修士論文

回路面積指向レジスタ・キャッシュ

Area-Oriented Register Cache

指導教員　五島正裕 准教授

東京大学大学院 情報理工学系研究科
電子情報学専攻

塩谷 亮太

概要

レジスタ・ファイルは，最近のスーパスカラ・プロセッサの構成要素のうち，もっとも高コストなものの1つとなっている．本稿では，このレジスタ・ファイルの回路面積の縮小を目的とした，回路面積指向レジスタ・キャッシュを提案する．既存のレジスタ・キャッシュは，通常のキャッシュと同様，アクセス・レイテンシの短縮とそれによるパイプライン段数の縮小を第一の目的としている．それに対し，回路面積指向レジスタ・キャッシュの第一の目的は，回路面積の縮小であって，アクセス・レイテンシの短縮を図らない．回路面積指向レジスタ・キャッシュにおけるレジスタ・キャッシュは，メイン・レジスタ・ファイルへのアクセスを減らすフィルタとして働く．メイン・レジスタ・ファイルへは，レジスタ・キャッシュにミスした命令のみがアクセスを行うため，そのポート数を非常に少なくすることができる．提案手法では，性能をほとんど落とすことなく，メイン・レジスタ・ファイルのポート数を4まで減らすことができ，その回路規模をおよそ1/9程度にすることが可能である．

目 次

図 目 次

表 目 次

第1章　はじめに

レジスタ・ファイルは，最近のスーパスカラ・プロセッサの構成要素のうち，もっとも高コストなものの 1 つとなっている．

Out-of-order スーパスカラ・プロセッサでは，レジスタ・ファイルの容量は，命令ウィンドウ・サイズの 1.5〜2 倍程度必要である．

また，SMT (Simultaneous Multi-Threading) などのマルチスレッディングを行うプロセッサでは，同時に実行されるスレッドのコンテキストを保持するため，スレッド数に応じた容量が必要となる．

これらの理由のため，レジスタ・ファイルは巨大化する傾向にある．たとえば，2-way SMT をサポートする Intel Pentium 4 Processor の整数レジスタ・ファイルは，128 エントリにもなる．RAM が消費する電力は面積に比例するため [9]，電力消費も多くなる．

レジスタ・ファイル は，多ポートの RAM で構成される．通常，1 命令あたり，2 つのリード・ポートと 1 つのライト・ポートが必要であるから，4 つの整数系命令を同時に実行するスーパスカラ・プロセッサの整数レジスタ・ファイルのポート数は，合計 12 にもなる．RAM の回路面積は，ポート数の 2 乗に比例するため，レジスタ・ファイルは，その容量の割に非常に大きなものとなる．[4, 8].

1.1　配線遅延の影響

レジスタ・ファイルのレイテンシは配線遅延に支配されるため，LSI が微細化してもほとんど短縮されない．そのため近年では，レジスタ・ファイルを 1 サイクル程度でアクセスすることはもはや不可能であり，通常 2〜3 程度のパイプライン・ステージが充てられている．

しかしレジスタ・ファイル・アクセスのパイプライン化は，新たにいくつかの問題を引き起こす．まず，パイプラインが深化することの一般的な悪影響として，IPC の低下が挙げられる．パイプラインが深くなればその分だけ，分岐予測をはじめとする各種予測ミス・ペナルティが増加する．また，命令ウィンドウ・エントリや物理レジスタ自体の解放が遅れるため，資源不足によってフロントエンドがストールする確率も増える [11]．さらに，レジスタ・ファイル・アクセスのパイプライン化に固有の問題として，バイパス・ネットワークの複

雑化がある [15].

1.2 回路面積指向レジスタ・キャッシュ

このような問題に対して，本稿では**回路面積指向レジスタ・キャッシュ（AORC：Area-Oriented Register Cache）**を提案する．AORC は，その名の通り，回路面積の縮小を第一の目的とするレジスタ・キャッシュ（RC：Register Cache）である．メイン・レジスタ・ファイル（MRF：Main Register File）に，小容量ゆえに高速な RC を追加するという点では，通常の RC と同様であるが，その目的が全く異なる．既存の RC は，通常のキャッシュと同様，アクセス・レイテンシの短縮とそれによるパイプライン段数の縮小を第一の目的としている．それに対して，AORC の第一の目的は，回路面積の縮小であって，アクセス・レイテンシの短縮を図らない．

既存の RC は，いわば,「キャッシュ・ヒットを仮定した命令パイプライン」となっている．ヒットした場合には実効的なレイテンシが短縮されるが，ミス時にはペナルティが発生する．多くの研究 [10, 2, 13, 12] にも関わらず，ミス率は最大 13%にもなる．そのため，6 章で詳しく述べるが，レイテンシの短縮による正の効果より，ミス・ペナルティによる負の効果のほうがむしろ大きいのが現状である．

一方 AORC では，いわば,「キャッシュ・ミスを仮定した命令パイプライン」となっている．RC へのアクセス・ステージの後には，MRF へのアクセス・ステージが設けられており，すべての命令は，RC のヒット・ミスに関わらず，必ず MRF のアクセス・ステージを通過する．そのため，レイテンシは短縮されない代わりに，ミスも発生しない．

AORC における RC は，レイテンシを短縮するのではなく，MRF へのアクセスを減らすフィルタとして働く．MRF へは，RC にミスした命令のみがアクセスを行うため，そのポート数を非常に少なくすることができる．6 章の評価では，ポート数は 4 程度で十分であることが分かる．前述したように，RAM の回路面積はポート数の 2 乗に比例するため，MRF の回路規模は $(4/12)^2 = 1/9$ 程度になる．

小型化は結局，MRF 自体のレイテンシの短縮につながる．後述するように，RC と MRF の回路規模はほぼ同程度であり，MRF は 1 サイクルでアクセス可能である．

1.3 構成

AORC は，物理的構成の点では既存の RC などと変わらないため，それらとの比較が欠かせない．そのため，本稿は，以下のような構成とした．まず，2

章で，レジスタ・ファイルを構成する RAM の特性について述べ，3 章で，レジスタ・ファイルのパイプライン化について述べる．続く 4 章で，AORC について簡単に説明したうえで，RC などとの違いを明確にする．その後，5 章で AORC の詳細な説明を行う．6 章で必要なポート数などの評価を行う．

第2章　レジスタ・ファイルの構成と遅延

レジスタ・ファイルが大きな遅延を持つのは，配線遅延の相対的増大と，レジスタ・ファイルを構成する多ポート RAM の特性による所が大きい．

1. **配線遅延の相対的増大**　LSI の微細化は，配線遅延を相対的に増大させる．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ためである．配線遅延が増すにつれ，演算器などのゲート遅延が支配的なユニットに替わり，長い配線を持つレジスタ・ファイルやバイパスなどがクリティカルとなってきている．

2. **多ポート RAM の特性**　通常，レジスタ・ファイルは多ポートの RAM で構成される．この RAM の回路面積は，ポート数の 2 乗に比例して大きくなる．レジスタ・ファイルは，発行幅に応じた，多くのポートを持つ必要があるため，レジスタ・ファイルは非常に大きな回路となる．

本章では，上記のうち，多ポート RAM の特性について詳しく述べる．

2.1　RAM の構成

通常，エントリ数の大きな RAM は，**セル・アレイ**によって構成される [14]. **図 2.1** に，RAM セル・アレイのブロック図を示す

セル・アレイでは，アドレスをロウ・アドレスとカラム・アドレスに分け，この 2 次元アドレスでアレイ中のセルを指定する．アドレスが与えられると，ロウ・デコーダが，ロウ・アドレスをデコードし，ワードラインをアサートする．カラム・セレクタが，カラム・アドレスに基づいてビットラインを選択する．

2.2　SRAM の構成

レジスタ・ファイルは，RAM の中でも **SRAM** によって構成される．SRAM は，ロジックと同じトランジスタで構成されるため，非常に高速に動作する．

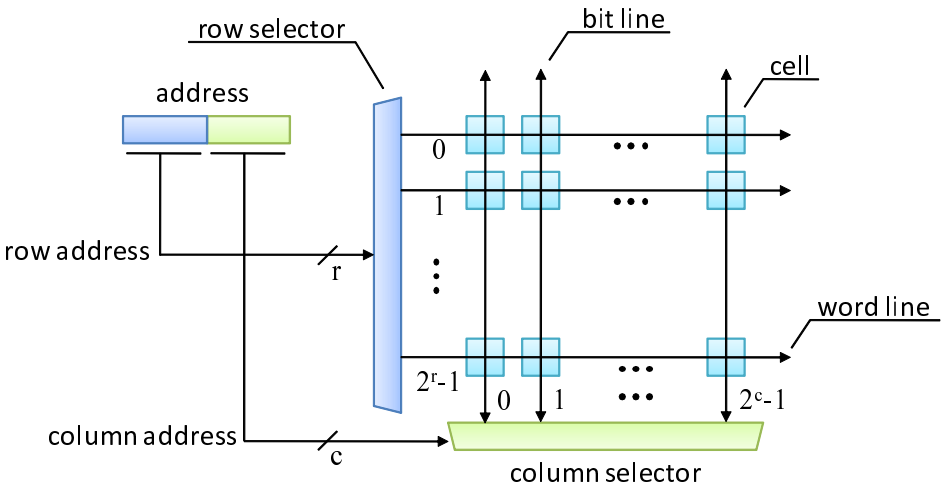

図 2.1: RAM セル・アレイ

図 2.2 に SRAM の回路図を示す．SRAM のセルは，フリップ・フロップから成る．各フリップ・フロップは，ワードラインによって制御されるアクセス・ゲートを介して相補的なビットラインに接続される．このようにして，SRAM のセルは 6 個のトランジスタによって構成される．

2.2.1　マルチ・ポート SRAM

SRAM をマルチ・ポート化した場合，その面積は，ほぼ配線の本数によって決まる．これは，ポート数の分だけ，ビットラインとワードラインが多重化されるためである．この多重化は，縦方向と横方向に対して同時に行われるため，面積はポート数の 2 乗に比例して大きくなる．

SRAM をマルチ・ポート化したもの（デュアル・ポート）を，**図 2.3** に示す．フリップ・フロップそのものについては，シングル・ポートの場合と同じである．しかし，ビットラインとワードラインは，ポートの数だけ多重化されており，アクセス・ゲートもその分だけ増えている．ポート数を 3，4，5 と，増やしていった場合も，これと同様であり，ビットラインやワードラインが多重化される．

マルチ・ポート RAM では，このようにして配線が縦方向と横方向に多重化されるため，配線の占める面積はポート数の 2 乗に比例して大きくなる．一方，アクセス・ゲートについては，ポート数に比例した増加となるため，ポート数の面積に与える影響は緩やかである．

2.2.2 回路レイアウト

ポート数によって面積が増加する様子は，実際の回路レイアウトにおいても容易に確認できる．**図 2.4** と**図 2.5** に，それぞれ 12 ポートと 2 ポートから成る，SRAM の回路レイアウトを示す．図中のレイアウトが示すものについては，以下の通りである．

