ビット分割構成による レジスタファイルのサイズおよびポート数削減手法

近藤正章[†] 中村宏[†]

動的命令スケジューリング，多命令同時発行を行うプロセッサにおいては，大容量・多ポートのレジスタファイルが必要不可欠である．近年では，さらなる高性能化のためにレジスタサイズやポート数が増加傾向にある．しかし，これはレジスタアクセス時間や消費電力の増大といった問題を引き起こす．そこで，本論文ではレジスタファイルサイズ，およびポート数の削減を目的として，ビット分割レジスタファイルを提案する．提案するレジスタ構成を用いることで，小容量かつ少ないポート数であっても高性能が達成できる．本論文は，ビット分割レジスタファイルのためのマイクロアーキテクチャの拡張について述べ，性能および消費エネルギーの評価を行う．評価結果より，提案するレジスタファイルを用いることで，従来のプロセッサに比べ，同程度の IPC を達成しつつ，サイズおよびポート数を大きく削減できることが分かった．

Reducing Register Port and Size Requirements by Bit-partitioning

Masaaki Kondo[†] and Hiroshi Nakamura[†]

A large multi-ported register file is indispensable for exploiting instruction level parallelism in recent dynamically scheduled superscalar processors. However, such a large register file causes problems of long access delay and huge power consumption. To tackle these problems, in this paper, we propose a *Bit-Partitioned Register File* to reduce required register file size and the number of ports. We show the basic idea and mechanism of proposed register file and evaluation results on performance and power consumption. Evaluation results reveal that the proposed register file achieves higher IPC even in small register file with less number of ports.

1. はじめに

動的命令スケジューリング・多命令同時発行を行うマイクロプロセッサでは，命令レベル並列性（ILP）を活用するために，大容量かつ多ポートのレジスタファイルが必須である．近年では，さらなる ILP 活用のために，命令ウィンドウサイズや同時発行命令数を増加させる傾向にあるが，それにともないレジスタファイルのサイズ・ポート数も増加している．しかし，このレジスタファイルの大容量・多ポート化は，アクセス時間や消費電力の増大という問題を引き起こす．

この問題への対処を目的として，アクセスすべきレジスタのサイズ・ポート数を削減するための手法がこれまでにも多く提案されている．たとえば，要求されるサイズ削減のために，物理レジスタファイルの割当てを遅らせる手法[1),2)] や，同じ値を持つオペランドを 1 つのレジスタエントリで共有化する手法[3),4)]，階層化レジスタファイル[5),6)] などが提案されている．また，ポート数を削減するものとしては，マルチバンク化[5),7)] などがよく知られる手法である．

我々は，必要とするレジスタファイルサイズを削減することを目的として，ビット分割レジスタファイル（*Bit-Partitioned Register file: BPRF*）と呼ぶレジスタファイル構成を提案している[8),9)]．BPRF は，多くのレジスタオペランドがデータパスの全ビット幅分を必要としていないという事実に基づき，レジスタファイルをビット方向に分割し，ビット幅は小さくなるがエントリ数を増やすことで，レジスタの記憶領域の有効活用を狙うものである．

従来のプロセッサでは，有効ビット幅が小さなデータに対しても，フルビット幅のレジスタエントリを割り当てるため，それらのエントリの上位ビット部分は無駄に使われていた．一方，BPRF では有効ビット幅が分割後のレジスタのビット幅よりも小さなオペランドに対しては 1 エントリのみを，また有効ビット幅が大きなオペランドに対しては複数エントリを割り当てることでデータの記憶を行う．これにより，有効ビッ

† 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology, The University of Tokyo

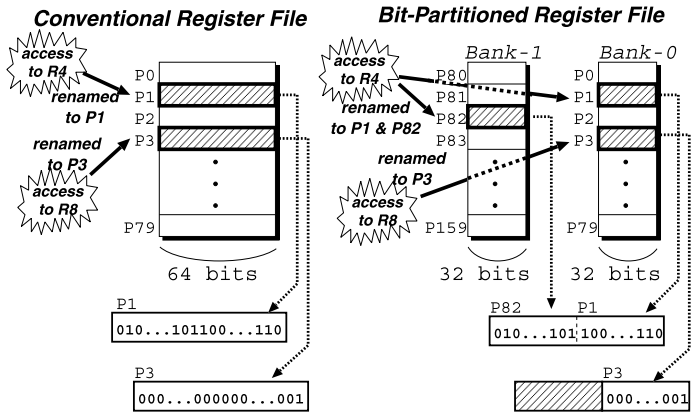

図 1 BPRF の概要
Fig. 1 An overview of BPRF.

ト幅の小さいデータが多い場合，従来構成のレジスタファイルに比べて同容量でもより多くのオペランドを保存することが可能となり，高い IPC 性能を達成できる．また逆に，同じ IPC 性能を得るためのレジスタ容量を削減することができるとも考えられるため，結果としてレジスタアクセス時間や消費電力の削減効果も期待できる．

以前の BPRF 手法ではレジスタファイルサイズは削減できるが，必要なポート数は従来のレジスタファイルと同様であった．本論文では BPRF を拡張し，サイズだけでなく，ポート数も同時に削減するための手法，およびマイクロアーキテクチャを提案する．

本論文の構成は以下のとおりである．次章において BPRF のアイデアについて述べ，3 章で BPRF を実現するためのマイクロアーキテクチャの拡張を示す．4 章では性能評価環境，および評価条件について説明し，5 章で評価結果を示す．6 章で関連研究を述べ，7 章でまとめと今後の課題について述べる．

2. ビット分割レジスタファイル

図 1 に BPRF の概要を示す．図は，64 ビットデータ幅，80 エントリの従来のレジスタファイルを，ビット方向に 2 分割した場合を示している．本論文では，分割されたレジスタファイルの各ブロックをバンクと呼び，オペランドの 64 ビットワードを 2 つに分割した場合の各部分をサブワードと呼ぶ．ここで，ビット方向に分割しても，分割後の各バンクのエントリ数は 80 で変わらず，両バンク合計で 160 エントリとなる（図では説明のために，エントリの番号を P0 から P159 までの通し番号で表している）．したがって，従来のレジスタに比べ，同じ記憶容量のもとでエントリ数が増えることになる．

