小容量 RAM を用いたオペランド・バイパスの複雑さの低減手法

三輪 忍[†] 一林宏憲[††] 入江英嗣[†††]
五島正裕[††] 富田眞治[†]

配線遅延の相対的な増大にともない，近年，長い配線を持つレジスタ・ファイルやオペランド・バイパスといったユニットがクリティカルになってきている．クリティカルなユニットをクリティカルでなくするためには，ユニットをパイプライン化することが有効である．ところが，レジスタ・ファイルのパイプライン化はオペランド・バイパスを複雑化する．オペランド・バイパスはそれ自体が既にクリティカルであり，それをこれ以上複雑にするのは受け入れがたい．この問題に対し，レジスタ・キャッシュが提案されている．レジスタ・キャッシュは，レジスタ・ファイルの一部を保持する，1 サイクルでアクセス可能な小型のバッファである．レジスタ・キャッシュを持つプロセッサは，それにヒットすれば，1 サイクルでレジスタにアクセスできる．そのため，そのようなプロセッサのオペランド・バイパスは，1 サイクルのレジスタ・ファイルを持つプロセッサのそれと同等で済む．しかし，レジスタ・キャッシュはミス・ペナルティが大きく，それを採用したプロセッサの性能は悪化してしまう．そこで我々は，レジスタ・キャッシュとほぼ同じ回路構成ながらミス・ペナルティをなくした，バイパス・バッファを提案する．本稿では，提案手法と理想化されたレジスタ・キャッシュとを比較し，提案手法を採用したプロセッサの方が高性能であることを示す．

Low-Complexity Operand Bypass Using Small RAM

Shinobu Miwa,[†] Hironori Ichibayashi,[††] Hidetsugu Irie,[†††]
Masahiro Goshima[††] and Shinji Tomita[†]

For the wire delay problem, the units with the long wires become critical such as register files and a bypass network. To prevent the units to be critical, the pipelining is an effective technique. However, the pipelining of register files complicates a bypass network. It is unacceptable that a bypass network is complicated because it is already critical. A register cache is proposed to resolve this problem. The register cache is a small buffer to cache register files. It is accessible in 1 cycle. If the instruction hits the register cache, the processor with the register cache behaves same as the processor with the non-pipelined register files. Therefore, the bypass network of the former processor is same as that of the latter processor. However, the processor with the register cache doesn't outperform because of the much register cache miss penalty. Then, we propose a bypass buffer. There is no miss penalty on the processor with it because it is not a cache. In this paper, we show that the processor with the bypass buffer achieves high performance rather than the processor with the ideal register cache.

1. はじめに

LSI の微細化により，近年，**配線遅延** の相対的な増大が問題となっている．LSI の微細化に対し，ゲート遅延 は比較的順調にスケーリングされるのに対し，配線遅延はほとんどスケーリングされないからである．

配線遅延の相対的な増大は，プロセッサ・アーキテクチャに多大な影響を及ぼす．従来は，演算器のような，ゲート遅延が支配的なユニットがクリティカルであった．配線遅延の影響が増すにつれ，それに代わって，長い配線をもつユニットがクリティカルになってきている．本稿で取り上げるレジスタ・ファイルやオペランド・バイパスなどは，そのようなユニットの代表格である．

そのようなユニットによってクロック速度が制限されるのを防ぐには，**パイプライン化**が有効である．あるユニットにより多くのパイプライン・ステージを割り当てれば，1 サイクルあたりの遅延時間を大幅に短縮でき，そのユニットをクリティカルでなくすることができる．

実際，Intel Pentium 4 プロセッサや，それ以降のプロセッサでは，レジスタ・ファイルもパイプライン化の対象となっている．レジスタ・ファイルのアクセ

† 京都大学（Kyoto University）
†† 東京大学（University of Tokyo）
††† 科学技術振興機構（JST）

ス・レイテンシは，Pentium 4 では読み出し/書き込みそれぞれ 2 サイクル[8]，Alpha 21464 ではそれぞれ 3 サイクル[13] となっている．

しかし，レジスタ・ファイルのパイプライン化は，以下の問題[7),16)] を引き起こす：

(1) 命令パイプラインの深化によって生じる一般的な問題：
  (a) 予測ミス・ペナルティの増大
  (b) 資源不足によるストールの増大☆
(2) レジスタ・ファイルのパイプライン化に固有の問題：バイパス・ネットワークの複雑化

命令パイプラインが深化すると，その分，分岐予測をはじめとする各種予測ミス・ペナルティが増す．また，命令ウィンドウや物理レジスタといった，種々の資源の解放が遅れる分，それらの不足によってフロントエンドがストールする確率が増大する．ただし，前者の問題が性能に与える影響は，予測精度の向上によってある程度補償できる．また，後者の問題が与える影響は，後述するように軽微である．

その一方で，バイパス・ネットワークは，上述のようにそれ自体が既にクリティカルであり，それをこれ以上複雑化することは受け入れがたい．

レジスタ・キャッシュ

こうした問題に対し，**レジスタ・キャッシュ**（register cache）が提案されている[1),4),7),18),19),27),28)]．レジスタ・キャッシュは，1 サイクル（以下）でアクセス可能な小型のバッファである．多くの場合，CAM を用いて実装される．メイン・メモリに対するキャッシュと同様，レジスタ・キャッシュは，**メイン・レジスタ・ファイル**（main register file）の一部を保持する．

レジスタ・キャッシュ・ミスに対する考慮点も，データ・キャッシュ・ミスに対するそれと同様である．ミスを起こした場合，かなり大きなペナルティを生じる．しかし，エントリの置き換えアルゴリズムを中心に，さまざまな研究が行われているにもかかわらず[3),4),18),27),28)]，レジスタ・キャッシュのミス率は最大 13% 程度[28)] と高く，IPC は最大 8% も低下してしまう[7)]．

バイパス・バッファ

それに対して我々は，**バイパス・バッファ**（bypass buffer）と呼ぶバッファを用い，バイパス・ネットワークの複雑さを低減する手法を提案する．

バイパス・バッファとは，レジスタ・キャッシュと同様の，1 サイクルでアクセス可能な小型のバッファである．バイパス・バッファとその周辺のデータ・パスの構成は，レジスタ・キャッシュのそれとほとんど同じである．ただし，バイパス・バッファはマルチポート RAM で実装できる．

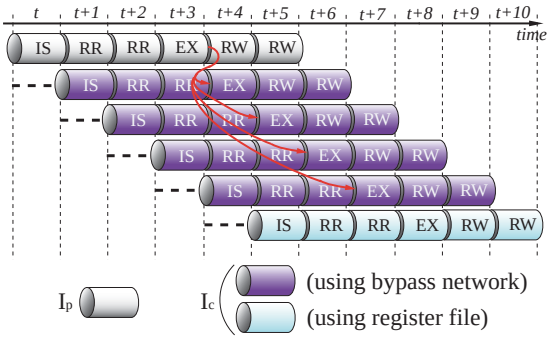

図 1 レジスタ・ファイルがパイプライン化されたプロセッサの命令パイプライン
Fig. 1 Instruction pipeline of with pipelined register file.

