Vol.2009-ARC-184 No.27
2009/8/5

面積効率を指向するプロセッサの設計

堀尾一生[†1]　塩谷亮太[†1]
五島正裕[†1]　坂井修一[†1]

本論文は面積効率の高いスーパスカラ・プロセッサの構成方式を提案するものである．
近年，ウェイ数の大きなスーパスカラ・プロセッサの面積効率を高める技術が多く提案されている．一つ一つの技術はスーパスカラ・プロセッサの一つの構成要素を対象としたものだが，既にスーパスカラ・プロセッサの全域をカバー可能な程，技術の選択肢は充実している．これらの技術を組み合わせれば，ウェイ数の大きなスーパスカラ・プロセッサを現実的な回路面積で実現可能だと考えている．
しかしこれら個別の技術は異なるスーパスカラ・プロセッサの構成を想定しており，単純に組み合わせることはできないという問題がある．
我々が提案するのは，これらの技術を一つに組み合わせられる特殊な構成のスーパスカラ・プロセッサである．本研究はこのプロセッサを実際のチップの形にすることを最終目標とする．そのファースト・ステップとして，現在 FPGA 上に実装することを目指している．

Design of Area-efficient Processor

KAZUO HORIO,[†1] RYOTA SHIOYA,[†1] MASAHIRO GOSHIMA[†1]
and SHUICHI SAKAI [†1]

This paper proposes an area-efficient design of superscalar processor.
Recent years, many techniques have been proposed which contribute to reducing the area of superscalar processor. Individual techniques only reduce the area of specific elements of processor, but the list of techniques is now sifficient to cover the entire chip. Combining all such techniques, a wide superscalar processor can be achieved at a realistic area.
The problem exists in combining the techniques, however. The individual techniques target different base designs of superscalar processor. They can't simply be put together as they are.
We propose a special design of superscalar processor that can incorporate all techniques into one chip. The goal of our research is actually fabricating the chip. As its first step, we will implement it on FPGA. Since RAMs on FPGA are equipped with severly limited number of ports for a superscalar processor, the implemented design will prove to be highly area-efficient if custom designed.

1. はじめに

スーパスカラ・プロセッサの性能——IPC を向上させる一次的な方法は，そのウェイ数を増やすことである．しかし，IPC はウェイ数に比例して増加する訳ではない．その一方で，標準的なスーパスカラ・プロセッサ のデザインでは，各部の回路規模はウェイ数のほぼ 3 乗に比例して増大してしまう[2]．これは各部の中心となるハードウェアが多ポートの RAM で構成されており，RAM の回路面積はポート数の 2 乗に比例するからである．そのため，ウェイ数を無闇に増加させると，回路面積は現実的な線を超えてしまう．したがって最近では，各コアのウェイ数は 2 程度に抑え，コアを多数並べることによって性能向上を目指すことが現実的な解だと認識されている[7]．

これに対し本研究室では，回路規模がウェイ数の 3 乗には比例しないような技術をいくつも提案してきた．発行幅を増やさずに命令のスループットを増やせる**ツインテール・アーキテクチャ**，スケジューリング・ロジックに対しては**マトリックス・スケジューラ**[3]，リネーミング・ロジックに対しては**リネームド・トレース・キャッシュ**[5]，レジスタ・ファイルに対しては**非レイテンシ指向レジスタ・キャッシュ・システム**[6] を提案した．

これらの技術はスーパスカラ・プロセッサの各部に対応している．これらを一つのプロセッサの中に全て組み合わせることができれば，比較的ウェイ数の大きなスーパスカラ・プロセッサを現実的な面積で実現することが可能になる．

しかし組み合わせることは容易ではない．これらの技術はそれぞれ異なるスーパスカラ・プロセッサの構成方式を想定しているため，単純に組み合わせようとすると矛盾が生じる．具体的には，ツインテール・アーキテクチャはフロントエンドでレジスタ読み出しを行うスーパスカラ・プロセッサの構成方式を想定しているが，非レイテンシ指向レジスタ・キャッシュはバックエンドでレジスタ読み出しを行う構成方式を想定している．さらにリネームド・トレース・キャッシュに関してはどちらでもない特殊な構成を想定しており，これらを矛盾なく，そのまま組み合わせることはできない．