BPRF では，オペランドの有効ビット幅に合わせて必要なエントリだけを割り当てることで，レジスタファイルの記憶領域を有効に利用できる．一方で，64 ビットデータも複数エントリを用いて記憶できるため，プロセッサの論理的な動作には影響を与えない．

BPRF の基本的なアイデアを，図 1 のレジスタアクセスの様子を用いて述べる．図において，アーキテクチャレジスタ R4 のオペランドの値は有効ビット幅が大きく，上位および下位サブワードともに有効な値を持ち，R8 のオペランドの値は有効ビット幅が小さく，下位サブワードのみに有効な値を持つものとする．従来のレジスタファイルでは，アーキテクチャレジスタ R4 が物理レジスタ P1 に割り当てられ，R8 が P3 に割り当てられている．ここで，R8 のオペランドは有効ビット幅が小さく，上位サブワードがすべて 0 であるにもかかわらず，64 ビット幅のエントリが割り当てられる．そのため，R4 と R8 の両オペランドを保存するためには，128 ビット分の記憶領域が必要となる．一方，BPRF では，アーキテクチャレジスタ R4 は物理レジスタバンク 0 の P1，およびバンク 1 の P82 に割り当てられ，R8 はバンク 0 の P3 のみに割り当てられている．したがって，両オペランドを記憶するためのレジスタ記憶領域は 96 ビットとなる．このように，BPRF では同じオペランドを保存する場合の必要とされるレジスタ記憶領域を小さくすることができる．なお，上位ビットがすべて “1” である場合も有効ビット幅が小さいと考えることができるが，本論文では以降，“0” の場合のみを対象とする．

さらに，上記の例において，アーキテクチャレジスタ R8 をアクセスする場合，バンク 1 をアクセスする必要はない．したがって，有効ビット幅の小さなオペランドが多い場合，必要なレジスタファイルの入出力

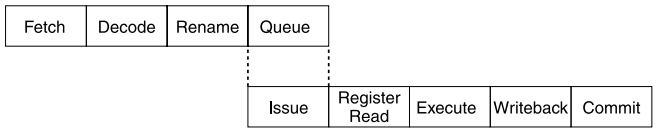

図 3 従来型プロセッサのパイプライン
Fig. 3 Pipeline of a conventional processor.

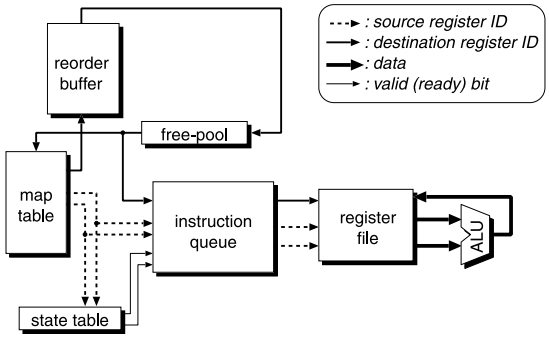

図 2 従来型プロセッサのブロック図
Fig. 2 Block diagram of a conventional processor.

バンド幅を小さく抑えることが可能となる．この事実は，BPRF では各バンクのレジスタファイルのポート数をある程度削減しても十分な性能を得ることができることを意味する．

ここで，BPRF においてバンク 0 とバンク 1 にオペランドをバランス良く割り当てるために，各演算命令，あるいはロード命令のデスティネーションレジスタごとに，上位サブワードと下位サブワードに割り当てるバンクを変更する．この目的で，*Least Significant Bank Pointer: LSBP* と呼ぶフラグを新たに導入する．LSBP は，オペランドごとに，どのバンクがオペランドの下位サブワードに割り当てられているかを示すものである．

3. マイクロアーキテクチャ

本章では，BPRF のアイデアを実装するためのマイクロアーキテクチャを提案する．以前の BPRF のマイクロアーキテクチャ[8),9)] では，レジスタアクセスと同時にオペランドの有効ビット幅チェックを行い，アクセス後にオペランドの再構築をしていたため，実際にはアクセスする必要のなかったバンクに対しても投機的にアクセスしなければならず，必要とされるポート数は従来のレジスタファイルと同様であった．本章では，その点を改良し，ポート数も削減可能なマイクロアーキテクチャの実装について述べる．なお，前提とするマイクロアーキテクチャとしては Alpha21264[10)] や MIPS R10000[11)] の構成をベースとし，物理レジスタ構成方式は *merged architectural and rename register file* 方式[12)] である．また，レジスタリネーミングの機構としては，RAM ベースのリネーミング機構を仮定する．

3.1 従来型プロセッサ

3.1.1 パイプラインおよび各機構

図 2，**図 3** に，従来型プロセッサのブロック図，およびパイプラインを示す．以下，このプロセッサの動作について簡単に説明する．

まず，*Fetch*，*Decode* された命令のソースレジスタ ID は，*Rename* ステージにおいて，*map-table* を参照することで物理レジスタ ID に変換される．また，それと同時にその命令のデスティネーションレジスタとして新しい物理レジスタエントリが *free-pool* から割り当てられる．free-pool から取得された物理レジスタ ID は，デスティネーションレジスタ ID に対応する map-table エントリに登録される．

また，各命令の *Commit* を命令順序どおりに行うため，Rename ステージではすべての命令の情報が *reorder-buffer* に命令順序どおりに記憶される．この際，命令のデスティネーションレジスタ ID と，それ以前にその ID に割り当てられていた物理レジスタ ID が reorder-buffer に登録される．命令が Commit される際には，reorder-buffer の該当エントリに登録されていた物理レジスタエントリが解放され free-pool に戻される．

各命令は，Rename ステージで得られたソース・デスティネーション物理レジスタ番号とともに，*Queue* ステージで命令キュー（*instruction queue*）に登録される．この際，レジスタエントリの状態が保持されている *state-table* がアクセスされ，その時点でのオペランドのアクセス可否の情報も命令キューに登録される．