使い方も異なる．その名が示すとおり，バイパス・バッファは，メイン・レジスタ・ファイルをバイパスされる値を FIFO で一次的に保持する．キャッシュではないため，レジスタ・キャッシュとは異なり，ミスは起こらない．

レジスタ・キャッシュ，および，バイパス・バッファを理解するには，バイパス・ネットワークの複雑化の問題に対する理解が欠かせない．そこで本稿は，以下のようなやや変則的な構成とした．まず次章で，バイパス・ネットワークの複雑化の問題について詳しく述べる．続く 3 章では，レジスタ・キャッシュについて説明し，この問題に対するレジスタ・キャッシュの位置付けを整理する．そして，4 章でバイパス・バッファについて説明し，5 章でシミュレーションによる性能評価結果を示す．なお，レジスタ・キャッシュを持つプロセッサの性能は，レジスタ・キャッシュのエントリの置き換えアルゴリズム，レジスタ・キャッシュ・ヒット/ミス予測精度などに強く依存する．そこで本稿では，理想化されたレジスタ・キャッシュを導入し，それに比べても提案手法の方が高性能であることを示す．

2. バイパス・ネットワークの複雑さ

レジスタ・ファイルのパイプライン化は，レジスタ・ファイルを介したデータの授受に要するサイクル数を増加させる．**図 1** に，レジスタ・ファイルの読み出しと書き込みに 2 + 2 サイクルを充てた場合のパイプライン・チャートを示す．図中，**IS**/**EX** は発行/実行ステージを，**RR**/**RW** はレジスタ・ファイル読み出

☆ 文献[7),16)] では物理レジスタ不足についてのみ言及しているが，この問題は，命令ウィンドウやリターン・アドレス・スタックといった，他の資源不足と同列に考えてよい．

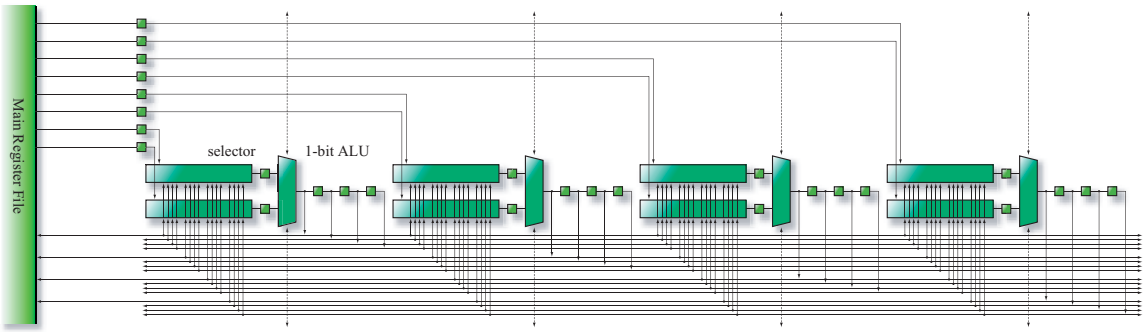

図 2 完全バイパス・ネットワーク（1 ビット・スライス分）
Fig. 2 Full bypass network for pipelined register file (1-bit slice).

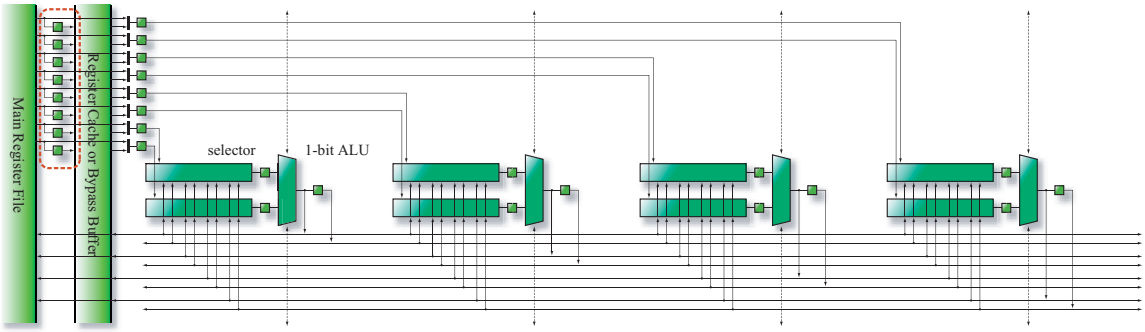

図 3 レジスタ・キャッシュとバイパス・バッファ
Fig. 3 Register cache or bypass buffer.

し/書き込みステージをそれぞれ表している．依存元の命令 I_p から，依存先の命令 I_c へと，レジスタ・ファイルを介してデータを受け渡すためには，I_p と I_c の発行間隔が 4 サイクル以上空いていなければならない（図の淡いパイプライン）．それ以下の場合には，レジスタ・ファイルを介さず，バイパス・ネットワークを介してデータを受け渡す必要がある（図の濃いパイプライン）．

そうするために，通常のバイパス・ネットワークを単純に拡張したものの回路図（1 ビット・スライス分）を**図 2** に示す．同図は，物理レイアウトを意識している．以下，この回路を**完全バイパス・ネットワーク**と呼ぶことにする．各演算器の下流には，1，2，3 サイクル前の実行結果を保持するパイプライン・ラッチが，そして，それぞれの内容をソース・ラッチへと転送するためのネットワークが必要となる．図に示したように，演算器数を 4 とすると，このネットワークは，16 入力 8 出力のクロスバー・スイッチとなる．1 ビット ALU セルにある 17 入力のセレクタの回路規模は，1 ビット ALU 本体に匹敵する．

複雑さを抑えるため，バイパスの一部を省略することが考えられるが，IPC 低下とのトレードオフになる．たとえば，最もあり得るものとして，直前のサイクルで得られた結果のみ，バイパスを介して直ちに利用する構成が考えられる．その場合，バイパスを介しても，レジスタ・ファイルを介しても結果が利用できないサイクルが生まれてしまう．運悪く，依存元の命令が発行されたサイクルの次サイクルで発行できなかった命令は，レジスタ・ファイルを介してデータが得られるタイミングまで，発行を待たされることになる．

レジスタ・キャッシュ[1),4),7),18),19),27),28)] や本稿で提案する**バイパス・バッファ**は，こうした問題に対処するものである．それでは，まずはレジスタ・キャッシュについて述べるとしよう．