- **12 ポート SRAM**　図 2.4 では，縦と横に規則正しく伸びている配線がビットラインとワードラインである[1]．また，中央付近にある回路がフリップ・フロップを構成するゲートであり，その周辺部にあるのがアクセス・ゲートとそれに接続されている配線である．12 ポート SRAM では，配線が隙間無く均等に配置されており，全体の面積が配線によって決定されていることがわかる．

- **2 ポート SRAM**　図 2.5 では，12 ポート SRAM と比較して配線の数が少なく，ちょうど 12 ポート SRAM の中央部付近を取り出したものとなっている．2 ポート SRAM では，その回路面積のほとんどはフリップ・フロップとアクセス・ゲートによって占められていることがわかる．

通常，レジスタ・ファイルには，1 つの命令あたり，2 つのリード・ポートと，1 つのライト・ポートが必要である[2]．このため，4 つの命令を同時に実行するスーパスカラ・プロセッサのレジスタ・ファイルのポート数は，合計 12 にもなり，その回路面積は図 2.4 のように非常に大きくなる．

面積の大きな回路では，当然その遅延も大きなものとなる．次章では，このレジスタ・ファイルの遅延を緩和する，パイプライン化について述べる．

[1] 同じ配線層には電源ラインも含まれるため，その数は若干ポート数よりも多い

[2] 1 つの命令が最大で，2 つのソース・オペランドと，1 つのディスティネーション・オペランドを持つことした場合．たとえば，Alpha 21464[5] は，このような構成をとる．

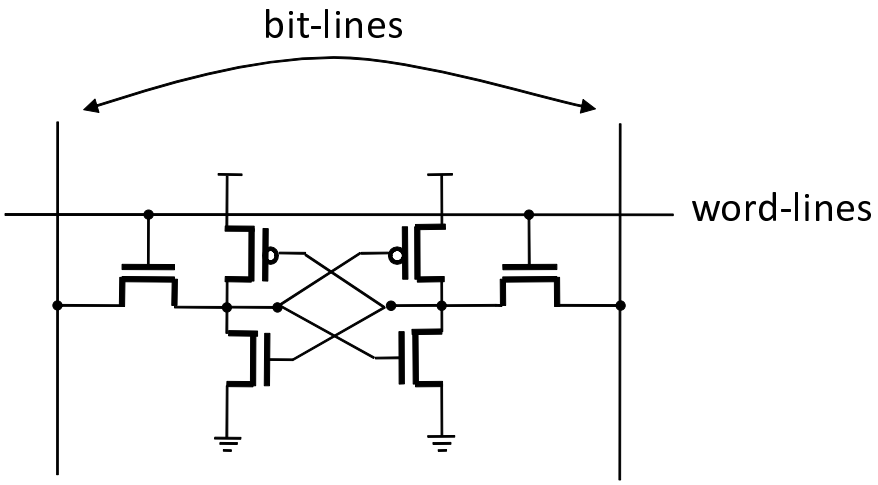

図 2.2: シングル・ポート SRAM

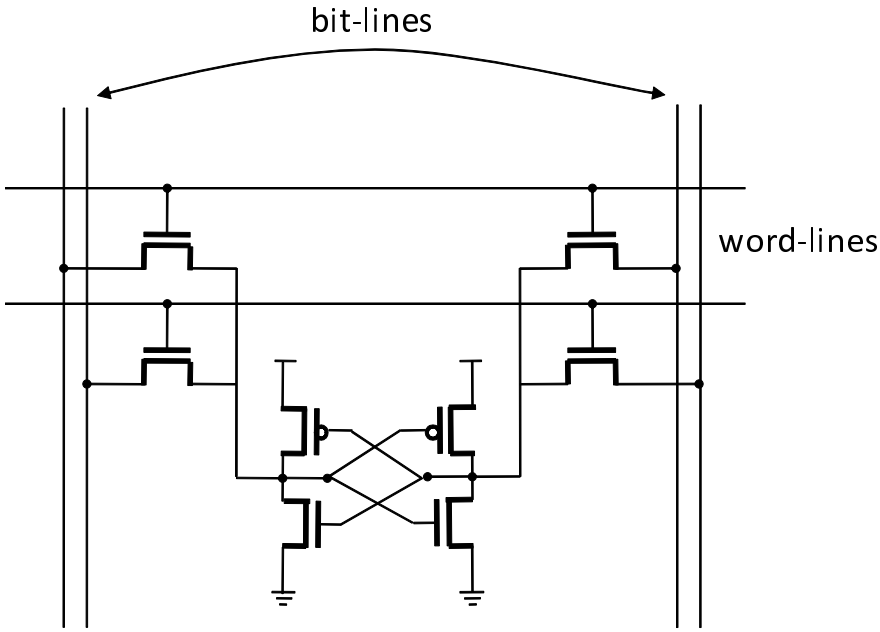

図 2.3: デュアル・ポート SRAM

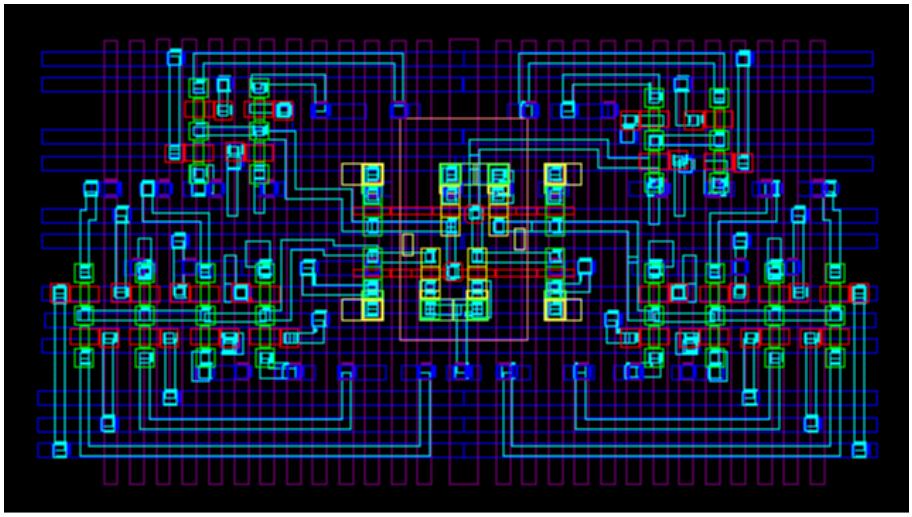

図 2.4: SRAM（12port：8read,4write）

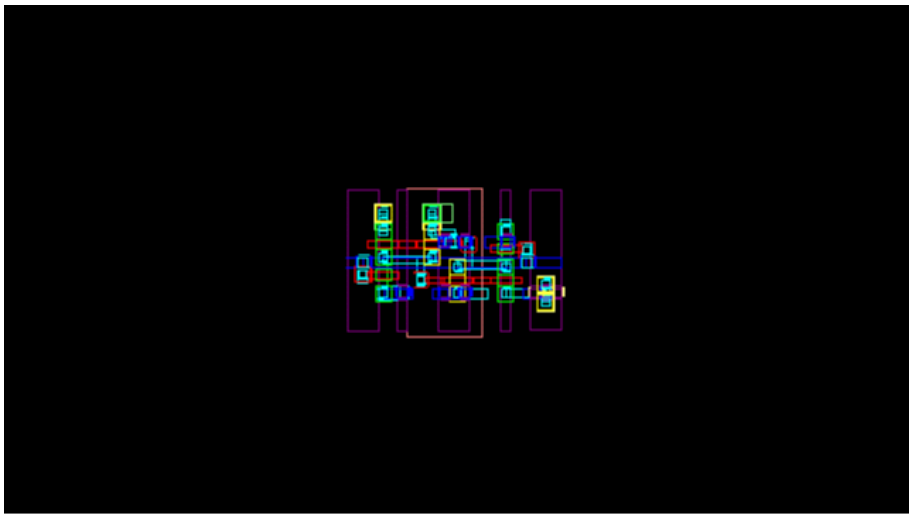

図 2.5: SRAM（2port：1read,1write）

第3章　レジスタ・ファイルのパイプライン化

遅延が大きなユニットがクリティカルとなるのを防ぐためには，**パイプライン化**を行うことが有効である．パイプライン化を行うことにより，1サイクルあたりの遅延時間を大幅に短くする事ができる．本節では，レジスタ・ファイルのパイプライン化と，それによって生じる問題について述べる．

3.1　パイプライン化時の動作

近年のプロセッサでは，レジスタ・ファイル・アクセスはパイプライン化されている．これらのプロセッサでは，レジスタ・ファイルを複数サイクルかけてアクセスする事により，1サイクルあたりの遅延を減らしている．たとえば，Alpha 21264では3サイクル[5]，Intel Pentium4では2サイクル[3]が，それぞれレジスタ・ファイル・アクセスに割り当てられている．

3.1.1　バイパス期間の延長

パイプライン化されたレジスタ・ファイルでは，バイパスを行う期間が長くなる．これは，レジスタ・ファイルを介して値の受け渡しを行う場合に，より長い時間がかかるためである．

図3.1に，レジスタ・ファイルの読み書きに2サイクルを割り当てた場合の，パイプライン・チャートを示す．ISとEXはそれぞれ発行と実行を，RRとRWはそれぞれレジスタの読み出しと書き込みを表す．

今，パイプライン上の命令I_pに対し，それに依存する，命令I_c群が依存しているものとする．I_pは，その実行によって得られた値を2サイクルかけてレジスタ・ファイルに書き込む．このため，レジスタ・ファイルから値が得られるようになるまでの4サイクル間，後続のI_cは，オペランド・バイパスによって値を得る．

3.2　パイプライン化によって生じる問題

上記のようにして，レジスタ・ファイル・アクセスのパイプライン化を行うことは，新たにいくつかの問題を引き起こす．このパイプライン化によって生じる問題は，以下のように，**パイプラインの深化よって生じる一般的な問題**と，**レジスタ・ファイルのパイプライン化に固有の問題**に分けて考えることができる．

1. パイプラインの深化による一般的な問題：
   (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・ファイルに固有の問題：
   バイパス・ネットワークの複雑化

前者に関しては，後の 6 章で示すように，その影響は比較的軽微である．しかし，バイパス・ネットワークの複雑化に関しては，バイパスそのものがクリティカルであるため，その複雑化は受け入れがたい．

3.2.1　バイパス・ネットワークの複雑化

前述したように，レジスタ・ファイルのパイプライン化は，値をオペランド・バイパスによって得なければならない期間を増加させる．これを行うために，通常のバイパス・ネットワークを単純に拡張したもののブロック図（1 ビット・スライス分）を**図 3.3** に示す．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 [15].