本研究が提案するのは，これらの技術を一つに組み合わせられる特殊な構成のスーパスカ

†1 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†2 東京大学大学院情報理工学系研究科・日本学術振興会特別研究員 DC
Graduate School of Information Science and Technology, The University of Tokyo

ラ・プロセッサである．提案する構成は物理レジスタ・ファイル及び命令キューをリング・バッファで構成し，フロントエンドにフューチャ・ファイルを配置する．この構成により上記の技術を全て一つのプロセッサに組み込むことができる．

本研究はこのプロセッサを実際のチップの形にすることを最終目標とする．そのファースト・ステップとして，現在 FPGA 上に実装することを目指している．

FPGA 上の RAM は 2 ポート程度しか備えていない．これを用いてスーパスカラ・プロセッサを実装するには，あらゆるハードウェアに対してポート数を削減する技術を活用することが必須である．よって FPGA 上で高速に動作するスーパスカラ・プロセッサが設計できれば，それはカスタムチップ上でも高い面積効率を達成できる．

本稿は以下の構成をとる．まず 2 章で一般的なスーパスカラ・プロセッサの構成について説明する．次に 3 章で組み合わせる個別の技術について簡単に説明する．4 章では提案する構成について述べる．最後に 5 章でまとめを述べる．

2. プロセッサの構成方式

本章では一般的なスーパスカラ・プロセッサの構成方式について簡単に説明する．次章で説明する特殊な技術は，本章で説明するいずれかの構成方式を想定している．

スーパスカラ・プロセッサの構成はデザイン毎に異なるが，概ね 2 種に大別して整理できると考えている．本研究ではスーパスカラ・プロセッサの構成方式を

- フロントエンドでレジスタ・ファイルを読み出す方式 (以下**フロントエンド方式**と呼ぶ)
- バックエンドでレジスタ・ファイルを読み出す方式 (以下**バックエンド方式**と呼ぶ)

の 2 種類に整理している．

図 1 と図 2 にそれぞれのブロック図を示す．

本稿ではフロントエンド方式とバックエンド方式において，命令ウィンドウに位置するハードウェアを，それぞれ**リザベーション・ステーション**と**命令キュー**と呼んで区別している．両者の違いは，リザベーション・ステーションはオペランドを保持するが，命令キューは保持しないという点である．

