修士論文

並列事前実行における
専用連想検索装置の設計と評価

指導教員　　富田 眞治

京都大学大学院情報学研究科
修士課程通信情報学専攻

高 洪波

平成18年2月3日

並列事前実行における専用連想検索装置の設計と評価

高 洪波

内容梗概

近年，スーパスカラ方式によるプロセッサの性能向上は頭打ちになってきており，他の性能向上手法である投機的マルチスレッディング（以下SpMT）の研究が多く行われている．ただし，従来のSpMTでは，投機実行中の先行スレッド１つのみを主スレッドが引継ぐことしかできないという欠点がある．これに対して提案されている，再利用および並列事前実行機構は，命令区間の再利用性に着目し，入力の一致した先行スレッドの実行結果を主スレッドが利用可能とする．また，専用コンパイラを使わず，手続きやループ構造の動的な検出により，既存ロードモジュールへの適用が可能である．

本機構では，入力比較のために，再利用バッファと呼ぶ連想検索装置を用いる．再利用バッファに登録されている命令区間について今後の入力値を予測し，主スレッドの実行と並行して投機スレッドを事前実行し，結果を再利用バッファに登録する．これにより，入力が単調変化する場合など，過去の実行結果の単純な再利用では効果がない命令区間に対しても高速化を図る．

しかし，汎用連想メモリにより構成した場合，登録および検索などの複合機能に多くのサイクル数が必要であり，機構全体のボトルネックになる．

本論文では，再利用バッファの主機能（検索&書き込みと検索&読み出し）を高速化する専用連想検索装置の設計を提案する．富士通社の0.18umプロセスを用い，高速連想メモリ，優先検出機構および制御装置などのトランジスタレベル設計を行い，各機能を，1サイクル内に実現することを目標とした．HSPICEを用いた評価では，64エントリの連想検索装置が2.2nsで動作した．この時の再利用バッファの面積は$0.339mm^2$で，消費電力は94mWであった．この64エントリの連想検索装置を8コアで32スレッドを実行可能なプロセッサUltraSparc T1に搭載すると仮定すると，面積は0.025%，消費電力は0.13%の増加で済むことから，本研究の成果がマルチコア型マイクプロセッサに対して有効であることがわかった．

Design and Evaluation of Special-Purpose Associative Memory for Parallel Early Computation

GAO HONGBO

Abstract

Recently, it becomes difficult to improve the performance of microprocessors by superscalar technique. Instead, speculative multithreading (SpMT) is attracting more attention and widely recognized as a core technique to eliminate power dissipation and improve the effectiveness of parallelism of high-end processors in the next generation.

In conventional SpMT, the main thread takes over the data from only one prior speculative thread. In order to improve the effectiveness of SpMT, I proposed a SpMT model called Parallel Early Computation which exploits the hardware based multi-level region-reuse mechanism with a shared associative buffer. Each set of input is memorized in the reuse buffer, and reused by main thread later. The nested regions are identified dynamically while executing the conventional load modules, which are generated by widely-used compilers. As a result, recompilation or binary annotation is unnecessary.

In this model, CAM is used to compare all inputs of the region. However, using the conventional CAM requires lots of cycles for search-and-write and search-and-read operations. It will result in poor performance of the system.

For the purpose of solving the problem described above, the design of a high-speed CAM for Parallel Early Computation is introduced in this thesis. By using Cadence IC Design Platform and FUJITSU 0.18us layout rule, Schematic and Layout of CAM, SRAM, and Control Circuit are dessigned. By using HSPICE simulation, the 64 entry reuse buffer can operate correctly in 2.2ns. The chip-area is $0.339mm^2$, and the power consumption is 94mW. Recently announced 8-core processor, such as UltraSparc T1, the area and power consumption are increased by 0.025% and 0.13% respectively. This thesis shows that above mechamism can improve the performance of SpMT.

並列事前実行における専用連想検索装置の設計と評価

目次

第 1 章　はじめに　1

第 2 章　研究の背景　3

2.1　投機実行 3
2.2　SpMT 4
2.3　再利用 4

第 3 章　並列事前実行機構　7

3.1　概要 7
3.2　命令区間の動的抽出 8
3.3　一時記録機構 12
3.4　入出力の木構造 14
3.5　ハードウェアモデル 15

第 4 章　再利用バッファの構成と動作　17

4.1　構成と動作の概要 17
4.2　具体例 17
4.3　汎用連想メモリを用いた場合の問題点 21

第 5 章　連想検索装置の設計　23

5.1　構成 23
5.2　動作 23
5.3　メモリ部の設計 (CAM，RAM) 28
5.3.1　RAM セル 28
5.3.2　CAM セル 29
5.3.3　センスアンプ 30
5.3.4　メモリ性能を左右する要素 32
5.4　空きエントリ検出器 34
5.5　制御装置 (GATE) 36

第 6 章　評価　40

6.1　単体性能 40

6.2 専用連想検索装置の性能 40
第 7 章 おわりに **47**
謝辞 **48**
参考文献 **49**

第１章　はじめに

近年のプロセッサ高速化手法として，複数命令を並列実行するスーパスカラ方式がある．しかし，一般的なプログラムにおいて並列実行可能な命令数は高々２程度と言われており，命令レベル並列性の追究だけでは今後の性能向上が見込めなくなってきている．また，回路の大規模化により，消費電力が膨大になる問題が生じている．

一方，次世代ハイエンドプロセッサにおいて，消費電力を抑えつつ並列度を向上させる中核的技術として，マルチコアが注目されている．マルチコアとは，2つ以上のプロセッサコアを１個のパッケージに集積したマイクロプロセッサである．各プロセッサコアは基本的に独立しているため、それぞれのプロセッサコアは他のプロセッサコアに影響されることなく動作できる．さらにマルチコアプロセッサで単一スレッドアプリケーションの性能を向上させるために，投機的マルチスレッディング (Speculative Multithreading：以下，SpMT)」と呼ばれる技術が提案されている．SpMT とは，値予測に基づく投機実行により，スレッドレベルの並列度を確保する手法である．一般的な SpMT は，投機実行スレッドと通常実行スレッド間で一対一のデータ受け渡しを行っている．従って，レジスタおよび主記憶入出力の比較機構が必要となるが，一般に再コンパイルを行い，静的解析に基づいて付加情報の埋め込み（アノテーション）を行うことで実現している．

これに対し，本論文では，区間再利用を用いた並列事前実行を提案している．再利用（Reuse）とは一般に，関数呼び出しやループなどの命令区間について，入出力セットを記憶しておき，再び同じ入力により命令区間を実行しようとする際に，記憶しておいた出力を利用して命令区間の実行を省略する高速化手法である．本機構では，プログラムを構成する命令区間を多入力多出力の複合命令と捉え，動的に検出した複数の複合命令を複数のコアにより並列実行するため，再コンパイルやアノテーションが全く不要である．本手法における並列事前実行機構では，入出力比較のための再利用バッファとして連想メモリ (CAM:Content Addressable Memory）を用いることを想定している．再利用バッファに登録されている命令区間の各々について将来の入力値を予測し，主スレッドの実行と並行して投機スレッドを事前に実行し，結果を再利用バッファに登録する．これにより，入力が単調変化する場合など，過去の実行結果の単純な再利用では

性能向上が見込めない命令区間に対しても，高速化を図ることができる．

しかし，汎用連想メモリを用いる構成では，CAM の検索の結果，複数のマッチラインがアサートされ得るために，アサートされたマッチラインのうち唯一のマッチアドレスを出力するためのプライオリティエンコーダを設け，最優先ラインに対応するマッチアドレスのみを生成する必要がある．その後，改めて当該マッチアドレスを用いて RAM を参照する構成とすることにより，連想検索装置全体におけるマッチアドレスの競合を防止できる．しかし，プライオリティエンコーダは，マッチライン数の増加とともに長い処理時間が必要となる．このため，深い CAM，すなわち，多くのマッチラインを有する CAM を構成する際には，プライオリティエンコーダがボトルネックとなり，高速化が困難となる．プライオリティエンコーダ自身の消費電力も問題となる．また，再利用バッファを連想検索する時，後続命令区間の入力パターンを予測するために，検索の結果一致した CAM の内容を履歴表に格納する必要がある．しかし，汎用 CAM のビットラインは検索と読み出し機能を兼ねることができないので，検索結果を読み出す際に改めて読み出しサイクルを設ける必要がある．

本論文では，以上のような汎用連想メモリを用いる構成の問題点を解消する効率的な専用連想検索装置を提案する．以下，第 2 章で研究背景について述べ，本研究の位置付けを明らかにする．次に，第 3 章では，並列事前実行の詳細について述べ，第 4 章で本論文が再利用バッファの構成について述べ．第 5 章で本論文提案する連想検索装置の設計について述べる．その後，第 6 章 HSPICE による評価を行う．

第2章　研究の背景

本章では，関連研究である投機実行，SpMT，および再利用について概観する．

2.1　投機実行

投機実行は，命令アドレスを対象とするものとオペランドを対象とするものに大別できる．命令アドレスに関する投機実行は，さらに静的分岐予測と動的分岐予測に分類できる．前者は，コンパイラが分岐予測を行い，命令中に予測情報を埋め込む手法である．後者は，コンパイラに頼らず，実行時に履歴を用いて分岐先を予測する手法である．分岐方向には偏りがあり，過去2回程度の分岐履歴情報を用いて十分に予測可能と言われており，比較的単純なハードウェア機構により実現できることから，多くの商用プロセッサが動的分岐予測を採用している．一方，オペランドの投機実行は，さらに，アドレス予測と値予測に分類できる．アドレス予測の代表として，Next Line Prefetch がある．あるキャッシュラインが参照されると，近い将来，次のラインを参照すると予測し，主記憶からプリフェッチする手法である．値予測では，履歴に基づいて命令の実行結果を予測し，予測値を入力として後続命令を投機的に実行する．予測が正しければ後続命令が先行命令の完了を待つ時間が短縮され，誤りの場合は，投機状態にある命令を無効化し，正しい入力値を用いて再実行する．このため，投機実行を適用しない場合より性能が低下する場合がある．また，予測値を常に検証する必要があるため，ハードウェアが複雑になる欠点もある．値予測には，大きく次の手法がある．

Last-value 予測: 同じ命令アドレスにおける前回の演算結果をそのまま使用する手法である．さらに，動的に予測可能かどうかを判断する CT（Classification Table）を用いて，予測ミスを削減する機構が提案されている [1].

Stride-based 予測: 最近の2回の演算結果の差分を S，最近の演算結果を B とした場合に，S+B を予測値とする [2].

Two-level 予測: 命令ごとに最近の4種類の演算結果と，各演算結果の過去の出現回数を記録する．後者が既定値に達した時に，対応する演算結果を予測値とする．

ハイブリッド予測: Last-value 予測と Stride-based 予測，または，Stride-based 予測と two-level 予測を組み合わせる手法である．

ところで，投機実行においては，予測値の検証が必要性であるため，命令の実行時間そのものを削減することはできない．このため，厳密な検証が必要となる値自身を投機対象とするのではなく，予測したアドレスに基づいて load 命令を事前に実行し，効果的なプリフェッチを図る予備実行（Precomputation）の研究が多く行われている [3，4，5].

2.2 SpMT

SpMT は，値予測に基づく投機実行により，スレッドレベル並列性を利用して高速化を図る手法である．主記憶一貫性を保証するために，一般的には，投機スレッドの開始後に，主スレッドが投機スレッドのソースオペランドを上書きした時点で投機実行を無効化する．Gopal ら [6] は，マルチプロセッサシステムにおいて，各主記憶値の予測値を複数保持する Speculative Versioning Cache を提案している．各キャッシュラインには順序づけされたラインセットが保持される．キャッシュはラインセットから無効化応答を受け取ると，無効化信号をプロセッサに送信する．Marcuello ら [7，8] は，トレースに基づく増分予測を提案している．増分は，当該トレースの前回実行時における，トレース終了時の値とトレース開始時の値の差として求める．Oplinger[9] らは，CALL された関数本体および関数復帰後に続く命令列を並列実行する手法を提案している．文献 [6，7，8，9] とは対照的に，Codrescu ら [10] は，ループイタレーションや関数呼び出しではなく，load 命令を契機にスレッドを起動することにより，細粒度の投機スレッドを生成可能な自動分割手法を提案している．投機的データを保持するキャッシュは，主スレッドを実行するプロセッサが発行するストアにより生じる擾乱を検出する．また Marcuello ら [11] は，コントロールグラフおよび到達可能性に基づく，プロファイルベースの投機スレッド起動手法を提案している．

2.3 再利用

再利用とは，実行時に入出力セットを再利用バッファに記憶しておき，再び同じ入力により命令が実行されようとした場合に，過去に記憶しておいた出力を利用することにより命令の実行自体を省略する高速化手法である．投機的手法ではなく，予測失敗による無効化を必要としない特長がある．ハードウェアによるものや，ソフトウェア支援によるものが提案されている．動的かつ効率よ

く基本ブロックを切り出すことは難しく，簡単化のために，コンパイラが再利用を行う専用命令を生成し，ハードウェアに伝える方法が一般的である．しかし，専用命令を使用する場合は，専用コンパイラが生成したロードモジュールのみが高速化対象となり，既存ロードモジュールは高速化できない欠点がある．

ハードウェアのみによる手法として，Sodani ら [12] は，最大 1024 エントリからなるフルアソシアティブの再利用バッファを用いる単命令を対象とした汎用的な再利用を提案している．各エントリは，命令オペランドの値および命令結果を保持する．また，ロード命令が再利用可能であることを保証するために，主記憶有効ビットおよび主記憶アドレスが保持される．ストア時には，主記憶アドレスが連想検索され，無効化される．Costa ら [14] は，ロード/ストア命令を対象から除外したうえで，フルアソシアティブの表による再利用手法を提案している．

ソフトウェア支援によるものには，Huang ら [15] の，コンパイラが再利用区間内に閉じたレジスタ出力の登録を省く，基本ブロック再利用方式がある．Connors ら [16] は，コンパイラによる区間切り出しとハードウェアサポートを組み合わせた手法を提案している．最後のロードから次のロードまでの間にストア命令が存在しなければ主記憶比較を省略する．再利用バッファの最大サイズは 128KB 以上である．Connors ら [17] はまた，コンパイル時の情報だけでなく，実行時情報も用いることにより，再利用バッファの割り当てを最適化する手法を提案している．Huang ら [18] は，一般に基本ブロックの入出力数は様々であるため，有限幅の再利用バッファを効率的に使うことができないとし，基本ブロックをサブブロックに分割することにより，再利用バッファに登録可能な入出力数を揃える方法を提案している．

他にも，入力の一致判定に寛容性を持たせる曖昧再利用 [19] や，ロード命令に限ってアドレスおよびロード値を再利用する手法 [21，22] が提案されている．投機実行の結果を一部再利用する研究も行われている．Roth ら [23] は，過去のレジスタマッピングを書き戻すことにより，以前の失敗した投機実行において書き込まれた物理レジスタを再利用する方法を提案し，SPEC ベンチマークプログラムの実行時間を最大 11%削減している．また，再利用とデータ駆動スレッドを統合した，投機的データ駆動マルチスレッディングを提案しており [24]，SPEC ベンチマークプログラムの実行時間を最大 25%削減している．事前実行の結果は，レジスタリネーミングを利用して物理レジスタに格納される．

再利用と投機実行を組み合わせた手法も研究されている．Wu ら [25] は，コンパイラが再利用区間の切り出しを行い，再利用不可能である場合には再利用区間の出力値を予測して，後続区間の実行を投機的に開始する手法を提案している．これにより，4 命令および 8 命令同時発行の構成において，SPEC ベンチマークプログラムの実行時間を 1.25 倍から 1.4 倍高速化している．この手法では，出力値の予測が外れた場合，後続区間の投機実行をキャンセルする必要があり，コストおよびオーバヘッドが問題となる．Molina らは，投機スレッドと主スレッドを組み合わせる手法を提案している．投機スレッドによる実行済みの命令は FIFO に格納され，主スレッドはそこから命令を取り出してソースオペランドを比較し，一致した場合は投機スレッドの結果を用いる．これにより，4 命令同時発行の構成において，SPEC ベンチマークプログラムの実行時間を，最大 1.33 倍，平均で 1.16 倍高速化している．

第3章　並列事前実行機構

本章では，本研究の対象である並列事前実行機構 [26，27] について説明する．

3.1　概要

並列事前実行は，主スレッドや投機スレッドが並列実行した結果を再利用するために，多対 1 のデータ引継ぎ構造を有する SpMT である．本機構では，プログラムを構成する命令区間を多入力多出力の複合命令と捉え，動的に検出した複数の複合命令を並列実行し，主スレッドの実行結果も含む複数の実行結果を再利用することにより，主スレッドが実行する命令を大幅に削減する．投機スレッドを事前に実行することにより，同一入力が出現する間隔が長い場合や入力が単調に変化する場合など，過去の実行結果の単純な再利用では効果がない場合においても再利用の効率を上げ，高速化を図る．この点において，事前実行は従来の予備実行とは異なる．また，命令区間の複雑な入れ子構造についても多重に再利用する機構を提案している．