キューに登録された命令は，すべてのオペランドが揃い次第，発行（*Issue*）される．発行された命令は，レジスタの読み込み（*Register Read*）が行われ，ALU において演算が実行される（*Execute*）．演算結果は *Writeback* ステージでレジスタに書き込まれる．

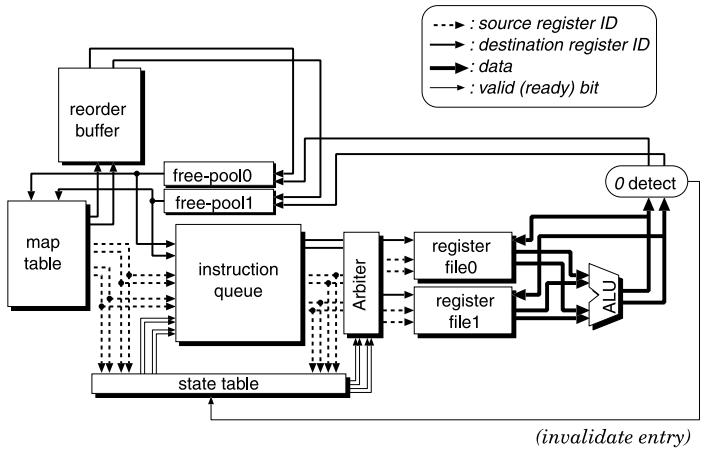

図 4 BPRF 用の拡張を行ったブロック図
Fig. 4 Block diagram with BPRF extension.

図 5 物理レジスタ番号フィールドの拡張
Fig. 5 Extension for physical register ID field.

3.2 BPRF の拡張

3.2.1 概　　要

図 4 に，BPRF の拡張を行った場合のプロセッサのブロック図を示す．図は，従来のレジスタファイルを 2 バンクに分割した場合のブロック図である．以降では 2 バンク構成の場合を前提として説明する．

本拡張では，Rename ステージにおいて，各命令のデスティネーションレジスタに対し，2 バンク分の物理レジスタエントリを割り当てる．これは，Rename ステージでは，その命令のオペランドの有効ビット幅が分からないためである．map-table や命令キューなど，物理レジスタ番号を保存するためのフィールドは，**図 5** のように 2 バンク分の ID が保存できるように拡張される．$PRegID_n$ は，それぞれバンク n 用の物理レジスタ ID である．また，どちらのバンクが下位ビットであるかを示す LSBP も追加される．さらに，free-pool もバンクごとに設けられる．上位/下位サブワードへのバンクの割当ては，ラウンドロビン方式で行うものとする．

0-detect 機構は，演算結果の有効ビット幅を判定するためのものであり，各サブワードごとにそのビットがすべて 0 かどうかを判定する．すべて 0 であった場合はそのサブワードに割り当てられていた物理レジスタエントリを *Writeback* ステージで解放し，その情報を state-table に登録する．

一般的に物理レジスタエントリの解放は，その物理レジスタと結びつけられたアーキテクチャレジスタ ID をデスティネーションレジスタに持つ次の命令が Commit した時点で行われる．ここで，本拡張のように物理レジスタのエントリを早い段階で解放することを *Early Register Deallocation*（*ERD*）と呼ぶ．ERD によりそのレジスタエントリを別の命令のデスティネーションレジスタとして用いることができるため，レジスタの使用効率が向上し，結果として小容量のレジスタサイズでも高性能を達成することができる．この ERD に関するマイクロアーキテクチャ的な拡張については 3.2.3 項で述べる．

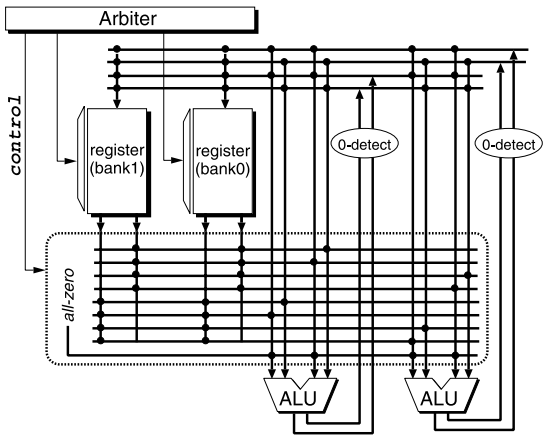

図 6 レジスタから ALU へのデータパス
Fig. 6 Data-path from Register to ALU.

3.2.2 レジスタアクセス

図 6 にレジスタから ALU へのデータパスを示す．64-bit プロセッサの場合，図中の各データ線は 32 ビット幅となる．

2 章で述べたように，BPRF では読み出すオペランドが 64 ビット幅である場合，バンク 0 とバンク 1

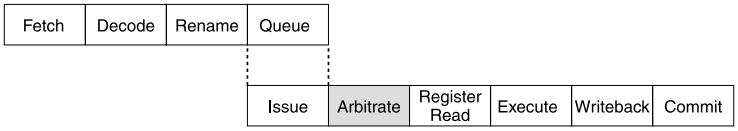

図 7 BPRF のパイプライン
Fig. 7 Pipeline with BPRF extension.

の両方からデータを読み出すが，32 ビット幅の場合，片方のバンクからのみデータの読み出しを行う．その際に，他方のバンクのポートを，他のオペランドの読み出しのために用いることで，ポートの使用効率が向上する．このために，図 4 や 図 6 に示す *Arbiter* 機構を設け，各オペランドがどのポートを使用してレジスタをアクセスすべきかの調停を行う．なお，図 3 に示す従来のパイプラインでは，Issue ステージの直後にレジスタの読み込み（Register Read）が行われるが，Issue ステージはクリティカルパスになりやすく，Issue ステージでポートの調停を行ったうえで命令を発行するのは現実的でない．そこで，BPRF ではパイプラインステージを拡張し，**図 7** に示すようにポートの調停を行う *Arbitrate* ステージを追加する．

