情報処理学会論文誌　コンピューティングシステム　Vol. 3　No. 3　44–55 (Sep. 2010)

# キャッシュを用いたレジスタ・マップ表の回路面積削減

三輪　忍[†1]　張　鵬[†1]　横山弘基[†1]
堀部悠平[†1]　中條拓伯[†1]

SMT の普及により，近年，レジスタ・マップ表は肥大化する傾向にある．マップ表は，通常，マルチポート RAM で構成される．同じくマルチポート RAM であるレジスタ・ファイルに対しては，小容量のキャッシュを用いて回路面積を削減する手法が提案されているが，この手法をマップ表に適用した例はまだない．また，この手法は，マルチポート RAM の回路面積を削減する一般的な手法，たとえばマルチバンク化などとの比較がまったく行われていなかった．そこで今回，小容量のキャッシュを用いる手法をマップ表に適用し，マルチバンク化した場合との比較を行った．本稿ではその結果を報告する．

# Area-efficient Register Map Table Using a Cache

Shinobu Miwa,[†1] Peng Zhang,[†1] Hiroki Yokoyama,[†1]
Yuhei Horibe[†1] and Hironori Nakajo[†1]

Area of register map tables is growing up in recent processors following the spread of SMT technologies. Register map tables are usually implemented with multi-port RAMs as well as register files. In order to reduce area of register files, a technique which uses a small cache has been proposed, but it has never been applied to register map tables. Moreover, the technique has never been compared with other techniques which aim to reduce area of multi-port RAM. This paper shows the result when both techniques are applied to register map tables.

†1 東京農工大学
Tokyo University of Agriculture and Technology

## 1. はじめに

近年の Out-of-Order スーパスカラ・プロセッサにおいて，**レジスタ・マップ表**（**Register Map Table**．以下 **RMT** とする）の回路規模は無視できなくなっている．RMT は，通常，デコード幅の 4 倍のポートを持つ RAM によって構成される[⋆1]．そのため，4-way のプロセッサでは，RMT のポート数は 16 にも達する．RAM の回路面積はポート数の 2 乗に比例する[16] ため，RMT の規模は，エントリ数やビット幅こそ小さいものの，決して無視できるものではない．

そのうえ，SMT（Simultaneous Multi-Threading）[19] の普及により，RMT の規模は拡大する傾向にある．SMT プロセッサでは，スレッドごとのコンテキストを保持するため，スレッド数に応じた RMT を必要とする[11]．そのため，そのようなプロセッサでは，リネーミング・ロジック全体の規模が 1 次キャッシュのそれを超えることもある[14]．

このような RMT が消費する電力，および，遅延に与える影響は大きい．RMT の消費電力はプロセッサ全体のそれの 4%程度と，リザベーション・ステーション，あるいは，グローバル・クロックのそれに匹敵する[10]．また，RMT の遅延は大きく，そのアクセスに 2 サイクルを要することもある[5]．RMT の回路面積を削減すれば，これらの影響を緩和できる．

マルチポート RAM の面積を削減する場合，RAM を複数のバンクに分割する，**マルチバンク化**がよく行われる[18]．分割によってバンクあたりのアクセス数は減少するため，各バンクには分割前と同数のポートは必要ない．単純には，$N$ バンクに分割した場合，ポート数を $1/N$ にまで削減できる．そうすれば，RAM の面積は分割前のおよそ $(1/N)^2$ になる．

一方，塩谷らはキャッシュを用いてレジスタ・ファイルの面積を削減する手法を提案している[21),22]．レジスタ・ファイルもまたマルチポート RAM によって構成されるユニットである．塩谷らの手法では，多ポート，かつ，小容量のキャッシュをレジスタ・ファイルのアクセス・フィルタとして用いる．キャッシュにヒットすればレジスタ・ファイルにアクセスする必要がないため，レジスタ・ファイルのポートはそれほど必要ない．文献 21) によると，キャッシュのエントリが 16 もあれば，ポート数を 1/3 にまで削減できる．

塩谷らは，文献 21) において，この方法を RMT にも適用できる可能性があることを指摘している．しかし，具体的にどのようにして適用するかまでは述べていない．また，マル

⋆1 CAM を用いる方式[7] もあるが，本稿では RAM を用いる方式[20] を議論の対象とする．ただし，CAM 方式の RMT にも本稿で述べる手法は適用できる．

チポート RAM の回路面積削減手法としてはマルチバンク化が一般的であるが，それを行った場合との比較も行っていない．

そこで我々は，キャッシュを用いる手法を RMT に適用し，その回路面積を削減した．そして，マルチバンク化よりも面積削減の効果が高いことを確認した．本稿では，その詳細について報告する．

以下まず次章では，リネーミング・ロジックを示し，RMT がその大部分を占めていることを述べる．続く 3 章では，RMT に適用する場合を例にマルチバンク化を説明した後，レジスタ・ファイルの回路面積をキャッシュを用いて削減した手法について説明する．4 章でキャッシュを用いて RMT の面積を削減する方法を述べ，5 章で評価し，6 章でまとめる．

## 2. リネーミング・ロジック

リネーミング・ロジックを**図 1** に示す．図はデコード幅が 4 の場合を表している．ロジックは主に 3 つのコンポーネントからなる．**RMT**，**フリー・リスト**（**Free List**），**依存チェック・ロジック**（**Dependence Check Logic**）である．

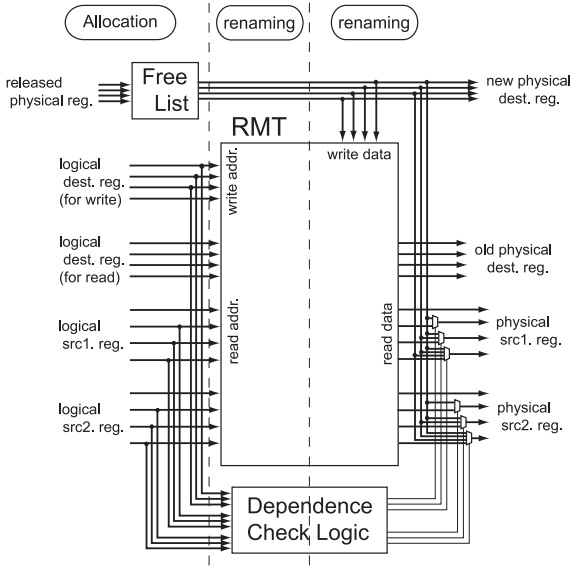


**図 1**　通常のリネーミング・ロジック
Fig. 1　Conventional renaming logic.

### 2.1 RMT

RMT は，論理レジスタと物理レジスタとの（投機的な）マッピングを表した表である．論理レジスタ数分のエントリを持ち．1 つのエントリが 1 つの論理レジスタに対応する．各エントリには，その論理レジスタに割当て中の物理レジスタ番号が記録される．

命令がデコードされると，論理ソース・レジスタ番号によって表を参照し，対応する物理レジスタ番号を得る．また，後述するフリー・リストから物理レジスタを取り出し，それを論理デスティネーション・レジスタに割り当てる．新しく割り当てられた物理レジスタ番号は RMT に記録される．