一般に，同一入力値が出現する間隔の長い命令区間や，入力値が単調変化し続ける命令区間に対しては，投機実行による効果が高いと予想できる．しかし，各命令区間の性質や投機実行による削減ステップ数は事前には分からない．このため，初めて実行する命令区間については，直ちに投機実行用プロセッサにより数回分の事前実行を試みる．その結果，対象の命令区間が主実行用プロセッサによる高い登録頻度を有し，かつ投機実行用プロセッサが登録したエントリの再利用頻度も高い場合，事前実行による効果が高いと考え，継続して投機実行用プロセッサによる投機対象とする．以下では，主スレッドを担当するプロセッサを MSP （Main Stream Processor），投機スレッドを担当するプロセッサを SSP（Speculative Stream Processor）と呼ぶ．

並列事前実行機構のスレッド間の関係を図 1 に示す．MSP は，通常実行するプロセッサであり，可能である場合は命令区間の再利用を行う．一方，複数の SSP は，MSP と並行して投機的実行する．演算器，レジスタ，一次キャッシュは各プロセッサごとに独立しており，再利用バッファ，二次キャッシュ，入力の予測機構および主記憶は全スレッドが共有する．本機構の特長は，コンパイラ支援を必要としない点，および再利用区間の入力値を予測の対象とするため，失敗した投機実行をキャンセルする必要がない点である．入力比較の際にキャッシュ

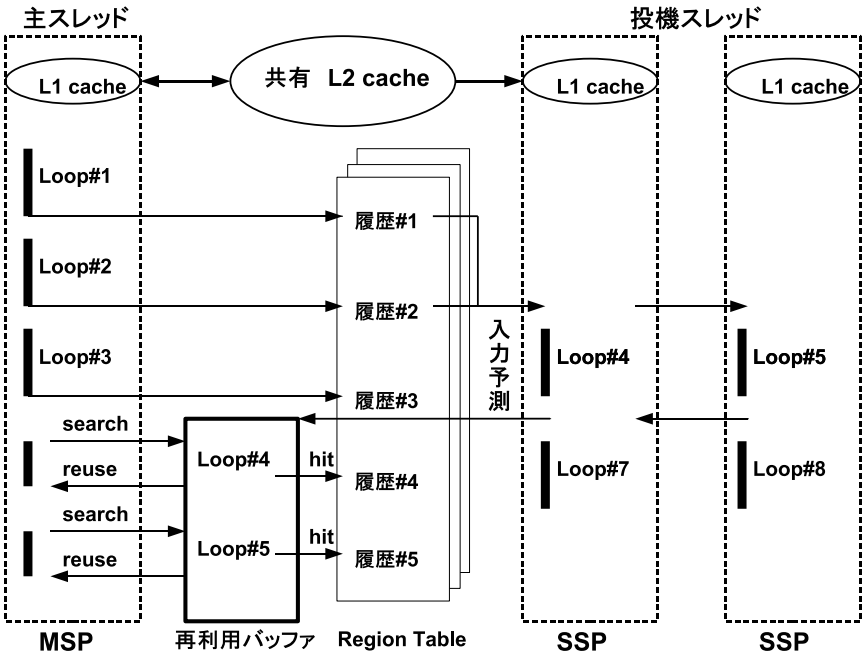

図 1: 並列事前実行の構造

内容まで比較することにより，主記憶参照を必要とする命令区間に対しても再利用が適用できる利点がある．また本機構では，主記憶一貫性保証の手法として，一般的な SpMT で採用されている無効化ではなく，一部比較を用いる．書き換えられたアドレスを記憶し，再利用時に当該アドレスのみを比較することにより，再利用率を低下させることなく主記憶入力値比較コストを軽減できる．

3.2 命令区間の動的抽出

再コンパイルや静的解析に基づく付加情報の埋め込み（バイナリアノテーション）を必要とすることなく，プログラムから適切な命令区間を抽出する機構は，SPARC アーキテクチャを前提としている．ロードモジュールが SPARC ABI（Application Binary Interface）に従って生成されていることを利用し，関数の多重構造を動的に把握することにより，関数の局所レジスタやスタック上の局所変数を再利用における入出力値から除外できる．特に関数については，複雑さに関わらず，最大 6 のレジスタ入力，最大 4 のレジスタ出力，および，局所変数を含まない最小限度の主記憶アドレスおよび値の登録により再利用と事前実行が可能である．

SPARC アーキテクチャでは，プログラムは常に計 32 個の汎用レジスタを使用

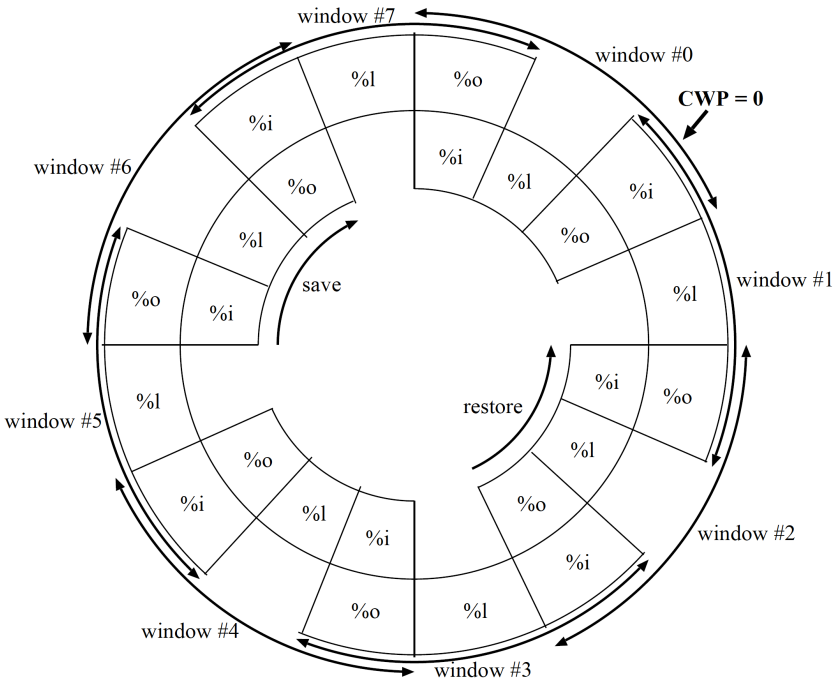

図 2: レジスタウィンドウ

できる．汎用レジスタには，大域変数を格納する global レジスタ（%g0〜%g7），引数および作業用に使用する out レジスタ（%o0〜%o7），引数および返り値を格納する in レジスタ（%i0〜%i7），局所変数を格納する local レジスタ（%l0〜%l7）がある．レジスタはレジスタウィンドウ（図 2）と呼ぶグループに編成される．各レジスタウィンドウはオーバーラップしており，隣接するウィンドウ間で一部のレジスタが共有される．各ウィンドウは上記の%on，%ln，%in（0 $\leq$ n $\leq$ 7）から構成され，%o7 は隣接するウィンドウの%i0 と同一である．%ln は各ウィンドウに固有である．

現在のウィンドウは，CWP（Current Window Pointer）レジスタの内容により決定される．CWP の値は save 命令によって増加し，restore 命令によって減少する．save 命令と restore 命令は，関数呼び出し時と終了時に実行される．save 命令により，以前に%o として参照していたレジスタは%i となり，新しく%l および%o が割り当てられる．一方，restore 命令により，以前に%i として参照していたレジスタは%o となり，新しく%l および%i が割り当てられる．

save 命令によるレジスタ割付けは，restore 命令を実行するまで保存される．しかし，save 命令が続くと，レジスタウィンドウの容量を超過し，新たなレジ

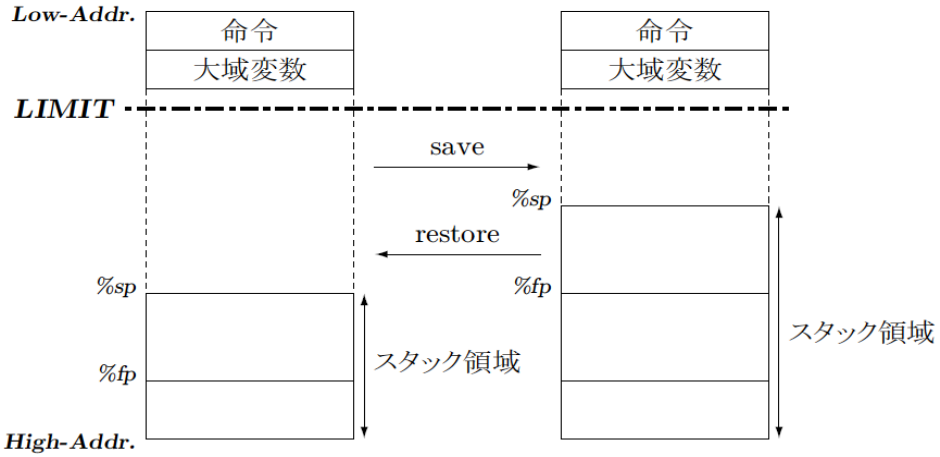

図 3: スタック

スタ割付けができない．この場合はウィンドウオーバフロー割り込みが発生し，レジスタの内容がスタックに退避される．逆に restore 命令が続くと，ウィンドウアンダフロー割り込みが発生し，スタック上の値がレジスタに戻る．すなわち，オーバフローおよびアンダフローの際には主記憶参照が生じ，命令の実行が遅れる．

スタックは，主記憶の高位アドレスから低位アドレスに向けて伸びる．有効なスタックの下限アドレスが，スタックポインタと呼ばれるレジスタ%o6（%sp）に格納され，図 3 に示すように，save 命令により積まれる関数フレーム分だけ減じされ，元の値はフレームポインタと呼ばれるレジスタ%i6（%fp）に格納される．restore 命令の場合は逆の操作が行われる．

また一般的に，主記憶上では，大域変数を格納するためのデータ域とスタックのためのデータ域との境界が OS により設けられる．この境界を LIMIT と呼ぶことにし，大域変数と局所変数の区別に用いる．LIMIT 以上%sp 未満の領域は無効データ領域である．

以上のレジスタウィンドウおよびスタックを用いて関数呼び出しの仕組みを説明する．関数は，CALL 命令，または，プログラムカウンタ（PC）を%o7 に書き込む jmpl 命令により呼び出される．関数の引数は%o0〜%o5 に入る．引数が 7word 以上ある場合には，前述のスタックに格納する．第 7word は%sp+92 に，第 8word は%sp+96 に格納される．以降の引数も同様にスタックに積まれる．さらに関数を呼び出す関数（非 leaf 関数）は save 命令を含む．save 命令の実

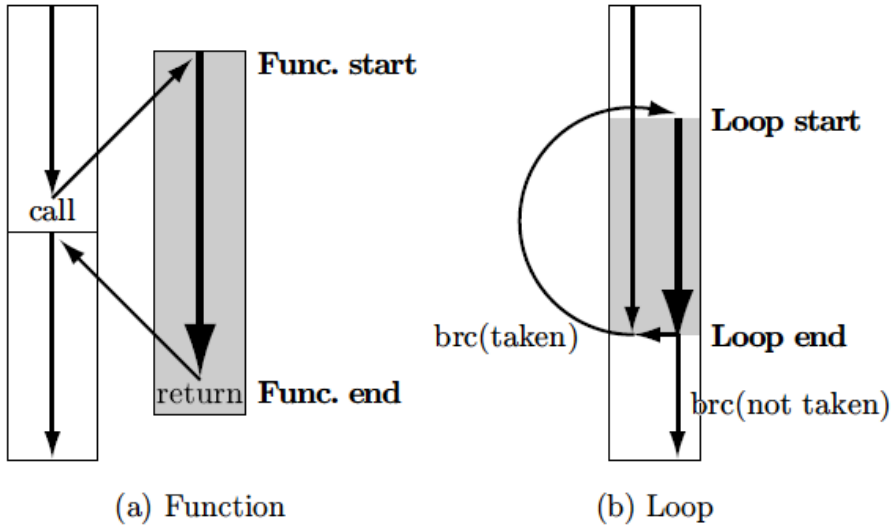

図 4: 関数とループの類似性

行により，%i0〜%i5 に引数，%i6（%fp）に以前のスタックポインタ，%i7 には関数呼び出し時の PC の値が格納される．返り値は 1word の場合%i0 に，2word の場合は%i0 および%i1 に格納される．さらに restore 命令によって，返り値は%o0 および%o1 として参照できる．

一方，leaf 関数の場合は，save 命令および restore 命令はなく，復帰には，第 1 オペランドが%o7+α（$\alpha > 0$）である jmpl 命令が用いられる．引数には%o0〜%o5，返り値には%o0 および%o1 が用いられる．

さて，動的に命令区間および入出力を特定するために，始点と終点を容易に特定できる関数およびループを対象とする．図 4 に，関数とループの類似性を示す．CALL 命令の飛び先を始点とし，最初に到達した return 命令を終点とする命令区間を関数と認識し，引数および大域変数を入力として記録する．同様に，後方分岐命令の飛び先から，同一後方分岐命令までの命令区間をループと認識する．ただし，後方分岐命令を検出して初めてループ始点がわかるため，ループ 1 回目のイタレーションは検出できない．また，ループ内の局所変数は動的に識別不可能であるため，参照したレジスタおよび主記憶アドレスを全て入力とみなして記録する必要がある．ループは，関数と異なり，複数の異なるループが同一先頭アドレスとなる場合がある．このため，ループの記録には分岐先アドレスも含める必要がある．ループの完了以前に関数から復帰したり，入出力の容量超過によりループの記録が中止されることがない場合，対応する後方

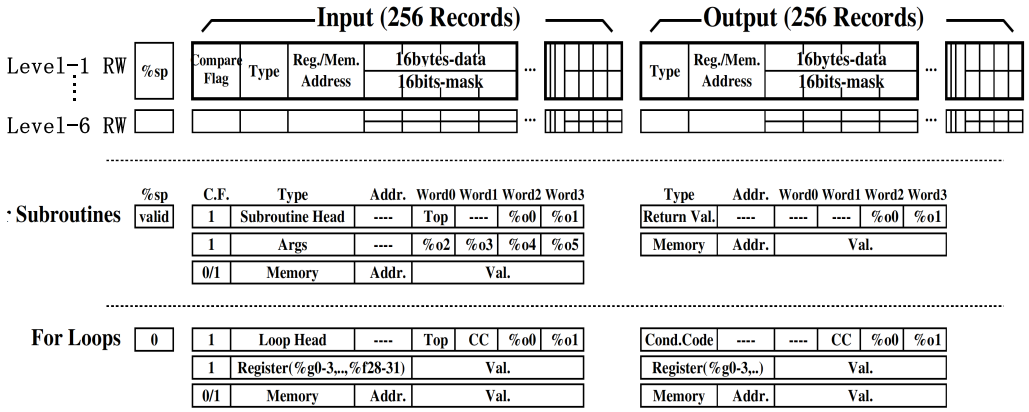

図 5: RW の構成

分岐命令を検出した時点でループが完結する．

3.3 一時記録機構

本節では，入出力を記録する一時記録機構 (RW) について述べる．

ループの場合，レジスタ参照や load のうち，自身が上書きする前の読み出しについては，レジスタ番号や主記憶アドレス，および各読み出し値の全てを入力として記録する．また，書き込みについては最終値が残るように逐次記録し，入力に対する上書きの検査にも用いる．

関数呼び出しについても同様に入出力を記録する．ただし，スタックフレーム上に配置する内部変数は，初期化後に読み出し，関数終了時に捨てる．SPARC アーキテクチャでは，大域変数は命令領域に続く低位アドレスに配置し，スタックフレームは高位アドレスから低位アドレスに向かって伸びる．大域変数とスタックフレーム下限の境界は OS が静的に決定すること，また，スタックフレーム間の境界は関数呼び出し直前のスタックポインタの値により決まることを利用して，あるアドレスが大域変数であるか，またはどの関数の局所変数であるかを識別できる．さらに SPARC アーキテクチャにはローカルレジスタの規定があり，同様に記録を除外できる．

命令区間実行中に入力を記録する際には，既に出力側に登録されているかどうか検査する必要がある．重複登録を避けるためには入力側の検査も必要である．出力についても，既に出力側に登録している場合には上書きしなければならない．

図5にRWの概要を示す．命令区間の種類（関数またはループ）に応じてType部にレコードの種別を格納し，各Typeに従って残りのフィールドに値を格納する．

RWは，現在実行中の最内命令区間（レベル1）から最外命令区間（レベル6）のそれぞれについて一組の入出力を時系列に記録する構造である．より内側に新たな命令区間を検出した場合は，最外命令区間の記録（レベル6）を破棄し，登録中のレベル2〜6に読み替え，空いたRWをレベル1として使用するリング構造としている．入力側（in），出力側（out）の総容量をそれぞれ一次キャッシュ程度の32KBに抑えつつ，入れ子関係にある6重の命令区間の入出力を記録するためには，1つの命令区間あたり約5KBが利用可能となる．1Byteごとに1bitのマスクビットを用意するとして，データに利用できるのは4KBである．16Byteを1レコードとして，連続するレジスタまたは主記憶アドレスの内容を記録する．各レコードには先頭アドレス番号または先頭アドレスを付加する．主スレッドについては以上の機構により，命令区間を実行しながら入出力をRWに記録する．

