ソース・タグ値による発行キューのセグメント化による電力削減

安藤 秀樹

2019 年 8 月 2 日

1 はじめに

現在のプロセッサは、非常に微細な LSI 技術で製造される。このような LSI の微細化に伴い、デバイスの信頼性低下 (ハード・エラー) の問題が深刻になっている [1]。その原因には、経時的絶縁破壊 (TDDB: time dependent dielectric breakdown)、負バイアス温度不安定性 (NBTI: negative bias temperature instability)、エレクトロマイグレーション (EM: electromigration) などがある。これらは、タイミング・エラーや誤動作といった摩耗故障 (wearout) を引き起こし、システムの寿命を決定づける。これらの現象は、いずれも温度に指数関数的に依存しており (TDDB については [2]、NBTI については [3]、エレクトロマイグレーションについては [4])、温度 10〜15°C の上昇で、デバイスの寿命は半分以下になる [5]。

プロセッサ・チップには、単位面積当たりの電力 (電力密度) が大きい場所を多数存在する。このような場所は**ホット・スポット**と呼ばれ、そうでない場所に比べて温度上昇が著しく、前述した故障を引き起こす確率が高くなる。このため、ホット・スポットを生成する回路は、消費する電力がたとえプロセッサの全消費電力に対し大きくなくとも、電力を低下させる必要がある。

ホット・スポットを生成する回路の 1 つに発行キュー (IQ: issue queue) がある。IQ のサイズは、プロセッサの世代が進むごとに性能向上のために大きくなっている。このため電力は増加しており、より深刻なホット・スポットとなっている。

IQ で最も大きな電力を消費する回路は、タグ比較器である。タグ比較は、発行幅分のデスティネーション・タグと IQ 内の全てのソース・タグについて全数比較が行われ、非常に電力効率が悪い。これに対して、本研究では、タグの下位数ビットが等しい命令についてのみ残りの上位ビットを比較する方式を提案する。これにより、動作する比較器の数を大きく減少させることができ、電力を削減できる。

具体的には、以下のように行う。

1. IQ を複数のセグメントに分割し、第 n セグメントには、第 1 ソース・タグの下位ビットが n である命令だけを保持する。

2. ウェイクアップにおけるタグ比較においては、デスティネーション・タグの下位ビットが n に等しい IQ のセグメントの比較器のみを活性化し、残りの上位ビットのタグの比較を行う。

以上のようにすれば、第 1 ソース・タグについての全ての比較器の内、その「1/セグメント数」の比較器しか動作しないので、電力が削減できる。また、タグの上位ビットだけの比較で済むので、比較器のビット幅が削減され、さらに電力は削減される。

なお、第 2 ソース・タグについては、通常通り、全ての命令についてタグ比較を行う。命令の中には、第 2 ソース・オペランドがない命令も多い。そのような場合、第 2 ソース・タグの比較は必要なく、あらかじめ比較器を非活性化しておけば良く、IQ をセグメント化した有効性は高く維持される。

2 概要

図 1 に、例として、IQ を $N = 4$ のセグメントに分割し、1 セグメントは $M = 2$ 個のエントリを持つ場合のウェイクアップ論理の左半分 (第 1 ソース・オペランド比較部分) の構成を示す。発行幅 IW とし、タグ・ビット幅 5 とする。

第 n セグメントのエントリには、第 1 ソース・タグ (stag) の下位 2 ビットが n である命令をディスパッチする。つまり、第 0 セグメントには、stag の下位 2 ビットが”00”、第 1 セグメントには、stag の下位ビットが”01” である命令が保持される (他のセグメントについても同様)。

ウェイクアップ時には、デスティネーション・タグ (dtag) の下位 2 ビットと等しい番号のセグメントのみ有効化される。つまり、この時点で dtag 下位 2 ビットは、stag の下位 2 ビットと一致していることが保証される。