3.2.2　Ahead Pipelining による回路規模の縮小

バイパス・ネットワークは，**Ahead Pipelining** を行うことによってその回路規模を若干縮小することができる．Ahead Pipelining では，1 サイクル以上前に結果が得られたオペランドについては，あらかじめセレクトを済ましておき，パイプライン・ラッチに保持する．このため，1 サイクルあたりにセレクトを行うオペランド数を少なくする事ができる．

図 3.3 より，単一の演算器周辺のバイパス・ネットワークついて取り出したものを**図 3.4** に示す．また，同一箇所に対し，Ahead Pipelining を行ったものを**図 3.5** に示す[1]．前者が，単一の 17 入力 1 出力のセレクタによってバイパスを行うのに対し，後者は，4 から 6 入力かつ 1 出力のセレクタをパイプライン化して用いる事により，バイパスを行っている．

Ahead Pipelining を行った場合，パイプライン・ラッチの数が増加する．Ahead Pipelining の場合，依存元の命令の結果が得られ次第，セレクトを行う．このため，パイプライン・ラッチに含まれる値は，セレクト済みの値であり，複数のソース・オペランド間で共有する事ができない．完全バイパス・ネットワークでは，オペランドが必要となる直前にセレクトを行う．このため，以前のサイクルに得られた結果は，複数のソース・オペランド間で共有する事ができる．

セレクタの制御

セレクタの制御については，依存元の命令が実行される前に決定されている必要がある．これは，ウェイクアップ時に得られる情報を用ることにより，解決することができる．**図 3.2** に，この時の様子を示す．**W/S** は，ウェイクアップ/セレクトを意味する．W/S でセレクトされた命令は，どの命令によってウェイクアップされたかを知る事ができる．このため，I_p の実行前に，その結果をバイパスするよう，セレクタを制御する事ができる．

レジスタ・キャッシュ[2][1] や**バイパス・バッファ**[15] は，こうしたバイパスの複雑化を緩和するものである．次章では，これらの手法について，AORC と比較を行いながら，その違いについて述べる．

[1] この図は，バイパスを行う全てのステージについて Ahead Pipelining を行ったものである．実際には完全バイパス・ネットワークとの中間の構成をとるものも考えられる．

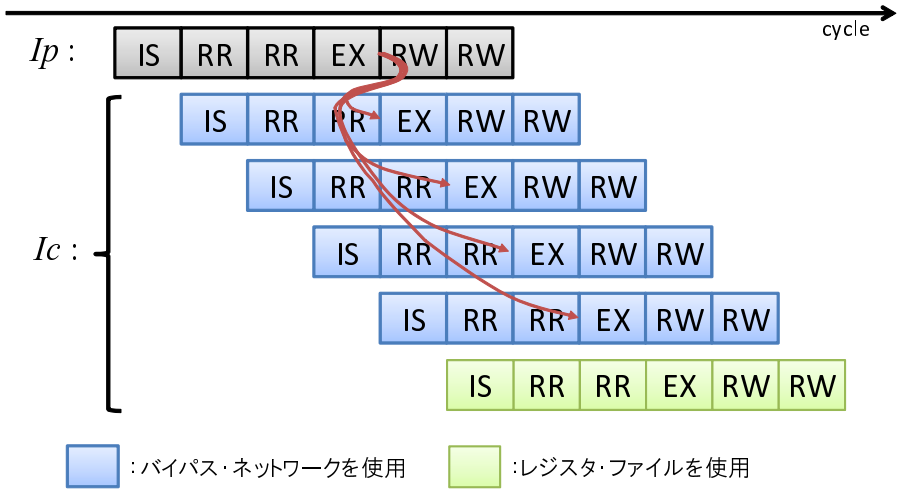

図 3.1: パイプライン化されたレジスタ・ファイルの動作

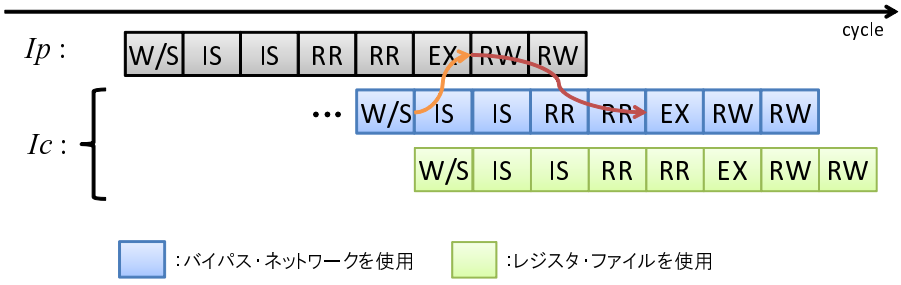

図 3.2: Ahead Pipelining を行った場合のパイプライン・チャート

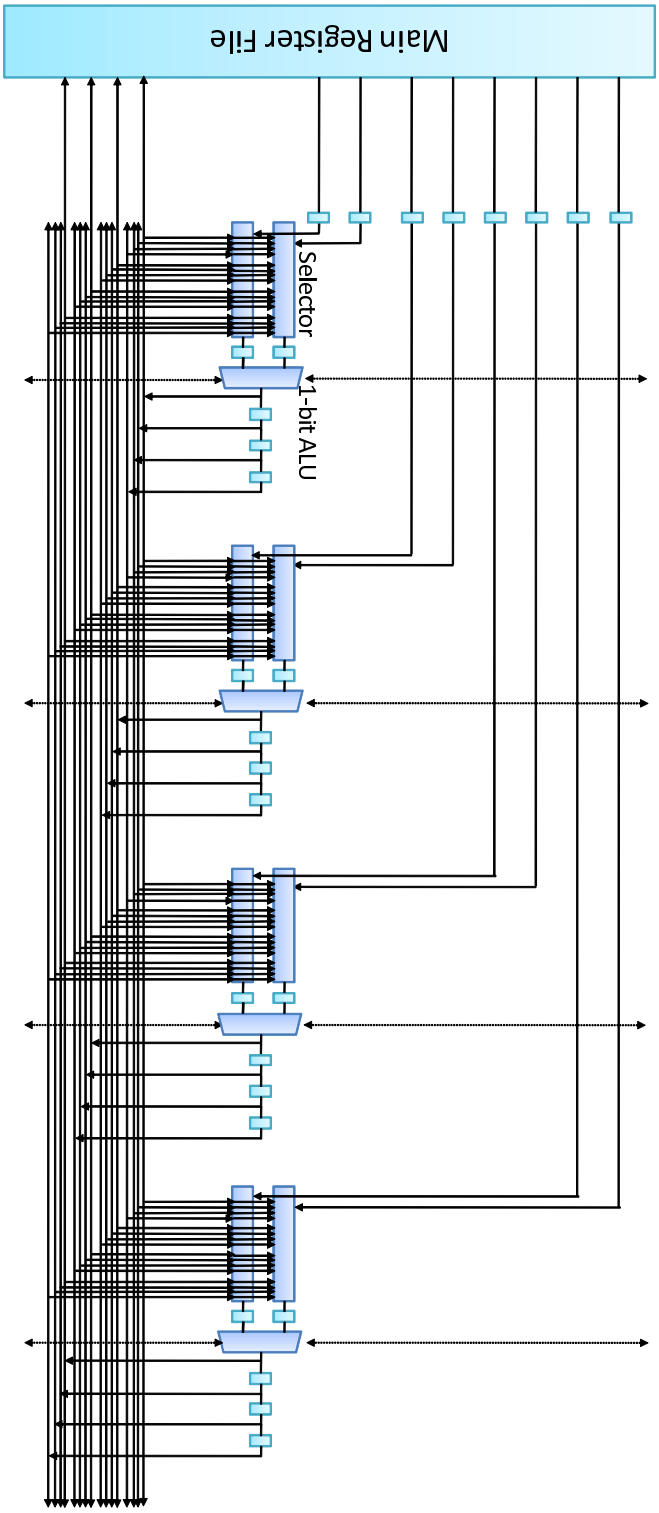

図 3.3: 完全バイパス・ネットワーク（1 ビット・スライス分）

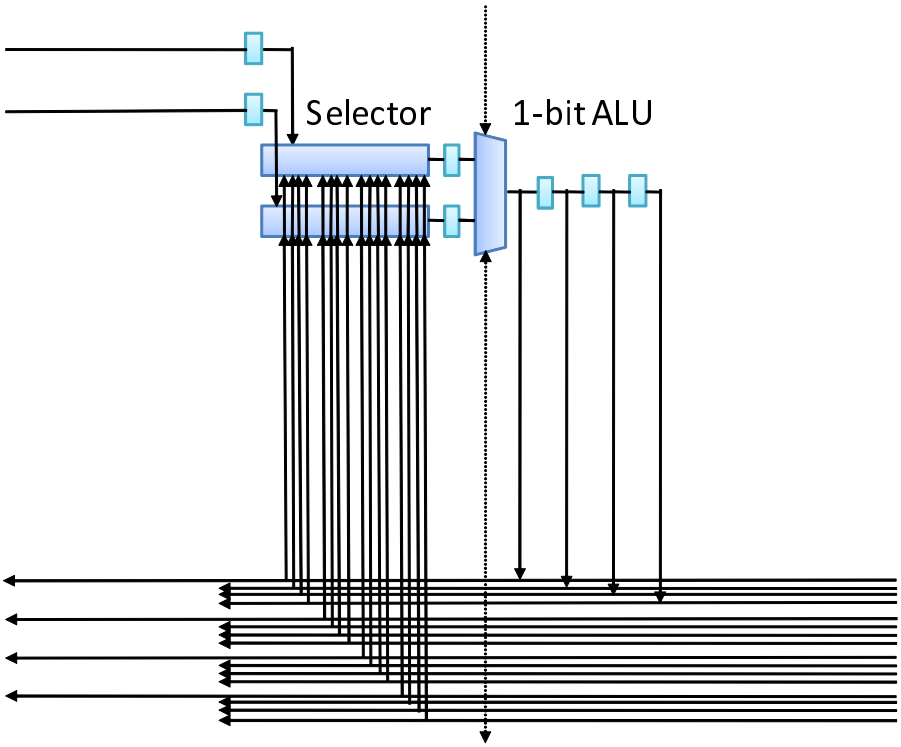

図 3.4: 完全バイパス・ネットワーク（演算器周辺）

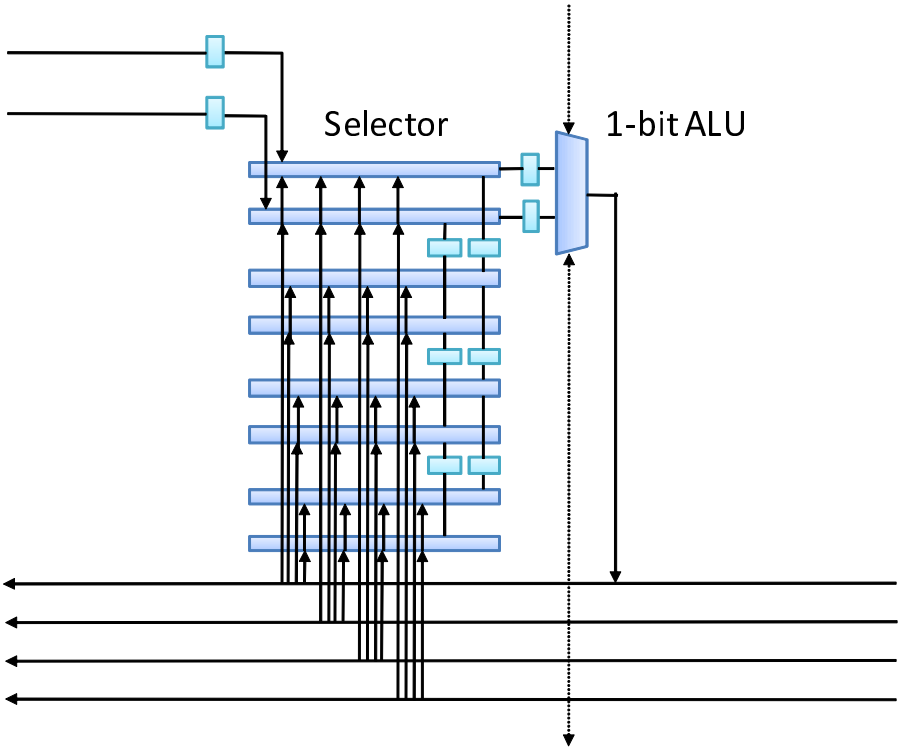

図 3.5: Ahead Pipelining を施した完全バイパス・ネットワーク

第4章　レジスタ・キャッシュなどとの違い

提案手法と類似の物理構成をもつ手法として，**レジスタ・キャッシュ（RC：Register Cache）**[10, 2, 13, 12] と**バイパス・バッファ**[15] がある．RC とバイパス・バッファは，共にレジスタ・ファイル・アクセスのパイプライン化によって生じる問題の緩和を目的としている．

