計算機アーキテクチャ 149－3
（２００２．８．２２）

オペランド再利用によるレジスタ・ファイルの低消費電力化

高村　拓志　　井上　弘士　　Vasily G. Moshnyaga

福岡大学工学部　電子情報工学科

〒814-0180　福岡県福岡市城南区七隈 8-19-1
E-mail: {takamura,inoue,vasily}@v.tl.fukuoka-u.ac.jp

あらまし　本稿では、オペランド再利用によるレジスタ・ファイル・アクセス数の削減手法を提案する。プログラム実行時、演算対象となるソース・オペランドはレジスタ・ファイルから読み出される。従来のプロセッサでは、2 つの連続した命令が同一ソース・オペランドを必要とする場合、それぞれの命令に関してレジスタ・ファイル読出しが実行される。これに対し、提案手法では、先行命令によって読み出されたソース・オペランドの値をパイプライン・レジスタ内に保存し、後続命令のオペランド・フェッチ時に再利用する。また、RAW ハザードを解決するために実装されたフォワーディング機能を活用することで、レジスタ・ファイル書込みに関するアクセス数も削減する。ベンチマーク・プログラムを用いて実験を行った結果、最大で 62%のレジスタ・ファイル・アクセス数を削減できた。
キーワード　低消費電力、レジスタ・ファイル、再利用、マイクロプロセッサ、動的最適化

Reducing Power Consumption of Register Files through Operand Reuse

Hiroshi Takamura,　Koji Inoue,　and　Vasily G. Moshnyaga

Department of Electronics Engineering and Computer Science,
Fukuoka University
8-19-1 Nanakuma, Jonan-ku, Fukuoka, 814-0180 Japan
E-mail: {takamura,inoue, vasily}@v.tl.fukuoka-u.ac.jp

Abstract　This paper proposes an energy reduction technique for register files. The proposed approach attempts to reuse operand data read from the register file in order to reduce the number of register-file accesses. If sequentially executed instructions, i and j, specify the same source operand, then the operand data read from the register file by the instruction i is reused for the instruction j. In this case, the operand fetch for the instruction j can be performed without register file activation, saving energy consumption. As well as the read operation, we can eliminate register-file write accesses by exploiting forwarding unit, which is used for solving RAW pipeline hazard problem. In our simulation, it is observed that the proposed approach can reduce the total number of register-file accesses by 62% from a conventional model.
Key words　low power, low energy, register file, microprocessor, reuse, dynamic optimization

１．はじめに

LSI の低消費エネルギー化は、バッテリ駆動型携帯電子機器システムの動作時間延長を実現するために極めて重要な要素技術である。また、高性能コンピュータ・システムにおいても、信頼性向上や低コスト化を実現するため、低消費電力化（消費電力は単位時間当りの平均消費エネルギー）を考慮しなければならない。特に、コンピュータ・システムの中心となるマイクロプロセッサの低消費エネルギー化は必要不可欠であり、これまでにも様々な研究が行われてきた。

マイクロプロセッサの低消費エネルギー化を実現する１つの手段として、プログラム実行時に不必要な処理を動的に検出し削除する方法がある。例えば、プログラム実行中、演算に必要となるデータ語長を検出し、不必要なビット位置（例えば上位ビット）に関連するデータパスの動作を停止する[1][2]。これにより、計算結果に影響を与えること無く消費エネルギーを削減できる。また、分岐予測の正確さを動的に検出し、分岐予測が外れるであろう場合には投機実行を中止する事で、不要な命令実行に伴う消費エネルギーを削減することも可能である[3]。

ここで、本稿では、マイクロプロセッサ構成要素の１つであるレジスタ・ファイルに着目する。一般に、１個の演算命令を実行するためには３回のレジスタ・ファイル・アクセス（２回のオペランド読出しと１回の演算結果書込み）が発生する。また、現在の高性能マイクロプロセッサにおいては、命令レベル並列性を有効に活用して高性能化を達成するため、より多くのファンクション・ユニットを搭載する傾向にある（最大命令発行幅の拡大）。ファンクション・ユニット数に見合った IPC の向上を実現するためには、レジスタ・ファイル・ポート数の増加は必要不可欠である。しかしながら、レジスタ・ファイルの消費エネルギーはポート数に比例する[4]。よって、今後の高性能/低消費電力マイクロプロセッサにおいては、レジスタ・ファイルの低消費エネルギー化が重要となる。