セグメントの有効化を指示するために、dtag の下位 2 ビットがセグメント番号に等しいことを検出する AND ゲートを配置し (セグメントを示す箱の左)、セグメント有効化端子 (E 端子) に信号を送る。有効化信号の生成に使用されなかった dtag の残りの 3 ビットは、セグメント内エントリにある比較器に送られ、比較が行われる (有効化されなかったセグメントのエントリでは比較は行われれない)。

たとえば、dtag=”01101” の場合、下位 2 ビットが”01” なので、第 1 セグメント内のエントリにおいて、上位のビット”011” がソース・タグの対応する 3 ビットと比較される。この 3 ビットが一致すれば、全ビット一致したこととなる。

3 ウェイクアップ比較器

ウェイクアップ論理の左半分の回路 (第 1 ソース・タグ比較からレディ・ビットのセットまでの回路)(1 エントリ分) を図 2 に示す。この回路は、2 節と同様、IQ のセグメント数 4、発行幅 IW、タグビット幅 5 の例である。回路を以下に説明する。

- ①は、当該命令が属するセグメント有効化信号 $E_0..E_{IW-1}$ を出力する AND ゲートである。
- ②は、セグメント有効化信号が真のとき、dtag の上位 3 ビットを比較器に送る AND ゲートである。有効化信号が偽のときは、②の AND ゲートの出力が L となり、比較器のプルダウン・トランジスタが全てオフとなり、比較は行われない。
- ③は、比較が行われた時にその結果の出力を送る AND ゲートである。セグメント有効化信号が真のとき出力は比較結果であり、偽のときは、L が出力される。
- ⑤、⑥は、通常のウェイクアップ論理に存在する回路であり、それぞれ、タグのいずれかが一致したことを検出する OR ゲートと、レディ・ビットを保持する FF である。