RMT は単純に更新してよいものではない．通常，各命令がコミットされる際，そのデスティネーションに以前割り当てられていた物理レジスタの解放が行われる．このため，新しく割り当てた物理レジスタ番号を RMT に記録する際は，以前の番号を読み出し，それをリオーダ・バッファに記録しておく．

### 2.2 フリー・リスト

解放された物理レジスタは，フリー・リストと呼ばれるバッファによって管理される．通常，FIFO で管理される．解放されたレジスタがリストの末尾に追加され，割当て時にはリストの先頭から空き物理レジスタが取り出される．

フリー・リストのアクセスには 1 つのパイプライン・ステージが割り当てられることもある．たとえば，Intel Pentium 4 では，フリー・リストのアクセスに 1 サイクル，RMT のアクセスに 2 サイクル，計 3 サイクルかけてリネーミングを行っている[5]．

### 2.3 依存チェック・ロジック

同時にリネーミングされるレジスタ間に，真の依存関係が存在する場合もある．RMT のアクセスには 1 サイクル以上要するため，依存元のレジスタに割り当てられた物理レジスタは，リネーミング時にはまだ書き込まれていない．したがって，依存先のレジスタが RMT をアクセスしても，正しい依存元の物理レジスタ番号は得られない．このような場合でも，正しいリネーミング結果を出力する必要がある．

依存チェック・ロジックは，同時にリネーミングされるレジスタ間の真の依存をチェックするロジックである．通常，図 1 のように，RMT に並列に配置される．RMT の後段に配置されたマルチプレクサにより，RMT から読み出した物理レジスタとそのとき割り当てた物理レジスタのどちらか一方を出力する．このマルチプレクサを依存チェック・ロジックは制御する．

依存チェックには，デコード幅が 4 の場合，12 個の比較器[⋆1]と 4 つのプライオリティ・エンコーダを必要とする．これは RMT の規模に比べれば十分小さい．

### 2.4　チェックポイント

RMT は投機的に更新されるため，分岐予測ミスなどが発生した際は回復処理が必要となる．そのため，分岐命令がリネーミングされたときを**チェックポイント**とし，チェックポイントごとに RMT の状態をバックアップしておく[⋆2]．分岐予測ミス時には，該当するバックアップを RMT に書き戻すことで回復を行う．

回復を高速にするため，RMT の各ビットに，現在のマッピングを保持するメイン・セルに加え，チェックポイント用のセルが組み込まれることもある[15]．その場合は，ビットあたりの面積が増加し，RMT 全体が肥大化することになる．

### 2.5　RMT の回路面積

2.1 で述べたように，命令 1 つにつき，RMT は以下の 3 種類のポートを必要とする．

(1)　各ソースのリネーミングのためのリード・ポート 2 つ
(2)　デスティネーションに割り当てた物理レジスタを書き込むためのライト・ポート 1 つ
(3)　解放する物理レジスタを読み出すためのリード・ポート 1 つ

したがって，同時に 4 命令をリネーミングするプロセッサの RMT は，12 個のリード・ポート，4 個のライト・ポートを持つ，計 16 ポートの RAM になる．

SRAM の回路面積は，ポート数の 2 乗にほぼ比例する[16]．64 個の論理レジスタ，256 個の物理レジスタを持つ，2 スレッドの SMT プロセッサがあったとする．すると，そのようなプロセッサの RMT の面積は，ビットあたりの面積を $\alpha$ とすると，およそ $16^2{*}64{*}8{*}2{*}\alpha = 262,144\alpha$ となる．

一方，2 ポート，8 KB のキャッシュのデータ・アレイの面積は，ビットあたりの面積を $\beta$ として，262,144 $\beta$ で表される．前節で述べたように，チェックポイント用のセルを含む分，ビットあたりの面積はキャッシュよりもマップ表の方が大きい．すなわち，RMT の面積は 1 次キャッシュのデータ・アレイのそれを超える．このように RMT の面積はエントリ数の割に大きい．

⋆1 各ソースは，それに先行する命令のデスティネーションとだけ比較すればよいため，比較器は $2{*}(3{+}2{+}1) = 12$ 個となる．

⋆2 RMT の回復方式にはリタイアメント・マップ表を用いる方式[5] もあるが，ここでは議論の対象としない．ただし，リタイアメント・マップ表を用いる方法であっても，本稿で述べる手法によってフロントエンド・マップ表のポートを削減することができる．しかも，本稿で述べる手法はリタイアメント・マップ表に対しても適用できる．

## 3.　マルチポート RAM の回路面積削減手法

RMT と同様，マルチポート RAM で構成されるユニットにレジスタ・ファイルがある．レジスタ・ファイルは RMT よりも多くのエントリ/ビット幅を必要とする．そのため，レジスタ・ファイルの面積は RMT のそれよりも大きく，プロセッサ全体で見てもかなり大きなウェイトを占めている[14]．

レジスタ・ファイルに対しては，そのポート数を減らすことで回路面積の削減を図る手法がいくつか提案されている[18),21),22]．前述のように，マルチポート RAM の回路面積はポート数の 2 乗に比例するため，回路面積を削減するうえでポートを減らす効果は大きい．

以下ではまず，こうした手法として一般的なマルチバンク化について，RMT に適用する場合を例にそれを説明する．次いで，今回 RMT への適用を行った，キャッシュを用いてレジスタ・ファイルの回路面積を削減する手法について述べる．

### 3.1　マルチバンク化

RAM の面積を削減する場合によく行われるのが，マルチバンク化である．文献 18) では レジスタ・ファイルにマルチバンク化を施しているが，本節では RMT に施す場合を例に説明する．

図 1 の RMT をマルチバンク化した場合のロジックを**図 2** に示す．図のロジックは，RMT を 2 つのバンクにインタリーブし，各バンクのポートを分割前の半分にしてある．

2 つのバンク even/odd は，それぞれ，偶数/奇数番号の論理レジスタに関するマッピングを管理する．すなわち，even バンクは $0, 2, 4, \cdots$ 番論理レジスタと物理レジスタとのマッピングを，odd バンクは $1, 3, 5, \cdots$ 番論理レジスタと物理レジスタとのマッピングを保持する．各バンクへのアクセスは，それらの手前に置かれたクロスバ・スイッチによって制御される．本稿では，この制御を **RMT アクセス・スケジューリング**と呼ぶことにする．

バンクに分割することにより，RMT へのアクセスはそれぞれのバンクに分散される．単純計算すると，2 つのバンクに分割したことで，バンクごとのアクセス数は RMT 全体のアクセス数の半分になる．すなわち，各バンクのポート数を半分に減らしたとしても，十分なデータ供給能力を維持できる可能性がある．

そこで，各バンクのポートを減らす工夫が行われる．図 2 では，2.5 で述べた 3 種類のポートをそれぞれ半分にしている．すなわち，ソース読み出し用の 8 つのリード・ポートを 4 つずつに，4 つのライト・ポートを 2 つずつに，そして 4 つの解放用のリード・ポートを 2 つずつにしている．