なお，投機スレッドは実行結果が保証できないため，キャッシュを経由した主記憶への書き込みはできない．命令区間の入出力に矛盾が生じないためには，ストア後のロードはキャッシュを参照してはならない．また，自身がストアしない限り同一アドレスからロードする値は同じでなければならない．前述のように，ループについては代わりにRWoutを出力先として利用できるものの，関数は内部変数の格納場所としてRW領域を利用できないため，一次キャッシュと同程度のローカルメモリを用意する必要がある．以上をまとめると，投機スレッドは，ローカルメモリ，RW，一次キャッシュを以下の優先順に参照しなければならない．

(1) ローカルメモリ: 内部変数を再利用対象としないためには，手続きが参照する内部変数はローカルメモリへ，大域変数および上位の命令区間が使用するスタックフレームの参照は入力としてRWへ，それぞれ振り分ける必要がある．このためには，図6に示すように，上位の命令区間が使用するスタックフレームを避けて，OSが静的に決定する大域変数とスタックフレーム下限の境界（LIMIT）にローカルメモリを割り付ける．さらにローカルメモリの最上位アドレスを投機スレッド開始時のスタックポインタ初期値とすることにより，この問題を解決できる．投機対象の最外区間がループの場合，フレームは作成せず，

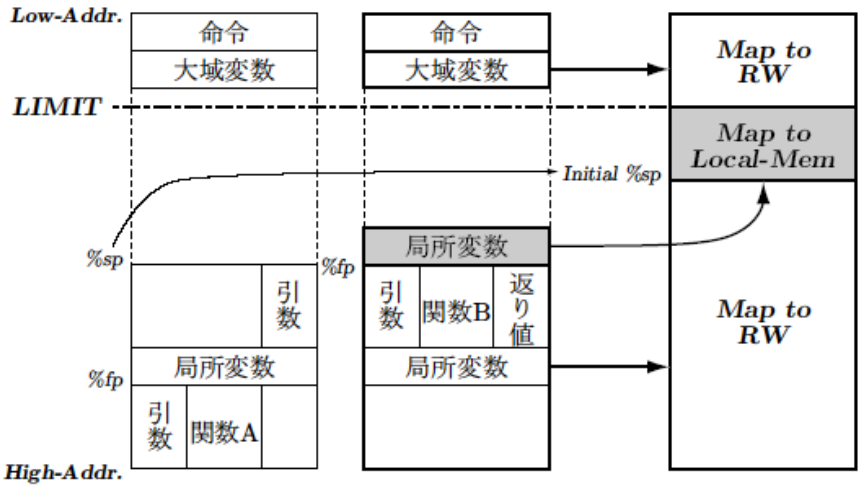

図 6: 投機スレッドにおけるスタックフレームと記録先の関連付け

正常なプログラムである限り，ローカルメモリに該当する領域（LIMIT からスタックポインタ値まで）のアドレスを使用することもない．

(2) レベル 1 の RW$_{\text{out}}$**:** ローカルメモリ範囲外からの load は，自身が書き込んだ値を最優先するために，レベル 1 の RWout を優先する．

(3) レベル 1 の RW$_{\text{in}}$**:** RWout にない場合は，過去に一次キャッシュから読み出して RWin に登録したものを優先する．

(4) 上位 RW: 以上を最高レベルまで繰り返す．

(5) 一次キャッシュ: いずれの RW にもない場合は，命令区間にとって初めての参照であるため，一次キャッシュを参照して RWin に登録する．

また，入出力セットがどの命令区間に属するかを識別するために，区間表 RF を用意する．RF は命令区間の先頭アドレスを格納し，256 種類程度の区間を管理することを想定している．

3.4 入出力の木構造

前節に述べたように，命令区間の入力とは，レジスタや主記憶アドレスの参照により得られる値であり，出力とは，レジスタや主記憶アドレスへ格納される値である．命令自身が変更されない限り，同じ入力に対して同じ実行結果となり，参照順の入力が異なる命令以降について実行結果が枝分かれしていく．すなわち，命令区間の入力セットは，図 7 に示すように，レジスタ番号や主記憶アドレスがノード，各内容が枝となる多分木中の 1 つのパスとして表現できる．

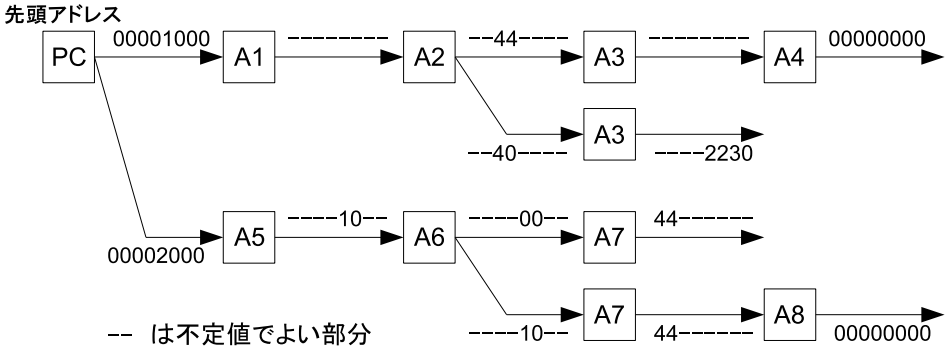

図 7: 入力セットの木構造

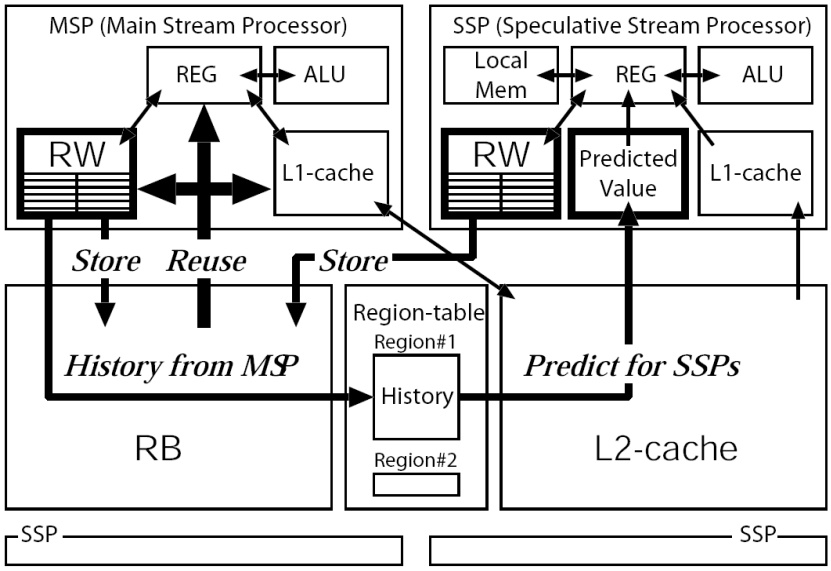

図 8: SpMT のハードウェアモデル

過去に出現した入力および予測入力をこのような木構造に格納すれば，可変長の入力セットに対応でき，枝に相当する記憶領域を節約できる．命令区間を認識すると，区間の識別子から木構造の根を選択し，各ノードに記録されたレジスタ番号または主記憶アドレスから現在の値を読み出し，複数の枝の中から値が同一である枝を順に選択することを繰り返す．最終的に末端に到達した場合，対応する出力を再利用する．

3.5 ハードウェアモデル

これまでに述べた機構のハードウェアモデルを図 8 に示す．主スレッドを担当する MSP および投機スレッドを担当する複数の SSP が，再利用バッファ（Reuse

Buffer）および二次キャッシュを共有する．Region-table では，ストライド予測により，MSP が実行あるいは再利用した命令区間の入力履歴から予測値を生成し，SSP 起動に間に合うように各 SSP の Predicted Value 領域へ送る．予測対象はレジスタ，定数アドレス，フレーム内定数アドレスである．RWin を再利用バッファへ蓄積する際，同時に入力履歴として RF に格納する．入力履歴は RWin の 1 行分が時系列に 2 セット並んだ FIFO であり，フラグを立てたレコード単位にストライド予測を適用して予測値を求める．予測値のレコードも RWin と同様に参照順に並ぶため，SSP は全予測値の転送を待たずに投機実行を開始できる．

SSP の load 命令は，Predicted Value 領域の予測値を優先的に使用し，RWin に登録する．以降は前述のように，(1）RWout，(2）RWin の優先順に参照するので，SSP から見た主記憶空間は他プロセッサの干渉を受けない．MSP および SSP は，各命令区間の入出力を各 RW へ記録し，命令区間実行完了時に再利用バッファへ送る．MSP は，後方分岐命令および関数呼び出し命令の検出と同時に RB の連想検索を行い，再利用可能なパスが存在する場合には，再利用バッファの出力値をレジスタおよび主記憶アドレスに書き込む．

第4章　再利用バッファの構成と動作

本章では，再利用バッファの構造を説明し，汎用連想メモリで構成した場合の問題点を述べる．

4.1　構成と動作の概要

命令区間の実行が終了すると，RW に記録した入出力セットを，再利用バッファ（RB:Reuse Buffer) に格納する．再利用バッファの構成を図9に示す．RBin は CAM 部および RAM 部から構成される連想検索装置である．RBin の CAM 部に各枝，RBin の RAM 部に各ノードが対応する．また，入出力セットがどの命令区間に属するかを識別するために，区間表 RF を用意する．RF は命令区間の先頭アドレスを格納し，命令バッファに格納した区間を管理することを目的している．

RBin の CAM 部の各エントリは，当該エントリの親ノードを指すインデクス（Key）と，入力値 (Val.) およびそのマスク (Mask) を格納する部分からなる．RBin の RAM 部の同一エントリが，次に参照すべき主記憶アドレス（next addr.），およびそのアドレスに対する書き換えが発生したか否かを記憶するフラグ (cont）を保持する．MSP は命令区間の実行開始時に RBin を参照し，入力セットが完全に一致するエントリが見つかった場合，当該エントリに対応する出力を RBout から一次キャッシュおよびレジスタに書き戻す．これにより，命令区間の実行を省略できる．SSP は，再利用バッファのこれまでの入力セットから予測される入力セットを受け取り，投機実行を行う．投機実行の結果得られた入出力セットは，MSP と同様に命令区間の実行終了時に RW から再利用表本体に登録する．

4.2　具体例

図 10 に RW への登録例を示す．Strlen(str) は，NULL 文字により終端した文字列引数のバイト数を数える関数とする．この関数を命令区間として実行した結果，引数に文字列 ”ABCDEF ”が与えられた場合と，”ABCDEFG ”が与えられた場合について，命令区間の入力および出力を記録すると，図 10 の下段に示す RWin および RWout が得られる．この 2 つの記録を木構造として表現したものを上段に示す．

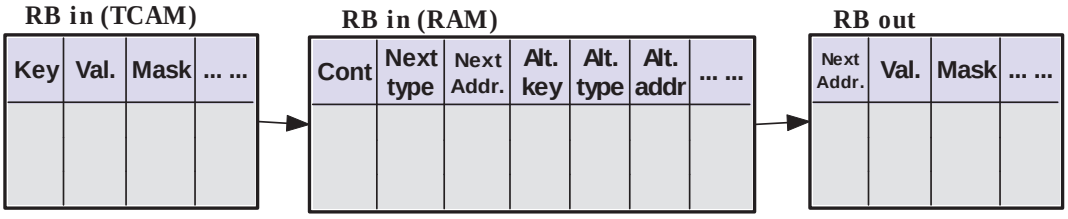

図 9: RB の構成

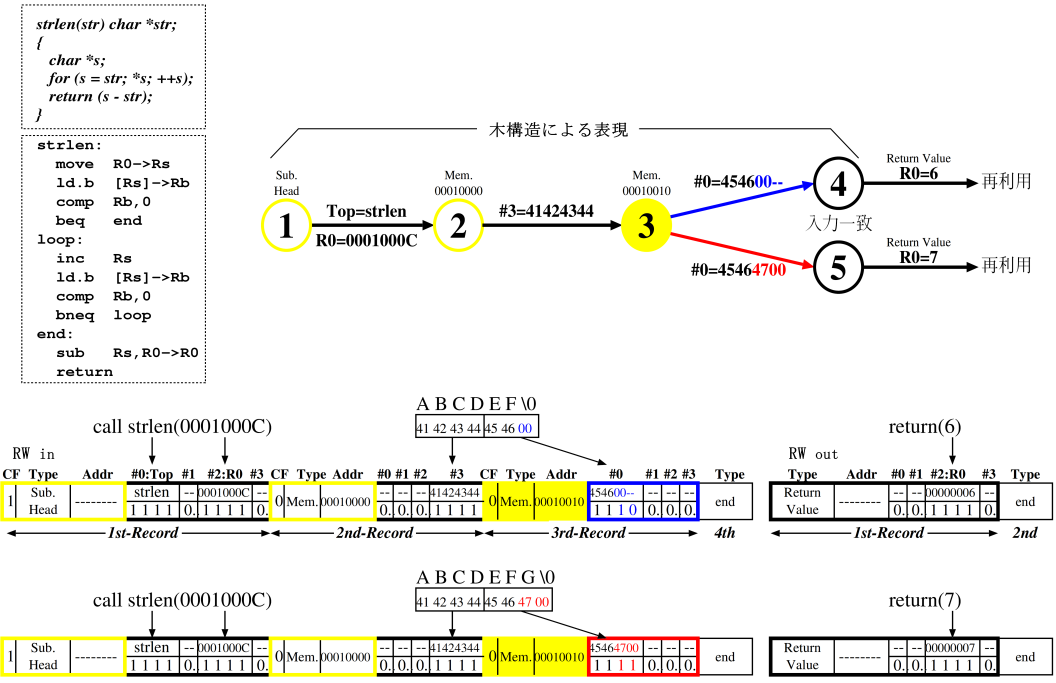

図 10: RW へ登録

図 11 に示すように，関数 strlen(”ADCDEF”) が実行される場合に，まず，第 1 引数に対応するレジスタ R0 から文字列の先頭アドレス (00010000 と仮定) を読み出してローカルレジスタ Rs に複写する．Rs が指すアドレスに NULL 文字を検出するまで Rs を 1 ずつ増加させ，文字列長 Rs-R0 を改めて R0 に格納し手続きを終了する．RWin には，第 1 レコードに手続きの先頭アドレスおよびレジスタ R0 の内容 00010000，第 2 レコードにアドレス 00010000 と 7 バイト値 ”41424344454600 ”，第 3 レコードに終端を記録する．RWout には，第 1 レコードにレジスタ R0 の内容 6，第 2 レコードに終端を記録する．

手続き終了時には，各 MSP/SSP の RW に一時的に登録しておいた入出力セットをまとめて再利用表に登録する．各エントリにタイムスタンプを設け，参照

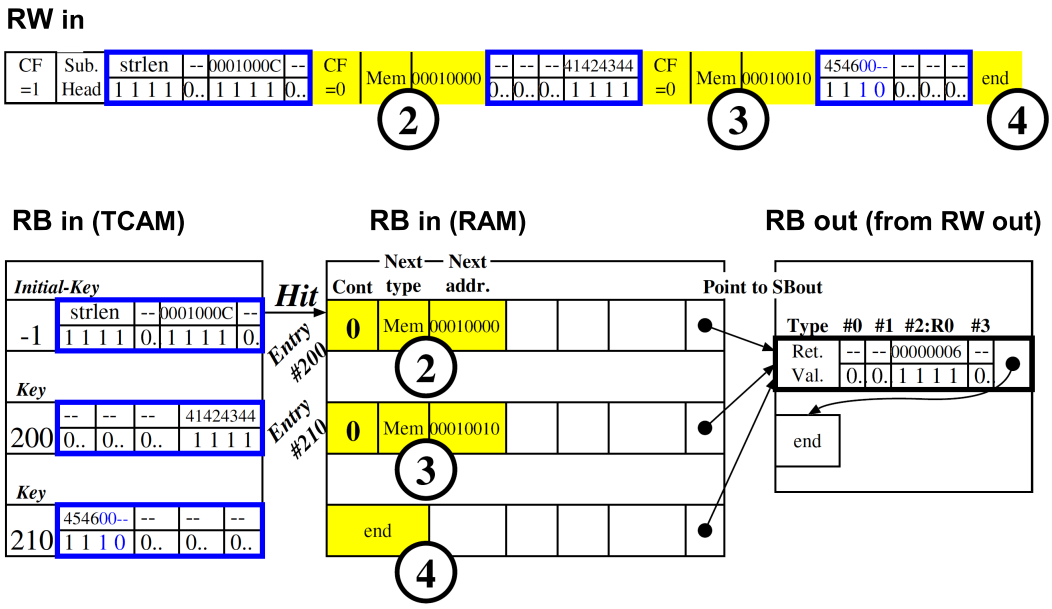

図 11: RB へ登録

したパスに属するエントリについてはタイムスタンプを更新し，古いタイムスタンプのエントリを定期的に消去することにより，長期間使用しない枝を刈り取ることができる．RBin から空きエントリを検索し，空きエントリがなければ，タイムスタンプの最も古いエントリを一括削除する．空きエントリが見つかれば，RW の各レコードを 1 ノードとして順に登録する．

登録の結果，ある一つの命令区間に対する入力セットのパターンは，図 10 上のような木構造を形成する．ノードは参照すべき入力アドレス，枝はその格納値であり，根から葉までの経路数が，命令区間の入力セットのパターン数となる．再利用表上では各枝と次に参照すべきアドレスを 1 エントリとする．RBin の CAM 部に当該枝の値と親エントリを指すインデクスを保存し，RBin の RAM 部の同一インデクスに，次に参照すべきアドレスを登録する．ただし，親が根である場合は-1 とする．