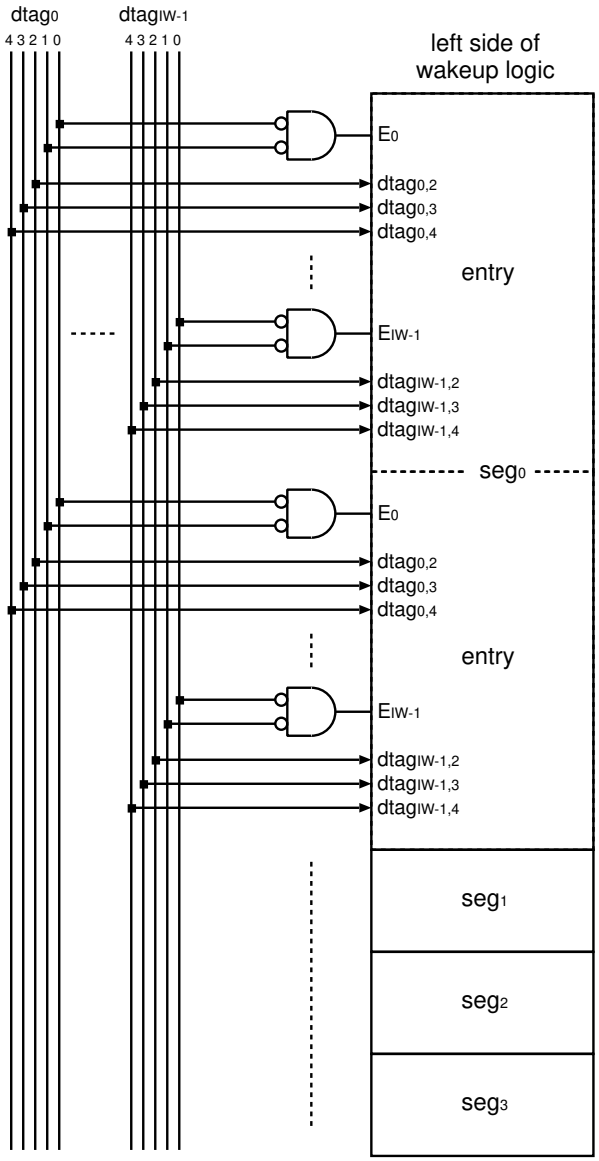

図 1: ウェイクアップ論理の左半分の構成

4 最適化

これまで説明した方式には、いくつか最適化の余地がある。

4.1 容量効率

IQ はソース・タグの下位ビットの値にしたがってセグメントに入れられるが、セグメント・サイズは固定なので、過不足が生じる。つまり、命令を書き込むべきセグメントが一杯ならば、他のセグメントに空きエントリがあってもディスパッチできず、ストールする。これを軽減する 2 つの手法を提案する:

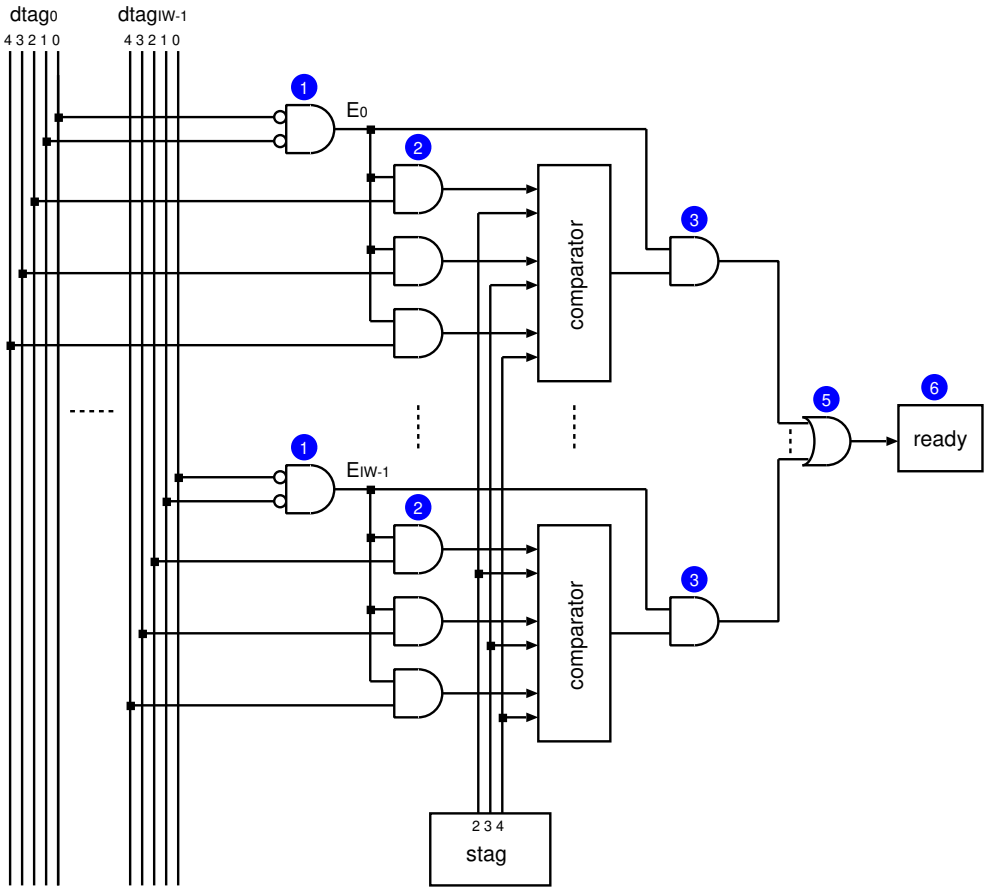

図 2: ウェイクアップ論理の左半分の回路

1. できるだけセグメントを均等に使用するよう、リネーム時にセグメントに命令が分散するようタグを割り当てる。

2. IQ に余分に数エントリを持つセグメント (予備セグメントと呼ぶ) 加える。予備セグメントのエントリは、いかなるソース・タグの命令も保持できるよう従来のウェイクアップ論理のエントリと同じとする。命令ディスパッチ時に、目的のセグメントに空きがなければ、この予備セグメントに書き込む。