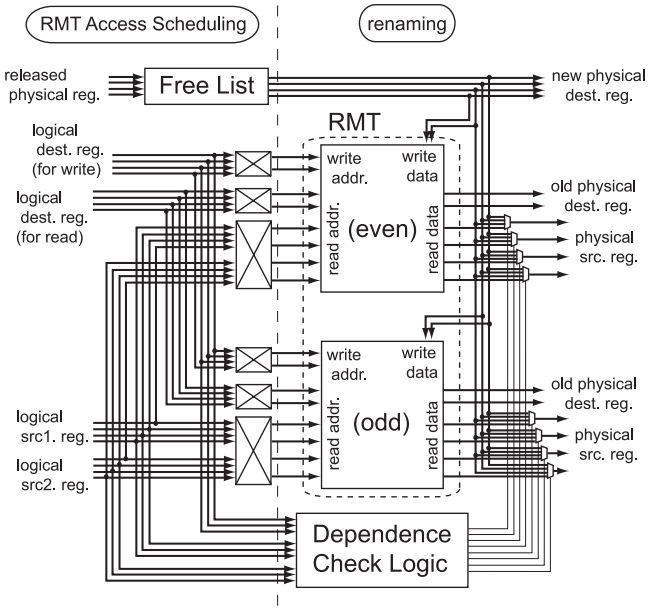


**図 2**　RMT をマルチバンク化した場合のリネーミング・ロジック
Fig. 2　Renaming logic with multi-banked map table.


**図 3**　面積を削減した RMT を用いたときの命令パイプライン
Fig. 3　Instruction pipeline on a processor with an area-efficient map table.

前述のように，SRAM の回路面積はポート数の 2 乗に比例する．したがって，2 バンクに分割し，ポートを半分にすることで，面積は $2 * 2 * (8^2 * 32 * 8 * \alpha) = 65,536\alpha$ となり，分割前（2.5）の 1/4 になる．

**動　　作**

マルチバンク化した RMT を用いるプロセッサのフロントエンド・パイプラインを**図 3** に示す．なお，後述するように，キャッシュを用いて面積を削減した RMT を用いる場合もまったく同じパイプライン構成をとる．

各ステージは，先頭から順に，フェッチ（IF），デコード（DEC），RMT アクセス・スケジューリング（RMTAS），RMT アクセス（RMT），ディスパッチ（disp.）を表している．ただし，回路面積が減ることで，RMT のレイテンシは 1 サイクルになるものとしている．RMT アクセス・スケジューリングで各リード要求がどのバンクをアクセスするかを制御し，次のサイクルでバンクへのアクセスを開始する．

バンクのポートを減らしたことで，**バンク・コンフリクト**が発生することもある．たとえば，同時にデコードされた命令の論理ソース・レジスタが 0，2，4，6，8 番だったとする．すると，even バンクにおいて，ソース読み出し用のポートが 1 ポート分不足してしまい，すべてのレジスタを同時にリネーミングできない．

このような場合は**ストール**によって対処する．バンク・コンフリクトが発生した際は，デコード以前のステージをストールさせたうえで，バンクへのアクセスを 2 回に分けて行う．このストールは性能低下の要因となる．

### 3.2　キャッシュを用いる手法

レジスタ・ファイルの面積を削減する一風変わった手法に，**レジスタ・キャッシュ**（**Register Cache**）を用いる手法がある．

レジスタ・キャッシュは，1 サイクル程度でアクセス可能な小型のキャッシュである．通常のキャッシュがメイン・メモリの一部を保持するのと同様，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．レジスタ・キャッシュは，それにヒットすれば高速にレジスタ・アクセスが行えるため，通常はレジスタのアクセス・レイテンシを短縮するために用いられる[1),3),4),23),24)]．

それに対して塩谷らは，キャッシュをレジスタ・ファイル・アクセスのフィルタとしての役に専念させた，**回路面積指向レジスタ・キャッシュ**（**Area-Oriented Register Cache**. 以下 **AORC** とする）を提案している[21),22)]．AORC では，レイテンシの短縮ではなく，レジスタ・ファイルへのアクセス数を抑えるためにレジスタ・キャッシュを用いる．そうすることで，レジスタ・ファイルのポート数を大幅に削減する．

AORC を用いた場合のレジスタ・ファイル周辺の回路構成を**図 4** に示す．AORC とアービタが追加されている点が通常とは異なる．

AORC は物理的には通常のレジスタ・キャッシュと変わらない．16 エントリ程度の小型のキャッシュであり，通常のレジスタ・ファイルと同数のポートを持つ．すなわち，発行幅が 4 の場合は，リード・ポート 8 つ，ライト・ポート 4 つである．なお，キャッシュはライト・スルーで動作する．

アービタはレジスタ・ファイルへのアクセスを制御する調停回路である．AORC にミス

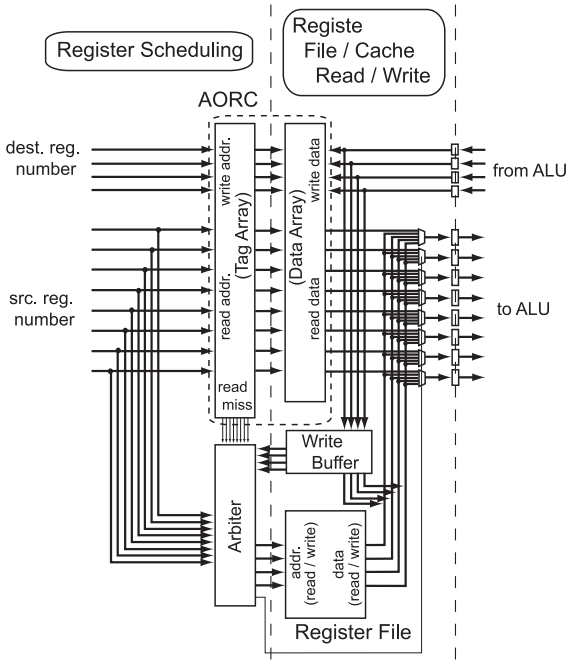


**図 4**　回路面積指向レジスタ・キャッシュ
Fig. 4　Area-oriented register cache.

したレジスタ・リード要求，および，ライト・バッファからの書き込み要求を調停する．

すべてのレジスタ・リードはまず AORC に対して行われる．タグ・アレイを参照し，

**ヒットした場合**　続けてデータ・アレイにアクセスし，該当するレジスタの値を読み出す．

**ミスした場合**　ミスしたリード要求間で調停が行われ，ポートを獲得した順に，続くサイクルからレジスタ・ファイルへのアクセスを開始する．

なお，タグ・アレイの参照，および，調停部分の処理をまとめて**レジスタ・スケジューリング**と呼ぶ．

上述のように，レジスタ・ファイルは，AORC にミスした場合にのみアクセスする．そのため，レジスタ・ファイルのポートは通常よりも少なくてよい．

AORC を用いたプロセッサのバックエンド・パイプラインを**図 5** に示す．各ステージは，先頭から順に，発行（IS），レジスタ・スケジューリング（RS），レジスタ・ファイル（キャッシュ）・リード（RR/CR），実行（EX），レジスタ・キャッシュ・ライト（CW）を表す．回路面積が減ることで，レジスタ・ファイルのレイテンシはレジスタ・キャッシュのそれと同じ 1 サイクルになるものとしている．