以後 strlen(str) を呼び出す前に初期キー (-1)，strlen の先頭アドレスおよびレジスタ R0 の値により RB の CAM 部分を連想検索し，エントリ番号 (200) にヒットした場合はさらに RAM 部分の nextkey(200) および nextaddr から読み出した主記憶値を用いて RB の連想検索を繰り返すことにより，木構造から 1 つのパスを選択する．各バイトごとに検索マスクを設けているため，文字列長が異なっていても正しく検索できる．検索を繰り返し，RAM 部分に終端 (end)

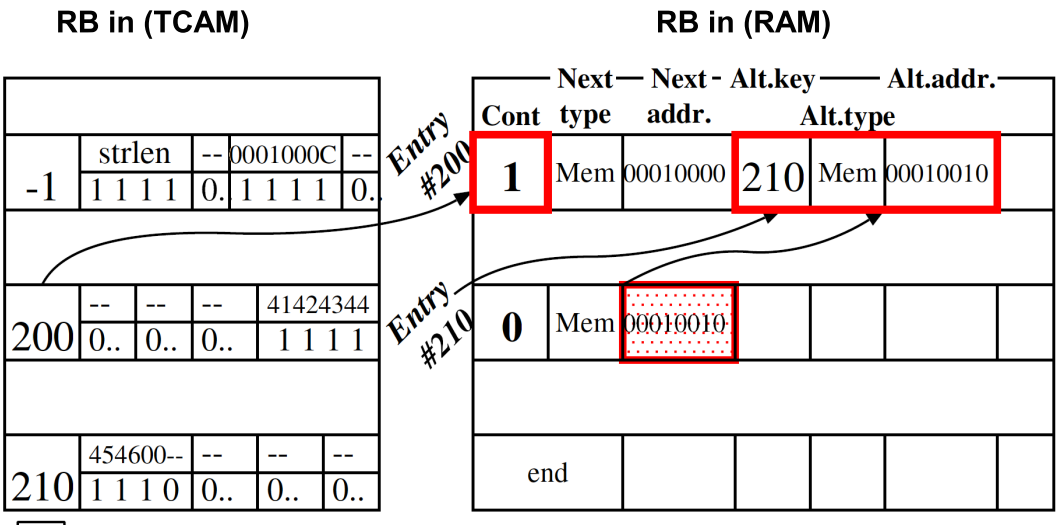

図 12: 主スレッドがストアを実行

を検出した時，対応する RBout が正しい出力値である．

また，図 11 に示すように，連想検索のオーバーヘッドを下げるために，比較フラグ cont を設ける．RBin に記録する際には，不要な比較を行わないよう cont フラグを 0 にリセットする．当該アドレスを更新するまでの間，第 1 レコードの一致のみにより cont=0 に到達し，直ちに命令区間を再利用できる．

MSP が文字列 ”ABCDEF ”を ”ABCDEFG ”に変更する時，すなわち主スレッドが ”G\0 ”を 10012 にストアする時，図 12 に示すように RBin に対して変更が行われる．RBin の CAM 部を連想検索し，更新した 10010 と一致するエントリ (210) から当該エントリの key 部 (200)，type 部 (mem)，addr 部 (10010) を取り出し，key 部により指定されるエントリ (200) の cont に 1 をセットし，alt.key，alt.type，alt.addr にそれぞれ 210，mem，10010 を格納する．以後，strlen(str) を再利用する際には，第 1 レコードの一致により cont=1 に到達し，本来の key/type/addr の代わりに，alt.key，alt.type，alt.addr を用いることにより，第 2 レコードを飛び越えて第 3 レコードを検査でき，連想検索回数を抑制できる．

図 13 は，strlen(”ABCDEFG”) が実行される時，RWin および RWout に再度記録される過程を示している．その後，当該 RWin および RWout を RBin および RBout に追加登録する．追加登録は，まず RBin の CAM 部を連想検索し，RW におけるアドレス 00010010 の内容が既に RBin に登録した内容と異なるので，nextkey を 210 とする新たなエントリを追加する．従って入力パターンが

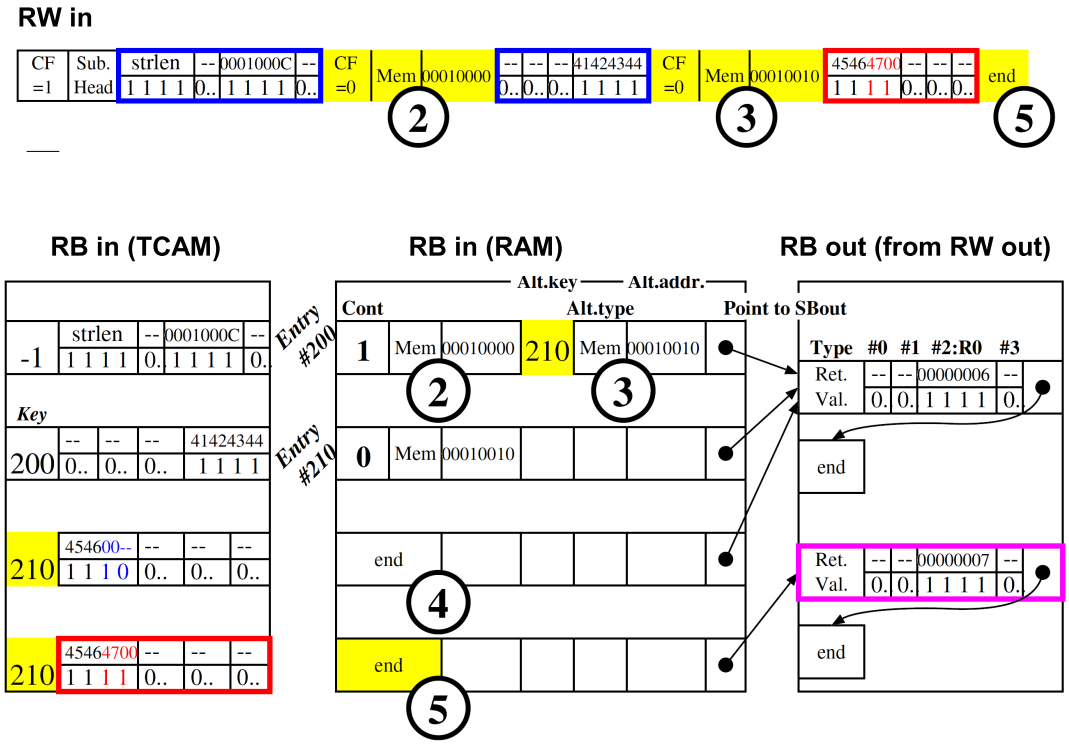

図 13: 主スレッドが再度 strlen(0001000C) を実行し，新しい RBin を登録

RBin に多分木構造を生成できるのである．

さて，連想検索装置により構成する RBin に必要な機能は次の通りである．

Search & Write 操作: RBin の内容を RWin に登録する際には，RBin の連想検索を行い，既登録であるか否かを判定し，既登録である場合には登録を行わず，未登録の場合には，連想メモリの空きエントリを選択して書き込む．

Search & Read 操作: MSP が RBin を検索する際には，RBin の連想検索を行い，既登録であるか否かを判定し， 既登録である場合には当該 RBin エントリに属する連想メモリ (CAM) および通常メモリ (RAM) を読み出し，未登録の場合には，検索を終了する．

4.3 汎用連想メモリを用いた場合の問題点

再利用バッファは，連想メモリ (CAM) と通常メモリ (RAM) を RBin および RBout として構成することを想定している．本節では汎用連想検索メモリを用いた場合の問題点について述べる．

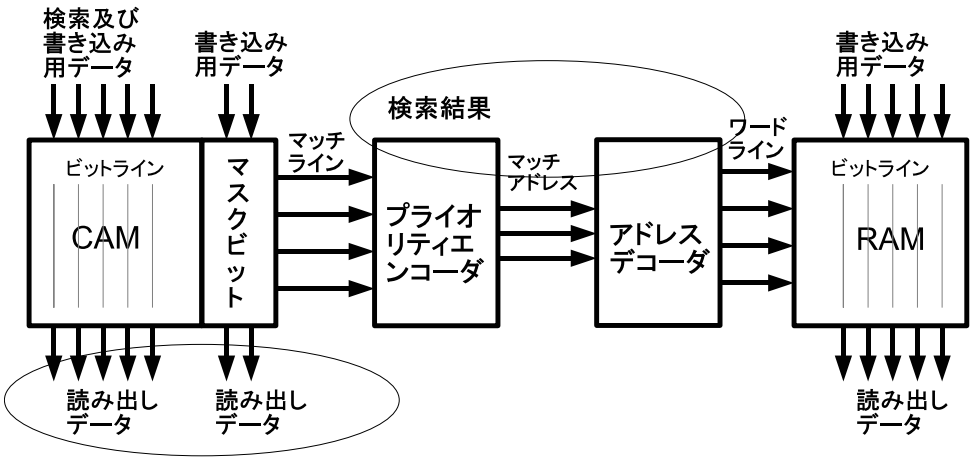

図 14: 汎用連想検索装置

汎用連想メモリを用いる構成 (図 14) では，CAM の検索の結果，複数のマッチラインがアサートされ得るために，アサートされたマッチラインのうち唯一のマッチアドレスを出力するためのプライオリティエンコーダを設け，最優先ラインに対応するマッチアドレスのみを生成する必要がある．その後，改めて当該マッチアドレスを用いて RAM を参照する構成とすることにより，連想検索装置全体におけるマッチアドレスの競合を防止できる．しかし，プライオリティエンコーダは，マッチライン数の増加とともに長い処理時間が必要となる．このため，深い CAM，すなわち，多くのマッチラインを有する CAM を構成する際には，プライオリティエンコーダがボトルネックとなり，高速化が困難となる．プライオリティエンコーダ自身の消費電力も問題となる．

また，再利用バッファを連想検索する時，後続命令区間の入力パターンを予測するために，検索の結果一致した CAM の内容を予測履歴表に格納する必要である．しかし，汎用 CAM のビットラインは検索と読み出し機能を兼ねることができないので，検索の結果を読み出す際に改めて読み出しサイクルを設ける必要がある．以上により市販の汎用連想メモリを再利用バッファとして使用する場合は，入出力パターンの検索と書き込み，検索と読み出しは別サイクルとなる．前節に述べた再利用バッファの登録および検索操作に数サイクルを必要とするため，高速化が困難である．

第5章　連想検索装置の設計

本章では，前述した問題を解決するために専用連想検索装置の構成を検討し，各部分の設計を進める．

5.1　構成

前章に汎用連想検索装置の問題点を検討した．当該問題点を解消するために，専用連想検索装置（図 15) を提案する [29].

専用連想検索装置では，Search & Write 機能，すなわち，指定した書き込みデータが登録済であれば登録せず，未登録であれば空きエントリに対して高速に登録し，唯一のマッチラインがアサートされることを保証する機構を設けることにより，汎用連想メモリのプライオリティンコーダを省略し，高速化を図る．また，Search & Read 機能，すなわち，検索の結果，一致した CAM の内容を直接読み出す代わりに，RAM が保持する CAM と同じマスクビットパターンを同時に読み出し，検索データ自身と組み合わせることにより，間接的に連想メモリの内容を読み出す機構により，高速化を図る．

専用連想検索装置の全体構成を図 16 に示す．左から順に，CAM 部，V ビット検索部 (CV)，制御部 (GATE)，V=0 空きエントリ検出器 (ENC)，V ビット読み出し部 (RV)，RAM 部である．検索データは，SD/XSD および V/XV により CAM 部および V ビット検索部に与えられ，検索結果が MATCH 信号により GATE 部に入力される．同時に，V ビット読み出し部から V ビット情報が空きエントリ検出器に与えられ，V=0 であるエントリが 1 つ選択される．制御部は以上の情報に基づいて，検索データに一致するエントリが存在する場合には RAM 部に対して Read 動作を指示し，存在しない場合には，CAM 部および RAM 部に対して Write 動作を指示する．専用連想検索装置では，Search & Read および Search & Write が各々1 サイクル内に動作することを目標とした．

さらに，再利用バッファをブロック化し [30]，CAM を深さ方向に分割し，登録または検索するデータの一部を用いて検索範囲を自動的にしぼり込むことにより，低消費電力と高速動作の両立を図る．

5.2　動作

本節では，専用連想検索装置における登録および検索動作について説明する．

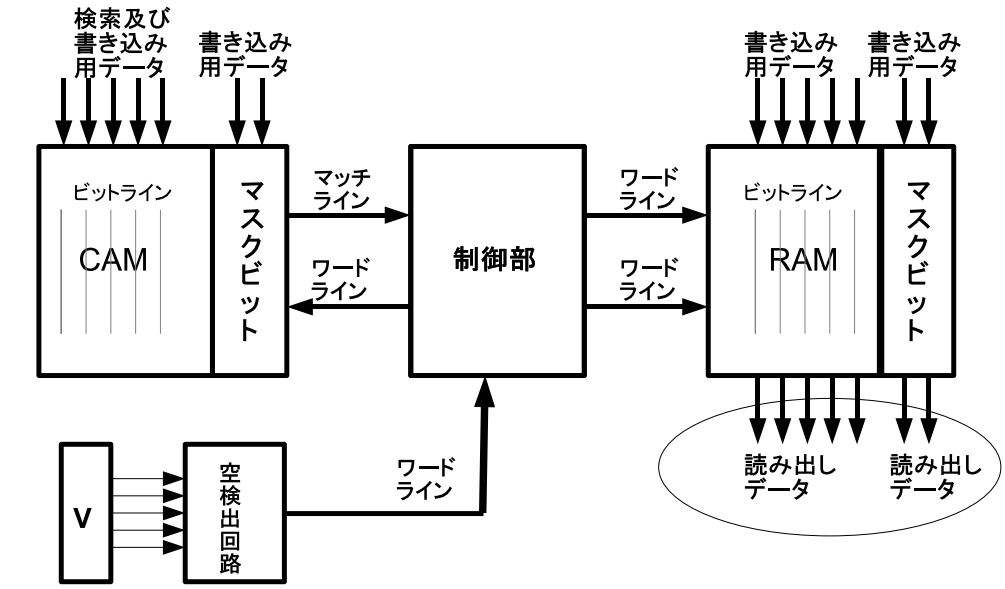

図 15: 専用連想検索装置の概要

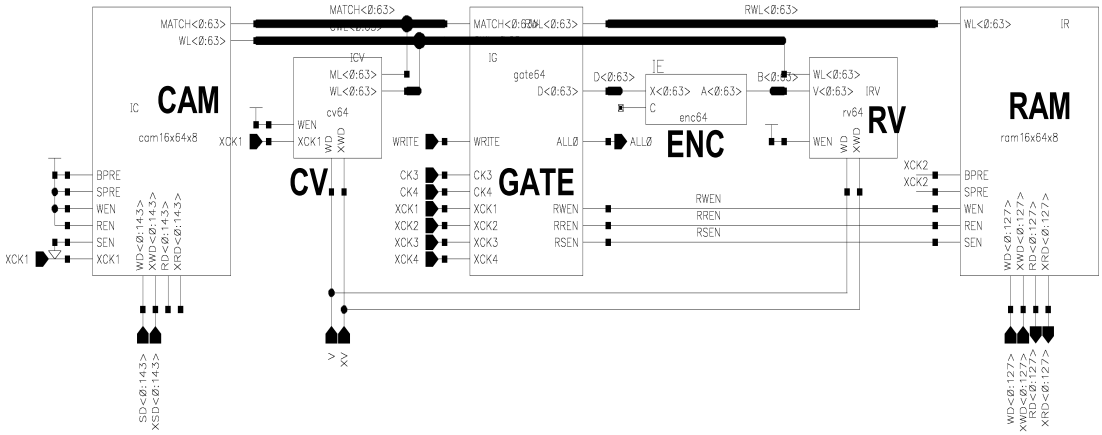

図 16: 各ブロックと信号線名

図17は，RWの各レコードを再利用バッファに登録する際のSearch & Write動作のタイミングチャートである．サイクル１Ａ前半では，CAM部の連想検索に備えて，CAM部の全マッチラインが１にプリチャージされる (MATCH Precharge)．同様に，RAM 部の読み出しに備えて，RAM部の全ビットラインが１にプリチャージされる (RAM-BL-precharge)．また，未登録であることが判明した際の空きエントリを１つ準備するために，VALID ビットが0である全エントリの中から最も優先順位が高い空きエントリを探す動作が，後述のENC64にて開始される (V=0:priority-0-detector)．本動作にはプライオリティエンコーダと同様に相当の時間を必要とするので，サイクル１Ａにて開始しておく．

サイクル１Ａ後半では，書き込みデータをCAM部のSD/XSDおよびRAM部のWD/XWDに与え，CAM部については連想検索を開始すると同時に，全てのマッチラインが0すなわち一致するエントリが皆無である（ALL0が高電位）ことを調査するために，GATE部のALL0信号のプリチャージを開始する．

