, like packet processing applications in network processors, the predictors can take advantage of additional information, like the flow number to which the packet being processed belongs (the data cache accesses performed by the processing of the first packet(s) of a new flow most likely will miss).

It will be apparent to the skilled artisan that there are many alterations that might be made in the embodiments of the invention taught herein without departing from the spirit and scope of the invention. The predictors may be implemented in various ways, for example, and different actions may be taken based on assigned probabilities. Further, the predictors may be used at different levels in a pipeline. For example, a predictor may have input from a decode stage, and output to a fetch algorithm. Further, the mechanisms to accomplish different embodiments of the invention may be implemented typically in either hardware or software. There are similarly many other alterations that may be made within the spirit and scope of the invention. The invention should be accorded the scope of the claims below.

What is claimed is:

1. In a multi-streaming processor, a system for fetching instructions from individual ones of the multiple streams to a pipeline, comprising:

a fetch algorithm for selecting from which stream to fetch instructions; and

a branch predictor for forecasting whether a branch alternative of a branch instructions will be taken;

wherein the prediction by the branch predictor is used by the fetch algorithm in determining from which stream to fetch.

2. The system of claim 1 wherein a prediction that a branch will not be taken precipitates no change in the fetching process.

3. The system of claim 1 wherein a prediction that a branch will be taken results in switching fetching to a different stream if no target address is provided by the predictor.

4. The system of claim 1 wherein the branch predictor determines a probability that a branch alternative will be taken, and the probability is used by the fetch algorithm in determining from where to fetch next instructions.

5. The system of claim 1 wherein the forecast of the branch predictor is also used by a dispatch algorithm in selecting instructions from the pipeline to dispatch to functional units.

6. In a multi-streaming processor, a system for fetching instructions from individual ones of the multiple streams to a pipeline, comprising:

a fetch algorithm for selecting from which stream to fetch instructions; and

one or both of a branch predictor for forecasting whether a branch alternative of a branch instructions will be taken, or a hit-miss predictor for forecasting whether instructions will hit or miss a data cache;

wherein the prediction by either or both of the predictors is used by the fetch algorithm in determining from which stream to fetch.

7. The system of claim 6 wherein a prediction that a branch will not be taken or that an instruction will hit the data cache precipitates no change in the fetching process.

8. The system of claim 6 wherein a prediction that a branch will be taken or that an instruction will miss a data cache results in switching fetching to a different stream if no target address is provided by the predictor.

9. The system of claim 6 wherein one or both of the branch predictors determine a probability that a branch alternative will be taken or that an instruction will miss the cache, and the probability is used by the fetch algorithm in determining from where to fetch next instructions.

10. The system of claim 6 wherein the forecast of one or both predictors is also used by a dispatch algorithm in selecting instructions from the pipeline to dispatch to functional units.

11. A multi-streaming processor comprising:

a fetch algorithm for selecting from which stream to fetch instructions; and

a branch predictor for predicting whether jumps proposed by branch instructions will be taken or not;

wherein a prediction by the branch predictor is used by the fetch algorithm in determining from which stream to fetch.

12. The processor of claim 11 wherein a prediction that a branch will not be taken precipitates no change in the fetching process.

13. The processor of claim 11 wherein a prediction that a branch will be taken results in switching fetching to a different stream if no target address is provided by the predictor.

14. The processor of claim 11 wherein the branch predictor determines a probability for whether a branch will be taken, and the probability is used by the fetch algorithm in determining from where to fetch next instructions.

15. The processor of claim 11 wherein the forecast of the branch predictor is also used by a dispatch algorithm in selecting instructions from the pipeline to dispatch to functional units.

16. A multistreaming processor, comprising:

multiple physical streams for running individual threads;

a data cache;

a fetch algorithm for selecting from which stream to fetch instructions; and

one or both of a branch predictor for forecasting whether a branch alternative of a branch instructions will be taken, or a hit-miss predictor for forecasting whether instructions will hit or miss a data cache;

wherein the prediction by either or both of the predictors is used by the fetch algorithm in determining from which stream to fetch.

17. The processor of claim 16 wherein a prediction that a branch will not be taken or that an instruction will hit the data cache precipitates no change in the fetching process.

18. The processor of claim 16 wherein a prediction that a branch will be taken or that an instruction will miss a data cache results in switching fetching to a different stream if no target address is provided by the predictor.

19. The processor of claim 16 wherein one or both of the branch predictors determine a probability that a branch alternative will be taken or that an instruction will miss the cache, and the probability is used by the fetch algorithm in determining from where to fetch next instructions.