**図 5**　回路面積指向レジスタ・キャッシュを用いたときのパイプライン
Fig. 5　Instruction pipeline on a processor with an area-oriented register cache.

ここで強調しておきたいことは，AORC のデータ・アレイとレジスタ・ファイルとが並列にアクセスされる点である．AORC にヒットした場合でもミスした場合でも，レジスタ・ファイルのポートが獲得できる限り，レジスタ・アクセスのレイテンシは変わらない．すなわち，ミスした場合のペナルティはない．

レジスタ・ファイルのポートを減らしたことで，AORC ミスが多数発生した場合にはポートが不足することもある．図 4 では，5 つ以上のリードが AORC ミスした場合にすべてのレジスタ・ファイル・アクセスを同時に処理できなくなる．そのような場合はバックエンドをストールさせて時分割でポートを使用する．

ストールの影響を緩和するため，レジスタ・ファイル・アクセスに初めから複数ステージを割り当てておくことも考えられる[⋆1]．レジスタ・ファイル・アクセスに 2 ステージを割り当て，1 サイクルでアクセス可能なレジスタ・ファイルのポートを時分割で使用する．このようにすれば，パイプラインは深くなるが，回路面積を増やさずにポート数を実質 2 倍にできる．

文献 21) によると，最大 0.5%の性能低下を許容すれば，ポートを 4 つにまで削減できる．その場合，レジスタ・ファイルの回路面積は $(4/12)^2 = 1/9$ となる．

## 4. キャッシュを用いる手法による RMT の回路面積削減

小容量のキャッシュを用いたレジスタ・ファイルの回路面積削減手法は，文献 21) において，RMT にも適用できる可能性があることが指摘されている．ただし，その具体的な方法までは述べてはいない．そこで我々は，キャッシュを用いる手法を実際に RMT に適用した．以下，その詳細について述べる．

⋆1 そのような場合は，タグ・アレイとデータ・アレイの間にラッチを挿入し，データ・アレイ側のレイテンシを調整する．

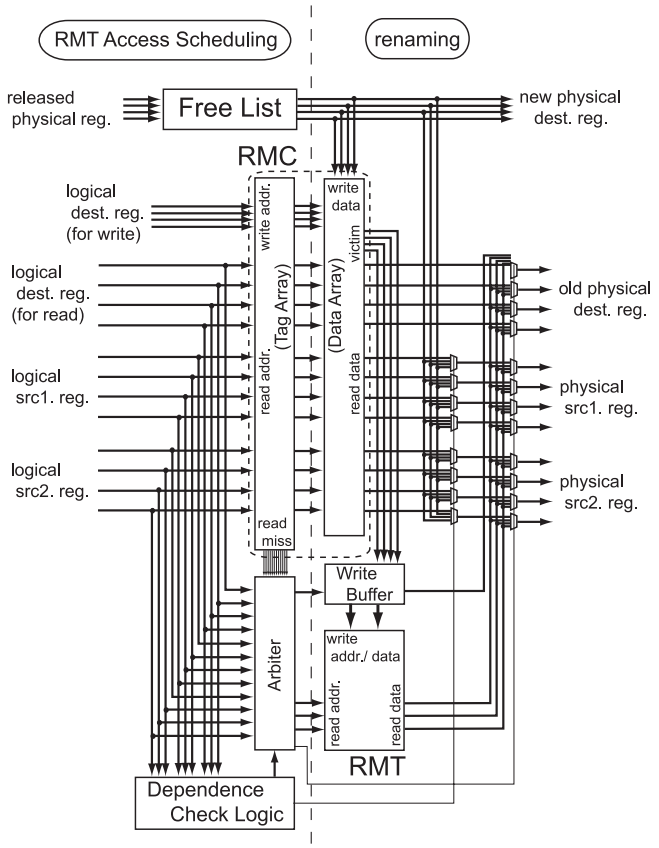


**図 6**　キャッシュを用いる手法を適用したリネーミング・ロジック
Fig. 6　Renaming logic to which an area reduction method using a cache is applied.

### 4.1 全体構成

キャッシュを用いる手法を適用した場合のリネーミング・ロジックを**図 6** に示す．レジスタ・ファイルに対してレジスタ・キャッシュとアービタを追加したのと同様，リネーミング・ロジックには**レジスタ・マップ・キャッシュ**（**Register Map Cache**．以下 **RMC** とする）とアービタが追加される．

RMC は RMT の一部を保持する，8 エントリ程度の小型のキャッシュである．置き換えは LRU とする．また，通常の RMT と同数のポートを持つ．デコード幅が 4 の場合は，リード・ポートが 12 個，ライト・ポートが 4 個である．

RMC，および，アービタの役割は，3.2 で述べた，AORC，および，アービタのレジスタ・ファイルに対する役割と同じである．すなわち，RMC は RMT のアクセス・フィルタとして機能し，アービタは RMC にミスしたリード要求を調停する．

RMC を加えることで，後述するように，アクセスの 8 割が RMC のデータ・アレイに集中する．RMT へのアクセスは全体の 2 割にまで低下するため，RMT のポートを大きく減らすことができる．図のように 5 ポートに減らした場合，RMT の回路面積は元の 1 割程度の大きさ（$(5/16)^2 = 0.098$）になる．

**レジスタ・ファイルに適用する場合との違い**

AORC とは異なり，RMC はライトバックで動作する．これは，AORC への書き込みと違って，RMC への書き込みはヒットしやすいためである．

この違いは，物理レジスタの使用頻度にはほとんど偏りがないのに対し，論理レジスタのそれには大きな偏りがあることに由来する．例外ハンドラ用のレジスタ（MIPS の 26, 27 番）はほとんど使用されない一方で，演算用のレジスタ（2，3 番）は使用頻度が高い．

紙面の都合により詳細は省略するが，RMC への書き込みがミスする確率はたかだか 1.5 回/サイクルである．この程度であれば，RMT のライト・ポートを 2 ポートにまで減らし，ライトバックで動作させた方がよい．

### 4.2 動　　作

基本的には，AORC の場合と同じである．RMT に対するすべてのリード要求はまず RMC に対して行われる．タグ・アレイを参照し，

**ヒットした場合**　続けてデータ・アレイにアクセスし，該当する物理レジスタ番号を読み出す．

**ミスした場合**　ミスしたリード要求間で調停が行われ，ポートを獲得した順に，続くサイクルから RMT へのアクセスを開始する．該当するデータがライト・バッファに存在するときは，ライト・バッファからフォワーディングする．

ライト要求とリード要求との間に真の依存関係があることもある．依存があった場合，リネーミング結果はフリー・リストから取得しなければならず，RMT にはアクセスする必要がない．そこで，タグ・アレイと並行して依存チェック・ロジックを参照し，そうした要求を除外する．