サイクル１Ｂでは，連想検索の結果に基づき，MATCHが高電位であることが判明した行に対応するRAM部のWord-Line（RWL）を駆動してRAMの読み出しを開始し，RBの次エントリの内容と比較して検証を行う．この時，実行しているプログラムが正常なプログラムである限り検証が失敗することはない．RAM読み出しのためのセンスアンプに必要なRAM-READ-ENABLE信号(RREN)は，ALL0信号をそのまま用いてオンにすることができる．引き続きサイクル２Ａでは，サイクル１Ａと同様，次のCAM部の連想検索に備えて，全マッチラインが１にプリチャージされる．同様に，RAM部の読み出しに備えて，RAM部の全ビットラインが１にプリチャージされる．また，未登録であることが判明した際の空きエントリを１つ準備するために，V ビット（V ビット=0時，対応される行が空いている) が0である全エントリの中から最も優先順位が高い空きエントリを探す動作を開始する．サイクル２Ｂでは，連想検索の結果に基づき，MATCHが低電位である場合には，RAMを読み出さず，１つもマッチしない場合にはALL0信号が１となることを受けて，あらかじめ探しておいた空きエントリに関するCAMおよびRAMのWL(CWL/RWL) をオンにし，当該空きエントリに次RBの内容を書き込む．同時に，SD/XSD/WD/XWDに用意しておいた書き込みデータをCAM/RAM内のビットラインに伝えるために，ALL0 信号を受けてセンスアンプのWRITE-ENABLE信号をオンにする．

図18はRBの検索および読み出し動作に関わるタイミングチャートである．

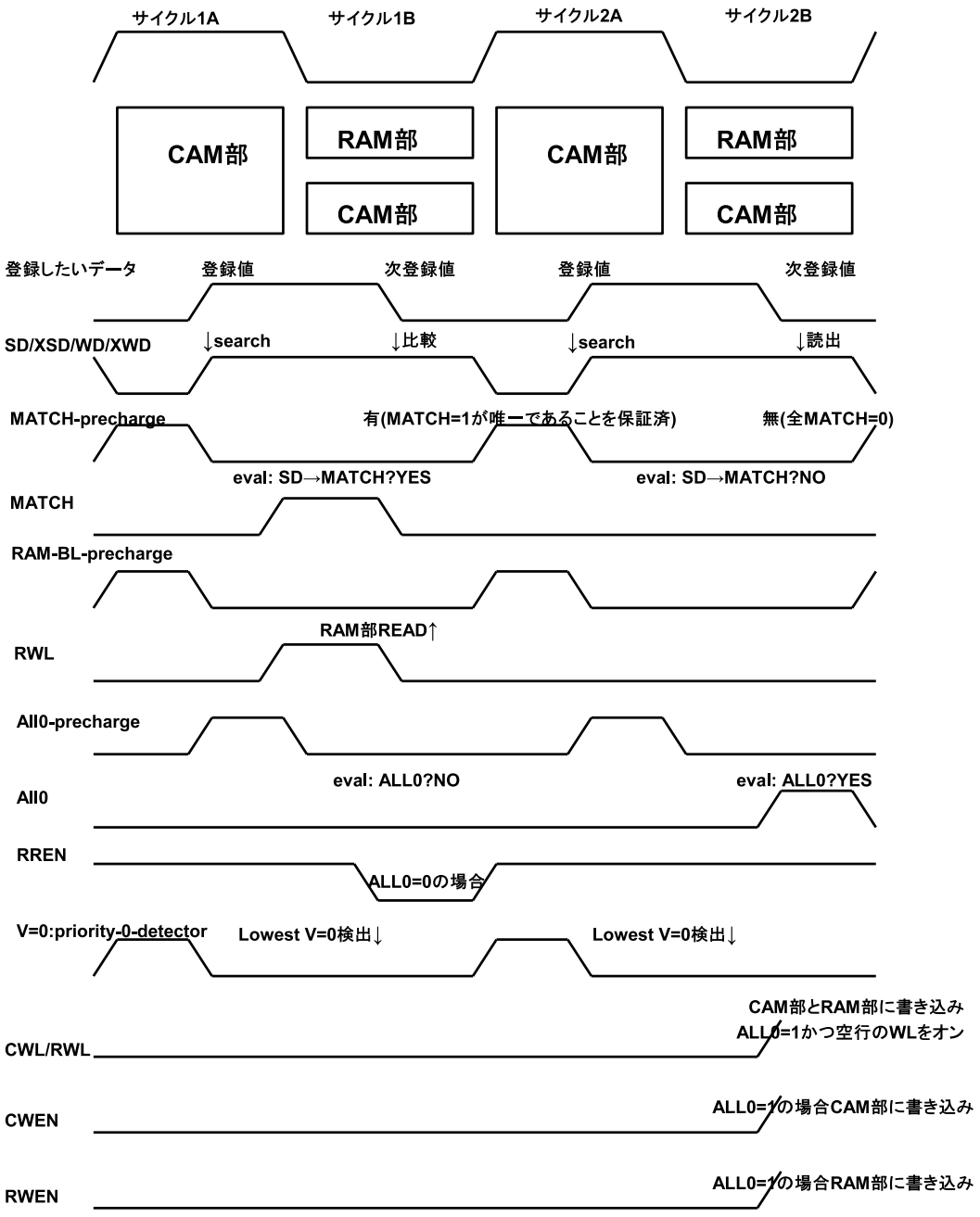

図 17: Search & Write のタイミングチャート

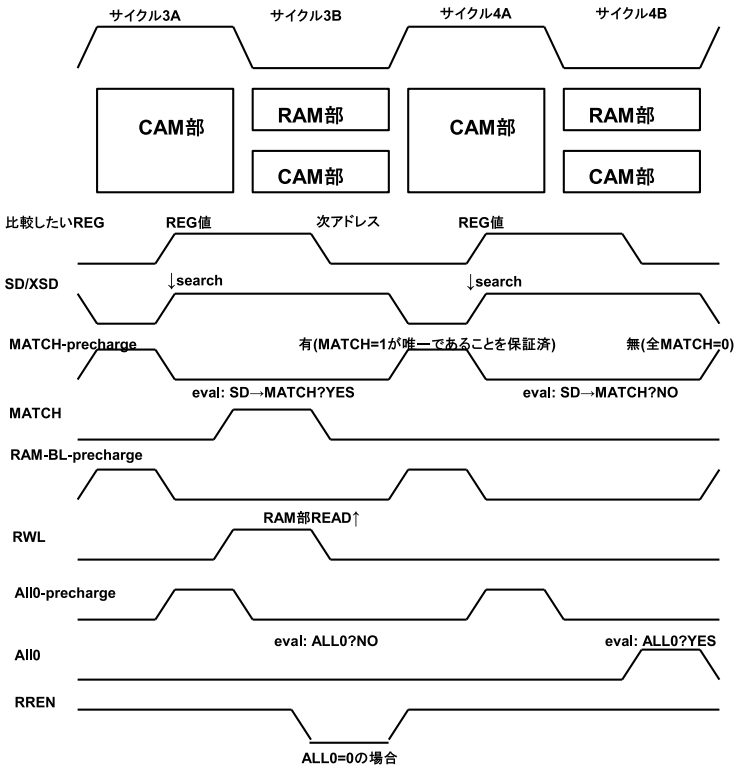

図 18: Search & Read のタイミングチャート

サイクル３Ａでは，CAM 部の連想検索に備えて，全マッチラインを１にプリチャージする．また，RAM 部の読み出しに備えて，RAM 部のビットラインが１にプリチャージされる．

サイクル３Ａ後半では，検索したい現レジスタ等の内容を CAM 部の SD/XSD に入力し，サイクル３Ｂに既登録であれば RAM 部が読み出される．RAM 部には CAM 部のマスクビットパターンが格納されており，検索したい現レジスタ等の内容と合わせて，CAM 部の直接読み出しに替えることができる．引き続きサイクル４Ａでは，CAM 部の連想検索に備えて，全マッチラインが１にプリチャージされる．また，RAM 部の読み出しに備えて，RAM 部のビットラインを１にプリチャージする．同時に，前回の検索の結果 RAM 部から得た，次に比較すべきレジスタ等のアドレスを元に，現レジスタ等の読み出しを開始する．サイクル４Ａ後半では，次に比較すべき現レジスタ等の内容を CAM 部の SD/XSD に入力し，未登録であればサイクル４Ｂにおいて全 MATCH が 0 となるので，これを検知して，検索を終了する．

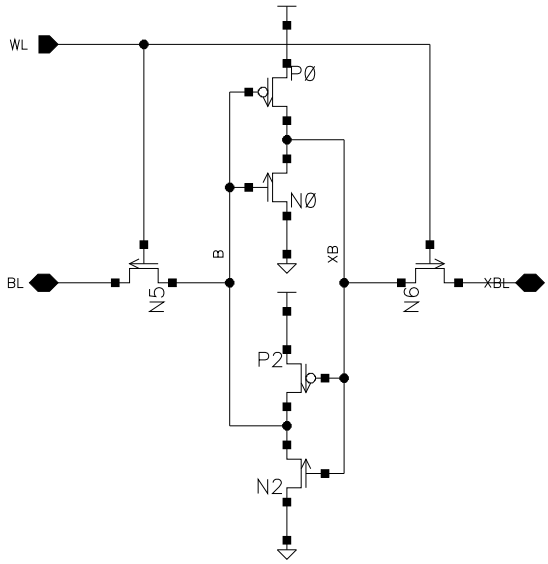

図 19: SRAM Cell

5.3 メモリ部の設計 (CAM，RAM)

RBin の CAM 部分は，最も古いエントリを消去する時に使用するタイムスタンプ tsid，RF から命令区間を削除する時に使用するインデクス rfid，親ノードの RBin インデクス key，および入力データ val からなり，幅 128Byte の CAM に格納できる．RBin の RAM 部は，CAM 部と同様の tsid と rfid，入力アドレスに対し書き換えがあったか否かを示す値 cont，レジスタかメモリのどちらからの入力であるかを示す値 type，次に参照すべきアドレス addr.，次の入力となる値のマスク値 valm からなり，幅 128bit の RAM に格納できる．RBout は RBin と同様の tsid と rfid からなり，幅 128bit の RAM に格納できる．

本節では CAM，RAM およびセンスアンプについて述べる．

5.3.1 RAM セル

RAM 部は命令区間の入力値のアドレスを保存する装置である．図 19 に示すように，6T SRAM Cell を採用した．

RAM Cell に書き込む場合には，1) 入力 DATA を BL 線上に置き，XDATA を XBL 線上に置く；2) 次に，書き込む目標行に対応する Word-Line に高電位を加え，N5 と N6 のゲート側に 1 を入力すると，BL および XBL が RAM Cell に接続される；3) 入力データが RAM Cell に保存される．

RAM Cell から読み出す場合には，1)BL および XBL を高電位にプリチャージする；2) 次に Word-Line に信号が加えられる；3) Cell 中のプルダウントラン

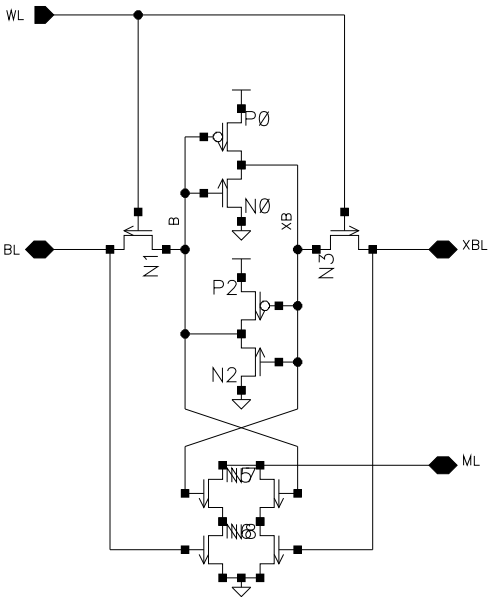

図 20: CAM Cell

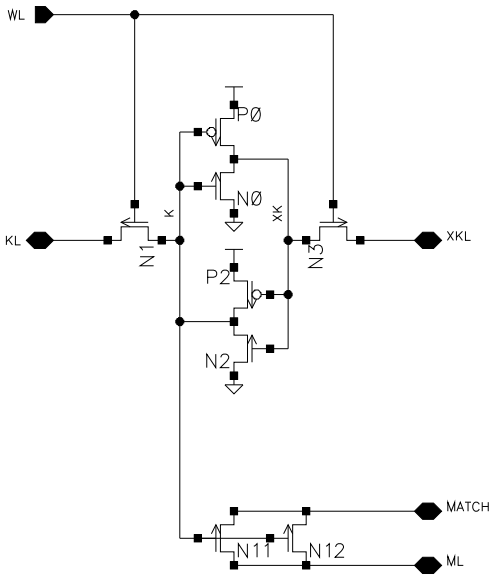

図 21: Mask Bit

ジスタ (N2，N0) によって，BL または XBL が放電される．例えば，B が高電位であり，XB が低電位である場合に，XBL 側が 0 に放電される．

5.3.2 CAM セル

図 20 に示すように，CAM Cell は 10 個のトラジスタにより構成される．正負 1 組のビットラインは書き込みデータと検索データの両方を伝える．

CAM は，まず，マッチラインを 1 にプリチャージした後に，Search DATA を BL および XBL 上に置くことにより検索を行う．一致しない Cell が存在する場合に，マッチラインが 0 に引き下げられる．図 21 が，CAM のマスク部である．1 バイトの CAM ごとに 1 ビットのマスクを設け，当該バイトを比較対象とするかどうかを判断する．マスクが 1 であればマスク部のマッチラインがデータ部のマッチラインと繋がり，当該バイトのデータが比較される．マスクが 0 であればマスク部のマッチラインはデータ部のマッチラインと繋がらず，当該バイトは比較されない．

CAM に登録する時，16 バイトを 1 レコードとして，連続するレジスタまたは主記憶アドレスの内容を記録する．従って，CAM の横幅（1 行分の長さ）は 16 バイトとなる．CAM の検索機能として，横幅が 16 バイトであればマッチライン線が長いため，遅延時間が大きくなる．図 22 は幅が 16 バイトの場合のシミュレーション結果であり，上から各々プリチャージ信号，1bit のみ mismatch

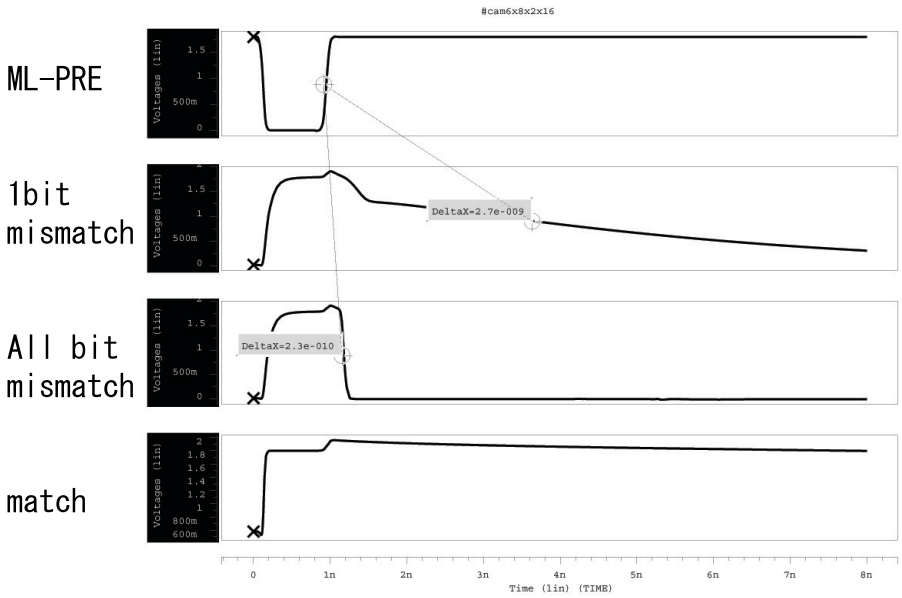

図 22: CAM の検索

である場合のマッチライン，すべての bit が mismatch である場合のマッチライン，Match である場合のマッチラインの波形を示している．このシミュレーションにより，全ビットが不一致である場合は放電速度が速いため，当該マッチラインは 0.23ns で 0 となった．しかし，1 ビットのみ不一致である場合には，マッチラインに充電された大量の電荷は，1 組のマッチトラジスタを通して放電され，当該 bit のマッチトラジスタがボトルネックになり，放電速度が 2.7ns 以上となった．

以上の予備実験の結果，検索速度を速くするために，マッチラインを 8 分割する方式を採用した．2 バイトごとに 1 つのサブマッチラインを設け，8 本のサブマッチラインからマッチラインを作る．分割方式の長所は，マッチライン上に充電される電荷が減少し，不一致である場合の放電速度が速くなる．本構成を図 23 に示す．P5，P6 および P90 によりサブマッチラインをプリチャージし，検索データを BL および XBL に置き，Search 動作を始める．検索データと合わない CAM Cell が存在する場合には，MLA 側が 0 にプルダウンされ，XMATCH 側が 1 になり，N83 を介してマッチラインが 0 になる．