Arbitrate ステージでは，state-table を参照し，アクセスすべきエントリが ERD により解放されているかどうかのチェックを行い，レジスタアクセスの際のポートの割当て制御を行う．それと同時に，レジスタ・ALU 間にあるスイッチのためのコントロール信号を生成する．読み出しオペランドのエントリが解放されていた場合，そのエントリの読み出しは行わずに，すべて 0 の値（図 6 中の *all-zero* のライン）を選択するように制御を行う．さら LSBP に依存した上位/下位サブワードの入れ替えも行われる．

なお，Arbitrate ステージは，マルチバンク化レジスタファイルの実装においても必要であり[7]，BPRF におけるポート調停や，競合が生じた場合の制御などはマルチバンクのレジスタ制御の延長として実装可能であると考えられる．また，レジスタ・ALU 間のオペランドのスイッチング制御については，上位/下位サブワードの入替え，および all-zero を選択するためのスイッチングが追加で必要となるが，その他は文献 7) と同様の回路で実現可能である．

3.2.3 Early Register Deallocation

これまでにも，ERD を用いたレジスタ使用効率の改善手法に関する提案が行われている[4),13),14)]．本項では，BPRF における ERD 実装方法について述べる．

ERD によりレジスタのエントリを Commit 時より早い段階で解放するためには，map-table，reorder-buffer，命令キュー，state-table の当該レジスタエントリを無効化し，その上で free-pool に解放する物理レジスタ ID を書き込む必要がある．しかし，それらの各機構に登録されているレジスタエントリを無効化するのは簡単ではなく，ハードウェアの複雑化や消費電力増大といった問題が生じる．そこで，本論文では実際にはそれらの機構のレジスタ情報を無効化せずに，低コストで ERD を実現する方法を提案する．

まず，低コストの ERD 実装のために，サブワードがすべて 0 であった場合に ERD によって解放できるエントリを，「最も下位のサブワード（すなわちオペランドをビット方向に分割した際の最下位ビットを含むワード）以外のバンクに割り当てられたエントリ」に制限する．ここで，オペランドの最も下位のサブワードに割り当てられたエントリを *master-entry*，それ以外の上位サブワードに割り当てられているエントリを *slave-entry* と呼ぶことにする．master-entry は，そのサブワードがすべて 0 であっても解放されることはないため，あるオペランドの生存期間中は master-entry は必ず有効であることが保証される．そこで，slave-entry が ERD により解放されたかどうかを，master-entry に付随する状態として管理することを考える．

従来のプロセッサでは，物理レジスタの各エントリの状態を管理するために，その各エントリに対して，**図 8** に示す 4 つの状態が定義されており，state-table に記録されている[12)]．これに対し，ERD のために状態遷移を **図 9** のように拡張する．また，slave-entry が有効であるか（すなわち ERD により解放されたかどうか）の状態を示す *pair-bit* と呼ぶ状態ビットを追加する．pair-bit は，そのエントリが master-entry の場合にのみ意味を持つ．なお，pair-bit のビット幅は，Rename ステージにおいて 1 つのオペランドに割り当てられる slave-entry の数と同じ，すなわち “合計バンク数 −1” となる．このビットは，該当する slave-entry が有効であれば（解放されていない場合）0，無効化された（解放された場合）は 1 をセットする．

図 9 の状態遷移について簡単に説明する．*Free* である，物理レジスタのエントリが，ある命令のデスティ

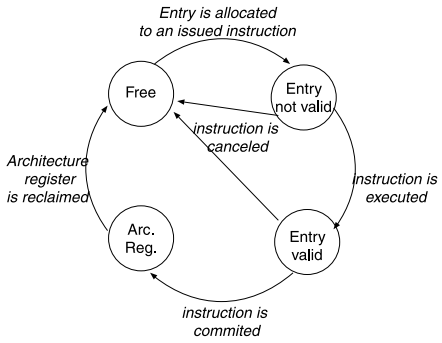

図 8 従来のプロセッサの状態遷移
Fig. 8 State transition diagram of a conventional processor.

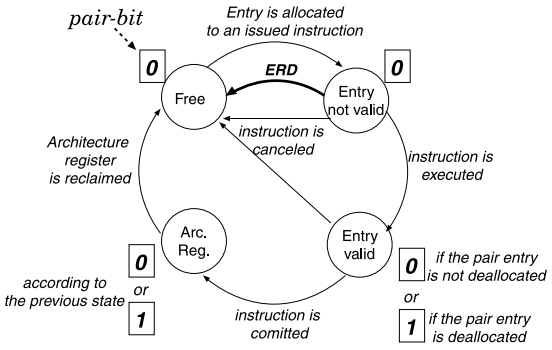

図 9 ERD 用の拡張を行った状態遷移
Fig. 9 State transition diagram with ERD extension.

ネーションレジスタとして割り当てられた際に，その状態は *Entry-not-valid* に移行する．同時に master-entry の pair-bit は 0 に初期化される．命令が実行され，もし slave-entry が ERD により解放された場合，その slave-entry 自信の状態は Free となる．一方，master-entry はこの時点で解放されることはないため状態は *Entry-valid* に移行する．ここで，slave-entry が解放されたことを示すために，該当する pair-bit ビットが 1 にセットされる．

上記の拡張により，map-table，reorder-buffer，命令キューに登録されているレジスタエントリの状態は，state-table を参照することで管理できるようになる．3.1 節で述べたように，Rename ステージで論理レジスタから物理レジスタへの変換が行われる際に，state-table は必ず参照されるため，slave-entry が有効であるかは state-table に登録されている master-entry の pair-bit を見ることで判断可能である．同様に，reorder-buffer についても Commit ステージにおいて，state-table を参照することで slave-entry の有効性が判断でき，二重解放なども防ぐことができる．さらに，命令キューの場合も，前項で述べたように発行された命令のソースオペランドに対して Arbitrate ステージで state-table を参照しつつ slave-entry の有効性のチェックを行うために，問題とはならない．このように，本拡張では各機構の物理レジスタ ID を無効化することなく，state-table の状態を参照することで，解放されたエントリの状態を管理することができる．