以下ではパイプラインの流れを追って，それぞれの動作を説明する動作の説明に当たっては，両方式に共通のフェッチやウェイク・アップ／セレクトなどの説明を省く．

2.1 フロントエンド方式の動作

フェッチの説明を省き，続くレジスタ読み出しの動作から説明する．

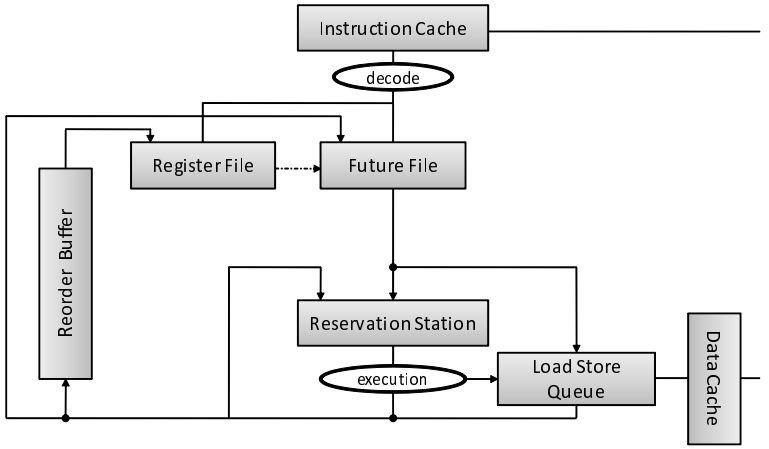

図 1 フロントエンド方式
Fig. 1 A system that reads the register file at the frontend

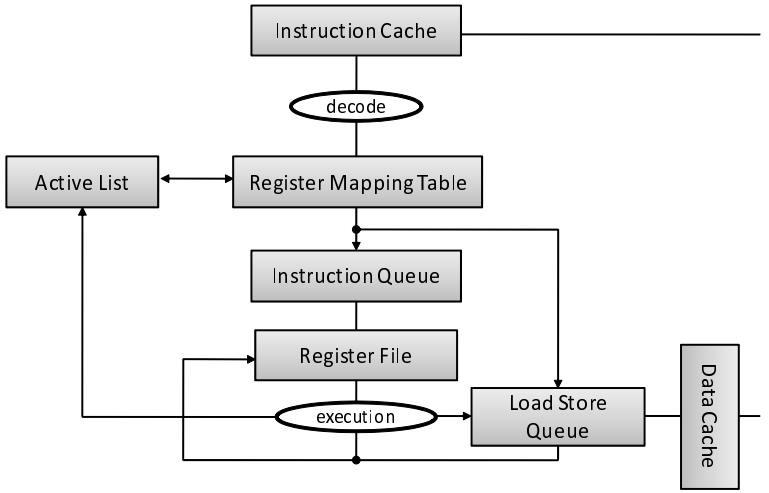

図 2 バックエンド方式
Fig. 2 A system that reads the register file at the backend

2.1.1 レジスタ読み出し

フューチャ・ファイルを読んでオペランドを得る．同時にそれぞれの命令が**リオーダ・バッファ**にエントリをアロケートし，そのインデックスをタグとして受け取る．

フロントエンドには命令をスケジューリングする機能はない．すなわち命令はオペランドがレディになるまで待機せずに，レジスタ読み出しを行う．従ってレジスタ読み出しの時点で全てのオペランドが揃うとは限らない．

2.1.2 ディスパッチ〜発行

リザベーション・ステーションに命令とオペランドを書き込み，読みだして演算器に送る．

2.1.3 実行〜書き込み

命令の実行結果はいくつかのハードウェアに重複して同時に書き込まれる．

リザベーション・ステーション 現在リザベーション・ステーションの中で待機中の (近い) 命令にオペランドを渡す．

フューチャ・ファイル これからリザベーション・ステーションに入る (遠い) 命令はオペランドをフューチャ・ファイルから入手する．

リオーダ・バッファ イン・フライトな命令全ての実行結果を保持している．正しいアーキテクチャ・ステートの再現に必要．

2.1.4 リ タ イ ア

リオーダ・バッファは命令をプログラム・オーダに格納しており，どの命令がリタイアしてよいかを判定する．リタイアの際には命令毎の実行結果を読み出し，**論理レジスタ・ファイル**へ移動する．論理レジスタ・ファイルにはリタイアして確定した，正しい値のみが書き込まれる．

2.2 バックエンド方式の動作

フェッチの説明は省き，続くリネームの動作から説明する．

2.2.1 リ ネ ー ム

命令の書き込む論理レジスタ毎に異なる物理レジスタを割り当てる．さらにレジスタ・マッピング・テーブル (以下 **RMT**) に対して同時にいくつかの読み書きを行う．

- 割り当てた物理レジスタを RMT に書き込む
- 同じ論理レジスタに割り当てられていた古い物理レジスタを読み出す
- 命令のソース・レジスタに割り当てられている物理レジスタを読み出す

同時にそれぞれの命令が**アクティブ・リスト**にエントリをアロケートし，

- その命令が書き込む論理レジスタ
- その論理レジスタに前に割り当てられていた物理レジスタ (RMT から読みだしたもの)

を記録しておく．命令はアクティブ・リストのインデックスをタグとして取得する．

2.2.2 ディスパッチ〜発行

命令を命令キューに書き込み，オペランドがレディになったら読みだして演算器に送る．

2.2.3 レジスタ読み出し

物理レジスタ・ファイルを読み出す．

2.2.4 実行〜書き込み

フロントエンド方式と異なり，書き込みの対象は物理レジスタ・ファイルだけである．アクティブ・リストに実行終了のフラグのみ書き込む．

2.2.5 リ タ イ ア

バックエンド方式において命令をプログラム・オーダに保持しているハードウェアはアクティブ・リストである．従ってアクティブ・リストがどの命令がリタイアしてよいかを判定する．リタイア時の操作は物理レジスタの解放 (及びストアのコミット) のみである．