RC とバイパス・バッファは，共に 1 サイクルでアクセス可能な小型のバッファである．**図 4.1** に，RC とバイパス・バッファのブロック図を示す．両者は，その周辺までを含めたデータ・パスの構成もほぼ同じである．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・ミス時などに，MRF の内容を RC にリフィルするためのものである．

提案する回路面積指向レジスタ・キャッシュ（AORC：Area-Oriented Register Cache）も，従来の構成と同様の **RC** と **MRF** を持つ．**図 4.2** に，AORC のブロック図を示す．

以下では，これらの手法について，比較を行いながら説明を行う．なお，提案する AORC との区別のため，従来の RC の構成を**レイテンシ指向レジスタ・キャッシュ（LORC：Latency-Oriented Register Cache）**と呼ぶことにする．また，RC と言った場合，MRF の一部を保持する小型のバッファそのものを示すものとする．

4.1　パイプライン構成

それぞれの手法の大きな違いは，そのパイプライン構成にある．LORC は**ヒットを仮定したパイプライン構成**をとり，バイパス・バッファと AORC は**ミスを仮定したパイプライン構成**をとる．以下では，まず，パイプライン構成を中心に概説し，その後 4.2 節で利害得失についてまとめる．

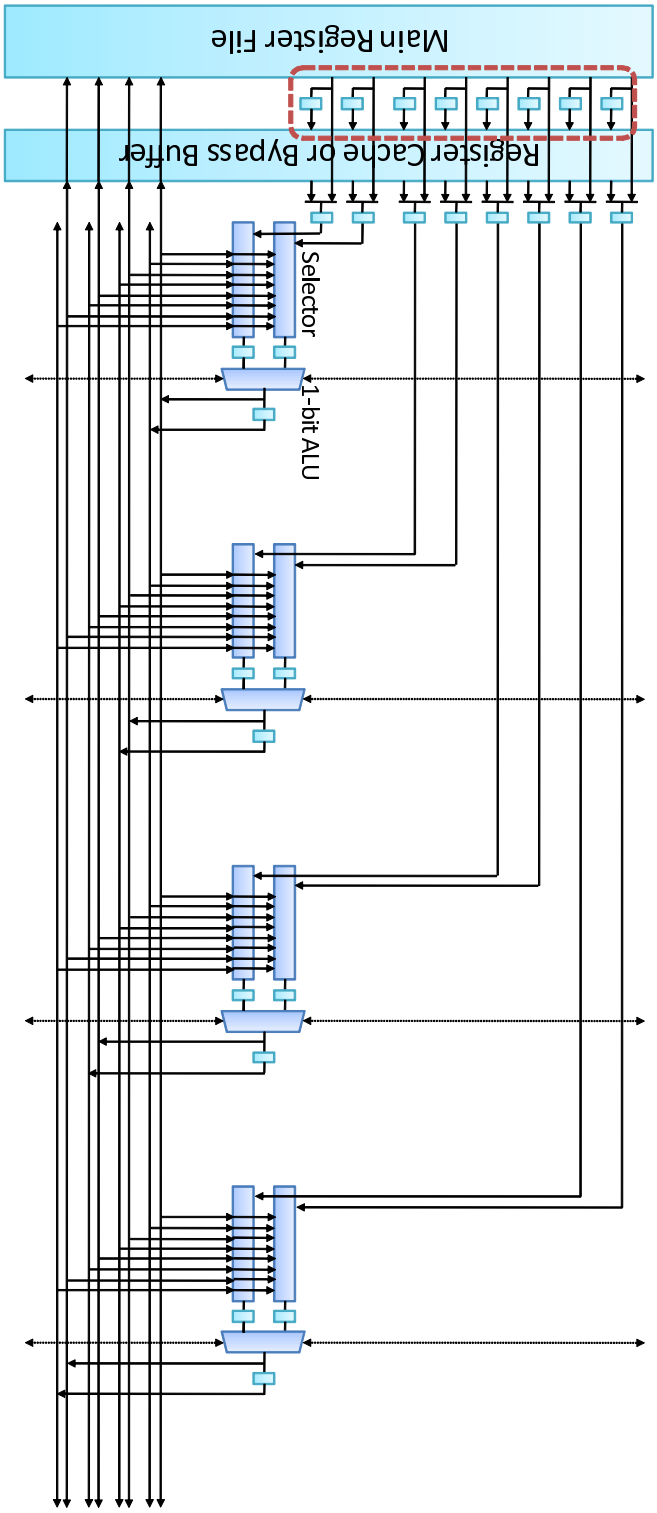

図 4.1: レジスタ・キャッシュとバイパス・バッファ

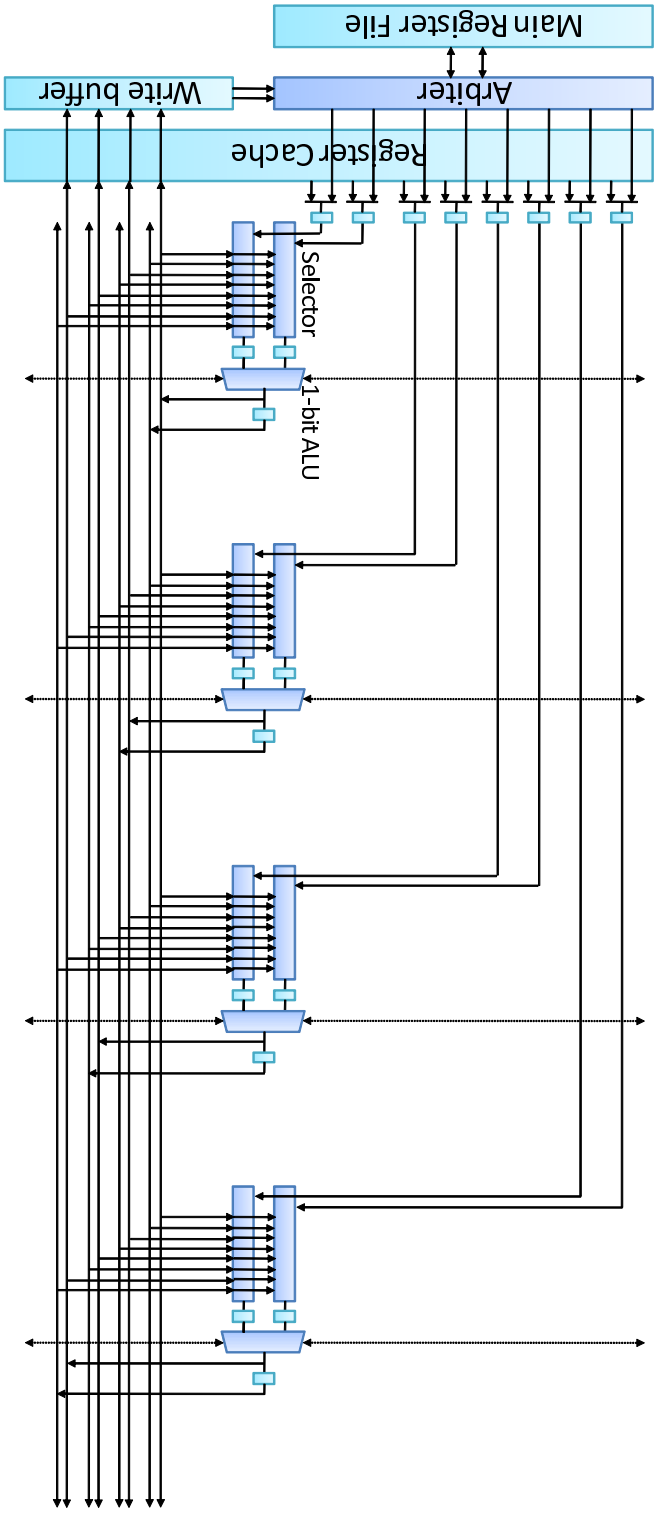

図 4.2: 回路面積指向レジスタ・キャッシュ

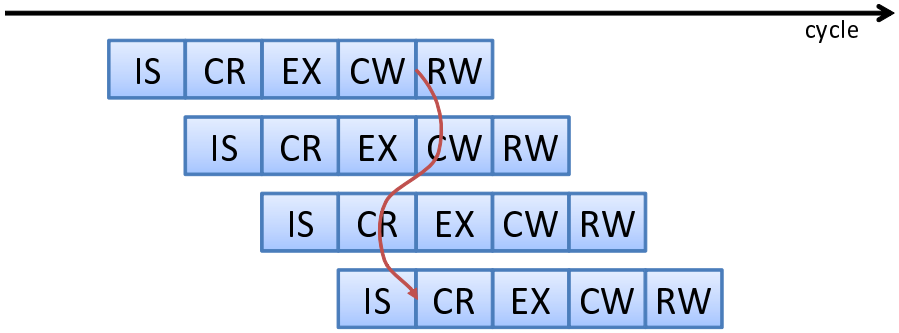

図 4.3: レイテンシ指向レジスタ・キャッシュのパイプライン

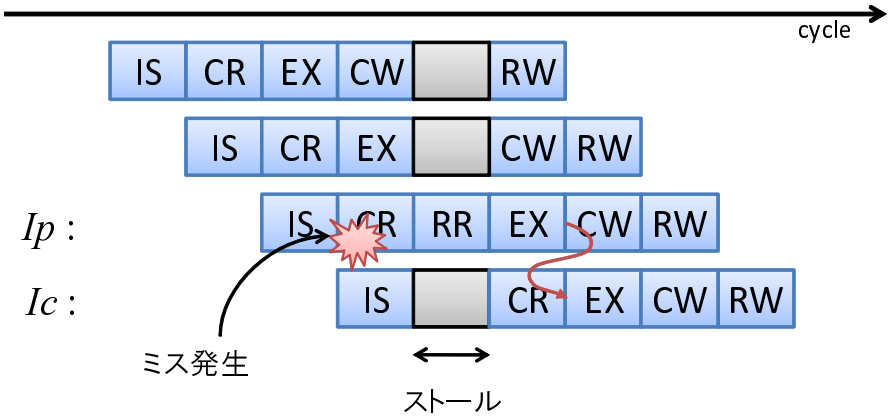

図 4.4: レイテンシ指向レジスタ・キャッシュのミス時の動作

4.1.1　レイテンシ指向レジスタ・キャッシュ

図 4.3 に，RC を持つプロセッサのパイプライン・チャートを示す．図中の **CR** と **CW** は，それぞれ，RC の読み出しと書き出しを意味する．