1 については、もう少し詳細に説明する。もしも、各命令が依存する命令の数が同数なら、デスティネーション・タグをセグメントに均等に対応するよう割り当てれば、容量効率は向上する。割当手法としては、2 節で示した例では、タグのフリーリストを 4 つのバケツに分割する。各バケツには、タグの下位 2 ビットの値が同じタグを保持する。命令にタグを割り当てるときには、バケツをラウンドロビンでアクセスしてタグを得る。この方法は、「各命令が依存する命令の数が同数」ならおよそうまく働くように思うが、依存命令のタグの下位ビットが偏っているとやはりセグメント間のエントリの過不足を生じる。しかし、命令が依存する命令の数は 1 の場合が多いので、前提条件はほぼ満たすと思われる。

4.2 第 2 ソース・タグの比較に要する電力は削減されない欠点について

第 2 ソース・オペランドがない命令も多いが、そうでない命令については、第 2 ソース・タグの比較に要する電力は削減されないことは欠点と言える。これについては、以下の方法で欠点を緩和できる:

> 一方のオペランドがレディで、他方がレディでなければ、レディでないオペランド・タグをセグメント化された側に書き込み、他方をセグメントされていない側に書き込む。

この方法はオペランドの順を交換しているように見え、命令においてオペランドの順に可換性がある場合 (たとえば、加算) にのみ可能のように思えるかもしれないが、そうではない。順に意味があるのは、ペイロード RAM に格納される命令であり、ウェイクアップ論理は、両オペランドがレディかそうでないかがわかりさえすればよいからである。

5 研究の進め方

5.1 シミュレータ

SimpleScalar シミュレータのコードが以下にあるので、本研究用にレポジトリを新たに作り、クローンし、提案手法を実装する:

```
http://hg/repos/simple_scalar/sss_alpha/trunk/
```

5.2 プロセッサ構成

近年のプロセッサに合わせて 8 命令発行、100 エントリの IQ の構成とする。`simplescalar/config` に SimpleScalar のコンフィグ・ファイルがある。

5.3 コーディングのヒント

5.1 節で示したコードでは、IQ の方式としては、シフティング・キュー、ランダム・キュー、エイジ・マトリクス付きランダム・キューが選べるが、本研究では、エイジ・マトリクス付きランダム・キューで研究を行う (多くのプロセッサがこの方式を採っている)。

IQ をこの方式にする場合、命令を IQ のどのエントリにディスパッチするかを以下の IQ のフリーリスト (FIFO バッファ) で管理している (SimpleScalar では、IQ のことを RS(reservation station) と呼んでいる):

```
IQ.[ch]:
struct RS_flist_t RS_flist; /* RS フリーリスト */
```

本研究では、IQ には、セグメントを意識してディスパッチしなければならないから、このフリーリストをセグメントごとに用意しなければならない。

次にどれほど効果があったかを示すために、動作した比較器の数の減少率を評価することとする。従来の IQ での動作した比較器の数は、ウェイクアップ時にレディでないオペランドの比較において、タグが一致しなかった数とする。ここで、次のような仮定を置いている。

- すでにレディなオペランドの比較器は動作しないとする。比較器のプリチャージを抑制することにより、容易に停止させることができるからである。
- デスティネーション・タグは、最大で、発行幅分送られてくるが、送られてこなかったデスティネーション・タグのタグ線につながっている比較器は動作しないとする。これは、比較器のプリチャージ期間にタグ線を L にするが、デスティネーション・タグが送られてこなければ、L のままとすることは容易であるからである。
- タグが一致した比較器は、プリチャージされた電荷がディスチャージされないので、電力を消費しない。

「レディでないオペランドの比較において、タグが一致しなかった数」を数えるには、ウェイクアップを行っている以下の関数内で行えば良い:

```
sim-outorder.c:
static void broadcast_to_Res_Station(int d_preg)
```