3. 要 素 技 術

本章では高い面積効率の達成に貢献する個別の技術について説明する．本章で説明する技術は以下の通りである．

- リネームド・トレース・キャッシュ
- マトリックス・スケジューラ
- 非レイテンシ指向レジスタ・キャッシュ
- ツインテール・アーキテクチャ

本稿では技術の詳細な仕組みについては解説しない．その貢献と概要を簡単に述べるに留める．しかしそれぞれの技術がどのようなプロセッサの構成方式を想定しているかは，技術の組み合わせを考える上で重要である．この点については強調する．

3.1 リネームド・トレース・キャッシュ

3.1.1 貢　　献

リネームド・トレース・キャッシュ(以下，RTC) は RMT の規模削減に貢献する技術である．RTC は RMT の規模を削減する一方で，命令キャッシュを複雑な RTC に置き換えることを必要とする．しかし，

- RTC が必要とするポート数は RMT よりずっと少ない (1-read/1-write).
- RMT より RTC の方がコアの外側に配置できる．

などの理由により，メリットの方が大きい．

3.1.2 概　　要

スーパスカラ・プロセッサでは一般的に，ループを回って同じ PC の命令を何度もフェッチする場合，新たにフェッチする度にリネームを行わなければいけない．

RTC を用いると，同じ PC の命令に対しては毎回リネームを行う必要がなくなる．従ってリネームを行うのはループの最初のイタレーションのみでよく，リネームのスループットを落としても全体のパフォーマンスに与える影響はほとんどなくなる．そのため RMT の規模を削減することができる．

毎回リネームを行う必要がなくなるのは，コンシューマとプロデューサ間の距離 (命令数) の情報を RTC に保存するからである．コンシューマはこの情報を用いてプロデューサを参照する．この情報はプログラムの実行パスが等しければ不変である．

直接物理レジスタ番号を用いて参照する場合，割り当てられる物理レジスタが毎回違うため，リネームし直さなくてはいけない．不変の情報を用いることで，その必要はなくなる．

3.1.3 プロセッサの構成方式

RTC はバックエンドでレジスタ・ファイルを読み出す構成方式を想定している．しかし RTC の導入はプロセッサの各部に影響を及ぼす．

RTC では，コンシューマはプロデューサとの間の距離を用いて，それを参照する．そのため物理レジスタ・ファイルをリング・バッファで構成し，命令はプログラム・オーダに物理レジスタをアロケートするようにする．コンシューマは自分のインデックスに距離を足すことで，プロデューサのインデックスを得て，オペランドを読み出す．

物理レジスタは一般的にプログラム・オーダに解放されるものではないので，プログラム・オーダにアロケートするだけではリング・バッファを構成できない．これに対する一つの解は，論理レジスタ・ファイルを配置し，リタイア時に物理レジスタ・ファイルから論理レジスタ・ファイルに値を移動することである．これにより物理レジスタ・ファイルのエントリはプログラム・オーダに解放でき，リング・バッファを構成できる．

3.2 マトリックス・スケジューラ

3.2.1 貢　　献

ウェイク・アップ・ロジックは通常パイプライン化できないため，発行幅を増やしていくとそこがクリティカル・パスになってしまう．マトリックス・スケジューラはウェイク・アップ・ロジックからタグの比較を行う CAM 構造を取り除く．これによりウェイク・アップ・ロジックの回路面積が削減され，クリティカルでなくなる．

3.2.2 概　　要

ウェイク・アップは命令ウィンドウに格納された命令の中から，プロデューサに依存する命令を探し出す操作である．通常これはプロデューサの実行結果のタグと，全ての命令のソース・オペランドのタグを比較することによって実現される．

マトリックス・スケジューラはコンシューマを探し出す操作を，シンプルな依存行列の読み出しによって実現する．依存行列はその列がプロデューサの，行がコンシューマの，それぞれ格納されたエントリに対応する．行と列の交差点のビットを立てることで依存関係を表現する．

依存行列を用いれば，プロデューサは自分の列を読み出すだけで，コンシューマの場所が分かるようになる．これによりタグの比較は必要なくなり，ウェイク・アップ・ロジックの回路面積が削減される．