4. 評　　価

4.1 評 価 環 境

BPRF の性能および消費電力を調べるため，SimpleScalar Tool Set [16] を用いたサイクルレベルシミュレーションにより評価を行う．なお 3.2 節で述べたプロセッサ構成を評価できるよう，SimpleScalar のマイクロアーキテクチャを大幅に変更している．また，消費エネルギーの評価には，Wattch [17] を用いる．図 4 中の命令キューやレジスタファイルといったメモリ構成の機構については，Wattch 上のメモリ機構のアクセスあたりの消費電力を算出するモデルを用い，ビット幅，エントリ数，ポート数をパラメータとして与え算出した．BPRF では，命令キューなどで各バンクごとにレジスタ ID フィールドが必要になる点を考慮し，各機構のベースのビット幅に対し，追加すべきレジスタ ID フィールドのビット幅を加えて消費電力を求めた．さらに，free-pool はバンクごとに必要であるため，バンク分の free-pool の消費電力を計算している．なお，Arbiter や 0-detect といった組合せ回路の消費電力については，レジスタファイルや命令キューなどに比べ小さいと考えられるため，本評価では無視する．

評価プログラムは，SPEC CPU2000 の整数ベンチマークのすべてのプログラム，および MediaBench [18] から adpcm，epic，g721，mpeg2（それぞれエンコードとデコード）を用いる．コンパイラは，Alpha 用の命令セットを生成する DEC C コンパイラを用い，オプションは “-arch ev6 -fast -O4 -non_shared” である．なお，SPEC CPU2000 ベンチマークには *ref* インプットセットを用い，最初の 10 億命令実行後の 200 万命令を評価した．また，mpeg2 エンコード以外の MediaBench のプログラムはプログラム実行の最初から最後までを，mpeg2 エンコードは SPEC 同様に最初の 10 億命令実行後の 200 万命令を評価した．

4.2 評価の仮定

表 1 に評価におけるプロセッサの仮定を示す．評価では，レジスタのサイズ，およびポート数を変化させ，従来のレジスタファイルと BPRF を比較する．ポートの調停のための Arbitrate パイプラインステージを

表 1 評価における仮定
Table 1 Processor configuration.

Data path width	64 bit
Fetch & Decode & Commit width	4
Branch prediction	Combined bimodal (4K-entry) gshare (4K-entry) selector (4K-entry)
BTB	1,024 sets, 4way
Mis-Prediction penalty	8 or 9 cycles
Instruction queue size	integer: 32, load/store: 32 floating-point 32
Issue width	integer: 4, load/store: 2 floating-point: 2
L1 I-Cache	32 KB, 32 B line, 2way 1 cycle latency
L1 D-Cache	32 KB, 32 B line, 2way 2 cycle latency
L2 unified Cache	1,024 KB, 64 B line, 8way 10 cycle latency
Memory latency	80 cycle
Bus width	16 B
Bus clock	1/4 of processor core

追加する場合，分岐予測ミスペナルティが 1 サイクル延びると仮定して評価を行った．なお，すべての演算器，およびロードストアユニットが同時にレジスタアクセスをした場合に必要となる整数レジスタファイルのポート数は，リードが 12 ポート，ライトが 6 ポートである☆．

また，0-detect は ALU の演算結果だけでなく，ロード命令のオペランドにも適用するものとして評価を行う．

5. 評価結果

5.1 性能

レジスタサイズの評価

まず，従来型のレジスタファイルおよび BPRF において，レジスタファイルサイズが性能に与える影響を調べるために，**図 10** にレジスタファイルサイズを変化させた場合の，評価に用いた全プログラムの平均 IPC を示す．図中の *Normal* は従来型のプロセッサを表し，*2-bank*，*4-bank* は，BPRF においてそれぞれ何バンクに分割したかを表している．また，横軸のレジスタファイルサイズは，各バンクのエントリ数である．なお，ポートに数ついては，12 リード/6 ライポートを仮定し，コンフリクトが生じない条件で評価

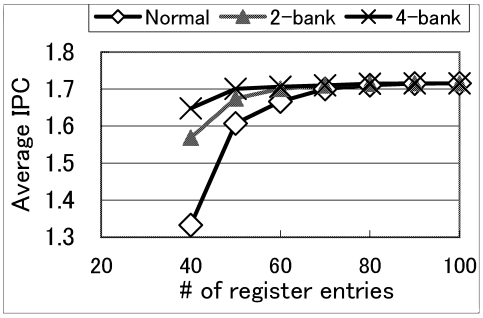

図 10 IPC（全プログラム平均）
Fig. 10 Average IPC.

を行った☆☆．

図 10 の結果より，すべての場合で，レジスタサイズが増加するにつれて IPC が向上しているのが分かる．これは，レジスタエントリが多い場合，命令キューにおいて発行するべき候補の命令数が増え，より ILP が活用できるためである．レジスタサイズが十分でないと，Rename ステージにおいて free-pool からデスティネーションレジスタに割り当てるべきエントリがなくなり，割当て可能なエントリが確保できるまで Queue ステージ以前のパイプラインをストールさせなければならない．その結果，少ないレジスタエントリでは ILP が十分に活用できなくなる．

しかし，ある程度までレジスタサイズが増えると IPC は飽和し，ほぼ同じ IPC に収束する．これは，レジスタサイズが性能上のボトルネックでなくなったときには，他の機構（命令キューサイズなど）の制限から，それ以上 IPC が向上しなくなるためである．

ここで，注目すべき点は，BPRF では Normal に比べて少ないレジスタサイズでも高性能を達成できる点である．Normal では 80 エントリ付近で最高 IPC に到達するのに対し，2-bank では 60 エントリ，4-bank では 50 エントリ程度でほぼ同じ IPC を達成できる．BPRF ではレジスタを分割し，有効ビット幅の小さいオペランドが検出された時点で，そのレジスタエントリを解放するため，より多くのエントリを新たなオペランドのために用いることができ，効率的にレジスタファイルを使うことができた結果である．

なお，BPRF の 2-bank と 4-bank 構成とを比較すると，50 エントリの場合でも両者にあまり差がないことが分かる．BPRF においてバンク数を増やすと，map-table などに記憶しなければならないレジスタ ID

☆ 演算，またはロードストアあたり，2 リード/1 ライトを行うため，整数演算，ロードストアの合計命令発行幅が 6 の場合，最大で 12 リード/6 ライトポートが必要となる．

☆☆ ポート調停のための Arbitrate ステージは必要ないため，Normal，BPRF ともに，分岐予測ミスペナルティは 8 サイクルとして評価した．

表 2 レジスタのアクセス時間と消費エネルギー
Table 2 Access time and energy consumption per register access.