3. レジスタ・キャッシュ

レジスタ・キャッシュは，**メイン・レジスタ・ファイル**の一部を保持する，1 サイクル（以下）でアクセス可能な小型のバッファである．レジスタ・キャッシュとその周辺回路のブロック図を**図 3** に示す．本稿の冒頭で触れたように，レジスタ・キャッシュとバイパス・バッファは，その周辺の回路を含めてデータ・パスの構成はほとんど同じである．ただし，同図中破線で囲んだポートは，バイパス・バッファでは必要ない．これは，レジスタ・キャッシュ・ミス時等にレジスタ・ファ

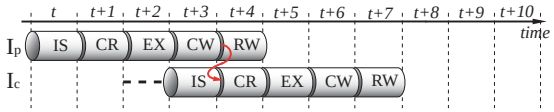

図 4 レジスタ・キャッシュを採用した命令パイプライン
Fig. 4 Instruction pipeline with register cache.

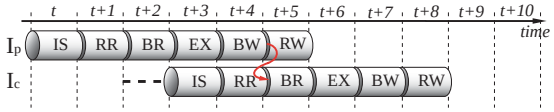

図 5 バイパス・バッファを採用した命令パイプライン
Fig. 5 Instruction pipeline with bypass buffer.

イルの内容をレジスタ・キャッシュにリフィル (refill) するためのものである．

ここで特に強調しておきたいことは，レジスタ・キャッシュは，通常の 1 次データ・キャッシュと同様，基本的には，**ヒットを想定した命令パイプライン**としていることである．**図 4** に，レジスタ・キャッシュを採用したパイプライン・チャートを示す．図中，**CR**/**CW** は，それぞれ，レジスタ・キャッシュ読み出し/書き込みステージを表している．通常のプロセッサでは，基本的には，ロード命令とそれに依存する命令は，ロード命令がキャッシュにヒットするものとして投機的にスケジューリングされる．それと同様，全ての命令は，基本的には，レジスタ・キャッシュにヒットするものとしてスケジューリングされる．

すなわちレジスタ・キャッシュを持つプロセッサは，レジスタ・キャッシュにヒットする限り，1 サイクル（以下）でアクセス可能なメイン・レジスタ・ファイルを持つプロセッサと同じと考えてよい．

3.1 レジスタ・キャッシュ・ミス

上述のようなパイプライン構成のため，レジスタ・キャッシュ・ミスに対する考慮点も，キャッシュ・ミスに対するそれと同様である．

レジスタ・キャッシュ・ミスが発生した場合のパイプライン・チャートを**図 6** に示す．ミスした命令 I_c は，改めてメイン・レジスタ・ファイルから値を読み出す．その結果，I_c，および，それに依存する I_h の実行は，ヒットした場合と比べ，2 サイクル遅れてしまう．

ミスした命令とそれに依存する命令のみを選択的に遅らせることが理想である．しかし，発行済みの命令に対してそれを行うことは，スーパスカラ・プロセッサの構造上，極めて困難である．そのため，

(1) バックエンドのストール
(2) 発行済みのすべての命令のキャンセル/再発行（フラッシュ）

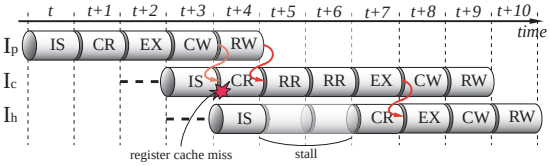

図 6 レジスタ・キャッシュ・ミス時の命令パイプライン
Fig. 6 Instruction pipeline in the case of a register cache miss.

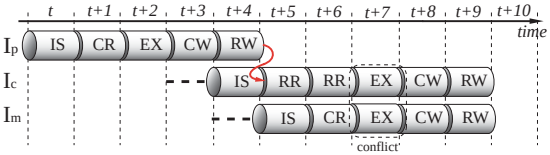

図 7 レジスタ・キャッシュ・ヒット・ミス予測した場合の命令パイプライン
Fig. 7 Instruction pipeline with a register cache hit/miss prediction.

(3) ミスを起こした命令と，それに依存する命令のキャンセル/再発行（選択的無効化）

などの手段が採られる．

メイン・レジスタ・ファイルのレイテンシが 2〜3 サイクル程度であるため，レジスタ・キャッシュのミス・ペナルティ自体は 1〜2 サイクル程度と，キャッシュのそれと比べてそれほど大きなものではない．そのため，キャッシュ・ミスの場合は選択的無効化を行うのがよいとされているが，レジスタ・キャッシュ・ミスの場合はストールで十分な可能性が高い．

一方，ミスを起こした命令，および，それに依存する命令が発行幅を浪費する影響は大きい．そのため，次節で述べる，レジスタ・キャッシュのヒット/ミス予測の援用が欠かせない．データ・キャッシュのヒット/ミス予測[17),24)] と同様，レジスタ・キャッシュ・ミスを起こすと予測した命令（とそれに依存する命令）の発行を最初から遅らせれば，余計な性能低下を抑えることができる．

3.2 レジスタ・キャッシュ・ヒット/ミス予測

レジスタ・キャッシュ・ヒット/ミス予測に対する考慮点もまた，基本的には，キャッシュ・ヒット/ミス予測に対するそれと同様である．1 次キャッシュにミスすると予測されたロード命令が 2 次キャッシュを参照するものとして発行されるのと同様，レジスタ・キャッシュにミスすると予測された命令はメイン・レジスタ・ファイルを参照するものとして発行される（**図 7**）．それにともない，ミスと予測された命令に依存する命令も遅れて発行される．最初からメイン・レジスタ・ファイルを参照するため，レジスタ・キャッシュ・ミスは

起こらず，したがってストールなども発生しない．

ただし，レジスタ・キャッシュのヒット/ミス予測は，キャッシュのそれと全く同じというわけではない．レジスタ・キャッシュ・ヒット/ミス予測では，演算器ユニットが競合する可能性を考慮する必要がある．図 7 のように，レジスタ・キャッシュにミスする命令 I_c に続き，ヒットする命令 I_m を発行したとしよう．すると，前者のレジスタ読み出しステージが後者のそれより長いため，両者は同じサイクルで実行されてしまう．

演算器ユニットの競合は，予約表を用いて回避できる．表には，各演算器が n サイクル後に使用されているかいないかを記録する．たとえば，レジスタ・キャッシュ/メイン・レジスタ・ファイルの読み出しにそれぞれ 1/2 サイクルかかるのであれば，表は，2 サイクル後のもの，および，3 サイクル後のものがあればよい．レジスタ・キャッシュにヒットする命令は 2 サイクル後のものを，ミスする命令は 3 サイクル後のものを見て，発行可能かどうか判断する．