20. The processor of claim 16 wherein the forecast of one or both predictors is also used by a dispatch algorithm in selecting instructions from the pipeline to dispatch to functional units.

21. In a multi-streaming processor, a method for fetching instructions from individual ones of multiple streams as instruction sources to a pipeline, comprising the steps of:

(a) on loading a branch instruction, making a prediction by a branch predictor as to whether a branch will be taken or not; and

(b) if the prediction is that the branch will be taken, altering the source of the fetch if no target address is provided by the predictor.

22. The method of claim 21 wherein the predictor determines a probability, and the probability is used in determining fetch source.

23. In a multi-streaming processor having a data cache, a method for fetching instructions from individual ones of multiple streams as instruction sources to a pipeline, comprising the steps of:

(a) on loading an instruction, making a prediction by one or both of a branch predictor as to whether a branch will be taken if the instruction is a branch instruction, or by a hit-miss predictor as to whether the instruction will hit the data cache; and

(b) discriminating from which stream to continue to fetch according to prediction made.

24. The method of claim 23 wherein the predictor or predictors determine a probability, and the probability is used in determining fetch source.

Fetch	INST A
Decode	INST B
Read	INST C
Dispatch	INST D
Execute	INST E
Access	INST F
Write	INST G

Fig. 1a

Fetch	INST H
Decode	INST A
Read	INST B
Dispatch	INST C
Execute	INST D
Access	INST E
Write	INST F

Fig. 1b

Fetch	INST I
Decode	INST H
Read	INST A
Dispatch	INST B
Execute	INST C
Access	INST D
Write	INST E

Fig. 1c

Fetch	INST J
Decode	INST I
Read	INST H
Dispatch	INST A
Execute	INST B
Access	INST C
Write	INST D

Fig. 1d

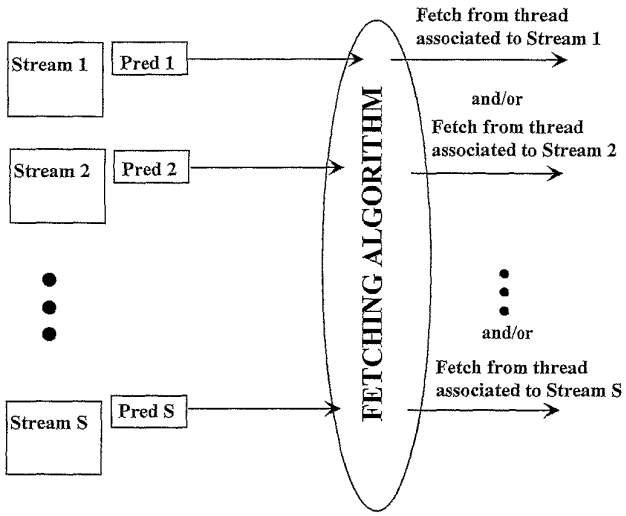

Stream 1
Pred 1
Stream 2
Pred 2
Stream S
Pred S
FETCHING ALGORITHM
Fetch from thread associated to Stream 1
and/or
Fetch from thread associated to Stream 2
and/or
Fetch from thread associated to Stream S

Fig. 2

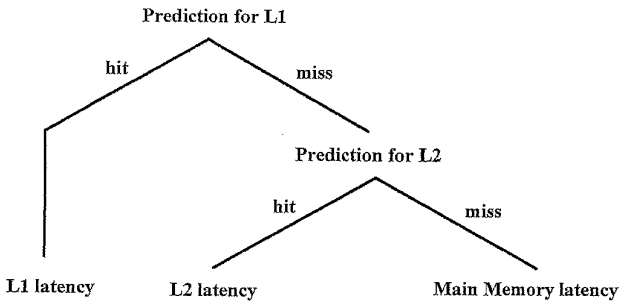

Prediction for L1
hit
miss
Prediction for L2
hit
miss
L1 latency
L2 latency
Main Memory latency

Fig. 3

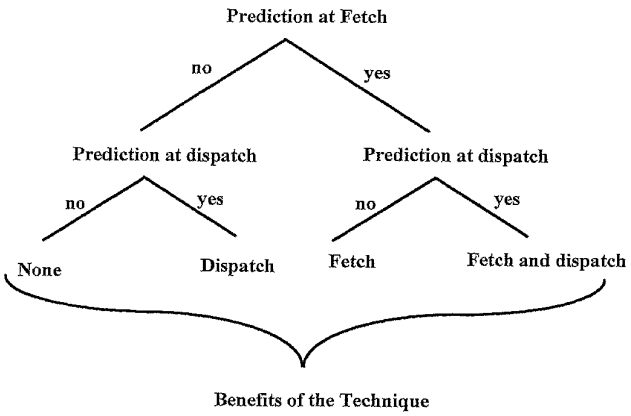

Prediction at Fetch
no
yes
Prediction at dispatch
Prediction at dispatch
no
yes
no
yes
None
Dispatch
Fetch
Fetch and dispatch
Benefits of the Technique