	Normal (80-entry)		BPRF (60-entry)	
# of read/write ports	Time [ns]	Energy [nJ]	Time [ns]	Energy [nJ]
2 / 1	0.500	0.0899	0.417	0.0842
4 / 2	0.571	0.0996	0.460	0.0915
6 / 3	0.643	0.1092	0.503	0.0988
8 / 4	0.718	0.1189	0.547	0.1061
10 / 5	0.795	0.1285	0.590	0.1135
12 / 6	0.876	0.1382	0.635	0.1208

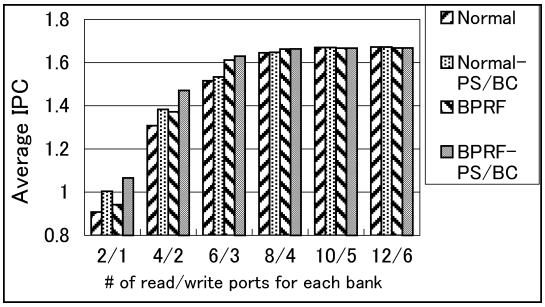

図 11 各バンクのポート数を変化させた場合の IPC
Fig. 11 Average IPC varying the number of ports for each bank.

が増え，ハードウェア量が大きくなることから，BPRF では 2-bank 構成が妥当であると考えられる．しがたって，以降の評価では，2-bank の BPRF について主に評価する．

ポート数の評価

図 11 に各バンクのリード/ライトポート数を変化させた場合の全プログラムの平均 IPC を示す☆．レジスタのサイズは，図 10 の評価結果より，Normal の場合で 80 エントリ，BPRF の場合で 60 エントリを仮定した．なお，従来型のレジスタファイルにおいても，*Port-Sharing*（*PS*），および *Bypass-Check*（*BC*）と呼ばれるポートの使用効率向上手法があり[7]，それらの手法を適用した場合についても評価している．PS は，同一サイクルに同じオペランドが読み出される場合に，複数オペランドで 1 つのポートを共有するものであり，BC はオペランドが演算器からフォワーディング（Bypass）される際にはレジスタからの読み出しを行わず，無駄なポート消費を防ぐものである．なお，PS および BC は BPRF でも適用可能であり，それらを組み合わせた場合の結果も示している．

図より，Normal の場合では，リード/ライトポート数が 12/6 の場合と比較して 8/4 ポートまではほとんど性能低下は見られないが，ポート数が 6/3 より少なくなると，性能が大きく低下してしまうことが分かる．これは，レジスタのポート数が十分でない場合，アクセスの競合が生じ，演算器に対して必要なオペランドの供給やレジスタへの書き込みがその時点では行えず，ストールが発生してしまうためである．

一方，BPRF では PS および BC 手法と組み合わせることで，6/3 ポートの場合でも，ほぼ 12/6 ポートの場合と同じ性能を達成している．BPRF ではすべての値が 0 であるサブワードには，レジスタエントリを読み出す必要がないため，少ないポート数でもアクセスの競合が起きにくいのが理由である．

なお，図 10 の最高 IPC と 図 11 の 12/6 ポートの場合を比較すると，Arbitrate ステージを設けた影響でわずかに性能が低下している．しかし，その差は 2.5%と小さく，Arbitrate ステージの追加による性能への影響はそれほど大きくないと考えられる．

5.2 レジスタのアクセス時間および消費電力

前節の評価結果より，BPRF では従来のレジスタファイルに比べ，小容量で少ないポート数であっても高性能が達成できることが分かった．そこで，次にレジスタのサイズおよびポート数が，レジスタのアクセス時間と消費エネルギーに与える影響を評価する．

表 2 に，80 エントリの従来型のレジスタファイルと，60 エントリの BPRF における 1 アクセスあたりのアクセスタイム，消費エネルギーを示す．アクセス時間は CACTI-3.2[19] を，消費エネルギーは Wattch[17] を拡張して求めた．本結果は，0.1μ プロセスを仮定した場合の値である．なお，BPRF における消費エネルギーは，64-bit のオペランド読み出す場合，すなわち 2 つのバンクをアクセスした場合の値である．

前節の評価結果では，ほぼ同じ IPC を達成するためには，PC および BC 手法を適用した Normal-PS/BC の場合で 80 エントリ・8/4 ポート，BPRF-PS/BC の場合で 60 エントリ・6/3 ポートでが必要であった．それらの場合を比較すると，BPRF では，レジスタアクセス時間が 30%，消費エネルギーは 17%ほど削減さ

☆ ポート調停のための Arbitrate ステージが必要であるため，分岐予測ミスペナルティを 9 サイクルとして評価した．

れている．また，ポートの調停を行わない場合，従来のプロセッサでは 12/6 ポートのレジスタファイルを用いなければならず，その場合を比較すると，Normal の 80 エントリ・12/6 ポートに対して BPRF の 60 エントリ・6/3 ポートでは，アクセス時間で 43%，消費エネルギーは 29%も削減することができる．このことより，提案する BPRF を用いることで，IPC をほとんど低下させずに，レジスタアクセス時間や消費エネルギーを大きく削減できると考えられる．

一般に，現在のマイクロプロセッサにおけるクリティカルパスは，Issue ステージや Register Read ステージ，またフォワーディングパスを含む Execute ステージであるといわれているが，BPRF のために追加で必要となる制御回路は Issue ステージや Register Read ステージにはない．また，Execute ステージにおける 0-detect 機構は，従来のプロセッサでも *divide-by-zero* の例外を早期に検出するためにすでに実装されていることもあり[20]，BPRF の拡張にともない Execute ステージの遅延時間が大きく延びることはないと考えられる．したがって，BPRF 手法によりプロセッサ全体の遅延時間が大きく増大することはなく，逆にクリティカルパスになりやすいレジスタアクセス時間を削減することができるため，プロセッサ全体の高速化に寄与できる可能性も大きい．