3.3 書き込みポリシー/置き換えポリシー

レジスタ・キャッシュのミス率を抑えるため，様々なレジスタ・キャッシュ**書き込みポリシー/置き換えポリシー**が提案されてきた[3),4),18),27),28)]．前者はどの命令の結果をレジスタ・キャッシュに書き込むかという方針であり，後者はその結果をどの結果と置き換えるかという方針である．

最も単純な方法は，全命令の結果をレジスタ・キャッシュに書き込み，LRU で置き換える方法だろう．ただし，この方法は，後続命令が参照しない結果もレジスタ・キャッシュに書き込んでしまうため，エントリの利用効率が悪い．

そこで，オペランド・バイパスによって使用されなかった結果のみを書き込むというポリシーが提案されている[7)]．これは，生成された結果の 85 ～ 88% は 1 度しか使用されないことによる．しかし，複数回使用される結果も存在するため，全命令の結果を書き込んだ場合と比べて性能は悪化してしまう[27)]．

ヒット/ミス予測の要素を取り入れた方法として，クリティカル・パス予測を利用した書き込みポリシーが提案されている[27)]．クリティカル・パス上にない命令は実行が遅れても，すなわち，レジスタ・キャッシュにミスしても構わない．そこで，そのような命令をレジスタ・キャッシュに書き込まないことで，エントリの利用効率を向上させる．加えて，クリティカル・パス上にない命令をクリティカル・パス予測機構[23)]によって予測し，そのような命令をレジスタ・キャッシュにミスするものとしてスケジューリングすることで，ミス・ペナルティを軽減する．

CAM を用いるとハードウェア・コストが増すことから，RAM を用いた実装も検討されている．RAM を用いた実装では，CAM を用いる場合と比べ，競合が発生し易い．そこで，レジスタ・プールを工夫し，レジスタ・キャッシュに対する書き込みがサイクリックになるよう，物理レジスタを割り当てる方法が提案されている[28)]．しかし，そのような工夫を行ったとしても，ミス率は平均 13% 程度と高く，CAM を用いた場合と比べ，性能は 4.9% 程悪化してしまう．

3.4 レジスタ・キャッシュの持つ意味

レジスタ・キャッシュは，本稿の冒頭で述べた，レジスタ・ファイルのパイプライン化によって生じる以下の問題点を，全て緩和すべく提案されたものである：

(1) 命令パイプラインの深化によって生じる一般的な問題：
  (a) 予測ミス・ペナルティの増大
  (b) 資源不足によるストールの増大
(2) レジスタ・ファイルのパイプライン化に固有の問題：バイパス・ネットワークの複雑化

レジスタ・キャッシュでは，端的に言えば，ヒットする限り，レジスタ読み出し/書き込みが 1 サイクル（以下）で行えるため，全ての問題は，1 サイクル（以下）でアクセス可能なメイン・レジスタ・ファイルをもつ場合と同等で済むことになる．

以下，それぞれについて詳しく述べる．

バイパス・ネットワークの簡単化

図 2 の完全バイパス・ネットワークの回路と，図 3 のレジスタ・キャッシュの回路とを比べてみよう．すると，両者は機能的には等価であり，回路構成上の違いであることが分かる．クロスバー・スイッチを，前者は，個別の論理 —— ラッチ，配線とセレクタで；後者はバッファ —— マルチポート RAM で構成している．

前者が複雑で許容できない一方で，後者が 1 サイクル（以下）でアクセス可能であるのは，マルチポート RAM がこのような機能の実現のために究極的に最適化された回路だからであると言える．

分岐予測などのミス・ペナルティの低減

一方で，分岐予測などのミス・ペナルティの低減の効果は，わずかである．メイン・レジスタ・ファイルの読み出しレイテンシを 2 サイクル，レジスタ・キャッシュの読み出しレイテンシを 1 サイクルとすると，分岐予測ミス・ペナルティの低減量は 1 サイクルである．近年のスーパスカラ・プロセッサでは，分岐予測ミス・ペナルティは，10 サイクル程度なので，低減

率は 10% 程度である．分岐出現率を 20%，ヒット率を 90% としても，4 命令発行のプロセッサでは，分岐予測ミス以外にパイプラインを乱す要因がない場合でさえ，性能に対する影響は 4.5% に過ぎない．

5 章で詳しく評価するが，この程度の低減率では，レジスタ・キャッシュ自体のミスによる性能低下に見合わない．エントリの置き換えアルゴリズムを中心に，さまざまな研究が行われているにもかかわらず，レジスタ・キャッシュのミス率は最大 13% 程度[28] と，1 次データ・キャッシュのそれに比べて著しく高い．さらに言えば，キャッシュ・ミスはロード/ストア命令のみに起こりえるのに対して，レジスタ・キャッシュ・ミスはほとんどすべての命令に対して起こりえる．その結果，IPC は，最大 8% も低下してしまう[7]．

資源不足の緩和

資源不足を緩和する効果もまたわずかである．

命令に割り当てられた資源は，一般に，コミット時に解放される．命令ウィンドウやリターン・アドレス・スタックの解放は，それらが割り当てられた命令のコミット時に行われる．物理レジスタの解放は，これらとはタイミングが異なるものの，同一の論理レジスタをデスティネーションとする後続命令の，やはりコミット時である．

命令がレジスタ・キャッシュにヒットすれば，実行は早くなる．しかし，コミットも早くなるとは限らない．命令がコミットされるタイミングは，その命令の実行レイテンシだけでなく，先行命令のそれにも支配される．先行命令の中にメイン・メモリにアクセスするような命令があれば，後続命令のコミットは，レジスタ・キャッシュにヒットしても早くならない．

そのため，資源の解放が早くなるのは，パイプラインがスムーズに流れている状態でレジスタ・キャッシュにヒットする，という極めて稀な状況に限られる．レジスタ・キャッシュ/メイン・レジスタ・ファイルのレイテンシをそれぞれ 1/2 サイクルとすると，そのような状況でようやく，解放が 1 サイクル早くなる．物理レジスタの寿命は 100 サイクル[14] 程度であるから，パイプラインがスムーズに流れ続けたとしても，寿命はわずか 1% 縮むだけである．

そこで我々は，レジスタ・キャッシュと同等（以下）の回路構成からなる，バイパス・バッファを提案する．提案手法は，レジスタ・キャッシュのように，分岐予測などのミス・ペナルティ，および，物理レジスタ不足といった，一般的な問題を緩和する効果はない．その一方，キャッシュではないためミスは起こらない．次章では提案手法について詳しく述べる．