3.2.3 プロセッサの構成方式

マトリックス・スケジューラはフロントエンド方式とバックエンド方式のどちらにも適用可能だが，構成方式によっては導入コストを低減できる．

依存行列の作成のために，コンシューマはプロデューサの格納されている命令ウィンドウのインデックスを知る必要がある．これはリネーム時に，コンシューマがプロデューサの格納されている物理レジスタ番号を調べているのと，全く同じ操作を要求する．すなわち RMT に相当するハードウェアを追加する必要がある．

ここでもし，物理レジスタ番号 (フロントエンド方式の場合，リオーダ・バッファのインデックス) と命令ウィンドウのインデックスが一致するなら，依存関係の検出を別々に行う必要はない．

バックエンド方式において，物理レジスタ・ファイルと命令キューのインデックスを一致させることは困難である．だがフロントエンド方式の場合，リオーダ・バッファとリザベーション・ステーションのサイズ，及びインデックスを一致させることができる．そうした場合，マトリックス・スケジューラを導入するコストは安くできる．

3.3 非レイテンシ指向レジスタ・キャッシュ

3.3.1 貢　　献

非レイテンシ指向レジスタ・キャッシュ(以下 NORCS) はレジスタ・ファイルの面積削減，さらにバイパス・ネットワークの簡単化に貢献する技術である．

レジスタ・キャッシュを用いると通常以下のようなメリットが得られる．

(1)　レジスタ・ファイルのアクセス・レイテンシの短縮

(2)　バイパス・ネットワークの簡単化
(3)　レジスタ・ファイルのポート数の削減

従来のレジスタ・キャッシュ・システムは主に 1 を目的としていた．アクセス・レイテンシを短縮することによりパイプライン長が短縮され，IPC を改善する効果がある．だが実際にはキャッシュ・ミスによるペナルティの方が大きく，IPC は逆に低下してしまう．実用的な価値は期待できなかった．

NORCS はレジスタ・キャッシュを用いつつも，1 のメリットを捨てることで，キャッシュ・ミスによるペナルティをほぼなくした．これにより 3 及び 2 のメリットを安全に得ることができる．

3.3.2 概　　要

従来のレジスタ・キャッシュ・システムは，レジスタ・キャッシュのヒットを仮定したパイプライン構成をとることで，アクセス・レイテンシを短縮していた．しかしこれはミス時にパイプラインを乱し，性能低下を招く．アクセス・レイテンシの短縮が性能に影響するのは分岐予測ミス時だが，レジスタ・キャッシュ・ミスの方が頻度がずっと高いため，トータルの性能は低下してしまう．

NORCS はレジスタ・キャッシュのミスを仮定したパイプライン構成をとる．すなわちレジスタ・キャッシュは初めからヒットしないものとして，後続の命令を発行する．アクセス・レイテンシは短縮されないが，ミスによってパイプラインが乱れることもない．従って NORCS の導入は性能にほとんど影響を与えない．

レジスタ・キャッシュはレジスタ・ファイルへのアクセスのフィルタとして働くため，レジスタ・ファイルのポート数は大きく削減できる．さらにバイパスの必要な期間を短縮してバイパス・ネットワークの簡単化を達成できる．NORCS は性能に悪影響を与えずにこの恩恵に預かることができる．

3.3.3 プロセッサの構成

NORCS はバックエンド方式の物理レジスタ・ファイルに適用することを想定しているが，参照の局所性を持つ RAM ならば，ポート数の削減の効果は得られる．バイパス・ネットワークの削減の効果については，バックエンドでレジスタ・ファイルを読み出す場合以外は該当しない．

3.4 ツインテール・アーキテクチャ

3.4.1 貢　　献

ツインテール・アーキテクチャは発行幅を増やさずに，命令の実行のスループットを向上させることを可能とする技術である．

3.4.2 概　　要

スーパスカラ・プロセッサの命令ウィンドウの回路面積は，その発行幅の 3 乗に比例する．演算器そのものは小さいので多数並べることができるが，発行幅を増やすことは難しい．命令の実行のスループットは発行幅によって制限されてしまう．