5.3 プロセッサの消費エネルギー

BPRF では，map-table や命令キューなどに必要なレジスタ ID のフィールドが増加するため，レジスタファイルの消費エネルギーを削減できたとしても，プロセッサ全体で見た場合に消費エネルギーが増加してしまう可能性がある．そこで，本節では，他の機構も含めた消費エネルギーを評価する．なお，Arbiter や 0-detect といった組合せ回路の消費電力については，レジスタファイルや命令キューなどに比べ小さいと考えられるため，ここでは無視する．

図 12 は，Normal-PS/BC，および BPRF-PS/BC について，ポートの調停を行わない従来のプロセッサに対する，キャッシュを除くプロセッサコア部の消費エネルギー削減率を示したものである．図より，従来のプロセッサに比べ，Normal-PS/BC，および BPRF ともに消費エネルギーを大きく削減できることが分かる．これは，レジスタアクセス 1 回あたりの消費エネルギーが削減されたことによる効果と，PS および BC 手法や BPRF により，レジスタアクセス回数そのものが削減された効果によるものである．そのため，ポート数が 2/1 や 4/2 など，少ないポート数の場合には，アクセスあたりの消費エネルギーは小さいものの，PS や BC の効果が減少しアクセス回数が増えるため，かえって消費エネルギー削減率が小さくなっている．

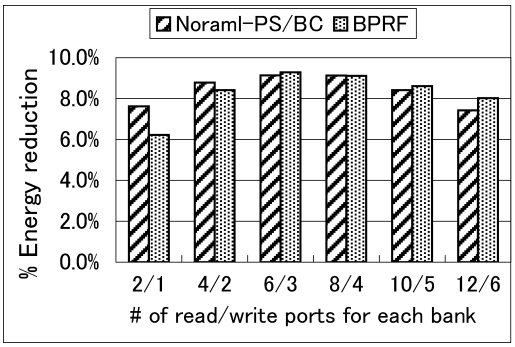

図 12 プロセッサコア部の消費エネルギー削減率
Fig. 12 Energy reduction of the processor core.

Normal-PSBC ではリード/ライトポート数が 8/4 の場合に最も削減率が高く，従来型のプロセッサに比べ，9.1%の消費エネルギーが削減されている．また，BPRF の場合はポート数が 6/3 の場合に最も削減率が高く，9.3%の消費エネルギーが削減されている．

以上の結果，BPRF では map-table や命令キューなどに保存すべきレジスタ ID が増えるものの，レジスタアクセスの消費電力を大きく削減できるため，プロセッサコア部全体でも従来のマルチバンクレジスタと同程度に消費エネルギーを削減できると考えられる．一方で，BPRF は従来のマルチバンクレジスタに比べレジスタアクセス時間を削減できるため，この点で従来のマルチバンクレジスタ手法よりも有効なアーキテクチャである．

6. 関 連 研 究

従来より，レジスタファイルのアクセス時間や消費電力の削減を目的に，レジスタの容量やポート数を削減するための手法が多く提案されている．また，本研究と同じ視点から，オペランドのビット幅を考慮したレジスタファイルの効率的使用に関する研究も行われている[14),15),21),22)]．

文献 14) では，有効ビット幅が小さいオペランドを検出し，有効ビット幅がある閾値よりも小さいオペランドは，レジスタリネーミング用の map table の物理レジスタ ID フィールドに保存し，そのオペランド用のエントリを割り当てないことで，レジスタファイルを効率的に使用する手法を提案している．文献 21) では，従来のレジスタファイル中にはオペランドの上位ビットの値が同じで，下位ビットのみが違うオペランドが多数存在することに着目し，上位ビットをそれら

のオペランドで共有して保存することで，レジスタを有効利用する手法を提案している．また，文献 15) では，有効ビット幅の小さないくつかのオペランドを，1 つのエントリにパッキングして保存することで，レジスタエントリを有効に活用する手法を提案している．さらに，文献 22) では，一部のポートのみが全ビットにアクセスできる一方，他のポートは下位ビット部分のみアクセス可能なレジスタファイル構成を提案している．上位ビット部分で消費されるエネルギーを削減することが目的である．

また，組み込み分野などにおける LSI 設計時に，有効ビット幅を意識して，レジスタだけではなくデータパス全体を最適化する手法も提案されている[23]．さらには，コンパイラがビット幅を意識することで，レジスタを有効利用する手法もある．たとえば，MIPS の命令セットアーキテクチャでは，浮動小数点データに関し，単精度（32 ビット幅）の場合は 32 個のレジスタエントリが使用可能であるが，倍精度（64 ビット幅）の場合は連続する 2 エントリを統合してオペランドを保存するように（偶数番のレジスタ ID のみを用いる）命令の生成が行われる．これらは基本的にユーザによる明示的なビット幅などにより最適化が行われるが，本論文で提案する BPRF は，動的に有効ビット幅を判定することで，実行するデータに依存して有効ビット幅が変わるような場合にも対応できることが利点である．

7. まとめと今後の課題

本論文では，レジスタファイルのサイズおよびポート数の削減を目的とした，ビット分割レジスタファイル（*Bit-Partitioned Register file: BPRF*）を提案した．BPRF ではレジスタファイルをビット方向に分割し，有効ビット幅の小さいオペランドに対しては必要な分だけのレジスタバンクを割り当てることで，レジスタの記憶領域の有効利用を狙うものである．

評価の結果，従来のレジスタファイルに比べて，BPRF では同程度の IPC を達成しつつ，サイズおよびポート数を大きく削減できることが分かった．また，その場合レジスタアクセス時間やレジスタアクセスの消費エネルギーを大幅に削減できることも分かった．今後は，従来のマルチバンク化レジスタファイルと組み合わせることで，さらに効率的なレジスタの利用法を検討することや，マルチバンク化レジスタと性能および消費電力を比較することなどが課題である．