4. バイパス・バッファ

我々は，一般的な問題の緩和を放棄することと引き換えに，レジスタ・キャッシュ・ミスをなくすことを考えた．前述したように，レジスタ・キャッシュはヒットを想定した命令パイプラインの構成である．一般的な問題の緩和とレジスタ・キャッシュ・ミスの効果の交換は，いわば**ミスを想定した命令パイプライン構成**をとることによって実現できる．

提案手法で用いる**バイパス・バッファ**は，レジスタ・キャッシュと同様の，1 サイクルでアクセス可能な小型のバッファである．前述したように，バイパス・バッファおよび周辺回路のデータ・パスの構成は，レジスタ・キャッシュのそれとほとんど同じである．

異なるのは，その使い方である．

その名が示すとおり，バイパス・バッファは，図 2 に示した完全バイパス・ネットワークと，機能的に等価である．バイパス・バッファは，完全バイパス・ネットワークのラッチが保持していたバイパスされる値を，代わりに FIFO で保持する．毎サイクル，各演算器が生成した値が次々に書き込まれる一方で，メイン・レジスタ・ファイルから取得できるようになった古い値は次々に捨てられる．

図 5 に，提案手法の命令パイプラインを示す．図中，**BR**/**BW** は，それぞれ，バイパス・バッファ読み出し/書き込みステージを表す．レジスタ・キャッシュとは異なり，提案手法では必ずメイン・レジスタ・ファイルにアクセスする．

バイパス・バッファは，上述のように，メイン・レジスタ・ファイルをバイパスされる値を FIFO で一次的に保持している．したがって，必要な値がメイン・レジスタ・ファイルから取得できないときは（また，そのときに限り）バイパス・バッファから取得できるので，レジスタ・キャッシュのようなミスは起こらない．

4.1 ハードウェア・コスト の比較

レジスタ・キャッシュに対するバイパス・バッファのハードウェア・コストは，以下のようにまとめられる：

バッファ

ポート数 図 3 から分かるように，ポート数はバイパス・バッファのほうが少ない．

容量 バイパス・バッファの容量 は，同時にメイン・レジスタ・ファイルをバイパスされ得るデータの総量となる．たとえば，メイン・レジスタ・ファイルの書き込み，読み出しレイテンシをそれぞれ 2 サイクル，演算器数を 4 とす

ると，容量は 16 ((2+2)∗4) エントリとなる．レジスタ・キャッシュの容量は 16〜32 エントリ程度が想定されており[1),4),7),18),19),27),28)]，バイパス・バッファの容量はそれと同等以下である．

したがって，レジスタ・キャッシュが 1 サイクルでアクセス可能なら，バイパス・バッファも 1 サイクルでアクセス可能である．

制御 メイン・レジスタ・ファイルをバイパスされる値を FIFO で一次的に保持するだけであるので，レジスタ・キャッシュの置き換えアルゴリズムのような複雑な制御は必要ない．

また，レジスタ・キャッシュのように，値を物理レジスタ番号で検索する必要もない．前述のように，バイパス・バッファの各エントリは完全バイパス・ネットワークの各ラッチと，完全に 1 対 1 に対応する．そのため，結果を生成した演算器と生成したサイクルとによって，その結果が格納されているエントリは一意に決まる．

したがって，バイパス・バッファは，レジスタ・キャッシュとは異なり，RAM でよい．

4.2 IPC の比較

レジスタ・キャッシュに対するバイパス・バッファの IPC は，以下のようにまとめられる：

予測ミス・ペナルティ パイプラインはメイン・レジスタ・ファイルに合わせて深いままなので，分岐予測などの予測ミス・ペナルティはレジスタ・キャッシュと比べてわずかだが大きい．たとえば，メイン・レジスタ・ファイルの読み出しレイテンシが 2 サイクルであるとすると，その差は 1 サイクルになる．

レジスタ・キャッシュ・ミス バイパス・バッファでは，レジスタ・キャッシュのようなミスは生じない．

したがって，性能差は：

(1) レジスタ・キャッシュにおける，分岐予測などの予測ミス・ペナルティの減少による性能向上 と，
(2) 同じくレジスタ・キャッシュにおける，レジスタ・キャッシュ・ミスによる性能低下

との大小関係による．前者が大きければレジスタ・キャッシュが，後者が大きければ提案手法が高い性能を示すことになる．

5. 評 価

前述のように，レジスタ・キャッシュを持つプロセッサの性能は，レジスタ・キャッシュのエントリの置き換えアルゴリズム，レジスタ・キャッシュ・ヒット/ミス予測精度などに強く依存する．そこで以下では，理想化されたレジスタ・キャッシュを導入し，それと提案手法とを比較する．

5.1 評価モデル

本稿では，以下の 6 つのモデルについて評価を行う：

LRU LRU によってエントリを置き換えるレジスタ・キャッシュを用いたモデル．簡単のため，レジスタ・キャッシュには演算結果を全て書き込むものとする．

LRU w H/M 上記のモデルにレジスタ・キャッシュ・ヒット/ミス予測を加えたモデル．ヒット/ミス予測は，当該物理レジスタの直前のヒット/ミスに基づいて行う．たとえば，前回そのレジスタを参照した際にレジスタ・キャッシュにヒットしたならば，次もヒットと予測する．

OPT 後述する OPT[22)] によってエントリを置き換えるレジスタ・キャッシュを用いたモデル．OPT による置き換えは，最小のミス率を保証する．

OPT w H/M perfect 上記のモデルにレジスタ・キャッシュ・ヒット/ミス予測を加えたモデル．予測は完全にヒットする．したがって，ストールなどのレジスタ・キャッシュ・ミス・ペナルティは一切発生しない．このモデルがレジスタ・キャッシュの性能の上限を与える．

PERFECT 完全にヒットするレジスタ・キャッシュを用いたモデル．メイン・レジスタ・ファイルと同エントリ数で，なおかつ，1 サイクルでアクセスできる，という仮想的なレジスタ・ファイルが存在した場合の性能．

BB バイパス・バッファを用いたモデル

OPT とその実装

OPT は，将来に渡って最も長期間使用されない値を置き換えの対象とする，理想的な置き換えアルゴリズムである．そのため，OPT によるミス率は，全ての置き換えアルゴリズムの中で最小となる．

また，OPT は，理想的な書き込みアルゴリズムへと容易に拡張できる．最も使用されない値がこれから書き込もうとする値だったならば，それを書き込まなければよい．

このように，OPT は，現実には実現不可能なアルゴリズムであるが，キャッシュの性能の指標となる．

OPT のレジスタ・キャッシュの実装には工夫が必要である．命令は Out-of-Order に実行されるため，最も使用されない値を求めることは難しい．

そこで，置き換えを厳密にシミュレートするのでは

なく，各レジスタ・キャッシュ・アクセスのヒット/ミスのみをシミュレートする．同じレジスタが以前参照されてから現在までに，レジスタ・キャッシュの容量を越える他のレジスタ・アクセスがなければ，現在のアクセスはヒットする．逆に，容量を越えていればミスする．このように，各アクセスのヒット/ミスは，将来の参照系列ではなく，参照履歴から求めることができる．