そこで本稿では、不必要なレジスタ・ファイル・アクセスを動的に検出・削除する新しい手法を提案する。また、ベンチマーク・プログラムを用いた定量的評価を行い、その有効性を明らかにする。連続して実行される命令が同一ソース・オペランドを参照する場合、先行命令によって読み出されたソース・オペランド値を後続命令が再利用する。これによりレジスタ・ファイル・アクセス数を削減し、低消費エネルギー化できる。また、RAW ハザードを回避するために実装されたフォワーディング回路を活用することで、レジスタ・ファイル書込みに関するアクセスも削減できる。

以下、第 2 節では、従来のプロセッサにおける無駄なレジスタ・ファイル・アクセスに関して説明し、問題点を整理する。次に、第 3 節ではオペランド再利用に基づくレジスタ・ファイル・アクセス削減手法を提案する。そして、第 4 節ではベンチマーク・プログラムを用いた定量的評価を行い、最後に第 5 節で簡単にまとめる。

２．従来型レジスタ・ファイルにおける問題点

レジスタ・ファイルが SRAM で実現される場合、プログラム実行時のその消費エネルギー(*Erf*)は以下の式で近似できる。

$$Erf = (Nread + Nwrite) * Eacc \quad (1)$$

ここで、*Nread* および *Nwrite* は、それぞれ、プログラム実行におけるレジスタ・ファイル総読出し回数、および、総書込み回数である。また、Eacc はレジスタ・ファイル・アクセス当りの平均消費エネルギーである（本稿では、簡単化のため、レジスタ・ファイル読出しアクセス当りの消費エネルギーと、書込みアクセス当りのそれは同じと仮定する）。式(1)から分かるように、プログラム実行におけるレジスタ・ファイルの消費エネルギーは、レジスタ・ファイル・アクセス回数と、アクセス当りの消費エネルギーに依存する。

ここで、5 段の命令パイプライン(IF, ID, EX, MEM, WB)において、以下の命令列を実行する場合を考える（シングル・パイプラインのイン・オーダ実行）。

```
add     $t0, $s1, $t1     ①
mul     $t3, $s1, $t1     ②
add     $t1, $t1, $s1     ③
sub     $t1, $s1, $t1     ④
lw      $t2, 20($s1)      ⑤
sub     $t4, $t1, $s1     ⑥
```

第 1 オペランドはデスティネーション・レジスタを、第２および第３オペランドはソース・レジスタを表す。この例の場合、命令①～④、および、命令⑥に関しては、それぞれ、２回のレジスタ読出しが必要となる。また、命令⑤では１回のレジスタ読み出しを実行する必要がある。一方、レジスタ・ファイル書込みに関しては、各命令それぞれにおいて１回ずつ発生する。つまり、従来型レ

ジスタ・ファイルのアクセス数は、

$$Nread = 11, \quad Nwrite = 6$$

となる。

ここで、各命令実行の詳細を考察する。①の加算命令では、ソース・オペランドとして ID ステージにてレジスタ s1 が読み出される。その後、次命令②の実行において、再度レジスタ s1 へのアクセスが発生する。しかしながら、レジスタ s1 の更新は、少なくとも命令⑥の実行が終了するまで更新されない。つまり、命令①～⑥で読み出されるレジスタ s1 の値は全て同じである。従来型のレジスタ・ファイル・アクセスでは、連続実行される命令間でのソース・オペランド相関関係を考慮しない。そのため、同一レジスタに対する連続アクセスが発生した場合、当該レジスタの値が更新されていないにも関わらず、毎回レジスタ・ファイルにアクセスする（活性化する）ため、多くの無駄なエネルギーを消費する。