提案手法では、第 1 ソース・タグについては、セグメントを意識して数える必要があるが、同様に上記関数内で数えることができると思われる。第 2 ソース・タグについては、従来と同様に数えればよい。

なお、本資料のパッケージ内の `simplescalar/about-simplescalar` に書いているように、数を数えてログに出力するには、インタフェース `stat_reg_counter()` を用いること。

6 ファイル

`jobdir` には、以下のファイルが入っている:

- `job`: この資料の TeX ソース。
- `simplescalar`:
  - `config`: 5.2 節で述べたコンフィグ・ファイル。
  - `about-XXX`: XXX についての注意。

7 関連論文

- 発行キューに関するサーベイ: [6]
- 発行キューの回路の基本: [7, 8] (論文の PDF は研究室の HP にリンクしている)
- エイジ・マトリクス: [9]
- Ernst *et al.* proposed decomposing the issue queue into three queues, with zero, one, or two sets of comparators in each entry corresponding to the number of non-ready operands of an instruction [10]. An instruction is inserted into the appropriate issue queue according to the number of non-ready operands. Power consumption is reduced by not performing useless tag comparison for an already-ready operand. However, decomposing the issue queue

reduces the capacity efficiency of the entire issue queue, because the ratio of instructions based on the number of non-ready operands does not necessarily match the ratio of the capacity of each decomposed queue.

- Folegnani *et al.* and Ponomarev *et al.* independently proposed resizing the issue queue based on demand [11, 12]. If the size of the issue queue is reduced, the comparators in the inactivated entries are disabled, thereby saving power. However, these schemes cannot reduce power if the demand is high, whereas our scheme can.

- Michaud *et al.* proposed a scheme that preschedules instructions by calculating their expected issue time based on dependency and latency, and places them in a simple FIFO buffer [13]. Since the latency of several instructions is variable, a small conventional issue queue is placed after the FIFO buffer. This organization replaces a large portion of the conventional issue queue with a simple FIFO buffer, thereby reducing power consumption. The drawback of this scheme is that the small conventional issue queue after the FIFO buffer is easily clogged by instructions that depend on long-latency operations (e.g., cache misses), and, as a result, the entire issue queue is blocked. Raasch *et al.* improved this approach by constructing dependency chains originating from a cache missed load and controlling the forward movement of an instruction between the segments of the queue [14]. However, implementation and control of the dependency chains is complex.

- Palacharla *et al.* proposed constructing an issue queue with several FIFO buffers, in which instructions are placed according to their dependencies [15]. By taking advantage of the dependency ordering in each FIFO buffer, readiness to issue can be checked by reading the scoreboard that holds the operand availability of each register, considering only the head instructions in the FIFO buffers. In this scheme, the scoreboard must have twice as many ports as the number of FIFO buffers. According to this study, a small number of FIFO buffers is sufficient to achieve high IPC, and thus, the complexity of the readiness check logic is acceptable. Although this is true for compute-intensive programs, the FIFO buffers are filled by long-latency cache missed loads and their consumers in memory-intensive programs, and a new independent dispatch instruction may be blocked owing to the lack of an empty FIFO buffer. This can be avoided by increasing the number of FIFO buffers, which in turn requires more ports for the scoreboard, causing unacceptable complications.

- Goshima *et al.* proposed a scheme that organizes the wakeup logic using RAM, instead of CAM [16]. The energy consumption of the matrix scheduler can be reduced using the scheme proposed in [17], which reduces the number of columns of the wakeup matrix RAM. This scheme, however, requires complicated large cells for the wakeup matrix RAM, lengthening the bit and wordlines, which increases energy consumption. Moreover, it requires several additional energy consuming structures. These include a *wakeup allocation table*, which dynamically allocates a column of the wakeup matrix RAM to a dependency, and a *wakeup free list*, which holds free columns of the wakeup matrix RAM. Both of these are implemented using a heavily ported RAM, where the energy consumption of a multi-ported RAM increases in proportion to the square of the number of ports [1]. These overheads offset the energy reduction of the wakeup matrix RAM.