完全にヒットするヒット/ミス予測は，以下のようにして実装する．まず，予測時に，1 サイクル後の，すなわち，実際にアクセスする際のレジスタ・キャッシュの状態を求める．同一サイクル内でのシミュレーションは，下流のステージから上流のステージに向かって行われる．そのため，発行ステージでは，次サイクルでレジスタ・キャッシュに読み書きする命令が全て判明しており，次サイクルのレジスタ・キャッシュの状態は容易に求まる．後は，求めた将来のレジスタ・キャッシュに対するヒット/ミスを調べ，それを予測結果とすればよい．

上述の 6 つのモデルを SimpleScalar ツールセット (ver. 3.0)[20] の sim-outorder シミュレータに対して実装し，評価を行った．命令セット・アーキテクチャは PISA を用いた．評価したプロセッサのパラメタを**表 1** に示す．表のパラメタは，Intel Core アーキテクチャに準じている．メイン・レジスタ・ファイルは int, fp 各 64 個，分岐予測ミス・ペナルティは 15 サイクルとした．リネームの際，命令ウィンドウ・エントリ/物理レジスタがフェッチ幅のそれぞれ 1/2 倍以上残っていない場合は，フロントエンドをストールさせるものとする．レジスタ・キャッシュ・ミス時にはバックエンドをストールさせ，メイン・レジスタ・ファイルの内容をリフィルする．

表 1 プロセッサの各パラメタ
Table 1 Parameters of the processor.

parameter	remarks
fetch width	4 inst.
issue width	4 inst.
instruction window	64 ent.
main register file	int : 64, fp : 64
branch prediction	2KB g-share
	misspenalty : 15 cycle
BTB	4K entry, 4-way
RAS	16 entry
L1C	32KB, 8-way, 64B/line, 3 cycle
L2C	1MB, 8-way, 64B/line, 10 cycle
main memory	first chunk : 32 cycle
	remain : 3 cycle
execution unit	6 (int : 2, fp : 2, LD/ST : 2)

測定には，SPEC CINT2000 の 8 本のプログラムを使用した．入力セットには train を用いた．プログラムは，最初の 1G 命令をスキップし，続く 100M 命令を実行した．

5.2 評価結果

各モデルの IPC を**図 8** に示す．2 つのグラフは，上がメイン・レジスタ・ファイルのレイテンシが 2 サイクルの場合，下が 3 サイクルの場合である．グラフの横軸はベンチマーク（一番右は平均）を，縦軸は IPC を表す．ただし， IPC は，完全バイパス・ネットワークを持つプロセッサのそれで正規化してある．

ベンチマーク毎の 6 本の棒グラフは，左から順に，前節で述べた 6 個のモデルに対応する．1 本目から 4 本目までは，棒グラフが 2 つ重なり合っている．それぞれ，手前の薄い方がレジスタ・キャッシュの容量がメイン・レジスタ・ファイルの 1/8（int, fp それぞれ 8 エントリ）の場合を，奥の濃い方が 1/4 （それぞれ 16 エントリ）の場合を表す．なお，バイパス・バッファのエントリ数は，演算器が int, fp それぞれ 2 個（表 1）であるから，2 サイクルの方はそれぞれ 8 エントリ，3 サイクルの方はそれぞれ 12 エントリである．

グラフより，メイン・レジスタ・ファイルが 2 サイクルの場合（図 8 上），レジスタ・キャッシュを採用したモデルは，OPT（左から 3 本目の薄い棒グラフ）でさえ BB の性能に及ばない．前述のように，バイパス・バッファは完全バイパス・ネットワークと機能的に等価であるから，その性能は完全バイパス・ネットワークのそれに等しい．一方，バイパス・バッファと同一エントリ数，すなわち，1/8 の OPT の性能は，BB のそれを最大 14.2%，平均 4.7% 下回る．

また，2 サイクルの場合，BB の性能は，ストールが一切発生しないモデル（一番右の薄い棒グラフ）と比べても遜色ない．1/8 の OPT w H/M perfect の性能は，BB のそれを平均でわずか 0.08% 上回るに過ぎない．加えて，レジスタ・キャッシュに完全にヒットした場合（右から 2 番目の濃い棒グラフ）でさえ，その性能は，BB のそれを平均 3.8% 上回るに過ぎない．

なお，命令ウィンドウのエントリ不足，および，物理レジスタ不足の発生は，命令パイプラインの深さと全く関係がない．どちらの不足の発生も，PERFECT が最も少なく，LRU が最も多かった．PERFECT で命令ウィンドウ・エントリが不足したサイクル数は，BB のそれと比べて，最大 18.8%，平均 11.1% 少なかった．一方，LRU のそれは，BB のそれと比べ，最大 278%，平均 24.4% 多かった．物理レジスタが不足