フリー・リスト，タグ・アレイ，依存チェック・ロジック，および，アービタの処理をま

とめて RMT アクセス・スケジューリングと呼ぶことにする．

RMC を用いたプロセッサのパイプライン構成は，機能的には，マルチバンク化した RMT を用いたプロセッサのそれ（図 3）とまったく同じである．RMT アクセス・スケジューリング・ステージにおいて RMT と RMC のどちらをアクセスするかをまず決定し，実際のアクセスは続くステージで行う．

マルチバンク化と同様，場合によっては RMT のポートが枯渇することがある．その場合の処理もマルチバンク化のときと同様である．デコード以前のステージをストールさせ，時分割でポートを使用する．

ストールの影響を緩和するため，AORC と同様，RMT のアクセスに初めから複数ステージを割り当てておくことも考えられる．すなわち，パイプラインが 1 段深くなるのと引き換えに，RMT のポートを実質倍にする．こうしたモデルの違いが性能に与える影響は次章で評価する．

### 4.3　チェックポイント

2.4 で述べたように，RMT は，分岐予測ミス時にチェックポイント回復が行われる．チェックポイント回復では，分岐命令が出現するたびにすべてのマッピングをバックアップしておく．このときバックアップするマッピングは，通常の RMT を用いた場合にバックアップされるそれとまったく同じ必要がある．

上述のように，RMC はライトバックで動作する．そのため，分岐が出現した際，論理レジスタと物理レジスタとの最新のマッピングが RMC にしか存在しないこともある．すなわち，RMT のバックアップを行うだけでは不十分である．RMT だけでバックアップを行ったとしても，そのバックアップは，そのチェックポイントにおける「正しいマッピング」ではない．

この問題は，最も単純には，チェックポイントにおいて RMC 上のマッピングもバックアップすることで解決できる．RMT と同様，RMC の各ビットにチェックポイント用のセルを組み込む．このようにすることで RMC の状態を高速に回復する．

## 5. 評　　価

キャッシュを用いた回路面積削減手法を RMT に適用し，性能に与える影響を評価した．また，回路面積についても評価した．以下詳しく述べる．

### 5.1　評価モデル

以下のモデルについて評価した．

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

| Parameter | Remarks |
|---|---|
| Way | 4 |
| Instruction Window | 64 entry |
| Register File | int：128, fp：128 |
| Execution Unit | int：3, fp：2, LD/ST：2 |
| L1 I-Cache | 32 KB, 4-way, 32B/line, 1 cycle |
| L1 D-Cache | 32 KB, 4-way, 32B/line, 2 cycle |
| L2 Cache（Unified） | 1 MB, 8-way, 32B/line, 10 cycle |
| Main Memory | 200 cycle |
| Branch Prediction | 8 KB g-share, penalty：14 cycle |
| BTB | 4 K entry, 4-way |
| RAS | 8 entry |

**BASE**　通常の RMT を持つプロセッサ

**BANK**　マルチバンク化した RMT を持つプロセッサ．

**CACHE**　キャッシュを用いる手法を適用した RMT を持つプロセッサ

**BANK** は，分割するバンク数，バンクごとのポート数によって，回路面積の削減率とバンク・コンフリクトの発生率が変化する．多くのバンクに分割し，バンクごとのポートを少なくすれば，面積は大幅に削減される一方，ポートの枯渇によってストールが発生する可能性が高くなる．バンク数とポートの削減量を抑えれば，面積削減量は減少するが，ストールの発生確率も抑えることができる．本稿では，ストールの影響をなるべく抑えるため，**BANK** のバンク数を 2 とし，各バンクのポートを 8，9，10 個と変化させた場合について評価した．

**CACHE** は，RMC のエントリ数が 4，8，16 の場合について評価した．マッピングはセットアソシアティブとし，way 数が 2，4 の場合を測定した．RMT のポート数は 3，4，5，6 と変化させた．それらのうちの 2 つはライト・ポートである．

プロセッサのパラメータを**表 1** に示す．表のパラメータは，Intel Core アーキテクチャに準じている．レジスタ・ファイルは int，fp 各 128 個，分岐予測ミス・ペナルティは 14 サイクルとした．また，最大 4 つまで，成立と予測された分岐を超えて命令フェッチする．

レジスタ・リネーミングには，基本的には，3 サイクル（アロケーションに 1 サイクル，RMT のアクセスに 2 サイクル）を要するものとした．ただし，後述するように，マルチバンク化，あるいは，キャッシュを用いる手法では，RMT の回路面積は **BASE** のそれの半分以下になる．そのため，**BANK**，**CACHE** では RMT のレイテンシを 1 サイクル減ら

し，基本的には 2 サイクルでリネーミングできるものとしている．

4.2 で述べたように，ポートを減らした RMT を用いる場合，ポートを時分割で使うことも考えられる．ポートを減らした RMT は 1 サイクルでアクセスできると考えられるが，時分割で使う場合は RMT のアクセスにあらかじめ 2 サイクルを割り当てておく．したがって，そのような場合はリネーミングに計 3 サイクルかかる計算である．

上述のモデルを SimpleScalar ツールセット（ver. 3.0）[2] 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．測定には，SPEC CINT2000 から 9 本，CFP2000 から 6 本のプログラムを使用した．入力セットには train を用いた．最初の 1 G 命令をスキップし，続く 256 M 命令を実行して評価した．

### 5.2 性能評価

**BASE** に対する **BANK**，**CACHE** の性能低下率を**図 7** に示す．3 つのグラフは，上から順に，RMC のエントリ数が 4，8，16 のときの結果である．way 数はいずれも 2 である．各グラフの縦軸は性能低下率，横軸はプログラムである．プログラムごとの 11 本の棒グラフは，左の 3 本が **BANK** を，残りの 8 本が **CACHE** を表している．**BANK** の 3 本のグラフは，左から順に，各バンクのポートを 8，9，10 個（うち 2 つはライト・ポート）にした場合を表している．**CACHE** の 8 本の棒グラフは，左半分（凡例の “TS” 無）がポートの時分割を行わなかった場合，右半分（“TS” 有）が行った場合である．それぞれ，左から右に向かって順に，RMT のポート数が 3，4，5，6 のときの結果である．

グラフより，ストールをなるべく起こさない構成にしたにもかかわらず，**BANK**（8R2W）は最悪 13.4%（平均 5.01%）の性能低下を引き起こしている．その一方で，**CACHE** は RMT に 5 ポートあればほとんど性能低下していない．特に RMC が 8 エントリで時分割を行わなかったときの性能低下率は，最悪でも 5.98%（平均 0.70%）にすぎない．

後述するように，8 エントリの RMC と 5 ポートの RMT をあわせた回路面積は **BANK**（8R2W）とほとんど同じである．したがって，**マルチバンク化よりキャッシュを用いる手法の方が効果が高い**といえる．

時分割を行った場合は，RMC を 4 エントリにし，RMT を 5 ポートにしてもよい．その場合の性能低下率は最悪 1.85%（平均 0.46%）である．このように，ポートを時分割で使用することで，性能を維持しつつ回路面積をさらに削減できる．