5.3.3 センスアンプ

センスアンプ回路は，メモリデータ読み出し時に生ずる微小な電位差をすばやく検出し，増幅する回路である．つまり高感度と高速性が要求される．本設

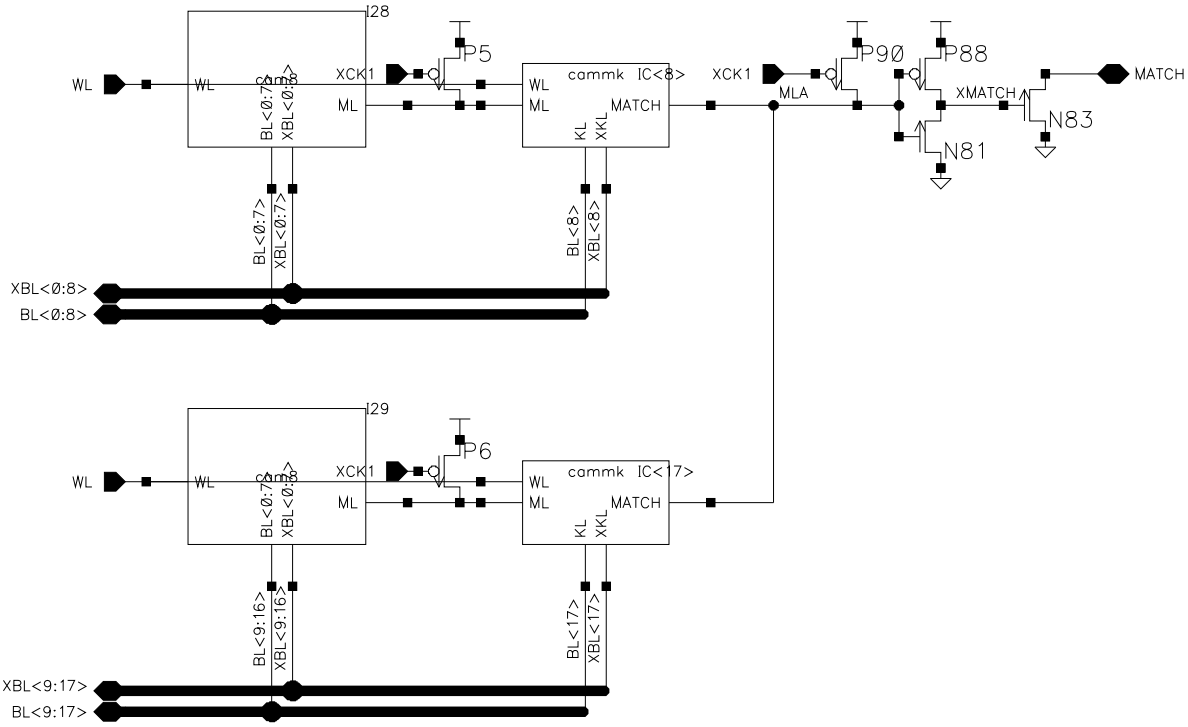

図 23: 2Byte CAM

計はラッチ形センスアンプを採用している．

ラッチ形センスアンプ回路の基本構造を図 24 に示す．センスアンプの制御ノードは Sense Amp Enable(SEN)，Bit Line Precharge(BPRE)，Sense Amp Prearge(SPRE) であり，入力データが WD と XWD，出力データが RD および XRD，アクセス制御が Write Enable(WEN) および Read Enable(REN) である．

書き込み時の動作は以下の通りである．

1) 入力データを WD および XWD に置く．

2) 書き込みを行うメモリセルに対応するワードラインを 1 にする．

3) WEN に VDD 電位を加え，入力データノードをビットラインに接続し，書き込み動作が開始する．

4) 入力データは N9 および N14 を通じビットラインの電位を上げ，プルアップされたワードラインに対応する RAM セルに書き込まれる．

読み出しの場合は，

1) BPRE に 0 電位を入力し，BL および XBL をプルアップトラジスタ P3，P0，P4 を通じて高電位にプリチャージする．ビットラインは長いので（本設計では 64 エントリの場合に bit-Line の長さが 452um である），長いプリチャージ時

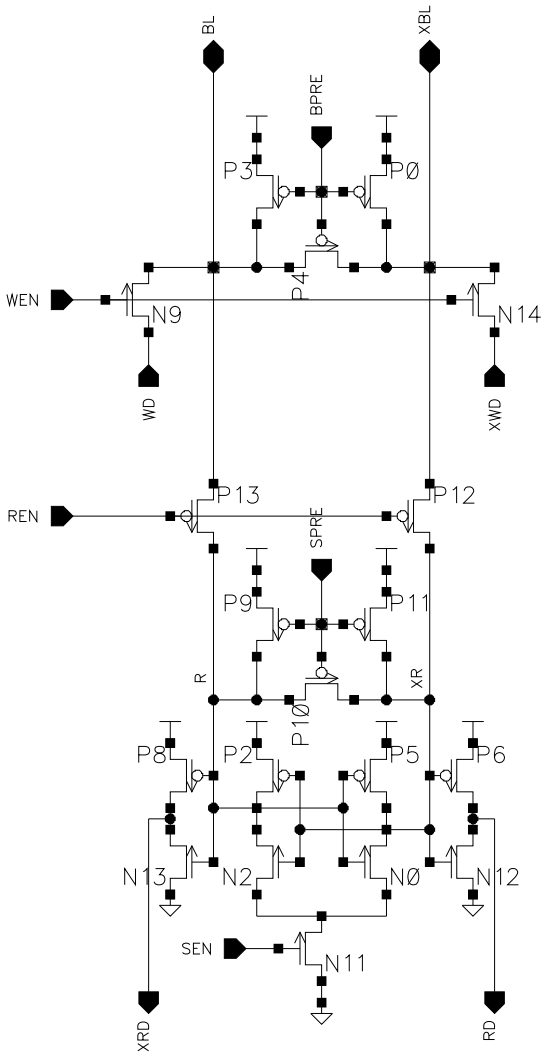

図 24: Sense Amp. Schematic

間が必要となるものの，BPRE トラジスタの幅（W）を増大することによって，プリチャージ動作を速くできる．

2) 同時に SPRE にも 0 信号を入力し，センスアンプの出力線をプルアップトラジスタ P9，P10，P11 を通じる高電位にプチチャージする．

3) 次に読み出す行に対応するワードラインをプルアップし，メモリセルをセンスアンプに接続させる．

4) センスアンプの REN および SEN を on にし，センス動作を開始する．

5.3.4 メモリ性能を左右する要素

図 25 に，CAM の構造を示す．以下にメモリ性能を左右する要素を列挙する．

アドレス デコード遅延 (address decoding latency) アドレスをラッチす

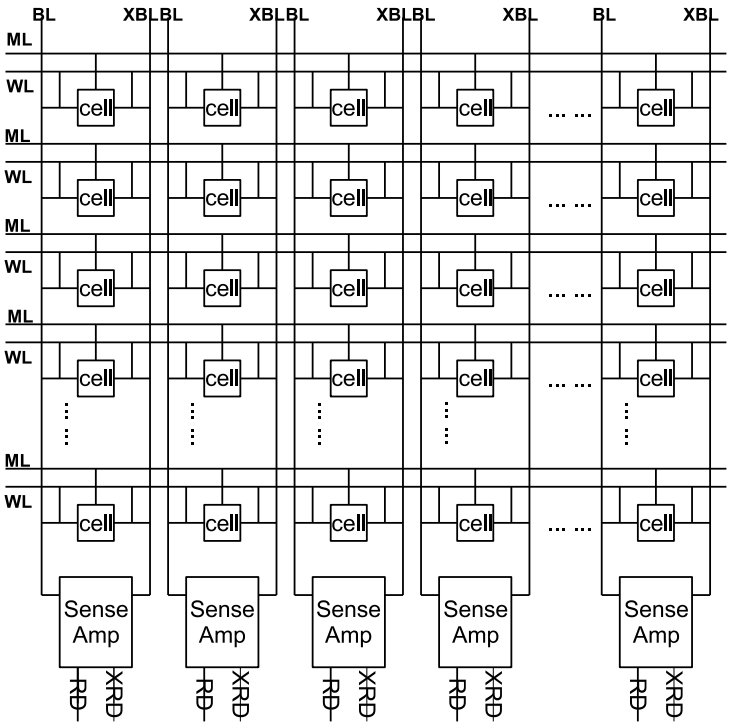

図 25: CAM の全体構成

る時間およびワードラインを選択する遅延時間である．この遅延の原因は，行，列アドレスのマルチプレクシングおよびデコーディング論理の伝播遅延である．

ワードライン遅延（word-line delay）ワードラインをプルアップする遅延時間である．

ビットライン遅延（bit-line delay）センスアンプを通して，メモリセルの内容が読み出されるための遅延時間である．この遅延はビットライン構造，配線の RC 遅延，Cell-to-bit の容量比率およびセンスアンプのトポロジ構造によって影響される．

出力遅延（Output delay）データがセンスアンプから出力パッドまで伝播される時間である．これも RC 遅延である．

本設計では，アドレス デコーダおよびプライオリティエンコーダを省略できるので，アドレス デコーディング遅延は無く，主な遅延はビットライン遅延である．

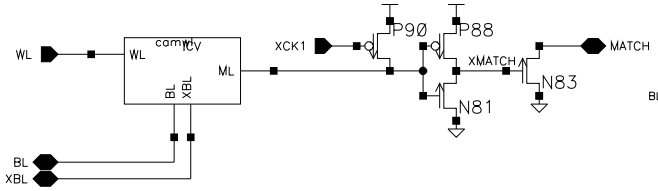

図 26: CV

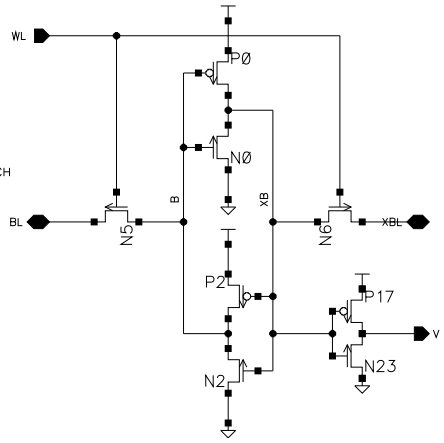

図 27: RV

5.4 空きエントリ検出器

空きエントリ検出器は，V ビット読み出し部において，V=0 を示す複数エントリから１つを選択する回路である．空きエントリ検出器は V ビット検索部，V ビット読み出し部，優先エンコーダから構成される．

CV は V ビット検索部である．V ビット検索部は，有効な CAM エントリのみを検索対象とするために，CAM 検索結果と一体となり，V=1 であるエントリのみを検索するための追加ビットである．V ビット読み出し部（RV）と同じ値を保持するよう制御される．V ビットが 0 である場合，CAM 部の同行が空いてることを表示している．CV セルは図 26 に示すように，構造は CAM セルと同じであり，マッチラインが一つのインバータを通して N83 のゲート側に接続される．N83 のドレインは同行 CAM 部の同行のマッチラインと繋がり，N83 のソースは GND と繋がる．N83 のゲート電位（XMATCH）が１となり，N83 のソースとドレインが導通し，プリチャージされた CAM のマッチラインが 0 にプルダウンされる．これにより，CAM の当該行の検索が省略される．

RV は V ビット読み出し部である，RV Cell を図 27 に示す．XBL 側を一つのインバータを通して出力する．検索時には，V ビット検索部に対して V=1 を用いて検索が行われると同時に，空きエントリ検出時には，V ビット読み出し部に対して V=0 エントリの選択が行われ，空きエントリ（行）に有効データが書き込まれる際には，V ビット検索部および V ビット読み出し部の両方に対して，当該行のビットに V=1 が書き込まれる．

V ビットの優先エンコーダは，最初に検出した V=0 エントリを選択し，当該

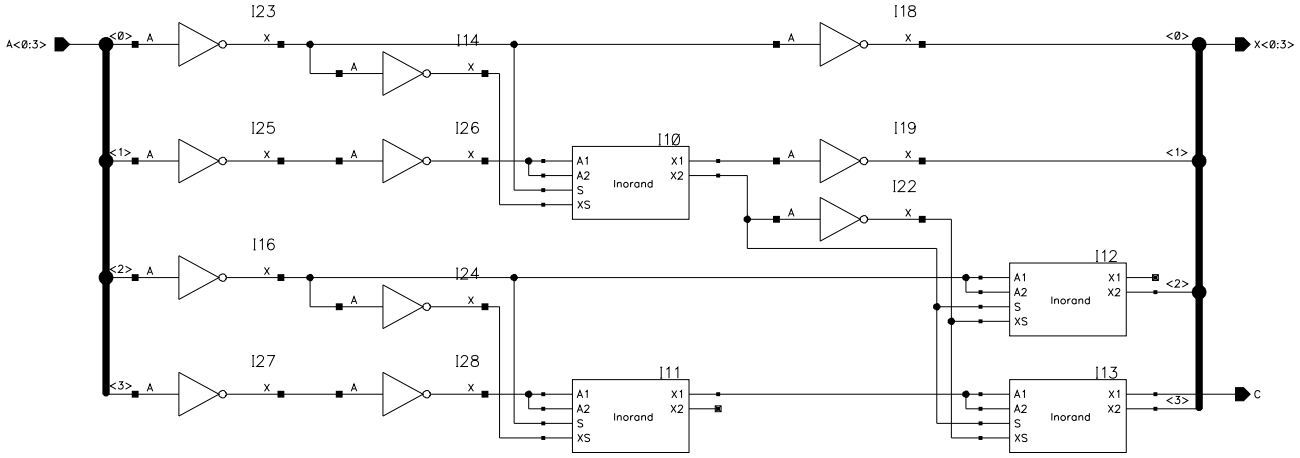

図 28: ENC4

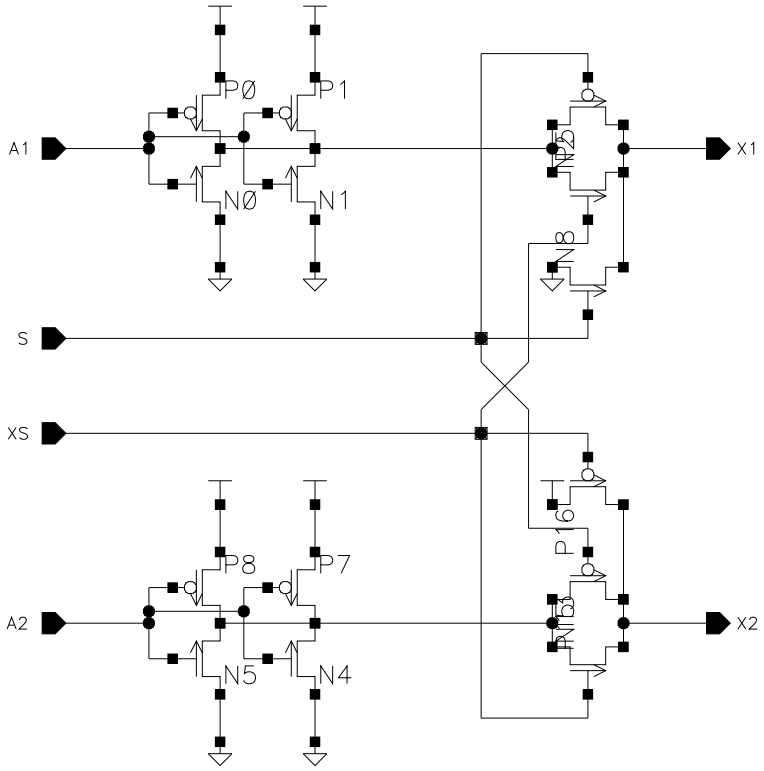

図 29: lnorand

表 1: lnorand の論理関係

A1	A2	S	XS	X1	X2
0	0	0	1	1	1
0	1	0	1	1	0
1	1	0	1	0	0
1	0	0	1	0	1
X	X	1	0	0	1

表 2: ENC4 入出力の論理関係

A0	A1	A2	A3	X0	X1	X2	X3
0	0	0	0	0	1	1	1
1	0	0	0	1	0	1	1
1	1	0	0	1	1	0	1
1	1	1	0	1	1	1	0

行を書き込み対象とする．図 28 に 4 入力 4 出力の優先エンコーダの schematic を示す．図 29 に優先エンコーダ中の lnorand ゲートの回路を示す．表に lnorand の論理関係を示す．

ENC4 は，RV から読み出された 4 つ V ビット（A(0，3)）を入力とする．なお，入力信号 (A0，A1，A2，A3) と，ENC4 の出力信号 (X0，X1，X2，X3) の対応関係を表 1 に示す．64 エントリの連想検索装置の場合に，16 個の ENC4 を並列接続する優先エンコーダを用いる．

表 2 の論理関係により，優先エンコーダ部は，CAM の空きエントリ中の最上行に最も近い空きエントリを書き込み対象として選択する．

5.5 制御装置 (GATE)

図 30 は，制御部の各行ごとの構成である．登録動作の場合には 1，検索動作の場合には 0 と与える WRITE 信号，CAM 部マッチラインからの MATCH 信号，および，空きエントリを検出する D 信号から，RAM 部のワードラインを駆動する RWL と，CAM 部のワードラインを駆動する CWL を生成する．RWEN，RREN，RSEN は，各々，RAM の Write enable 信号，Read enable 信号，Sense