LORC は，一般的な L1 データ・キャッシュと同様に，**ヒットを仮定したパイプライン構成**をとる．すなわち，LORC では，全ての命令は，そのオペランドが RC にヒットするものとしてスケジューリングされる．

レジスタ・キャッシュ・ミスが発生した場合の考慮についても，通常のデータ・キャッシュの場合と同様である．**図 4.4** に，レジスタ・キャッシュ・ミスが発生した場合の動作を示す．図中の **RR** は，MRF からの読み込みを表す．オペランドが RC にミスした命令 I_p は，MRF からの値の読み込みを行う．その結果，I_p と，それに依存する I_c の実行は，ヒットした場合と比べて 2 サイクル遅くなってしまう．

4.1.2　バイパス・バッファ

バイパス・バッファ[15] は RC と同様の，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．バイパス・バッファは，完全バイパス・ネットワーク内のラッチが保持していた，バイパスされる値を，FIFO によって保持するものである．バイパス・バッファには，毎サイクル，演算結果の値が書き込まれると同時に，MRF から読み込めるようになった値は順次捨てられる．

図 4.5 に，バイパス・バッファのパイプライン・チャートを示す．図中の **BR** と **BW** は，それぞれ，バイパス・バッファの読み出しと書き込みを意味する．バイパス・バッファは LORC とは逆に，**ミスを仮定したパイプライン構成**をとる．このため，LORC とは異なり,バイパス・バッファの場合は必ず MRF にアクセスが行われる．したがって，レイテンシは短縮されないかわりに，ミスも発生しない．

4.1.3　回路面積指向レジスタ・キャッシュ

図 4.6 に，AORC の基本的なパイプライン・チャートを示す．この図は，MRF のアクセスに 2 ステージを割り当てたものである．

RS（Register Scheduling）は，RC のヒット・ミス判定と，後で述べる MRF アクセスのスケジューリングを意味する．また，**RR** は MRF の読み出しステージを意味する．**RR/CR** は，MRF 読み出しと並列して，さらに RC のデータ・アレイ読み出しを行うステージである．

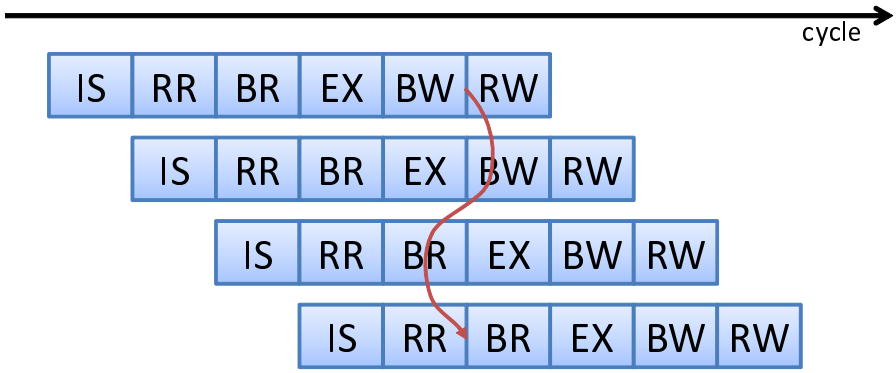

図 4.5: バイパス・バッファのパイプライン

図 4.6: 回路面積指向レジスタ・キャッシュのパイプライン

基本的な動作

発行されてきた命令は，まず RS で RC に対するオペランドのヒット・ミス判定を行う．従来の LORC と異なるのは，AORC が，バイパス・バッファと同様に**ミスを仮定したパイプライン構成**をとる点である．AORC では，命令はオペランドのヒット・ミスに関わらず，後続のレジスタ読み出しステージを通過する．このため，レジスタ・キャッシュ・ミスが起きただけでは，ペナルティは発生しない．

RS においてヒットと判定されたオペランドは，後続のステージにタグの一致情報だけを送る．RC のデータ・アレイへのアクセスは，実行ステージの直前にある RR/CR で行う．通常のセット・アソシアティブ構成をとるキャッシュでは，タグとデータ・アレイのアクセスは並列に行われるが，このようにして逐次で行うことにより，アレイ・アクセス後のセレクタを省略することができる．

また，ミスと判定されたオペランドは，後続の RR と RR/CR で MRF へのアクセスを行う．このため，MRF はアクセス数に見合った少数のポートだけを持つ．ミスしたオペランドが多数あり，ポートが足りない場合は，バックエンドのその他のユニットをストールさせ，MRF から値の取得を行う．

MRF の時分割アクセス

1 章でも触れたように，MRF の回路規模は 1/9 程度になるため，この MRF は 1 サイクルでアクセス可能である．このため，MRF からオペランドを時分割で取得することを行う．たとえば，図 4.6 に示すパイプライン構成の場合，命令は MRF に対するアクセスの機会を RR と RR/CR の 2 回持つ．

この，MRF へのアクセス・タイミングのスケジューリングは，RS ステージでヒット・ミス判定を行った後に行う．ヒット・ミス判定に必要なタグ・アクセスは，RC のアクセスよりも大幅に短い時間で行うことが可能なため，タグ・アクセスと MRF のアクセス・スケジューリングは 1 サイクルで実行可能であると考える．

MRF への書き込み

実行後に行う MRF への書き込みは，一旦小容量のライト・バッファに書き込んだ後，サイクル・スチールによって，MRF へ順次書き込みを行う．また，このライト・バッファへの書き込みは RC に対して，ライトスルーで行う．

4.2　各方式の得失

LORC とバイパス・バッファは，レジスタ・ファイルのパイプライン化によって生じる問題の緩和を目的としている．これに対し，AORC は，RC を MRF へのアクセスのフィルタとして用いることにより，MRF のポート数を削減することを目的としている．以下では，これらの得失について議論を行う．

4.2.1　RC とバイパス・バッファ

LORC は，それがヒットする限りにおいては，1 サイクルでアクセス可能なレジスタ・ファイルを持つプロセッサと機能的に等価となる．したがって，理想的には，前述したパイプライン化による問題の全てを緩和する効果がある．しかし，実際には，ミス・ペナルティのため，大幅な性能低下を起こすことがある [2].

一方，バイパス・バッファは，機能的には完全バイパス・ネットワークと等価であり，MRF から値が取得できない場合にのみバイパス・バッファにアクセスが行われる．このため，キャッシュ・ミスが発生することは無いが，パイプラインの短縮も行われない．したがって，バイパス・バッファは，上記の問題のうち，バイパスの複雑化のみを緩和するものである．

バイパス・バッファは，レジスタ・ファイルのパイプライン化によって生じる問題のうち，(1）一般的な問題の緩和を放棄することにより，RC のミスを無くしたものであると考えることができる．バイパス・バッファがこのようなアプローチをとるのは，後の 6 章で示すように，RC のミスによる性能低下が大きいことと，パイプラインの短縮による効果が小さく，性能低下に見合うものではないためである．

4.2.2　AORC

理想的に動作した場合，AORC は，パイプライン化されたレジスタ・ファイルと機能的に等価となる．AORC は，バイパス・バッファと同様に，レイテンシの短縮によるパイプラインの縮小を放棄しており，キャッシュ・ミスが起きただけでは，ペナルティは発生しない．AORC のペナルティは，レジスタ・アクセスのために用意された期間に，全てのオペランドを揃えられなかった場合に発生する．

RC のミス率は，最大で 13%程度 [13] と L1 データ・キャッシュなどと比べると著しく高い．RC をレイテンシ短縮のために用いようとした場合，ミスの発生はペナルティに直結するため，これほど高いミス率では受け入れがたい．

しかし，AORC において，アクセスのフィルタとして用いる場合には，このミス率でも十分有効に機能する．6 章で詳しく述べるが，AORC では，MRF の

ポートをミス率に応じた数だけ設ければ，ほぼペナルティは発生しなくなる．このため，たとえばRCのミス率が20%であるならば，MRFのポート数も，元の20%程度に減らすことができる．

第5章　回路面積指向レジスタ・キャッシュの詳細

本章では，提案する **AORC** について，その詳細や考慮すべき点を述べる．

5.1　回路規模の検討

4 章で述べたように，AORC のパイプラインでは，RC にミスしたオペランドのみが MRF に対してアクセスを行う．このため，MRF は，アクセス数に見合った少数のポートを持つだけでよい．たとえば，RC のヒット率を 80%とし，毎サイクル 8 個のオペランドを供給する必要があった場合，MRF へアクセスを行うオペランドは，サイクルあたり平均 1.6 個となる．

2 章で述べたように，SRAM の回路面積は，およそポート数の 2 乗に比例する．このため，ポート数の少ない MRF は非常に小さなものとなる．

たとえば，**表 5.1** に示す構成の場合，RC と MRF の面積は同程度の大きさとなる．したがって，RC が 1 サイクルでアクセス可能であるのならば，MRF も 1 サイクルでアクセス可能であると言える．

5.1.1　MRF の時分割アクセス

4.1.3 項でも触れたように，AORC では MRF へのアクセスを時分割で行う．値の取得を時分割で行うことにより，MRF のポート数不足を緩和することができる．

図 5.1 に，時分割アクセスを行った場合の動作を示す．MRF はポートを 1 つだけ持ち，1 サイクルでアクセス可能なものとする．また，MRF のアクセスには 2 ステージが割り当てられているものとする．