一方、レジスタ・ファイル書込みに着目した場合にも、読出し時と同様に無駄なアクセスが存在する。例えば、命令③において生成されたレジスタ t1 の値は、次命令④によって消費される。通常の命令パイプラインでは、RAW ハザードの発生を回避するため、フォワーディング機構を搭載している。よって、命令③の演算結果がフォワーディングされて命令④に引き渡される。その後、命令④の計算結果がレジスタ t1 に書き込まれる。つまり、命令③が生成したレジスタ t1 の値の消費者は命令④のみとなる。したがって、命令③の WB ステージにてレジスタ・ファイルに書き込まれたレジスタ t1 の値は、他のどの命令からも参照されることなく、命令④によって上書きされる。この場合、命令③によるレジスタ・ファイル書込みアクセスは無駄となる。

3　オペランド再利用によるレジスタ・ファイル・アクセス数の削減

3．1　読出しアクセス

第 2 節で説明したように、従来のマイクロプロセッサにおいては多くの無駄なレジスタ・ファイル・アクセスが存在する。そこで、このような無駄なアクセスを動的に検出・削除し、レジスタ・ファイル消費エネルギーを削減する手法を提案する。

第 2 節で用いた例の場合、命令②で読み出すべきオペランド s1 の値は、命令①によってすでにレジスタ・ファイルから読み出されている。そこで、命令②のオペランド・フェッチにおいて、命令①で読み出したオペランド値を再利用する。具体的には、命令 i の実行において、以下のように動作する。

1. 命令 i の IF ステージの後半において、先行命令（命令 i-1）のソース・オペランド・レジスタ番号と、命令 i のそれとを比較する。
2. もし、一致であれば、命令 i の ID ステージにてレジスタ・ファイル読出しアクセスを中止する。それと同時に、先行命令(命令 i-1)によって読み出されたソース・オペランド値を失わないよう、レジスタ読出しデータを保持するための ID/EXE パイプライン・レジスタに対するクロック信号の供給を停止する。
3. もし、上記１における比較結果が不一致であれば、通常の命令実行と同様にレジスタ・ファイル・アクセスを行う。

例えば、第 2 節で示した例の命令②実行において、その IF ステージの後半で、命令①のソース・オペランドが命令②のそれと等しいか否かを判定する。そして、その判定結果を IF/ID パイプライン・レジスタに格納しておき、ID ステージにてレジスタ･ファイル･アクセス実行/中止の判定に用いる。本例の場合、命令②の第１および第２ソース・オペランド、ならびに、命令⑤の第１ソース・オペランドに対して読出しデータの再利用を適用でき、総レジスタ・ファイル・アクセス数は

$$Nread = 8, \quad Nwrite = 6$$

となる。

また、加算命令や乗算命令の場合には交換則が成り立つため、第１ソース・オペランドと第２ソース・オペランドを入れ替えることも可能である。例えば、加算命令③のソース・オペランドを入れ替える。これにより、命令③に関してもソース・オペランドの再利用が可能となり、総レジスタ・ファイル・アクセス回数は

$$Nread = 4, \quad Nwrite = 6$$

となる。この手法を本稿では **Swap** 方式と呼ぶ。

さらに、命令⑥のソース・オペランドを入れ替えた場合、その第２ソース・オペランドは、命令④の第２ソース・オペランドと同一である。それに加え、これらの命令間には第２ソース・レジスタ･オペランドを持たない命令⑤のみが存在する。このような場合､命令④の ID ステージにて読み出された第２ソース・オペランド値を２クロック・サイクル保持することにより（パイプライン・ストールが発生しない場合)、命令⑥のソース・オペランド・フェッチにおいても再利用できる。その

結果、総レジスタ・ファイル・アクセス回数は

$$Nread = 2, \quad Nwrite = 5$$

となる。この手法を本稿では **Skip** 方式と呼ぶ。

３．２　書込みアクセス

通常、命令の演算結果は、後続命令によって使用することができるよう、レジスタ・ファイルに書き込まれる。しかしながら、命令パイプラインには RAW ハザードの発生を回避するためのフォワーディング機構が搭載されているため、第 2 節で説明したように、レジスタ・ファイルに書き込まれたにも関わらず、将来一度も参照されない場合が出てくる。そこで、レジスタ・ファイル書込みを伴うある命令 i を実行する時、以下の条件が検出された場合にはレジスタ・ファイル書込みアクセスを中止する。