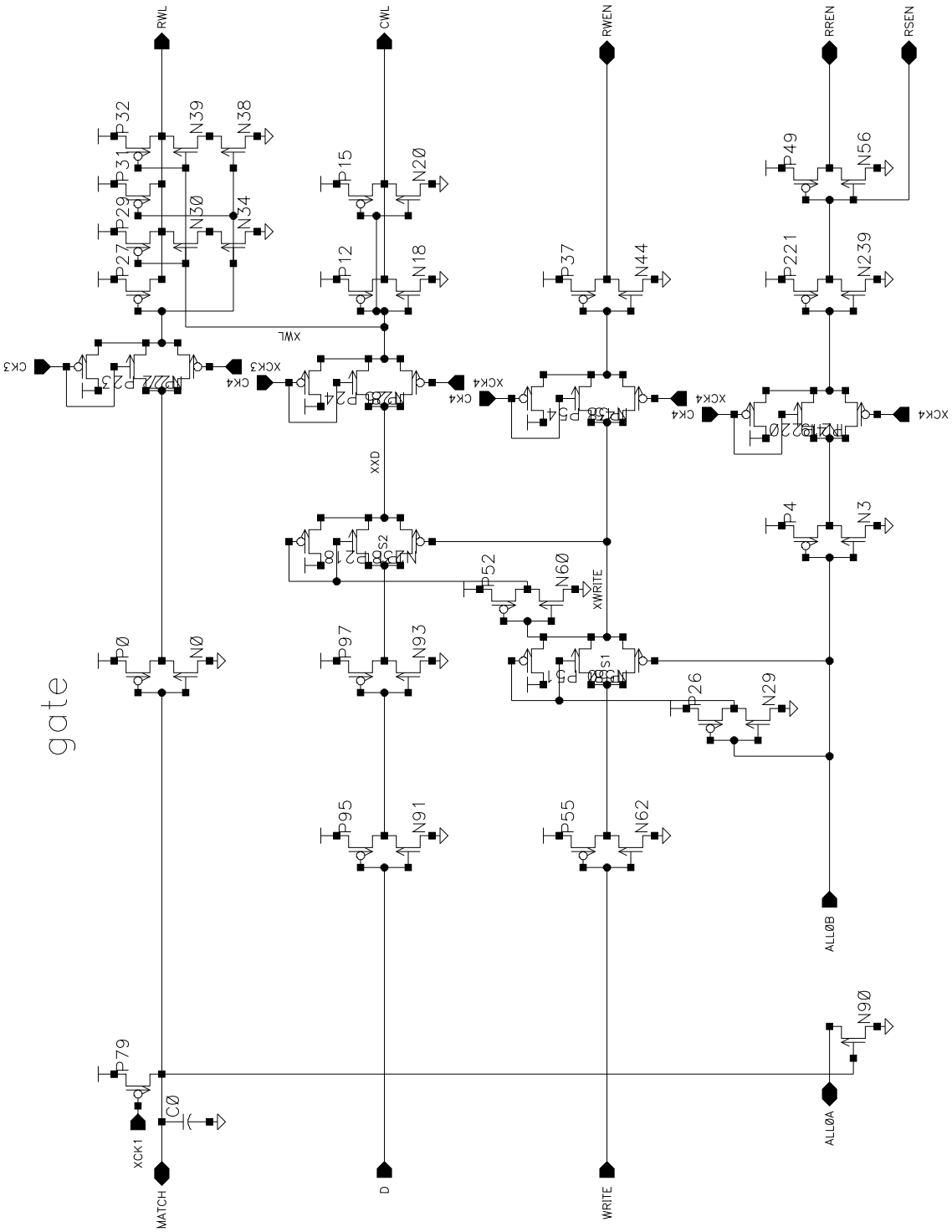
gate
RWL
CWL
RWEN
RREN
RSEN
MATCH
D
WRITE
ALL0B
ALL0A
XCK1
XWL
XXD
XWRITE
CK3
CK4
C0
P0
N0
P79
N90
P32
N39
N38
P31
P29
N30
N34
P27
P15
N20
P12
N18
P37
N44
P49
N56
P221
N239
P4
N3
P52
N60
P26
N29
P97
N93
P95
N91
P55
N62

図 30: GATE の回路

enable 信号である．

マッチラインの全てが 0 である場合，検索が失敗するため，書き込み動作を行う必要がある．このために，全てのマッチラインが 0 であるかどうかを示す ALL0B 信号を作る．１つもマッチしない場合には ALL0B 信号が 0 となる．登録時，Write 信号が 1 であり，ALL0B が 1 により相補形スイッチ S1 がオンになる．XWRITE の電位が 0 になり，インバータを通してセンスアンプの WRITE-ENABLE 信号（RWEN）をオンにして，書き込みデータを CAM/RAM 内のビットラインに伝える．同時にあらかじめ探しておいた空きエントリ情報を D 信号として入力し，XWRITE の電位が 0 であると，スイッチ S2 が ON になり，S1 を通して CAM の Word-Line（CWL）をプルアップし，当該空きエントリに登録データを書き込むことができる．同時に，XWL の電位（N30 と N39 のゲートの電位）が 0 であるので，RAM Word － Line（RWL）が放電されず，高電位を保持したままとなるため，当該行に書き込む．また，ALL0B はインバータを通して，センスアンプの Read Enable(RREN) および Sense Amp Enable（RSEN）と接続する．ALL0B がの時 0 は，RREN と RSEN が無効となる．

検索データが CAM に既登録である場合には，このデータに対応するマッチラインが高電位に保持され，ALL0B 信号が１となる．インバータを通して RREN が 0 になり，RSEN が 1 になり，センスアンプを通してデータを読み出すこと．スイッチ S1 がオフになるため，XWRITE の電位が 1 になり，インバータを通して RWEN が 0 になる．XWRITE の電位が１であるので，スイッチ S2 がオフとなり，CWL が 0 に保持される．なお，検索データを読み出す時は，CAM から直接に読み出す必要はない．検索データと一致した CAM の行により，当該行の RAM の WL をオンにして，RAM からマスク値を読み出すことができるためである．

以上は 1 行に対応する GATE の回路動作である．GATE8 は図 31 に示すように，GATE に基づいて 8 行のマッチラインおよびワードラインを並列制御するものである．ALL0 が 8 行分共有され，ALL0A 信号が GATE 部から出力する ALL0 信号であり，当該信号を増幅し，ALL0B 信号として GATE 部に入力する．P90 および P6 は充電用トラジスタであり，クロック信号 XCK2 が P90 のゲートに入力され，CK2 時点（前節に述べたように，本装置では，一つ動作クロックが四つサブクロックからなる）に XMATCH 側を充電する．GATE 部では，全てのマッチラインがプルダウンされていれば，検索が失敗であり，図 30

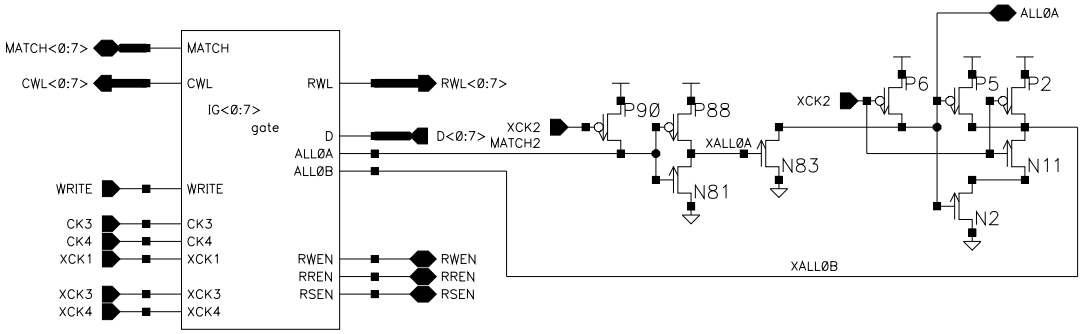

図 31: GATE8

の N90 のソースとドレインが導通せず，充電された XMATCH は放電されず，高電位を保持できる．XMATCH はインバータを通して N83 のゲートに入力され，XALL0A の電位が 0 であるため，N83 のソースとドレインが導通しない．CK2 が終了すると，XALL0B が N11 および N2 を通して GND と接続し，放電される．この結果，ALL0B 信号が 0 となり，CAM の書き込み動作を開始する．CAM の検索が成功である場合は，アサートされたマッチラインが高電位を保持し，ALL0A 信号を放電する．XALL0A 側の電位が 1 になり，N2 のゲート入力電位が 0 となる．従って充電された XALL0B 側の電位が高電位となり ALL0B に入力する．ALL0B 信号が 1 であるため，CAM 部の読み出し動作を開始する．

以上は ALL0 信号の制御の概要である．64 エントリの場合には，8 個の GATE8 を並列接続し，クロック入力，ALL0 信号，RWEN，RSEN および RREN が共有される．

第6章　評価

本章では，各々エントリ数が異なる連想検索装置回路を評価し，動作周波数を求める．具体的には，CAD ツールを用いて，設計回路のレイアウトから容量抽出を行い，HSPICE による回路シミュレーションを行った．

6.1　単体性能

図 32 と図 33 は，32 エントリと 64 エントリの CAM のレイアウトから抽出したネットリストにおける Read &Write 動作をシミューレートした時の波形である．このシミュレーションでは，アドレスデコーダのゲート容量と相当するインバータとそれに付随するワードラインを用いて，ワードライン遅延をシミュレートした．実際は，第 4 章で述べたように，専用連想検索装置にはアドレスデコーダはない．この評価では，メモリ全体の性能を測るため，ワードライン遅延も加えて考察した．

32 エントリの CAM の読み出し時間（ワードライン遅延も含む）は 0.3ns であり，64 エントリは 0.38ns であった．一方，32 エントリの CAM の書き込み時間（ワードライン遅延も含む）は 0.31ns であり，64 エントリでは 0.33ns であった．

図 34 は，CAM の検索時の波形である．グラフは上からそれぞれプリチャージ信号，1bit が mismatch 時のマッチライン，2bit が mismatch 時のマッチラインである．CAM の横幅が長い（128 bit）ため，検索速度を速くするためマッチラインを 8 分割する方式を採用した (第 4 章参照)．1 ビットのみ不一致の時，横幅が 128bit の CAM の検索時間は 3.9ns であり，2 ビットが不一致の時，検索時間は 2.6ns であった．

6.2　専用連想検索装置の性能

図 35 に，センスアンプの入力データから RAM の 1 行目に書き込む時，ビットライン中の高電位側の電位が 1 から 0 に下がる波形を示す．各エントリ数ごとの結果をグラフ中の折れ線により示しており，左からそれぞれ 32，64，128，256，1024 エントリに対応する．求めた Write Time を表 3 に示す．

図 36 は，センスアンプを付けず，メモリの 1 行目からデータを読み出す時の，高電位側のビットラインの電位変化波形である．表 4 に，BL と XBL の電位差が 60mV および 100mV になるまで必要時間，すなわち読み出す行のワードラ