謝辞 本研究の一部は，文部科学省科学研究費補助金（基盤研究（B）No.14380136），および東レ科学技術研究助成金によるものである．

参 考 文 献

1) González, A., González, J. and Valero, M.: Virtual-Physical Registers, *Proc. 4th Intl. Symp. on High-Performance Computer Architecture*, pp.175–184 (1998).
2) Monreal, T., et al.: Delaying Physical Register Allocation Through virtual-Physical Registers, *Proc. 32nd Intl. Symp. on Microarchitecture*, pp.186–192 (1999).
3) Jourdan, S., et al.: A Novel Renaming Scheme to Exploit Value Temporal Locality through Physical Register Reuse and Unification, *Proc. 31st Intl. Symp. on Microarchitecture*, pp.216–225 (1998).
4) Balakrishnan, S. and Sohi, G.S.: Exploiting Value Locality in Physical Register Files, *Proc. 36th Intl. Symp. on Microarchitecture*, pp.265–276 (2003).
5) Cruz, J.-L., González, A., Valero, M. and Topham, N.P.: Multiple-Banked Register File Architectures, *Proc. 27th Intl. Symp. on Computer Architecture*, pp.316–325 (2000).
6) Balasubramonian, R., Dwarkadas, S. and Albonesi, D.H.: Reducing the Complexity of the Register File in Dynamic Superscalar Processors, *Proc. 34th Intl. Symp. on Microarchitecture*, pp.237–248 (2001).
7) Tseng, J.H. and Asanovic, K.: Banked Multiported Register files for High-Frequency Superscalar Microprocessors, *Proc. 30th Intl. Symp. on Computer Architecture*, pp.62–71 (2003).
8) 近藤正章，中村 宏：ビット分割によるレジスタファイルサイズ削減手法，情報処理学会研究報告，ARC-159, pp.13–18 (2004).
9) Kondo, M. and Nakamura, H.: A Small, Fast and Low-Power Register File by Bit-Partitioning, *Proc. 11th Intl. Symp. on High-Performance Computer Architecture*, pp.40–49 (2005).
10) Kessler, R.E.: The Alpha 21264 Microprocessor, *IEEE Micro*, Vol.19, No.2, pp.24–36 (1999).
11) Yeager, K.C.: The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, Vol.16, No.2, pp.28–40 (1996).
12) Sima, D.: The Design Space of Register Renaming Techniques, *IEEE Micro*, Vol.20, No.5, pp.70–83 (2000).
13) Martin, M.M., Roth, A. and Fischer, C.N.: Exploiting Dead Value Information, *Proc 30th Intl. Symp. on Microarchitecture*, pp.125–135 (1997).

14) Lipasti, M.H., Mestan, B.R. and Gunadi, E.: Physical Register Inlining, *Proc. 31st Intl. Symp. on Computer Architecture*, pp.325–335 (2004).
15) Ergin, O., Balkan, D., Ghose, K. and Ponomarev, D.: Register Packing: Exploiting Narrow-Width Operands for Reducing Register File Pressure, *Proc. 37th Intl. Symp. on Microarchitecture*, pp.304–315 (2004).
16) Austin, T., Larson, E. and Ernst, D.: SimpleScalar: An Infrastructure for Computer System Modeling, *IEEE Computer*, Vol.35, No.2, pp.59–67 (2002).
17) Brooks, D., Tiwari, V. and Martonosi, M.: Wattch: A Framework for Architectural-Level Power Analysis and Optimizations, *Proc. 27th Intl. Symp. on Computer Architecture*, pp.83–94 (2000).
18) Lee, C., Potkonjak, M. and Mangione-Smith, W.H.: MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems, *Proc. 30th Intl. Symp. on Microarchitecture*, pp.330–335 (1997).
19) Shivakumar, P. and Jouppi, N.P.: CACTI 3.0: An Integrated Cache Timing, Power, and Area Model, *WRL Research Report 2001/2*, Compaq Computer Corporation, Western Research Laboratory (2001).
20) Brooks, D. and Martonosi, M.: Value-Based Clock Gating and Operation Packing: Dynamic Strategies for Improving Processor Power and Performance, *ACM Trans. Computer Systems*, Vol.18, No.2, pp.89–126 (2000).
21) Gonzalez R., et al.: A Content Aware Integer Register File Organization, *Proc. 31st Intl. Symp. on Computer Architecture*, pp.314–324 (2004).
22) Aggarwal, A. and Franklin, M.: Energy Efficient Asymmetrically Ported Register Files, *Proc. 21st Intl. Conf. on Computer Design*, pp.2–7 (2003).
23) Shackleford, B., et al.: Embedded System Cost Optimization via Data Path Width Adjustment, *IEICE Trans. Inf. Syst.*, Vol.E80-D, No.10, pp.974–981 (1997).

(平成 17 年 1 月 24 日受付)
(平成 17 年 4 月 26 日採録)

近藤　正章（正会員）

平成 10 年筑波大学第三学群情報学類卒業．平成 12 年同大学大学院工学研究科博士前期課程修了．平成 15 年東京大学大学院工学系研究科先端学際工学専攻修了．博士（工学）．独立行政法人科学技術振興機構戦略的創造研究推進事業 CREST 研究員を経て，現在東京大学先端科学技術研究センター特任助手．計算機アーキテクチャ，ハイパフォーマンスコンピューティング，ディペンダブルコンピューティングの研究に従事．電子情報通信学会，IEEE，ACM 各会員．

中村　　宏（正会員）

1985 年東京大学工学部電子工学科卒業．1990 年同大学大学院工学系研究科電気工学専攻博士課程修了．工学博士．同年筑波大学電子・情報工学系助手．同講師，同助教授を経て，1996 年より東京大学先端科学技術研究センター助教授．この間，1996〜1997 年カリフォルニア大学アーバイン校客員助教授．高性能・低消費電力プロセッサのアーキテクチャ，ハイパフォーマンスコンピューティング，ディペンダブルコンピューティング，ディジタルシステムの設計支援の研究に従事．情報処理学会より論文賞（平成 5 年度），山下記念研究賞（平成 6 年度），坂井記念特別賞（平成 13 年度），各受賞．IEICE，IEEE，ACM 各会員．