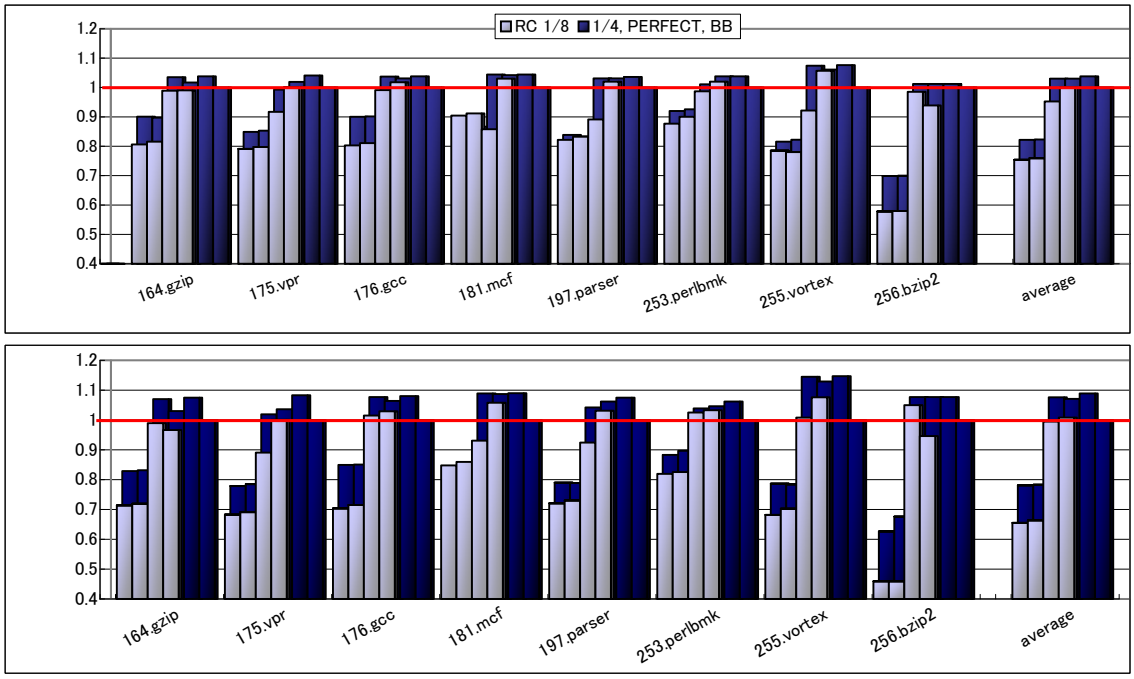

図 8 IPC（上：メイン・レジスタ・ファイルのレイテンシが 2 サイクルの場合，下：3 サイクルの場合）
Fig. 8 IPC (upper : in the case that the main register file's latency is 2 cycles, below : in the case of 3 cycles)

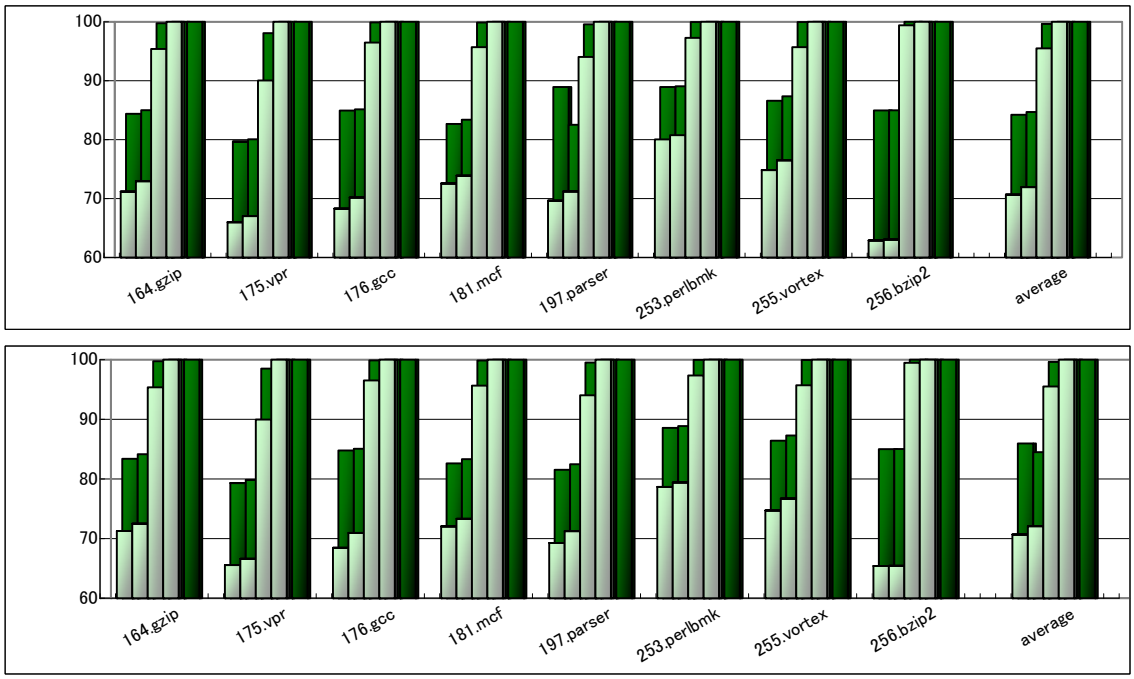

図 9 レジスタ・キャッシュヒット率（上：メイン・レジスタ・ファイルのレイテンシが 2 サイクルの場合，下：3 サイクルの場合）
Fig. 9 Hit rate of register cache (upper : in the case that the main register file's latency is 2 cycles, below : in the case of 3 cycles)

したサイクル数は，BB のそれと比べて，PERFECT のそれが平均 9.8% 少なく，LRU のそれが平均 69.2% 多かった．

メイン・レジスタ・ファイルが 3 サイクルの場合もまた，BB の性能は，1/8 の OPT のそれを上回る（図 8 下）．その向上率は，平均 0.63% である．

レジスタ・キャッシュを採用したモデル（BB を除く 5 つのモデル）の レジスタ・キャッシュ・ヒット率を**図 9** に示す．グラフの見方は，縦軸がヒット率になった点を除き，図 8 と同様である．

上述のように，レジスタ・キャッシュを採用したモデルの性能は，1/8 の OPT でさえ，BB のそれに及ばない．グラフより，そのレジスタ・キャッシュ・ヒット率は，どちらのレイテンシの場合も平均 95.5% である．

言い換えると，これ以上のヒット率でなければレジスタ・キャッシュを採用する意味はない．

6. 関 連 研 究

バイパス・バッファと同様，メイン・レジスタ・ファイルにまだ書き込まれていない演算結果を FIFO で保持する機構に，Alpha 21464 のレジスタ・キャッシュや**フォワーディング・バッファ**（forwarding buffer）がある[2),12),13)]☆．これらの機構では，バイパス・バッファと同様，バッファから結果をフォワーディングすることで，まだ書き込まれていない結果を利用する．

ただし，バイパス・バッファとは異なり，これらのバッファはメイン・レジスタ・ファイルのライトバック・キューとして機能する．各演算器が生成した結果は全て，このバッファの末尾へ書き込まれる．そして，書き込みポートが獲得されると，バッファの先頭から順に，メイン・レジスタ・ファイルへ結果を書き込む．このようにライトバック・キューを兼ねているため，これらのバッファは CAM を用いて実装される．

4 章で述べたように，バイパス・バッファでは，結果を生成した演算器とサイクルとによって，書き込まれるエントリが一意に決まる．そのため，これらとは異なり，バイパス・バッファは RAM でよい．

7. ま と め

メイン・レジスタ・ファイル以外のユニットがパイプライン化されることで，命令パイプラインは，将来，評価に用いた 15 サイクルよりも深くなると予測される．その場合，予測ミス・ペナルティに占めるメイン・レジスタ・ファイルのレイテンシの割合が低下するため，バイパス・バッファに対するレジスタ・キャッシュの利点であった，「予測ミス・ペナルティを軽減する効果」が今以上に失われる．