時分割を行わない場合，16 エントリ/6 ポートのモデルなど，いくつかのモデルで性能向上が見られる．これは，RMT の面積が削減されることでリネーミング・ステージが 1 サイクル減ると仮定しているが，このパイプライン短縮の効果が現れたためである．ただし，これらの RMT が実際に 1 サイクルでアクセスできるかどうかは検証の余地がある．

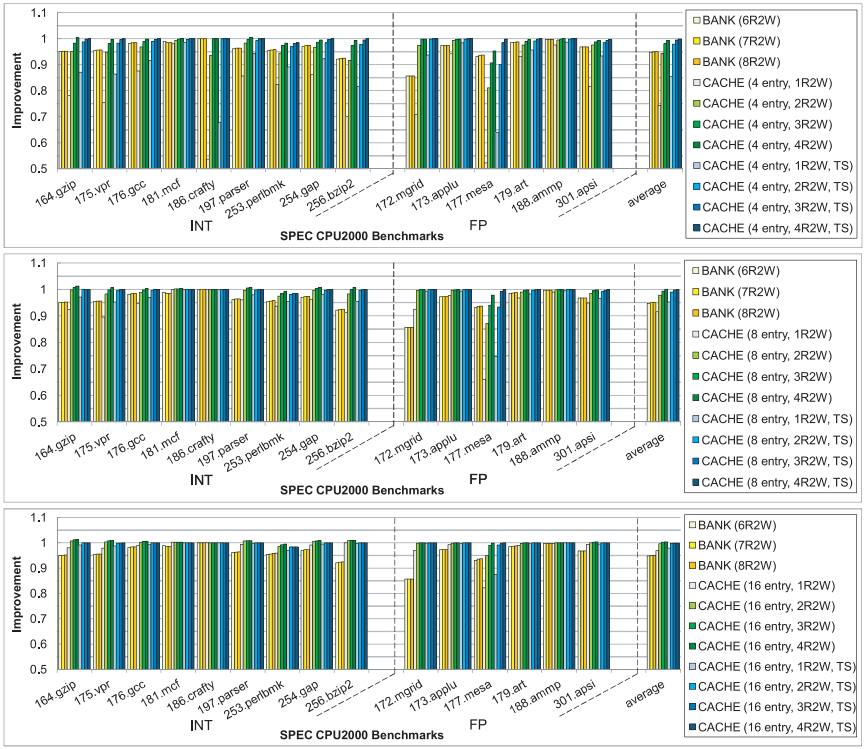


**図 7**　性能低下率（上：4 エントリ，中：8 エントリ，下：16 エントリ）
Fig. 7　IPC degradation (upper: 4 entry RMC, middle: 8 entry RMC, below: 16 entry RMC).

エントリ数を変化させた場合の RMC ヒット率を**図 8** に示す．縦軸がヒット率，横軸がプログラムである．3 本の棒グラフは，左から順に，エントリ数が 4，8，16 の場合である．way 数は，先ほどと同じく 2 である．

グラフより，INT 系のプログラムでは，4 エントリから 8 エントリに増やした場合にヒット率が大きく改善されるが，16 エントリに増やしてもあまり効果がない．一方，FP 系では，8 エントリのヒット率と 16 エントリのそれとの間に開きがある．ただし，図 7 より，RMT が 5 ポートもあれば性能差はほとんどない．時分割を行わない場合，RMC は 8 エン

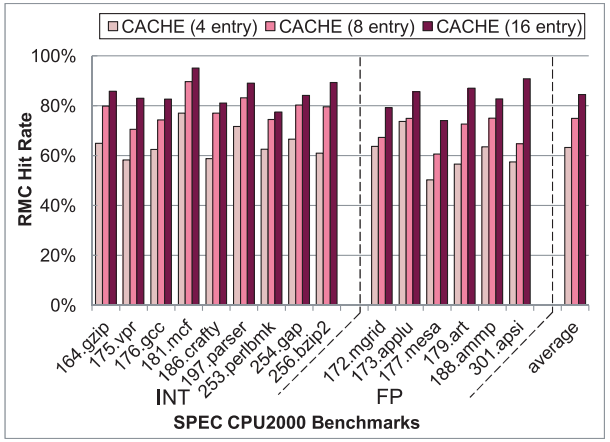


**図 8**　エントリ数を変えた場合の RMC ヒット率
Fig. 8　RMC hit rate in varying the number of entries.

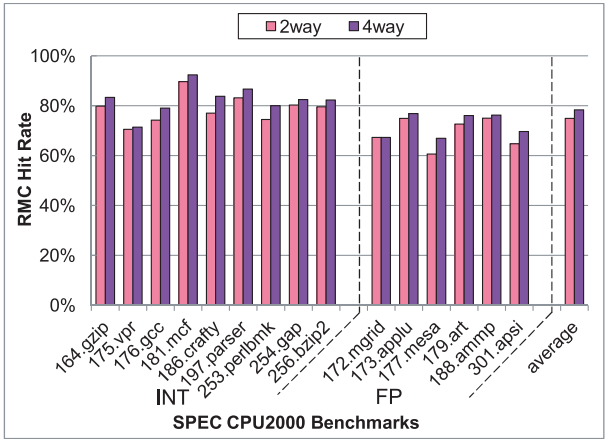


**図 9**　way 数を変えた場合の RMC ヒット率
Fig. 9　RMC hit rate in varying the number of ways.

トリもあれば十分である．

way 数を変えた場合の RMC ヒット率を**図 9** に示す．グラフの見方は先ほどと同様である．2 本の棒グラフは，左が 2 way の場合，右が 4 way の場合を表している．エントリ数は 8 とした．

グラフより，way 数を増やしてもヒット率はほとんど変わらない．したがって，2 way セットアソシアティブ・キャッシュでよい．

### 5.3　回路面積の評価方法

各モデルの回路面積，および，遅延を評価した．評価には，CACTI 5.3[17] を使用した．

CACTI 5 はそれ以前のバージョンにいくつかの改良が加えられているが，最も大きな違いはモデルが変更された点である．それまでの CACTI は，0.8 $\mu$m テクノロジを想定したモデルを入力パラメータに応じて単純にスケーリングしていたため，ディープ・サブミクロン世代のメモリに対してはあまり正確な見積りができなかった．それが，CACTI 5 からは，ITRS のロードマップ[6] に基づいたモデルに変更されたことで，ディープ・サブミクロン世代のメモリにも対応できるようになっている．本評価では 32 nm テクノロジを仮定した．

マルチバンク化された RMT のクロスバ・スイッチは，通常のマルチバンク RAM のそれとは構成が若干異なる．ライト・ポート，および，解放用のリード・ポートに対するクロスバ・スイッチは，図 2 に示したように，ソースのリネーミング用のクロスバ・スイッチと分離できる．こうすれば，単純に 16 入力 8 出力のクロスバ・スイッチを用いた場合よりも面積を削減できる．クロスバ・スイッチの与える影響が **BANK** に不利に働かないようにするため，評価に用いた CACTI にはこの修正を加えてある．