- 命令 i の演算結果が後続命令によってフォワーディングされ、かつ、フォワーディング可能命令と命令 i のデスティネーション・レジスタが同一である。

ここで、フォワーディング可能命令とは、命令 i の演算結果をフォワーディングによってソース・オペランドとして使用可能な後続命令を示す。例えば、第 2 節で示した例の命令③において、その演算結果は後続命令④によって参照され、かつ、命令③と命令④のデスティネーション・レジスタは同じである。つまり、この場合、上述した条件を満足するため、命令③のレジスタ・ファイル書込みアクセスを削除できる。その結果、総レジスタ・ファイル・アクセス回数は

$$Nread = 2, \quad Nwrite = 5$$

となる。

一般に、マイクロプロセッサは正確な割り込みをサポートしなければならない。しかしながら、前述したようにレジスタ・ファイル書込みを中止した場合、正確な割り込みを保証することができなくなる。つまり、第 2 節の例の場合、命令③の演算結果は命令④にフォワーディングされ、レジスタ・ファイルには書き込まれない。もし、命令③の実行が終了した後、命令④によってレジスタ t1 が更新されるまでの間に割り込み（もしくは例外）が検出された場合、プロセッサはレジスタ・ファイルの内容を退避する。割り込み処理終了後、退避したレジスタ・ファイルの内容を復元し、命令④から実行を再開する（正確な割り込みを保証するため）。しかしながら、この時、レジスタ t1 の値は、命令③によって更新される前の値が保持されている（命令③ではレジスタ・ファイル書込みを行わないため）。その結果、命令④のオペランド・フェッチ時には正しい t1 の値を得ることができず、正確な割り込みを保証できなくなる。よって、本節で説明したレジスタ・ファイル書込みアクセスの中止を行う場合、割り込みを禁止する等の処置が必要である（例えば命令④の実行が終了するまで割込みを禁止する）。

表１：ベンチマーク・プログラム

benchmark	input	benchmark	input
129.compress	train	MPEG2 decode	mei16v2
	test		tennis
	ref		trace_mei16v2
099.go	train	MPEG2 encode	trace_clinton
ADPCM decode	trace		
		pegwit	my.pub
ADPCM encode	trace		trace_my.pub
			race_pegwit

４　評価

４．１　実験環境

ミネソタ大学で開発されたマイクロプロセッサ・シミュレータ(fast)[5]を用いて命令レベル・シミュレーションを行い、以下に示す各手法におけるレジスタ・ファイル総アクセス回数を測定した。なお、本評価で用いたベンチマーク・プログラムを表１にまとめる。

- Conv：従来型レジスタ・ファイル・アクセスを行うモデル。
- R：レジスタ読出しに関して、連続した命令間でデータ再利用を行うモデル。
- Rsw：R モデルにおいて、第 3 節で示した Swap 方式（第一オペランドと第２オペランドの入れ替え）を適用したモデル。
- Rsk：R モデルにおいて、第 3 節で示した Skip 方式を適用したモデル。
- Rsw+sk：Rsw と Rsk の組合せ。
- W：フォワーディング機構を利用してレジスタ書き込みを中止するモデル。
- W+Rxx：W モデルと各 R モデルの組合せ。

４．２　実験結果

４．２．１　レジスタ・ファイル読出し回数

各プログラム実行におけるレジスタ・ファイル総読出しアクセス回数を図１に示す。全ての結果は、従来型読出しアクセス数(Conv)の結果で正規

化している。

ほとんどのプログラムにおいて、連続実行された命令間でオペランド・データを再利用することで約 15%～30%のレジスタ・ファイル・アクセス回数を削減できている（R モデル）。また、第 2 節で示した Swap 方式、ならびに、Skip 方式を用いることで、更なるアクセス数の削減を達成できた。実際、Swap 方式ならびに Skip 方式を組み合わせることで、最大 63%のレジスタ読出しアクセスを削減できた。

ここで、Swap 方式と Skip 方式を比較した場合、MPEG エンコードを除く全てのプログラムにおいて、Swap 方式の方が大きな削減率であった。これは、Skip 方式を適用するための条件と比較して、Swap 方式を適用するための条件を満足する実行命令パタンの出現率が高いためである。