ツインテール・アーキテクチャはフロントエンド方式を想定している．フロントエンド方式ではフロントエンドでレジスタ読み出しを行うため，リザベーション・ステーションにディスパッチする前にソース・オペランドが得られる．命令の中にはレジスタ読み出しの時点でオペランドが揃い，実行可能になるものがある．

ツインテール・アーキテクチャではリザベーション・ステーションと並列に，演算器を行列状に配置する．この演算器にはレジスタ読み出しを終えた命令を，リザベーション・ステーションのようなバッファを介さず直接投入する．これによりバッファのポート数が実行のスループットを制限することがない．従って発行幅に制限されずに，命令の実行のスループットを向上させられる．

ツインテール・アーキテクチャでは行列状に配置した演算器からなる実行系をイン・オーダ・テール，命令ウィンドウを介する通常の実行系をアウト・オブ・オーダ・テールと呼んでいる．

3.4.3 プロセッサの構成方式

ツインテール・アーキテクチャはフロントエンド方式を想定している．フロントエンド方式のように，命令ウィンドウの外でオペランドを入手する手段がなければ，この技術を適用することはできない．

4. 提案する構成

本章では提案するプロセッサの構成について説明する．
提案するプロセッサの構成はバックエンド方式をベースとしている．
提案する構成とバックエンド方式との違いは以下の通りである．

- 物理レジスタ・ファイルと命令キューをリング・バッファにより構成
- 論理レジスタ・ファイルの追加
- アクティブ・リストの削除
- フューチャ・ファイルの追加
- イン・オーダ・テールの追加

- 命令キャッシュを RTC に置き換え
- 物理レジスタ・ファイル，論理レジスタ・ファイル，フューチャ・ファイルへの NORCS の適用

以下では，3 章で挙げた技術がいかに提案する構成に組み込まれたかについて説明する．

4.1 RTC

RTC の導入により，RMT のスループットを減らし，その回路面積を大きく削減できる．

RTC を導入するために，物理レジスタ・ファイルをリングバッファにより構成し，論理レジスタ・ファイルの追加を行った．

3.1.3 節で述べた通り，これらの構成は RTC が要求するものである．

4.2 マトリックス・スケジューラ

マトリックス・スケジューラの導入により，最もクリティカルなウェイク・アップ・ロジックの回路面積を減らし，それがクロックを制限することがなくなる．

RTC には，コンシューマとプロデューサ間の距離 (命令数) の情報が保存されている．マトリックス・スケジューラを用いることで，この情報を自然な形でウェイク・アップに活かすことができる．

マトリックス・スケジューラの導入のため，命令キューを物理レジスタ・ファイルと同じくリング・バッファで構成し，命令をプログラム・オーダに格納する．これにより，コンシューマはプロデューサの格納されているインデックスを物理レジスタ・ファイルと同様にして知ることができる．コンシューマはプロデューサの格納されているインデックスのビットを立てることで，依存行列が作成できる．

通常マトリックス・スケジューラの導入には，RMT に相当するハードウェアの追加が必要である．しかし命令キューに命令をプログラム・オーダに格納することにより，物理レジスタ・ファイルと同じ距離の情報で依存行列の作成が可能となった．従って RMT 相当のハードウェアの追加は必要ない．

さらに，物理レジスタ・ファイルと命令キューのサイズを等しくし，アロケート，解放のタイミングを一致させる．これにより両者に命令が格納されるインデックスを一致させることができる．

このメリットは命令キューと物理レジスタ・ファイルが同時に読み出せることである．バックエンド方式では発行の後でレジスタ読み出しを行っていたが，提案する構成の場合は並列に行える．これは発行レイテンシ，ひいてはパイプライン長の短縮につながり，性能向上が期待できる．

4.3 NORCS

NORCS の導入により，物理レジスタ・ファイルやフューチャ・ファイルなど，大きなハードウェアの回路面積を削減できる．

NORCS は以下のハードウェアに対して適用する．

- 物理レジスタ・ファイル
- フューチャ・ファイル

それぞれ元々NORCS が想定している，バックエンド方式における物理レジスタ・ファイルとは異なる．しかしいずれも NORCS の適用により，高い面積効率を達成できる．以下ではそれぞれに適用した場合の効果を説明する．