なお，以下で述べる評価には，**CACHE** に必要な一部の回路——アービタとライト・バッファ——を含んでいない．これは，CACTI には，メモリ周辺の回路，および，数エントリのバッファを評価する機能がないことによる．これらの回路を含んでいないことは **CACHE** に有利に働くが，以下で述べるように，その影響は軽微である．

まずアービタであるが，その回路規模はクロスバ・スイッチに比べて十分小さい．8 入力 4 出力のクロスバ・スイッチは，4 つの 8 入力マルチプレクサからなる．RMT の 1 エントリは 8b（表 1）であるから，ソース用のクロスバ・スイッチ 1 つをとっても，32 個の 8 入力マルチプレクサを必要とする．この回路規模は，12 入力 5 出力のアービタを単純にカスケード方式[13] で実現した場合のそれを優に超える．後述するように，図 2 のクロスバ・スイッチ全体でさえ，その規模は SRAM 本体と比べれば小さなものである．

また，ライト・バッファの規模も十分小さい．前述のように，ライト・バッファへの書き込み頻度はたかだか 1.5 回/サイクルである．そのため，RMT のライト・ポートを 2 つ用意す

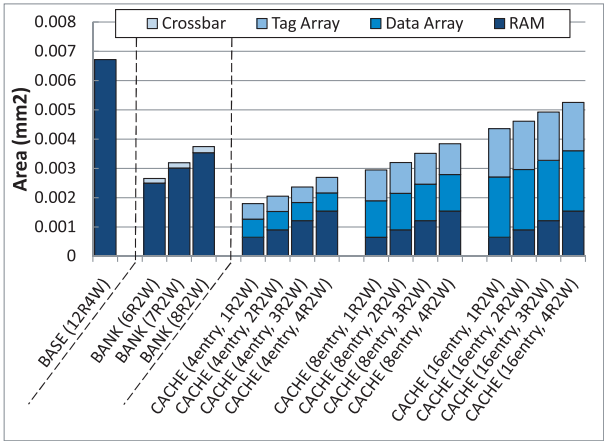


**図 10**　各モデルの回路面積
Fig. 10　Area of each model.

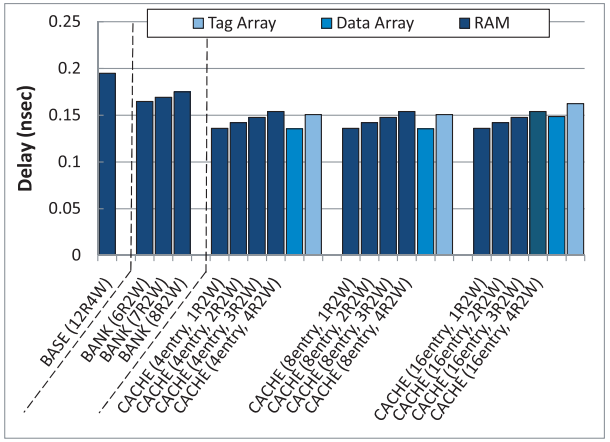


**図 11**　各モデルの遅延
Fig. 11　Delay of each model.

れば，ライト・バッファに書き込み要求が貯まることはほとんどない．試しに 4 エントリ，4 リード，4 ライトのキャッシュの面積を CACTI で測ったところ，その面積は $3.74*10^{-4}\,\mathrm{mm}^2$ であった．これは，4 エントリ RMC の 1/3 の規模でしかない．実際には，ライト・バッファは 2 エントリ程度で十分と考えられるため，それが **CACHE** 全体に占める割合はごくわずかといえる．

### 5.4　回路面積の評価結果

評価結果を**図 10** に示す．グラフの縦軸は面積，横軸はモデルである．14 本の棒グラフは，一番左が **BASE** を，次の 3 つが **BANK** を，残りが **CACHE** に対応する．各棒グラフは 4 色に塗りわけられており，色の濃い方から順に，RAM，データ・アレイ（**CACHE** のみ），タグ・アレイ（**CACHE** のみ），クロスバ・スイッチ（**BANK** のみ）を表している．

グラフより，**CACHE**（8 エントリ，3R2W）の面積は **BANK**（8R2W）のそれとほとんど同じである．前者は **BASE** の面積を 47.7%削減するのに対し，後者は 44.3%面積を削減する．前々節で述べたように，前者の方が性能低下率が小さいことから，**CACHE** の方が面積を削減する効果が高いといえる．

前述のように，時分割でポートを使用した場合は，**CACHE** を 4 エントリにまで減らすことができる．その場合（3R2W）の面積削減率は 64.8%に達する．

また，クロスバ・スイッチの規模は SRAM 本体と比べて十分小さい．その規模は，**BANK**（6R2W）の場合でさえ，SRAM 本体の 5.92%である．このように，メモリ以外の部分の面積は小さい．

RMT の遅延を**図 11** に示す．グラフの縦軸は遅延，横軸はモデルである．**CACHE** に関しては RMC の遅延も並べて表示してある．

グラフより，RMT のポート数を 16（**BASE**）から 5（3R2W）に減らした場合，遅延は 24.3%削減される．また，RMT の遅延と RMC のそれとを比較すると分かるように，RMC が 4 エントリ，あるいは，8 エントリの場合，クリティカル・パスはデータ・アレイではなく RMT である．したがって，上記の恩恵をそのまま受けることができる．

## 6. 関 連 研 究

Liu ら[9] は，RMT のレイテンシを短縮するため，RMT に小容量のキャッシュを追加する手法を提案している．Liu らの方法では，通常のレジスタ・キャッシュにヒットした場合にレジスタ・アクセス・レイテンシが短縮されるのと同様，キャッシュにヒットした場合に RMT のアクセス・レイテンシが短縮される．一方，キャッシュにミスした場合には，フロントエンドをストールさせ，RMT 本体に改めてアクセスする，というペナルティが発生す

る．本稿で述べた手法は，AORC と同様，RMC のデータ・アレイと RMT を並列にアクセスする．したがって，RMC にミスしても上記のペナルティはない．

RMT の消費電力を削減するため，小容量のキャッシュを追加した研究もある．Kucuk ら[8]の手法では，キャッシュのタグ・アレイのアクセスと RMT のアクセスを並列に開始する．キャッシュの方が容量が小さいため，タグ・マッチングの結果は，RMT の読み出しよりも早期に完了する．そこで，キャッシュ・ヒットした場合に該当する RMT の読み出しを中止することにより，動的消費電力を削減する．本稿で述べた手法は，まずタグ・アレイを参照し，ミスした場合に RMT を参照する．ミスした場合にのみ RMT を参照するため，RMT のポートを減らすことができ，回路面積が削減される．回路面積の削減は，動的消費電力だけでなく静的消費電力の削減にもつながる．

4.2 で述べたように，フリー・リストからリネーミング結果を供給できるソース・オペランドについては，RMT を参照する必要がない．そこで，そのような工夫を RMT に施すことで，RMT のポートを削減する手法が提案されている[12]．しかし，同時にリネーミングされる命令間に真の依存関係があるケースはそれほど多くないため，この方法ではポートをあまり削減できない．ポートを半分にまで減らすと，最悪の場合，5% 以上も性能が低下してしまう．前章で述べたように，本稿で述べた手法はポートを 1/3 にしても平均 0.7%の性能低下で済む．