また，分岐予測器も，本稿では 2KB g-share という精度の低いものを用いていたが，将来的には，さらなる精度向上が期待できる．実際，g-share より 10% 優れたもの[9)] やそれ以上の精度のもの[10),11),15),25),26)] も既に提案されている．分岐予測精度の向上もまた，レジスタ・キャッシュの利点を失わせる．

前述のように，バイパス・バッファのハードウェア量はレジスタ・キャッシュのそれ以下である．したがって，今回の結果から，バイパス・バッファに対するレジスタ・キャッシュの利点は，その将来性も含め，全くないと結論づけることができる．

謝辞

本研究の一部は，日本学術振興会 科学研究費補助金 基盤研究 S（課題番号 16100001），および 21 世紀 COE プログラム（課題番号 14213201）による．

参 考 文 献

1) Blasubramonian, R., Dwarkadas, S. and Albonesi, D. H.: Reducing the Complexity of the Register File in Dynamic Superscalar Processors, *Proc. of the 34th International Symposium on Microarchitecture*, pp. 237–248 (2001).
2) Borch, E. and Tune, E.: Loose Loops Sink Chips, *Proceeding of 8th International Symposium on High Performance Computer Architecture*, pp. 299–310 (2002).
3) Butts, J. A. and Sohi, G. S.: Characterizing and Predicting Value Degree of Use, *Proc. of the 35th International Symposium on Microarchitecture*, pp. 15–26 (2002).
4) Butts, J. A. and Sohi, G. S.: Use-Based Register Caching with Decoupled Indexing, *Proc. of the 31st International Symposium on Computer Architecture*, pp. 302–313 (2004).
5) B.Wijeratne, S. et al.: A 9GHz 65nm Intel Pentium 4 Processor Integer Execution Core, *Ineternational Solid-State Circuits Conference*, pp. 353–365 (2006).
6) B.Wijeratne, S. et al.: A 9-GHz 65-nm Intel Pentium 4 Processor Integer Execution Unit, *IEEE Journal of Solid-State Circuits*, Vol. 42, No. 1, pp. 26–37 (2007).
7) Cruz, J.-L., González, A., Valero, M. and Topham, N. P.: Multiple-Banked Register File

☆ Intel Pentium 4 には，バイパス・キャッシュ（bypass cache）という機構がある．文献[5),6)] からはその詳細は不明であるが，特許内容[21)] からはフォワーディング・バッファなどと同様の機構であると推定できる．

Architectures, *Proc. of the 27th International Symposium on Computer Architecture*, pp.316–325 (2000).
8) Intel Corp.: *A detailed look inside the NetBurst micro-architecture of the Intel Pentium 4 processor* (2000).
9) Jiménez, D. A., Keckler, S. W. and Lin., C.: The Impact of Delay on the Design of Branch Predictors, *Proceedings of the 33rd Annual International Symposium on Microarchitecture*, Monterey, CA, pp. 67–76 (2000).
10) Jiménez, D. A. and Lin., C.: Fast Path-Based Neural Branch Prediction, *Proceedings of the 36th Annual International Symposium on Microarchitecture*, San Diego, CA (2003).
11) Jiménez, D. A. and Lin., C.: Piecewise Linear Branch Prediction, *Proceedings of the 32nd International Symposium on Computer Architecture*, Madison, Wisconsin, pp. 382–393 (2005).
12) Kim, N. S. and Mudge, T. N.: Reducing register ports using delayed write-back queues and operand pre-fetch, *Proc. of the 17th Annual International Conference on Supercomputing*, pp. 172–182 (2003).
13) Preston, R. P. et al.: Design of an 8-wide Superscalar RISC Microprocessor with Simultaneous Multithreading, *International Solid-State Circuits Conference*, pp. 334–335 (2002).
14) Rama Sangireddy and, A. K. S.: Exploiting Quiescent States in Register Lifetime, *Proc. of the 22nd IEEE International Conference on Computer Design*, pp. 368–374 (2004).
15) Seznec, A.: Analysis of the O-GEometric History Length branch predictor, *Proceedings of the 31st International Symposium on Computer Architecture*, Madison, Wisconsin, pp. 394–405 (2005).
16) Tullsen, D. M., Eggers, S. J., Emer, J. S., Levy, H. M., Lo, J. L. and Stamm, R. L.: Exploiting Choice : Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor, *Proc. of the 23rd International Symposium on Computer Architecture*, pp. 191–202 (1996).
17) Yoaz, A., Erez, M., Ronen, R. and Jourdan, S.: Speculation Techniques for Improving Load Related Instruction Scheduling, *Proc. of the 26th International Symposium on Computer Architecture*, pp. 42–53 (1999).
18) Yung, R. and Wilhelm, N. C.: Caching Processor General Registers, *Proc. of the International Conference on Computer Design*, pp. 307–312 (Oct. 1995).
19) Zalamea, J., Llosa, J., Ayguadé, E. and Valero, M.: Two-Level Hierarchical Register File Organization for VLIW Processors, *Proc. of the 33th International Symposium on Microarchitecture*, pp. 137–146 (2000).
20) http://www.simplescalar.com/.
21) http://www.eipaweb.org/CPUs/Find-patent-Multiported-bypass-cache-in-a-bypass-network-135933.htm.
22) 萩原宏, 津田孝夫, 大久保英嗣: 現代 オペレーティングシステムの基礎, オーム社, chapter 6 (1988).
23) 千代延昭宏, 佐藤寿倫, 有田五次郎: 低消費電力プロセッサアーキテクチャ向けクリティカルパス予測器の提案, 情報処理学会研究報告 2002–ARC–149, pp. 1–6 (2002).
24) 福田祥貴, 片山清和, 島田俊夫: ライン・バッファ・ヒット/ミス予測を利用した動的命令スケジューリングの高精度化手法, 先進的計算基盤システムシンポジウム SACSIS 2003, pp. 227–234 (2003).
25) 石井康雄, 平木敬: 実行パス履歴情報を利用した分岐予測手法, 情報処理学会論文誌, Vol. 47, No. SIG 3 (ACS 13), pp. 58–72 (2006).
26) 三輪忍, 福山智久, 嶋田創, 五島正裕, 中島康彦, 森眞一郎, 富田眞治: パス情報を用いた分岐フィルタ機構, 情報処理学会論文誌, Vol. 47, No. SIG 12 (ACS 15), pp. 108–118 (2006).
27) 小林良太郎, 梶山太郎, 島田俊夫: クリティカル・パスに着目した階層型レジスタ・ファイル, 先進的計算基盤システムシンポジウム, 大阪, pp. 33–40 (2006).
28) 小林良太郎, 堀部大介, 島田俊夫: 物理レジスタ番号の割当順に着目したレジスタ・キャッシュの高精度化手法, 先進的計算基盤システムシンポジウム, 大阪, pp. 13–22 (2006).