4.3.1 物理レジスタ・ファイル

提案する構成における物理レジスタ・ファイルは，リング・バッファにより構成され，リタイアした命令の値を保持しない．これらの特徴はレジスタ値の参照の局所性を失わせるものではないので，ほぼバックエンド方式における物理レジスタ・ファイルと同じように，ポート数の削減の効果が得られる．

またバイパス・ネットワークの簡単化の効果についても，同様に得られる．

4.3.2 フューチャ・ファイル

フューチャ・ファイルは論理レジスタ番号で参照する．命令の実行終了時に結果を書き込むが，タグ (物理レジスタ番号) が一致しなければ書き込まない．また，新たに論理レジスタに書き込む命令がリネームされると，その論理レジスタ番号のエントリは invalid とする．

エントリが valid になるのは命令の実行終了後である．それまでに読み出しを済ませてしまった後続の命令は値が得られない．従ってプロデューサに近い命令ほど，値を得られない確率が高い．コンシューマとプロデューサの距離は近いことが多いので，値を得られないコンシューマが多く存在する．従って，ある瞬間に valid であるエントリの数は，論理レジスタの数よりも少ない．

invalid な値を優先的にリプレースすることで，少ない valid な値をキャッシュ上に集中させることができる．従ってフューチャ・ファイルに対してはキャッシュは有効に機能すると考えている．

フューチャ・ファイルはバイパス・ネットワークを持たないので，簡単化の効果は得られない．

4.4 ツインテール・アーキテクチャ

ツインテール・アーキテクチャを適用することで，命令ウィンドウの規模を削減すること

Vol.2009-ARC-184 No.27
2009/8/5

ができる．

フロントエンドにフューチャ・ファイルを配置したことで，ディスパッチ前にオペランドを得ることができる．従って提案する構成ではツインテール・アーキテクチャを用いることができる．

5. ま と め

本論文は面積効率の高いスーパスカラ・プロセッサの構成方式を提案するものである．

ウェイ数の大きなスーパスカラ・プロセッサの面積効率を高める技術は，すでに多く提案されている．これらを組み合わせることで高い面積効率を達成できる．

しかしこれらの技術はそれぞれ異なるスーパスカラ・プロセッサの構成方式を対象にしているため，単純に組み合わせることはできない．

本論文はこれらを組み合わせる特殊な構成のスーパスカラ・プロセッサを提案した．

本論文で提案したスーパスカラ・プロセッサに関して，現在 FPGA 上に実装を進めている．年度内の完成を目指している．

参 考 文 献

1) 堀尾一生，平井遥, 五島正裕，坂井修一: ツインテール・アーキテクチャ, 先進的計算基盤システムシンポジウム *SACSIS2007*, pp.303-311, May, 2007
2) 五島正裕: *Out-of-order ILP* プロセッサにおける命令スケジューリングの高速化の研究, 京都大学 *(*博士論文*)*, March, 2004
3) M. Goshima, K. Nishino, Y. Nakashima, S. Mori, T. Kitamura, and S. Tomita: *A High-Speed Instruction Scheduling Scheme for Superscalar Processors*, *MICRO-34*, pp. 225-236 (2001)
4) 服部 直也, 高田 正法, 岡部 淳, 入江 英嗣, 坂井 修一, 田中 英彦: 発行時間差に基づいた命令ステアリング方式, 情報処理学会論文誌 コンピューティングシステム *(ACS-7)*, pp.80-93, Oct 2004.
5) 一林 宏憲, 塩谷 亮太, 入江 英嗣, 五島 正裕, 坂井 修一: 逆 *Dualflow* アーキテクチャ, 先進的計算基盤システムシンポジウム *SACSIS2008*, pp.245-254
6) 塩谷 亮太, 入江 英嗣, 五島 正裕, 坂井 修一: 回路面積指向レジスタ・キャッシュ, 先進的計算基盤システムシンポジウム *SACSIS2008*, pp.229-236
7) Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Doug Burger, Stephen W. Keckler, Charles R. Moore: *Exploiting ILP, TLP and DLP with the Polymorphous TRIPS Architecture ISCA 2003* pp.422-433