４．２．２　レジスタ・ファイル書込み回数

各プログラムにおける総書込み回数を図２に示す。全ての結果は、従来型書込みアクセス数（Conv）の結果で正規化している。実験結果より、多くのプログラムで 30%以上のレジスタ・ファイル書込みを削減できていることが分かる。これは、連続した演算命令が実行される場合、演算結果の生存期間は極めて短く、かつ、それらは直後の後続命令によって消費される場合が多いためである。

４．２．３　総レジスタ・ファイル・アクセス回数

各プログラム実行における総レジスタ・ファイル・アクセス回数（読出し+書込み）を図３に示す。全ての結果は、従来型総アクセス数(Conv)の結果で正規化している。　本評価では、レジスタ・ファイル読出しに関してのみ削減を試みるモデル（Rxx）、ならびに、読出しおよび書込みの両方に関してアクセス削減を試みるモデル（W+Rxx）を比較する。

レジスタ・ファイル読出しにのみ着目した Rxx モデルでは、Swap 方式ならびに Skip 方式を組み合わせた場合で、総レジスタ・ファイル・アクセス回数を約 16%～33%削減できた。更に、RF 書込みも考慮した場合、総レジスタ・ファイル・アクセス回数を 32%～61%削減できる。しかしながら、第 3.2 節で説明したように、正確な割込みを保証する場合、レジスタ・ファイル書込みアクセスの削除は難しくなる。したがって、実用的なマイクロプロセッサへの応用を考えた場合、レジスタ・ファイル読出しに関してのみデータ再利用を行う Rxx モデルの方が適していると考える。

４．終わりに

本稿では、オペランドの再利用によるレジスタ・ファイル低消費エネルギー化手法を提案した。また、ベンチマーク・プログラムを用いた定量的評価を行い、レジスタ・ファイル読出しのみ着目した場合で最大 44%、読出し/書込みの両方に着目した場合で最大 62%のアクセス数削減を達成できた。

本評価においては、命令レベル・シミュレーションによりレジスタ・ファイル・アクセス回数を測定した。また、レジスタ・ファイル読出し当たりの平均消費エネルギーと、書込み当たりのそれとは同一であると仮定した。今後、サイクル・レベル・シミュレーションを行い、かつ、より詳細な消費エネルギー・モデルを用いた評価を行う予定である。

謝辞

日頃から御討論頂く、福岡大学工学部モシニャガ・ワシリー研究室の諸氏に感謝します。なお、本研究は、一部、文部省科学研究費補助金(科研番号 14702064,14580399) による。

参考文献

[1] D. Brooks and M. Martonosi, ”Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance,” The International Symposium on High-Performance Computer Architecture, Jan. 1999.

[2] R. Canal, A. Gonzalez, and J. Smith,”Very Low Power Pipelines using Significance Compression,” Proc. Of the International Symposium on Microarchitecture, Dec. 2000..

[3] S. Manne, A. Klauser, and D. Grunwald, “Pipeline Gating: Speculation Control for Energy Reduction,” Proc. Of the International Symposium on Computer Architecture, June-July, 1998.

[4] V. Zyuban and P. Kogge, “The Energy Complexity of Register Files,” Proc of the International Symposium on Low-Power Electronics and Design, Aug. 1998..