Fig. 4

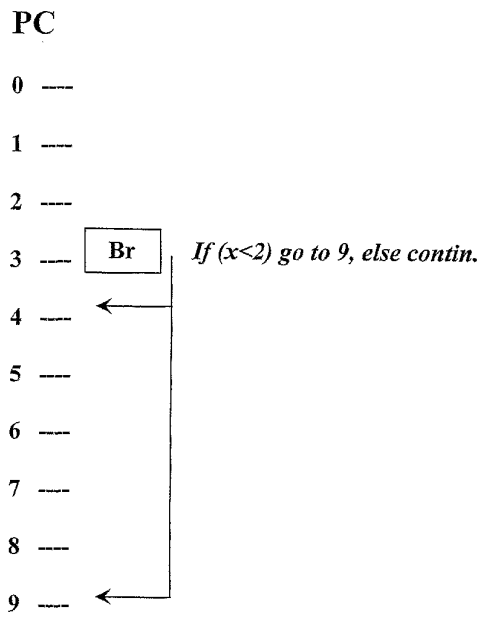


Fig. 5

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/21372

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : G06F 9/48
US CL : 709/107; 712/233,205
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 709/107; 712/233,205

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Please See Continuation Sheet

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 6,061,710 A (EICKEMEYER et al) 9 May 2000 (09.05.2000), fig.2 col. 5, .line 28-col. 9, line 52)	1-3,6-8,11-14,16-18,21-24
Y	US 6,076,157 A (BORKENHAGEN et al) 13 June 2000 (13.06.2000), column 5, lin8-column 9,line 25, col. 12 lines 1-37.	1,6,11,16,21,23
Y	US 6,029,228 A (CAI et al) 22 February 2000 (22.02.2000), see abstract, column 27, line 4-col. 32, line 65.	1-24
Y	US 5,561,776 A (POPESCU et al) 01 October 1996 (01.10.1996), column 5, lines 7-65).	1,6,11,16,21,23
A	US 4,200,927 A (HUGHES et al) 29 April 1980 (29.04.1980), column 2, lines 34-54.	1,6,11,16,21,23

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search
06 September 2001 (06.09.2001)

Date of mailing of the international search report
11 OCT 2001

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703)305-3230

Authorized officer
Eric Coleman
Telephone No. (703)-305-3900

Form PCT/ISA/210 (second sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/21372

Continuation of Item 4 of the first sheet:
The title is too long and contains superfluous language (i.e., improving).
New title:
Instruction fetch and dispatch in multithreaded system

Continuation of B. FIELDS SEARCHED Item 3:
EAST USPAT FILE
search terms: branching, probability, fetching, algorithm, predictor, multiple,multi, stream, thread, speculative

Form PCT/ISA/210 (second sheet) (July 1998)

フロントページの続き

(81)指定国 AP(GH,GM,KE,LS,MW,MZ,SD,SL,SZ,TZ,UG,ZW),EA(AM,AZ,BY,KG,KZ,MD,RU,TJ,TM),EP(AT,BE,CH,CY,DE,DK,ES,FI,FR,GB,GR,IE,IT,LU,MC,NL,PT,SE,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GW,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AT,AU,AZ,BA,BB,BG,BR,BY,BZ,CA,CH,CN,CO,CR,CU,CZ,DE,DK,DM,DZ,EC,EE,ES,FI,GB,GD,GE,GH,GM,HR,HU,ID,IL,IN,IS,JP,KE,KG,KP,KR,KZ,LC,LK,LR,LS,LT,LU,LV,MA,MD,MG,MK,MN,MW,MX,MZ,NO,NZ,PL,PT,RO,RU,SD,SE,SG,SI,SK,SL,TJ,TM,TR,TT,TZ,UA,UG,UZ,VN,YU,ZA,ZW

(74)代理人 100103920
弁理士 大崎 勝真
(72)発明者 ムソル，エンリケ
アメリカ合衆国、カリフオルニア・９５１３９、サン・ホセ、ビア・ロメラ・７２１０
(72)発明者 ネミロフスキイ，マリオ・デイ
アメリカ合衆国、カリフオルニア・９５０７０、サラトーガ、ノーサンプトン・ドライブ・１９７５０

Ｆターム(参考) 5B013 AA00 AA05 AA20 BB01
5B098 GC01