表 5.1: RC と MRF の面積オーダ

ユニット	エントリ数	ポート数	面積のオーダ
RC	16	12	16*12*12=2304
MRF	128	4	128*4*4=2048

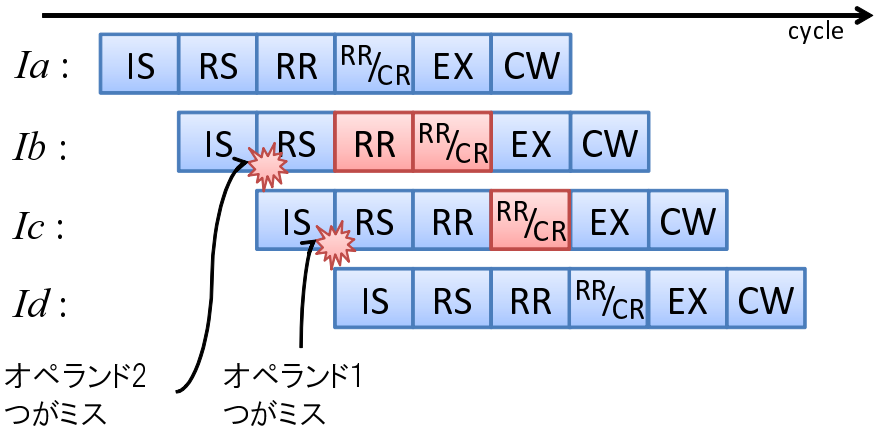

図 5.1: MRF への時分割アクセス

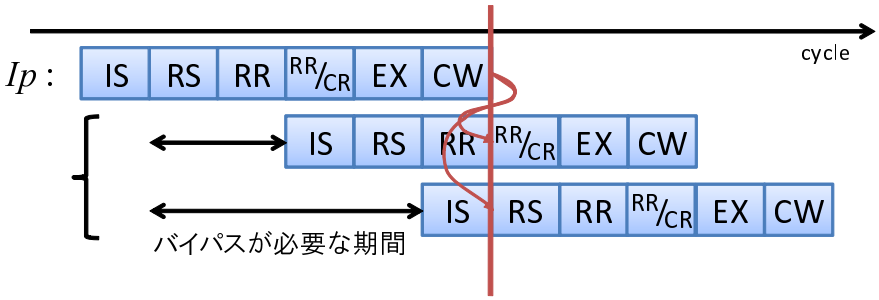

図 5.2: AORC におけるバイパス期間の短縮

今，命令 I_b の持つオペランドのうち 2 つと，I_c の持つオペランドのうち 1 つがミスを起こしたとする．I_b は，RR と RR/CR で MRF から値の読み込みを行う．I_c は，MRF のポート数が 1 しかないため，RS ステージの次のサイクルでは読み込みを行うことができない．しかし，さらに次の RR/RC ステージではポートが空いているため，MRF へアクセスを行うことができる．このようにして，値を時分割で得ることにより，MRF のポートの空きを命令間で共有することができる．

5.2 アレイ・アクセスの遅延による効果

AORC のパイプラインでは，ヒット・ミス判定とデータ・アレイ・アクセスを逐次的に別のステージで行う．これにより，バイパス期間の短縮を行うことができる．

5.2.1　バイパス期間の短縮

データ・アレイ・アクセスを実行ステージの直前まで遅らせることにより，バイパスを行わなければならないサイクル数を短くすることができる．**図 5.2** に例を示す．今，パイプライン上の命令 I_p に対し，I_c が依存しているとする．I_c が，RC を介して値を得ようとした場合，I_p の RC への書き込みが終了した後に読み出しを開始しなければならない．このため，RS ステージに読み出しを行った場合には，4 サイクル間バイパスを行う必要がある．しかし，RR/CR に読み出しを行った場合には，2 サイクル間バイパスを行うだけで済み，2 サイクルの短縮となる．

5.2.2　リフィルの省略

上記のようにして，データ・アレイ・アクセスを遅延させた場合，RC への**リフィル**を行うことは難しくなる．

これは，あるオペランドがヒットと判定された後，データ・アレイ・アクセスまでの間に，RC の置き換えが発生すると，動作に矛盾が生じるためである．このため，AORC では RC へのリフィルを省略する．

なお，RC への書き込みについては，この問題は発生しない．これは，発行されてきた段階で，各命令が RC に書き込むレジスタ番号は判明しているため，タグの更新をデータ・アレイの更新よりも前に早めることで対処できるためである．

リフィルの省略は，RC のヒット率をある程度低下させる．LORC では，ヒット率の低下は性能低下に直結するため，このリフィルの省略は受け入れがたい．しかし，AORC では，リフィルの省略を行ったとしてもほとんど問題とはならない．

たとえば，4 命令同時発行可能な構成で，各命令が 2 つのソース・オペランドを持つ場合，毎サイクル最大 8 個のソース・オペランドがレジスタ・ファイルにアクセスを行う．この場合，ミス率が 5%増加すると，1 サイクルに増加する MRF へのアクセス数は最大で 0.4 となる．実際には全ての命令がソース・オペランドを 2 つ持つわけではなく，全てのサイクルで 4 命令が同時発行されることも無いため，MRF へのアクセスの増加はより小さなものとなる．このため，リフィルを省略したことによる性能の低下は，MRF のポートを 1 つ程度増やすことで十分緩和可能である．また，リフィルを省略した場合，そのためのポートを RC から削減できるため，その回路規模を縮小することができる．

5.3 RC への書き込み

4 章で述べたように，実行後の MRF への書き込みは，RC に対してライトスルーで行う．通常のデータ・キャッシュの場合とは異なり，リネーム後の物理レジスタでは，同一レジスタへの上書きが生じない．このため，ライトバックとした場合でも，MRF への書き込みを減らせないためである．

前述したように，MRF への書き込みは，一旦ライト・バッファ上へバッファリングされた後，サイクル・スチールによって行われる．レジスタ・アクセスの際，このライト・バッファ上にある値に対してアクセスがおきる場合がある．このような場合，動作の正しさを保証するため，バックエンドをストールさせ，MRF への値の書き込みを完了した後に，あらためて MRF から値を得るものとする．これは，ライト・バッファ上に存在する値は，同時に RC にも書き込まれているため，ほとんどの場合は RC から値を取得できることが期待できるためである．また，ライト・バッファからのフォワーディングを行わないことにより，そのためのポートを増やさずに済む．

第6章　評価

6.1　評価方法

研究室で開発したシミュレータである**鬼斬弐**に対し，提案手法を実装して評価を行った．鬼斬弐はプロセッサのサイクル・レベル・シミュレータであり，一般にプロセッサ・アーキテクチャの研究で広く用いられている SimpleScalar ツールセット [6] と比較して，より精度の高いシミュレーションを行うことが可能である．

6.1.1　ベンチマーク

評価には，SPEC CPU2000[7] ベンチマーク・セットの中から，21 本のアプリケーションを使用した．アプリケーションのコンパイルには gcc 4.2.2 を用い，コンパイル・オプションについては，SPEC CPU2000 に含まれる環境設定スクリプトが指定するものに従った．各ベンチマークの入力データ・セットには train を用い，最初の 1G 命令をスキップして，続く 100M 命令を実行した．

6.1.2　評価モデル

LORC，バイパス・バッファ，そして AORC について，以下のモデルを実装して評価した．

- **LORC**　LORC を用いたモデル．LORC の性能は，その置き換えアルゴリズムやヒット・ミス予測精度に強く依存するため，ある程度の理想化を行っている．RC の構成は容量に関わらずフルアソシアティブとし，置き換えは LRU で行った．また，ヒット・ミス予測に関しては完全にヒットするものとした．
- **LORC PERFECT**　LORC において，RC に必ずヒットするモデル．このモデルは，1 サイクルでアクセス可能なレジスタ・ファイルと等価であり，LORC の性能向上の上限を与える．
- **BB**　バイパス・バッファを用いたモデル．このモデルはパイプライン化されたレジスタ・ファイルと性能的に等価である．

- **AORC**　AORC を用いたモデル．RC の構成はフルアソシアティブであり，置き換えは，LRU よりも単純な実装で済む NLU（Not-last used）を用いた．なお，ヒット・ミス予測については，その必要が無いため行っていない．

RC のエントリ数については，LORC では 8,16,32,64 の 4 通りに，AORC では，8 と 16 の 2 通りに幅を振って測定を行った．また，AORC のライト・バッファの容量は，一律 8 エントリとした．

6.1.3　構成

評価したプロセッサのパラメータを**表 6.1** に示す．分岐予測ペナルティが範囲を持つのは，モデルによってパイプライン段数が異なるためである．

6.2　評価結果

各モデルの IPC を**図 6.1** と**図 6.2** に示す．この IPC は，各ベンチマークごとの IPC をバイパス・バッファのそれによって正規化し，さらにベンチマーク全体で平均化したものである．

AORC の構成については，図 6.1 は MRF のアクセスに 1 ステージを割り当てたもの，図 6.2 は MRF のアクセスに 2 ステージを割り当てたものの性能である．

グラフからは，以下のことが読み取れる：

- MRF のポート数が 4 ポートの場合，AORC の性能低下率は最大で 0.5%程度であり，ほとんど性能低下を起こしていない．
- AORC は，MRF のポート数が 3 ポートより少ない場合，徐々に性能が低下する．特にポート数が 1 の場合の性能低下は激しく，40%以上の性能低下を起こす．
- LORC PERFECT の性能向上は，MRF のレイテンシが 2 サイクルの場合で 1.6%，3 サイクルの場合で 3.5%である．これは，3 章で述べた（1）パイプラインの深化による一般的な問題による影響が軽微であり，改善による LORC の伸びしろが少ないことを意味する．
- LORC LRU は，物理レジスタのエントリ数の半分となる 64 エントリをもつものであっても，4%程度の性能低下を起こす．

表 6.1: プロセッサの構成

パラメータ	値
ISA	Alpha
fetch width	4 inst.
execution unit	int : 2, fp : 2, mem : 2.
instruction window	int : 32, fp : 16, mem : 16
main register file	int : 128, fp : 128
branch prediction	8KB g-share
misspenalty	10〜13 cycle
BTB	2K entry, 4-way
RAS	8 entry
L1C	32KB, 4-way, 64B/line, 3 cycle
L2C	4MB, 8-way, 64B/line, 10 cycle
main memory	200 cycle