[5] URL: http://www-mount.ee.umn.edu/~okeefe/mcerg/fast-dlx/

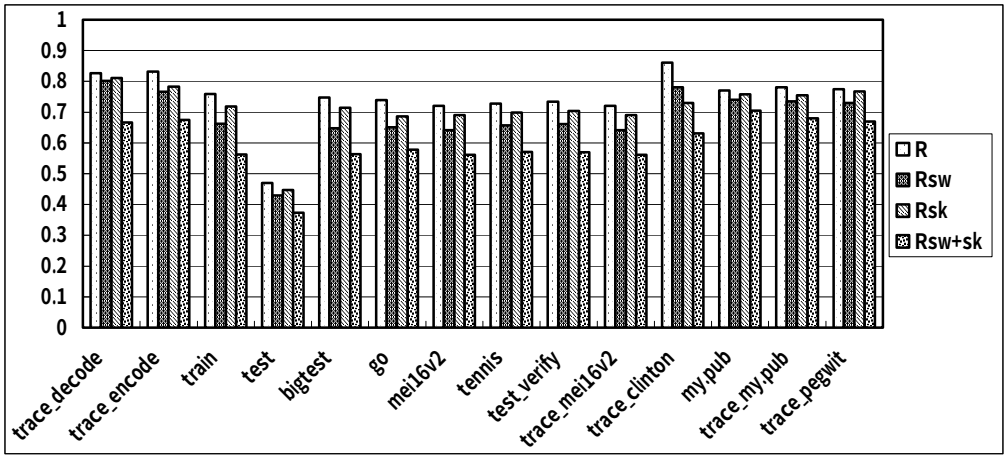

図 1：レジスタ・ファイル読出し回数

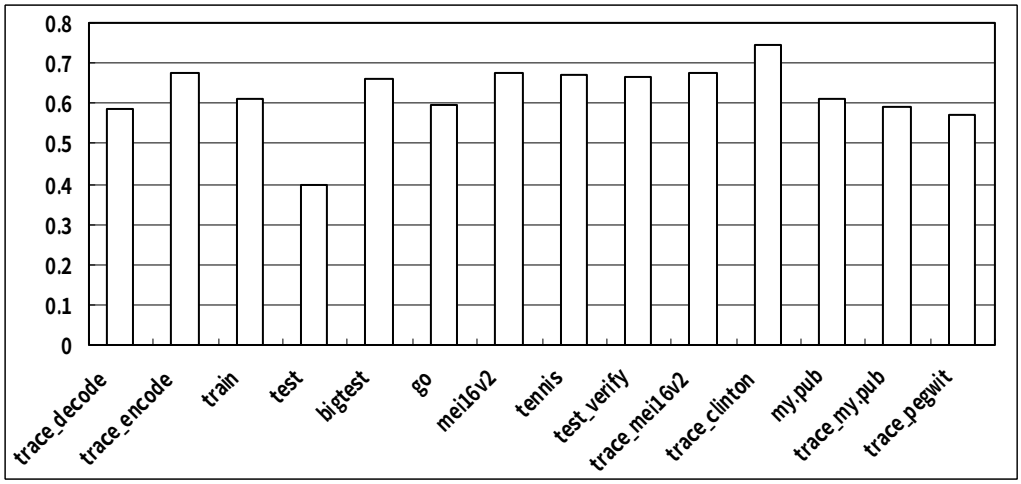

図 2：レジスタ・ファイル書き込み回数

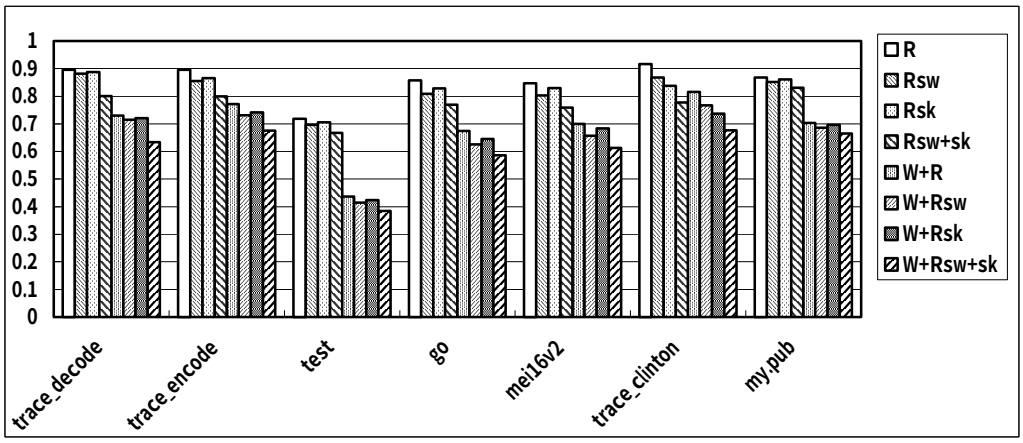

図 3：総レジスタ・ファイル・アクセス数（読出し+書込み）