## 7. ま と め

本稿では，レジスタ・ファイルの回路面積を削減する手法である，キャッシュを用いる手法を RMT に適用した．マルチバンク化と比較した結果，キャッシュを用いる手法の方が削減効果が高いことが分かった．

**謝辞**　本研究の一部は文部科学省共生情報工学推進経費による．

## 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. 34th International Symposium on Microarchitecture*, pp.237–248 (2001).
2) Burger, D., Austin, T.M. and Bennett, S.: Evaluating Future Microprocessors: The SimpleScalar ToolSet, Technical ReportCS-TR-1308, Univ. of Wisconsin-Madison (1996).
3) Butts, J.A. and Sohi, G.S.: Use-Based Register Caching with Decoupled Indexing, *Proc. 31st International Symposium on Computer Architecture*, pp.302–313 (2004).
4) Cruz, J.-L., González, A., Valero, M. and Topham, N.P.: Multiple-Banked Register File Architectures, *Proc. 27th International Symposium on Computer Architecture*, pp.316–325 (2000).
5) Hinton, G., Sager, D., Upton, M., Boggs, D., Carmean, D., Kyker, A. and Roussel, P.: The Microarchitecture of the Pentium 4 Processor, *Intel Technology Journal*, Issue 1, Vol.5 (2001).
6) ITRS: International Technology Roadmap for Semiconductors 2005 Edition.
7) Keller, J.: The 21264: A Superscalar Alpha Processor with Out-of-Order Execution, *9th Annual Microprocessor Forum* (1996).
8) Kucuk, G., Ergin, O., Ponomarev, D. and Ghose, K.: Reducing Power Dissipation of Register Alias Tables in High-Performance Processors, *Proc. Computers and Digital Techniques*, pp.739–746 (2005).
9) Liu, T. and Lu, S.-L.: Performance Improvement with Circuit-Level Speculation, *Proc. 33rd International Symposium and Workshop on Microarchitecture*, pp.348–355 (2000).
10) Manne, S., Klauser, A. and Grunwald, D.: Pipeline Gating: Speculation Control For Energy Reduction, *Proc. 25th Annual Int'l Symp. on Computer Architecture*, pp.132–141 (1998).
11) Marr, D.T., Binns, F., Hill, D.L., Hinton, G., Koufaty, D.A., Miller, J.A. and Upton, M.: Hyper-Threading Technology Architecture and Microarchitecture, *Intel Technology Journal*, Issue 1, Vol.6 (2002).
12) Moshovos, A.: Power-Aware Register Renaming, Technical report, University of Tronto (2002).
13) Palacharla, S., Jouppi, N.P. and Smith, J.E.: Quantifying the Complexity of Superscalar Processors, Technical report, University of Wisconsin-Madison (1996).
14) Preston, R.P., Badeau, R.W., Bailey, D.W., Bell, S.L., Biro, L.L., Bowhill, W.J., Dever, D.E., Felix, S., Gammack, R., Germini, V., Gowan, M.K., Gronowski, P., Jackson, D.B., Mehta, S., Morton, S.V., Pickholtz, J.D., Reilly, M.H. and Smith, M.J.: Design of an 8-wide Superscalar RISC Microprocessor with Simultaneous Multithreading, *Proc. 2002 IEEE Int'l Solid-State Circuit Conference Digest and Visuals Supplement* (2002).
15) Safi, E., Akl, P., Moshovos, A., Veneris, A. and Arapoyianni, A.: On the Latency, Energy and Area of Checkpointed, Superscalar Register Alias Tables, *Proc. Int'l Sympo. on Low Power Electronics and Design*, pp.379–382 (2008).
16) Tatsumi, Y. and Mattausch, H.J.: Fast Quadratic Increase of Multiport-Storage-Cell Area with Port Number, *Electronics Letters*, Vol.35, No.25, pp.2185–2187 (1999).

17) Thoziyoor, S., Muralimanohar, N., Ahn, J.H. and Jouppi, N.P.: CACTI 5.1, HPL technical report hpl-2008-20, HP Labs (2008).
18) Tseng, J.H. and Asanovic, K.: Banked Multiported Register Files for High-Frequency Superscalar Microprocessors, *Proc. 30th Annual Int'l Symp. on Computer Architecture*, pp.62–71 (2003).
19) 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. 23rd International Symposium on Computer Architecture*, pp.191–202 (1996).
20) Yeager, K.C.: The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, Vol.16, No.2, pp.28–40 (1996).
21) 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，pp.229–236 (2008).
22) 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・キャッシュの評価，情報処理学会研究報告，vol.2008-ARC-178, pp.13–18 (2008).
23) 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，pp.33–40 (2006).
24) 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，pp.13–22 (2006).

(平成 22 年 1 月 26 日受付)
(平成 22 年 5 月 17 日採録)

**三輪　忍**（正会員）

1977 年生．2000 年京都大学工学部情報学科卒業．2002 年同大学大学院情報学研究科通信情報システム専攻修士課程修了．2005 年同大学院情報学研究科通信情報システム専攻博士後期課程学習認定退学．同年京都大学法学研究科助手．2008 年 1 月より東京農工大学工学府特任助教，現在に至る．博士（情報学）．計算機アーキテクチャ，並列処理，ニューラルネットの研究に従事．電子情報通信学会，人工知能学会各会員．

**張　鵬**（学生会員）

1980 年生．2002 年中国寧夏大学物理与電気通信工程系卒業．2002 年中国寧夏大学計算機工程卒業．2010 年東京農工大学工学府情報工学専攻工学専攻博士前期課程修了．2010 年 4 月富士ソフト株式会社入社．マイクロ・アーキテクチャに興味を持つ．

**横山　弘基**（学生会員）

1987 年生．2008 年沼津工業高等専門学校電気電子工学科卒業．2010 年 3 月東京農工大学工学部情報工学科卒業．同年 4 月同大学大学院工学府情報工学専攻博士前期課程入学．マイクロアーキテクチャに興味を持つ．

**堀部　悠平**（学生会員）

1986 年生．2007 年 4 月東京工業高等専門学校電気電子工学科卒業．2009 年 3 月東京農工大学工学部情報工学科卒業．同年 4 月同大学大学院工学府情報工学専攻博士前期課程入学．マイクロアーキテクチャに興味を持つ．

**中條　拓伯**（正会員）

1961 年生．1985 年神戸大学工学部電気工学科卒業．1987 年同大学大学院工学研究科修了．1989 年同大学工学部助手の後，1998 年より 1 年間 Illinois 大学 Urbana-Champaign 校 Center for Supercomputing Research and Development（CSRD）にて Visiting Research Assistant Professor を経て，現在，東京農工大学大学院工学研究院准教授．プロセッサアーキテクチャ，並列処理，クラスタコンピューティング，リコンフィギャラブルコンピューティングに関する研究に従事．電子情報通信学会，IEEE CS，ACM 各会員．博士（工学）．