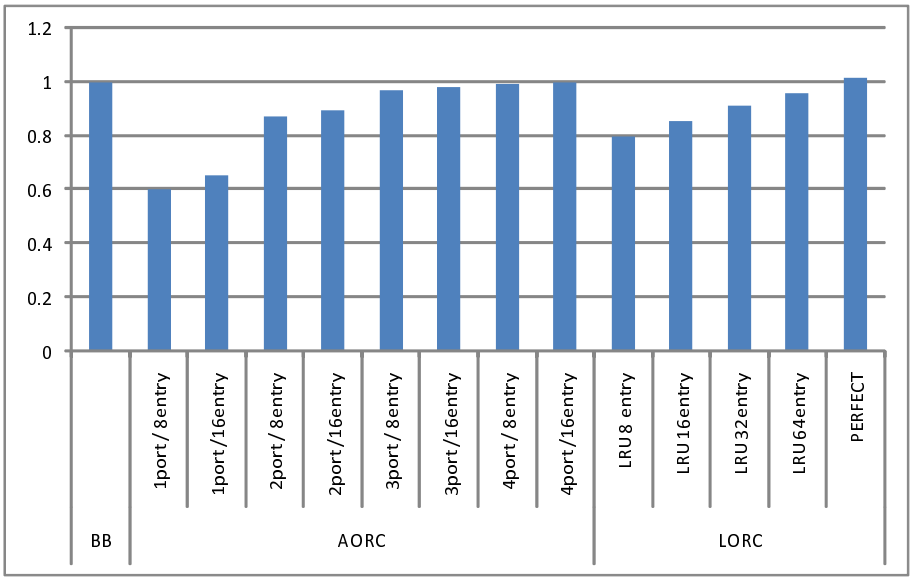

図 6.1: 平均相対 IPC（MRF のレイテンシ：2）

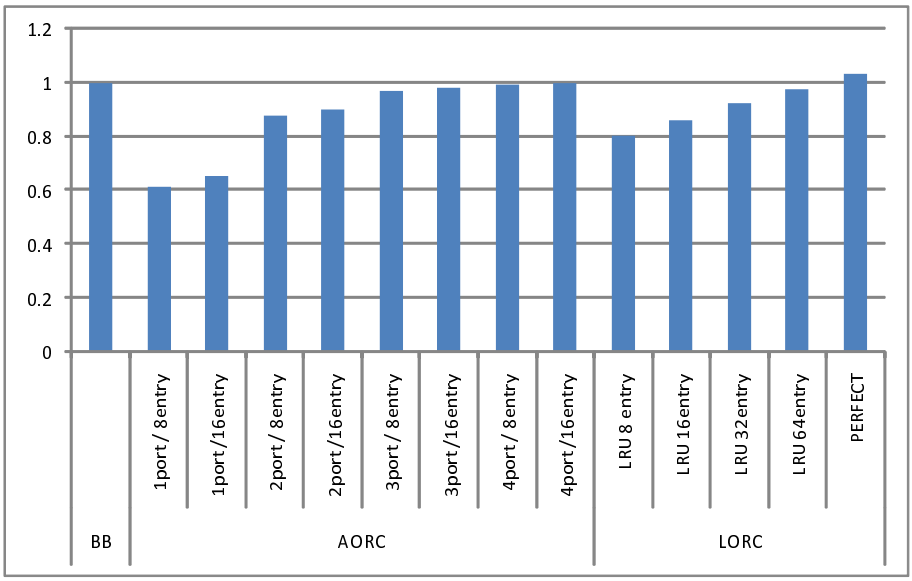

図 6.2: 平均相対 IPC（MRF のレイテンシ：3）

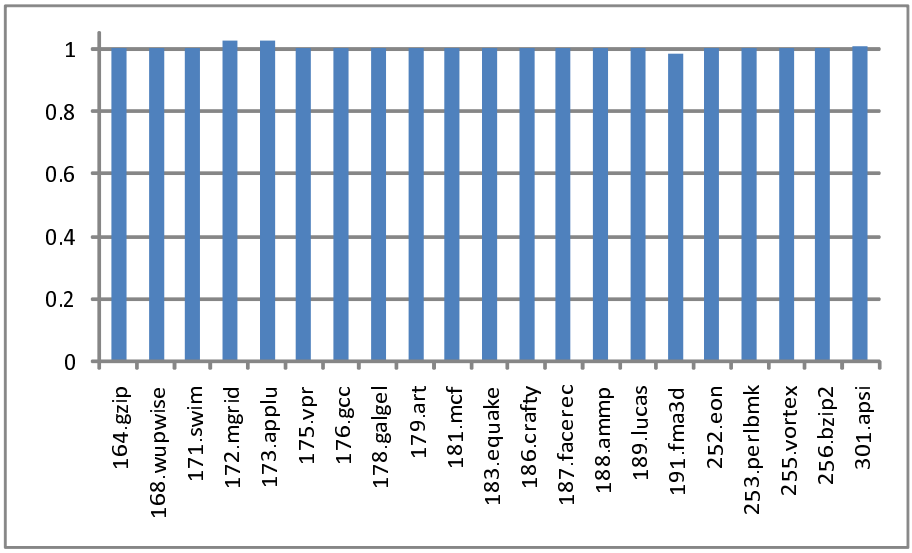

図 6.3: AORC の相対 IPC（MRF のレイテンシ：2，ポート数：4）

6.2.1 AORC の IPC

ポート数が 4 の場合

図 6.3 に，MRF のレイテンシを 2 サイクル，ポート数を 4 とした場合の，AORC の IPC を示す．この IPC は，各ベンチマークごとの IPC を，バイパス・バッファのそれによって正規化したものである．

ポート数が 4 の場合，全てのベンチマークにおいて，ほとんど性能が落ちていないことがわかる．

ポート数が 2 の場合

図 6.4 に，MRF のレイテンシを 2 サイクル，ポート数を 2 とした場合の，AORC の IPC を示す．この IPC は，各ベンチマークごとの IPC を，バイパス・バッファのそれによって正規化したものである．

ポート数が 2 の場合，ベンチマーク毎にその傾向は大きく異なることがわかる．172.mgrid や，301.apsi では 20%以上の性能低下があるものの，178.galgel や 179.art のようにほとんど性能低下が起きていないものもある．

6.2.2 RC・ヒット率

各モデルにおける，RC へのヒット率を**図 6.5** に示す．このヒット率は，各ベンチマークにおけるヒット率を平均化したものである．なお，MRF のレイテンシやポート数に関しては，それぞれ変化させた場合でも，傾向はほぼ一致していたため，MRF のレイテンシを 2 サイクルとし，ポート数を 4 としたもののデータを示す．

AORC と LORC では，置き換えアルゴリズムがそれぞれ NLU と LRU と言う違いがあるが，エントリ数が同じ場合，そのヒット率はほとんど変化しない．

先の図 6.1 より，RC のエントリ数を 8 ないしは 16 とした場合，LORC は前者で 20%，後者で 15%の性能低下を見せている．一方，同図の AORC では，ポート数が 4 の場合，その性能低下は RC のエントリ数が 8 の場合で 0.5%，16 の場合で 0.2%である．これにより，LORC で深刻な性能低下を起こすようなヒット率であっても，AORC では，ポート数が十分な場合には性能低下を起こさないことがわかる．

AORC の RC・ヒット率

AORC における，ベンチマーク毎の RC へのヒット率を図 6.5 に示す．RC のエントリ数を 8 から 16 に増加させた場合，168.wupwise や 188.ammp では，大

幅にヒット率が改善していることがわかる．一方，172.mgrid や 189.lucas などはほとんどヒット率が改善していない．

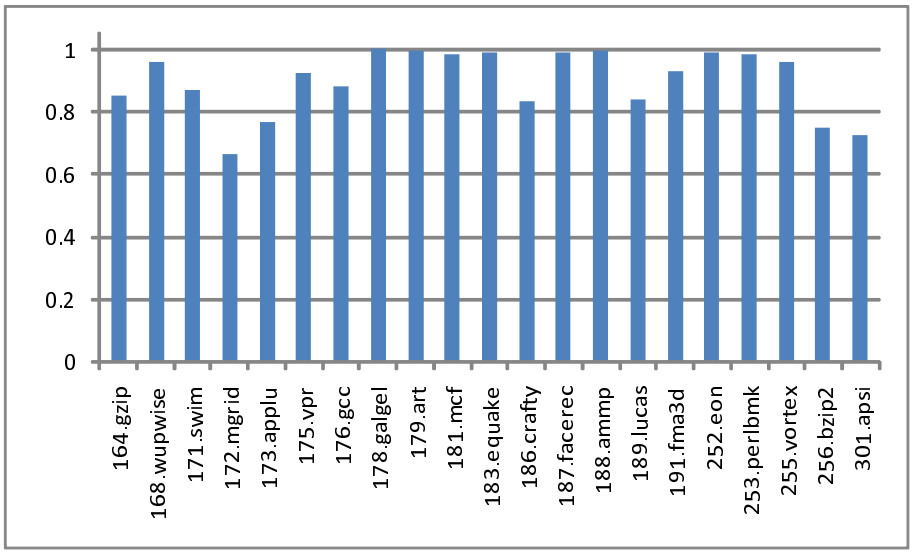

図 6.4: AORC の相対 IPC（MRF のレイテンシ：2，ポート数：2）

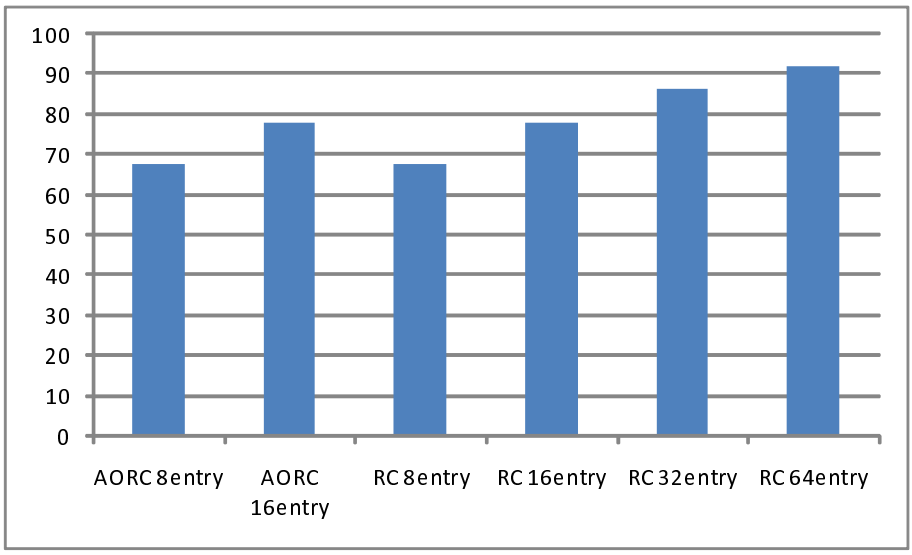

図 6.5: キャッシュ・ヒット率（MRF のレイテンシ：2）

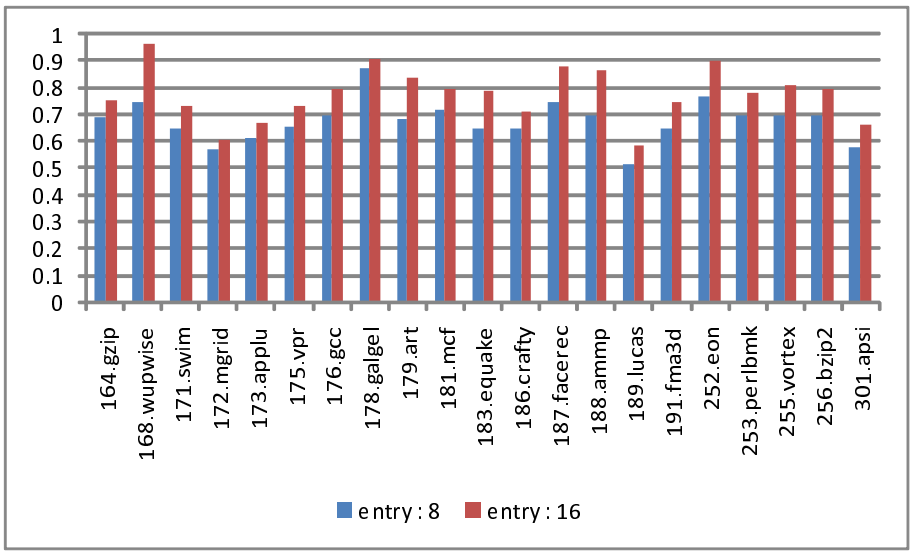

図 6.6: AORC のキャッシュ・ヒット率（MRF のレイテンシ：2，ポート数：4）

第7章　おわりに

本稿では，回路面積の縮小を目的とする，AORC の提案を行った．評価の結果，提案手法では性能をほとんど落とすことなく，MRF のポート数を大きく減らすことができることを確かめた．

今後の課題としては，提案手法を SMT に適用した場合の評価がある．1 章で述べたように，SMT は，物理レジスタを巨大化させる大きな動機であり，提案手法による回路面積削減の恩恵を大きく受ける．このため，今後はこの SMT に対して実装を行った場合の評価を行いたい．

また，ミスを仮定したパイプライン構成において，RC をフィルタとして用いる提案手法のアプローチは，物理レジスタ以外のモジュール—たとえば RMT やそのチェックポイントなど—にも有効であると考えている．今後は，これらについて新たな手法を検討する予定である．

参考文献

[1] J. A. Butts and G. S. Sohi. Use-Based Register Caching with Decoupled Indexing. In *Proceedings of the 31th International Symposium on Computer Architecture(ISCA)*, pp. 302–313, 2004.