参考文献

[1] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective, 4th edition*, Addition Wesley, 2010.

[2] F. Monsieur, E. Vincent, D. Roy, S. Bruyre, G. Pananakakis, and G. Ghibaudo, Time to breakdown and voltage to breakdown modeling for ultra-thin oxides (Tox<32Å), In *Proceedings of the 2001 IEEE International Integrated Reliability Workshop*, pp. 20–25, October 2001.

[3] S. Khan and S. Hamdioui, Temperature dependence of NBTI induced delay, In *Proceedings of the 2010 IEEE 16th International On-Line Testing Symposium*, pp. 15–20, July 2010.

[4] J.R. Black, Electromigration–a brief survey and some recent results, *IEEE Transactions on Electron Devices*, Vol. ED-16, No. 4, pp. 338–347., April 1969.

[5] R. Viswanath, V. Wakharkar, A. Watwe, and V. Lebonheur, Thermal performance challenges from silicon to systems, *Intel Technology Journal*, Vol. 4, No. 3, p. 116, August 2000.

[6] J. Abella, R. Canal, and A. Gonzalez, Power- and complexity-aware issue queue designs, *IEEE Micro*, Vol. 23, Issue 5, No. 5, September-October 2003.

[7] K. Yamaguchi, Y. Kora, and H. Ando, Evaluation of issue queue delay: Banking tag ram and identifying correct critical path, In *Proceedings of the 29th International Conference on Computer Design*, pp. 313–319, October 2011.

[8] K. Yamaguchi, Y. Kora, and H. Ando, Delay evaluation of issue queue in superscalar processors with banking tag ram and correct critical path identification, *IEICE Transactions on Information and Systems*, Vol. E95-D, No. 9, pp. 2235–2246, September 2012.

[9] R. P. Preston, R. W. Badeau, D. W. Bailey, S. L. Bell, L. L. Biro, W. J. Bowhill, D. E. Dever, S. Felix, R. Gammack, V. Germini, M. K. Gowan, P. Gronowski, D. B. Jackson, S. Mehta, S. V. Morton, J. D. Pickholtz, M. H. Reilly, and M. J. Smith, Design of an 8-wide superscalar RISC microprocessor with simultaneous multithreading, In *2002 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pp. 334–472, February 2002.

[10] D. Ernst and T. Austin, Efficient dynamic scheduling through tag ellimination, In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pp. 37–46, May 2002.

[11] D. Folegnani and A. González, Energy-effective issue logic, In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pp. 230–239, June 2001.

[12] D. Ponomarev, G. Kucuk, and K. Ghose, Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources, In *Proceedings of the 34th Annual International Symposium on Microarchitecture*, pp. 90–101, December 2001.

[13] P. Michaud and A. Seznec, Data-flow prescheduling for large instruction windows in out-of-order processors, In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, pp. 27–36, January 2001.

[14] S. E. Raasch, N. L. Binkert, and S. K. Reinhardt, A scalable instruction queue design using dependence chains, In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pp. 318–329, May 2002.

[15] S. Palacharla, N. P. Jouppi, and J. E. Smith, Complexity-effective superscalar processors, In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pp. 206–218, June 1997.

[16] M. Goshima, K. Nishino, Y. Nakashima, S. Mori, T. Kitamura, and S. Tomita, A high-speed dynamic instruction scheduling scheme for superscalar processors, In *Proceedings of the 34th Annual International Symposium on Microarchitecture*, pp. 225–236, December 2001.

[17] P. G. Sassone, J. Rupley II, E. Brekelbaum, G. H. Loh, and B. Black, Matrix scheduler reloaded, In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, pp. 335–346, June 2007.