[2] J. L. Crutz, A. Gonzalez, and M. Valero. Multiple-Banked Register File Architecture. In *Proceedings of the 27th International Symposium on Computer Architecture(ISCA)*, pp. 316–325, 2000.

[3] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel. The Microarchitecture of the Pentium 4 Processor. *Intel Technology Journal Vol.5 Issue 1*, feb 2001.

[4] K.J. Kim, J.M. Youn, S.B. Kim, J.H. Kim, S.H. Hwang, K.T. Kim, and Y.S. Shin. A novel 6.4 μm2 full-cmos sram cell with aspect ratio of 0.63 in a high-performance 0.25 μm-generation cmos technology. *VLSI Technology, 1998. Digest of Technical Papers. 1998 Symposium on*, pp. 68–69, 1998.

[5] Ronald P. Preston, Roy W. Badeau, Daniel W. Bailey, Shane L. Bell, Larry L. Biro, William J. Bowhill, Daniel E. Dever, Stephen Felix, Richard Gammack, Valeria Germini, Michael K. Gowan, Paul Gronowski, Daniel 6. Jackson, Shekhar Mehta, Shannon V. Morton, Jeffrey D. Pickholtz, Matthew H. Reilly, and Michael J. Smith. Design of an 8-wide superscalar risc microprocessor with simultaneous multithreading. *International Solid-State Circuits Conference*, pp. 334–335, 2002.

[6] SimpleScalar LLC. *SimpleScalar version 3.0. http://www.simplescalar.com/*.

[7] The Standard Performance Evaluation Corporation. *SPEC CPU2000 suite http://www.spec.org/cpu2000/*.

[8] Y. Tatsumi and H.J. Mattausch. Fast quadratic increase of multiport-storage-cell area with port number. *Electronics Letters*, Vol. 35, No. 25, pp. 2185–2187, 1999.

[9] Neil H. E. Weste, David Harris, and Addison Wesley. *CMOS VLSI Design -a circuits and systems perspective 3rd edition*. Pearson/Addison-Wesley, 2005.

[10] N.C. Yung, R.; Wilhelm. Caching processor general registers. *Proceedings of the International Conference on Computer Design(ICCD)*, pp. 307–312, 1995.

[11] 山本哲弘, 安藤秀樹, 島田俊夫. 先行実行を利用したロード命令のレイテンシ削減および正確なスケジューリング手法. 先進的計算基盤システムシンポジウム SACSIS, pp. 403–410, 2006.

[12] 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 SACSIS, pp. 33–40, 2006.

[13] 小林良太郎, 梶山太郎, 島田俊夫. 物理レジスタ番号の割り当て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 SACSIS, pp. 13–22, 2006.

[14] 五島正裕. ディジタル回路. 数理工学社, 2007.

[15] 三輪忍, 一林宏憲, 入江英嗣, 五島正裕, 富田眞治. 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. 情報処理学会論文誌 コンピューティングシステム ACS, 第 48 巻, pp. 58–69, 2007.

発表文献

主著論文

1. マルチコア・プロセッサの不均質共有キャッシュにおける LRU 大域置き換えアルゴリズム
   塩谷 亮太
   卒業論文，東京大学 工学部 (Feb. 2006)

2. マルチコア・プロセッサの不均質共有キャッシュにおける LRU 大域置き換えアルゴリズム
   塩谷 亮太，ルォン ディン フォン , 入江 英嗣 , 五島 正裕，坂井 修一
   先進的計算基盤システムシンポジウム 2006(SACSIS2006), 於 大阪国際会議場 (グランキューブ大阪), Vol.2006, No.5, pp. 23-31, May, 2006.

3. マルチコア・プロセッサの不均質共有キャッシュにおける LRU 大域置き換えアルゴリズム
   塩谷 亮太，ルォン ディン フォン , 入江 英嗣 , 五島 正裕，坂井 修一
   情報処理学会論文誌コンピューティングシステム (ACS 17), Vol.48, No.SIG3, pp.59-74, Feb, 2007.

4. 回路面積指向レジスタ・キャッシュ
   塩谷 亮太，入江 英嗣 , 五島 正裕，坂井 修一
   先進的計算基盤システムシンポジウム SACSIS 2008
   （投稿中）

5. 回路面積指向レジスタ・キャッシュ
   塩谷 亮太，入江 英嗣 , 五島 正裕，坂井 修一
   情報処理学会論文誌コンピューティングシステム ACS 23
   （投稿中）

共著論文

1. アドレスオフセットに着目したデータフロー追跡による注入攻撃の検出
   勝沼 聡，栗田 弘之，塩谷 亮太，清水 一人，入江 英嗣，五島 正裕，坂井 修一
   先進的計算基盤システムシンポジウム SACSIS2006, May, 2006.

2. Base Address Recognition with Data Flow Tracking for Injection Attack Detection
   Satoshi Katsunuma, Hiroyuki Kurita, Ryota Shioya, Kazuto Shimizu, Hidetsugu Irie, Masahiro Goshima, and Shuichi Sakai
   IEEE International Symposium on Pacific Rim Dependable Computing (PRDC 2006), 於 Riverside, California, USA, pp.165-172, Dec, 2006.

3. 動的なインフォメーションフロー制御による情報漏洩防止手法
   栗田 弘之，塩谷 亮太，入江 英嗣，五島 正裕，坂井 修一
   情報処理学会報告 2007-ARC-172, 於 北海道大学学術交流会館, Vol.2007, No.17, pp.227-232, Mar, 2007.

4. ツインテール・アーキテクチャにおける ハーフパンプ FU アレイ
   亘理 靖展，堀尾 一生，渡辺 憲一，塩谷 亮太，入江 英嗣，五島 正裕，坂井 修一
   先進的計算基盤システムシンポジウム SACSIS 2008
   （投稿中）

5. ツインテール・アーキテクチャにおける ハーフパンプ FU アレイ
   亘理 靖展，堀尾 一生，渡辺 憲一，塩谷 亮太，入江 英嗣，五島 正裕，坂井 修一
   情報処理学会論文誌コンピューティングシステム ACS 23
   （投稿中）

6. インジェクションアタック防止のための文字列に着目した情報フロー追跡方式
   勝沼 聡，塩谷 亮太，入江 英嗣，五島 正裕，坂井 修一
   先進的計算基盤システムシンポジウム SACSIS 2008
   （投稿中）

7. インジェクションアタック防止のための文字列に着目した情報フロー追跡方式
   勝沼 聡，塩谷 亮太，入江 英嗣，五島 正裕，坂井 修一
   情報処理学会論文誌コンピューティングシステム ACS 23
   （投稿中）

8. 逆 Dualflow アーキテクチャ
   一林 宏憲，塩谷 亮太，入江 英嗣，五島 正裕，坂井 修一
   先進的計算基盤システムシンポジウム SACSIS 2008
   （投稿中）

9. 逆 Dualflow アーキテクチャ
   一林 宏憲，塩谷 亮太，入江 英嗣，五島 正裕，坂井 修一
   情報処理学会論文誌コンピューティングシステム ACS 23
   （投稿中）

10. 可変長タグをサポートする低オーバヘッド・タグ・アーキテクチャ
    金 大雄，塩谷 亮太，入江 英嗣，五島 正裕，坂井 修一
    先進的計算基盤システムシンポジウム SACSIS 2008
    （投稿中）

11. 可変長タグをサポートする低オーバヘッド・タグ・アーキテクチャ
    金 大雄，塩谷 亮太，入江 英嗣，五島 正裕，坂井 修一
    情報処理学会論文誌コンピューティングシステム ACS 23
    （投稿中）

12. メモリアクセス最適化を適用した汎用プロセッサと Cell の性能比較
    原 健太朗，塩谷 亮太，田浦 健次朗
    先進的計算基盤システムシンポジウム SACSIS 2008
    （投稿中）

13. メモリアクセス最適化を適用した汎用プロセッサと Cell の性能比較
    原 健太朗，塩谷 亮太，田浦 健次朗
    情報処理学会論文誌コンピューティングシステム ACS 23
    （投稿中）

14. 情報漏洩防止のための暗黙的インフォメーションフロー追跡
    横田 侑樹，塩谷 亮太，入江 英嗣，五島 正裕，坂井 修一
    情報処理学会 第 70 回全国大会
    （投稿中）

15. 文字列に着目した情報フロー追跡によるインジェクション攻撃の検出
    勝沼 聡，塩谷 亮太，入江 英嗣，五島 正裕，坂井 修一

組込技術とネットワークに関するワークショップ 2008（ETNET2008）
（投稿中）

謝辞

非常に多くの方々から多大なご指導、ご協力、励ましを頂き、本論文を完成させることができました。この場を借りて、感謝の意を表したいと思います。

指導教官である五島正裕准教授には学部 4 年からの 3 年間に渡って多くのご指導，ご助言を頂きました．また，坂井修一教授からも，大変多くのご指導を頂きました．ここに深く感謝の意を表します.

入江英嗣博士には，論文の書き方や研究についてなど，様々な形で意見やアドバイスを頂きました.

Luong Dinh Hung 氏には，学部 4 年次に直接指導を行って頂いたのを始めとして，様々な点でお世話になりました.

渡辺憲一氏，一林宏憲氏，亘理靖展氏をはじめ，鬼斬開発チームの皆様には大変お世話になりました．特に，渡辺憲一氏の尽力なくしては，この研究は完成しなかったと思います.

八木原晴水さん，月村美和さんには，研究室における設備の導入や各種事務手続きなど，研究室で過ごすための様々なご支援を頂きました.

また，ここでは紹介しきれなかった研究室のメンバーの皆様にも様々な形でお世話になりました．本当にありがとうございます.