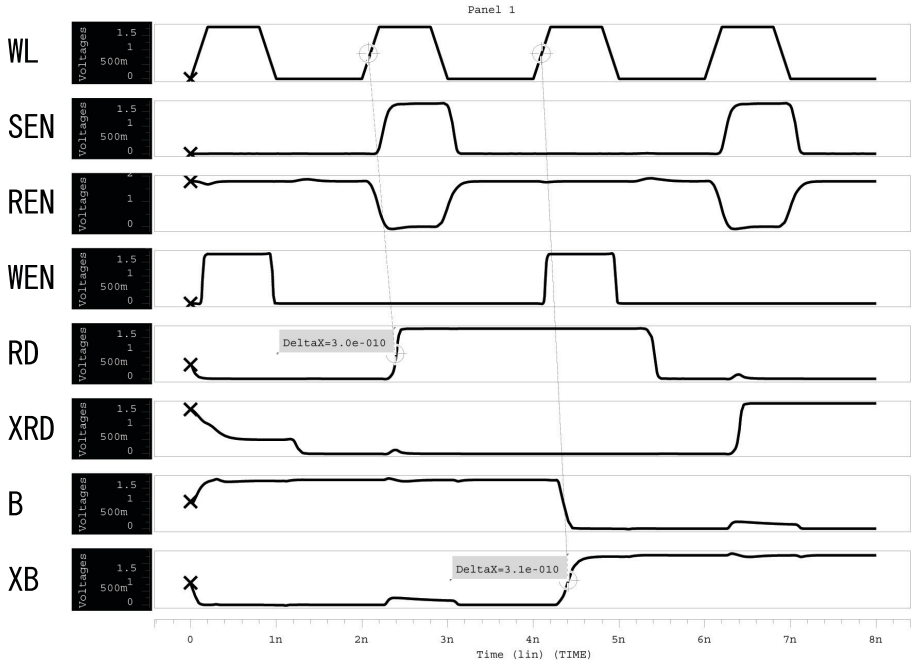

図 32: 32 エントリ CAM Read & Write

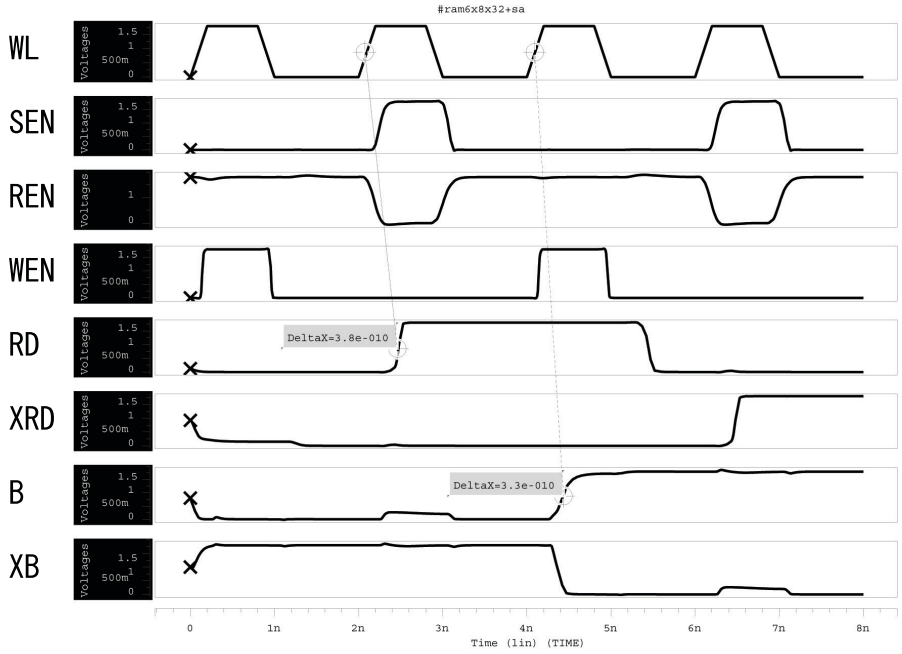

図 33: 64 エントリ CAM Read & Write

表 3: Write Time

エントリ	16	32	64	128	256	1024
Time(ns)	0.28	0.28	0.29	0.33	0.45	1.17

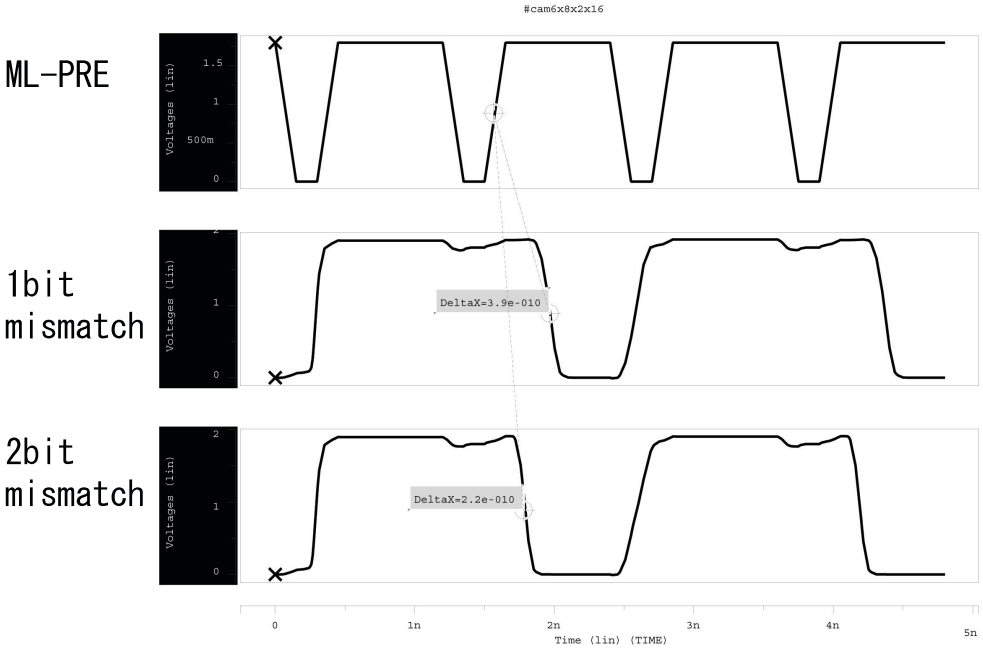

図 34: 128bit CAM の Search 時間

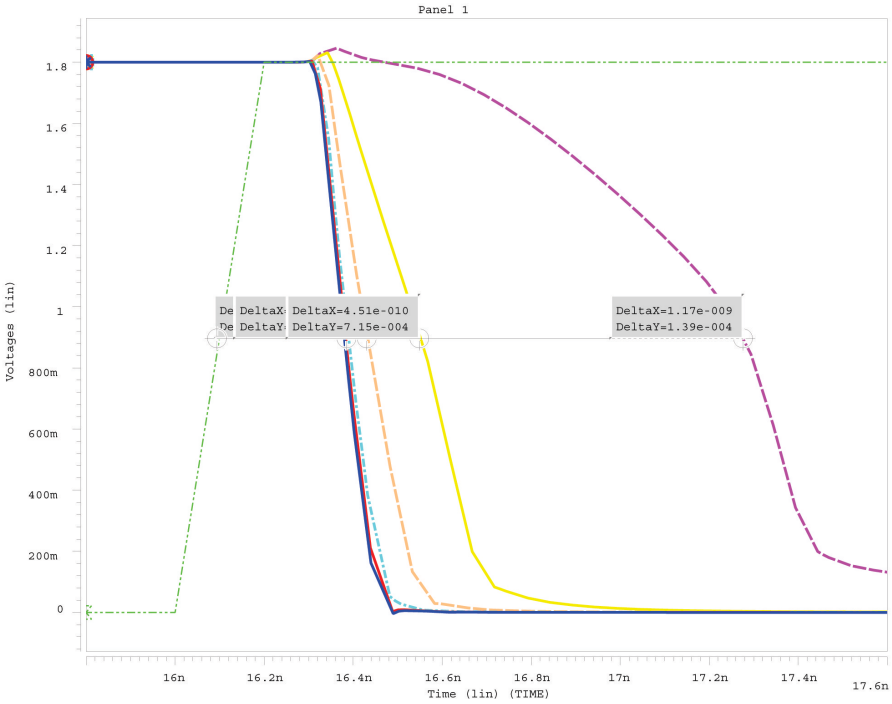

図 35: Write 遅延

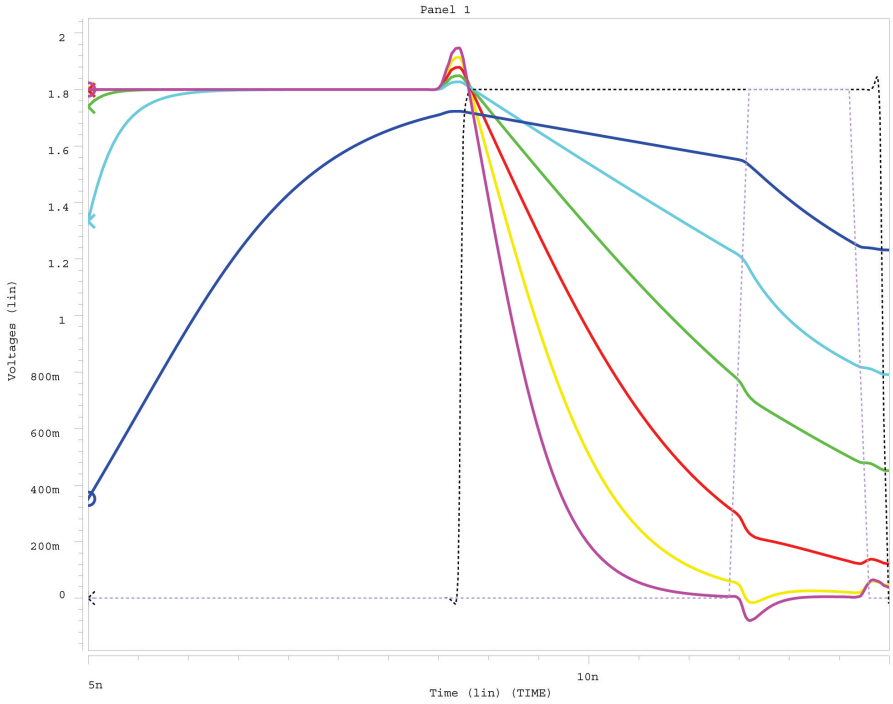

図 36: Read 遅延

表 4: センスアンプの電位差が発生するまでの時間

エントリ	16	32	64	128	256	1024
電位差 60mV	0.042	0.064	0.104	0.183	0.351	0.877
電位差 100mV	0.063	0.099	0.163	0.306	0.585	1.734

表 5: 放電率

エントリ	16	32	64	128	256	1024
放電率（mV/ps)	1.600	1.085	0.667	0.381	0.210	0.057

インがプルアップされてからセンスアンプが動作を開始するまでの必要時間を示す．表 5 はエントリ数が異なるメモリのビットラインの放電率である．

膨大なエントリ数（256 以上）のメモリブロックはビットラインが長いため，ビットラインの放電率が低くなり，アクセス遅延が大きくなる．メモリブッロクの容量としては，256 エントリ数以下の CAM を再利用バッファとして採用すべきであると言える．

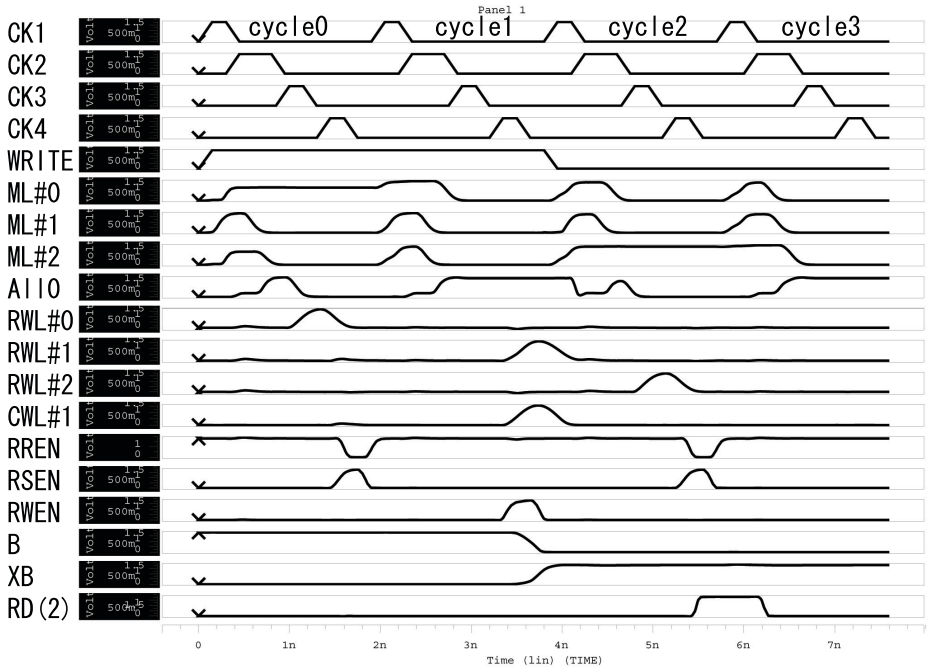

図 37: 32 エントリの専用連想検索装置の動作シミュレーション

図 37 に，専用連想検索装置の動作を示す．CAM のエントリ数が 32 の場合，動作クロックは 1.9ns であった．Cycle0(Search & Found) では，WRITE 信号を１とする，すなわち，登録サイクル (Search & Write) である．まず，CAM 部の全マッチラインが，1 にプリチャージされ，同時に RAM 部のビットラインもプリチャージされる．ここで CAM の 0 行目のマッチラインがアサートされるため，入力値が既登録であることを検知し，ALL0 信号が放電される．ALL0 信号により，RAM の Read Enable(REN) および Sense Amp Enable(SEN) 信号をオンにする．また，CAM の 0 行目のマッチラインにより RAM の 0 行目のワードラインをオンにし，0 行目からデータを読み出す．Cycle1(Search & Not-Found & Write) では，CAM のマッチラインが１つもマッチしないため，ALL0 信号が１となり，あらかじめ探しておいた空きエントリ (CAM の 1 行目) に関する CAM および RAM のワードライン (CWL/RWL) をオンにし，センスアンプの WRITE ENABLE 信号（CWEN および RWEN）もオンにし，入力データを CAM の 1 行目に登録している．Cycle2(Search & Found & Read) では，WRITE 信号が 0 となるので，検索動作となる．検索したい入力値が，CAM の 2 行目に既登録であることを検知し，2 行目のマッチラインに対応する RAM の Word-line をオンに

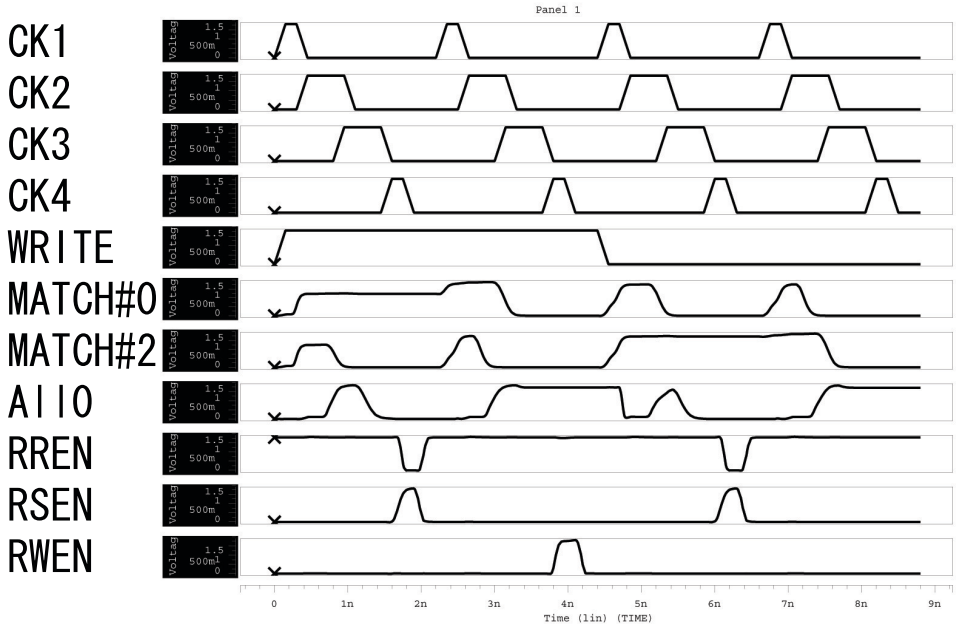

図 38: 64 エントリの専用連想検索装置の動作シミュレーション

表 6: 正常動作クロック

エントリ	16	32	64	128	256	1024
動作クロック (ns)	1.7	1.9	2.1	2.5	3.4	5.6

表 7: 平均電流と電力

エントリ	32	64	128
平均電流 (mA)	38.3	52.4	55.2
電力 (mW)	69.1	94.1	99.7

表 8: CAM Layout 面積

エントリ	32	64	128	1024
面積 (mm^2)	0.180	0.339	0.658	5.42

し，次入力のアドレスを読み出すと，同時に間接的に CAM の内容を読み出し，履歴として履歴表 (Region Table) に登録している．Cycle3(Search & Found) では，検索データが CAM に未登録であることを検知し，検索を終了している．

図 38 は 64 エントリの場合の波形である．動作クロックは 2.2ns 以上であった．

第 4 章に述べたように，本装置を用いて連想検索する時，ALL0 信号を用いて入力値の検索が成功したかどうかを判断する．しかし，ALL0 の充放電動作には一定の時間が必要であり，特に制御装置のエントリ数が多くなるほど，ALL0 の充放電回路のゲート数も多くなり，ALL0 信号の放電速度が低下する．従って ALL0 信号により駆動される RREN，RSEN，CWEN，RWEN などの Enable 信号も遅くなり，機構全体の速度に影響する．本装置が正しく動作できるクロックは次の式により表すことができる．

$$T_{clock} = T_{ml-pre} + T_{ALL0,search} + T_{ren,sen,wen} + T_{read,write}$$

T_{clock} は動作クロック，T_{ml-pre} はマッチラインのプリチャージ時間，$T_{ALL0,search}$ は ALL0 信号の充放電時間および CAM の検索時間（同時開始）,$T_{ren,sen,wen}$ は REN，SEN および WEN が駆動される時間,$T_{read,write}$ は CAM および RAM のアクセス時間である．表 6 に，エントリ数が 32，64，128，256，1024 の連想検索装置が正常動作できるクロックを示す．256，1024 エントリの連想検索装置は，ALL0 充放電に長い時間が必要なために，メモリアクセス遅延も長いので，装置全体の正常動作するクロックサイクル時間が長いことがわかる．従って 32，64，128 エントリの連想検索装置を再利用バッファブロックとして選択すれば，再利用率の低下というトレードオフを考慮しても，全体の性能向上を期待できる．

また，Vdd 電圧を 1.8V に設定した時，全体装置が正常動作する時の平均電流および平均消費電力を表 7 に示す．全体装置の面積は表 8 に示す．最近発表された 8 コアで 32 スレッドを実行可能なプロセッサ UltraSparc T1 は，面積が $340mm^2$，消費電力が 70W である．64 エントリの再利用バッファを UltraSparc T1 に搭載すると仮定すると，面積が 0.025%の増加，消費電力が 0.13%の増加で済むことから，今後，本研究の成果がマルチコア型マイクプロセッサに対して有効であることがわかった．

第7章　おわりに

本論文では，再利用および並列事前実行機構に必要となる再利用バッファの高速化および動作時間の評価を行った．本手法では，汎用連想メモリを用いる場合の問題点に対して，専用連想検索装置を設計することにより，再利用バッファを高速化した．汎用連想メモリのプライオリティエンコーダを省略し，指定した書き込みデータが登録済であれば登録せず，未登録であれば空きエントリに対して高速に登録し，唯一のマッチラインがアサートされることを保証できる．また，データ検索時の CAM からの読み出しを不要とすることで，従来多くのサイクル数が必要であった登録及び検索動作を 1 サイクル内に実現でき，再利用バッファの小型化，高速化および省電力化が可能となった．評価の結果，64 エントリの連想検索装置がクロックサイクル時間 2.2ns で正常動作でき，これは設計に用いた 0.18um の世代のクロックサイクル時間としては最適なものである．この時の再利用バッファの平均電流は 52 mA で，平均消費電力は 94mW であり，この 64 エントリの再利用バッファを 8 コアで 32 スレッドを実行可能なプロセッサ UltraSparc T1 に搭載すると仮定すると，面積が 0.025%，消費電力が 0.13%の増加で済むことから，本研究の成果が単一スレッドの高速実行を可能とする電力効率のよい次世代マルチコア型マイクロプロセッサの構成方式として有望であると言える．

謝辞

本研究の機会を与えてくださった富田眞治教授に深甚なる謝意を表します．また，本研究に関して適切なご指導を賜った中島康彦助教授，森眞一郎助教授，嶋田創特任助手に心から感謝いたします．また，お世話になった富士通 LSI 事業本部久保田勝久氏に心から感謝します．さらに，日頃からご助力頂いた京都大学大学院情報学研究科通信情報システム専攻富田研究室の諸兄に心から感謝いたします．

参考文献

[1] Lipasti, M.H. and Shen, J.P.:Exceeding the Dataflow Limit via Value Prediction, 29th MICRO, pp.226-237（1996）.

[2] Wang, K. and Franklin, M.: Highly Accurate Data Value Prediction Using Hybrid Predictors, 30th MICRO, pp.281-290（1997）.

[3] Collins, J.D., Wang, H., Thllsen, D.M., Hughes, C., Lee, Y., Lavery, D. and Shen, J.P.: Speculative Precomputation: Long-range Prefetching of Delinquent Loads, 28th ISCA, pp.14-25（2001）.

[4] Luk, C.: Tolerating Memory Latency through Software-Controlled Pre- Execution in Simultaneous Multithreading Processors, 28th ISCA, pp.40.51（2001）.

[5] Collins, J.D., Tullsen, D.M., Wang, H. and Shen, J.P.: Dynamic Speculative Precomputation, 34th MICRO, pp.306-317（2001）.

[6] Gopal, S., Vijaykumar, T.N., Smith, J.E. and Sohi, G.S.: Speculative Versioning Cache, 4th HPCA, pp.195-205（1998）.

[7] Marcuello, P., González, A. and Tubella, J.: Speculative Multithreaded Processors, International Conference on Supercomputing（ICS）, pp.77-84（1998）.

[8] Marcuello, P., Tubella, J. and González, A.: Value Prediction for Speculative Multithreaded Architectures, 32nd MICRO, pp.230-237（1999）.

[9] Oplinger, J.T., Heine, D.L. and Lam, M.S.: In Search of Speculative Thread-Level ParALLelism, 8th PACT, pp.303-313（1999）.

[10] Codrescu, L., Wills, D.S. and Meindl, J.: Architecture of the Atlas Chip- Multiprocessor: DynamicALLy ParALLelizing Irregular Applications, IEEE Trans. Comput., Vol.50, No.1, pp.67-82（2001）.

[11] Marcuello, P. and González, A.: Thread-Spawning Schemes for Speculative Multithreading, 8th HPCA, pp.55-64（2002）.

[12] Sodani, A. and Sohi, G.S.: Dynamic Instruction Reuse, Proc. 24th ISCA, pp.194-205（1997）.

[13] González, A., Tubella, J. and Molina, C.: Trace-Level Reuse, ICPP-1999, pp.30-37（1999）.

[14] Costa, A.T., Fran, ca, F.M.G. and Filho, E.M.C.: The Dynamic Trace Memorization Reuse Technique, PACT, pp.92-99（2000）.

[15] Huang, J. and Lilja, D.J.: Exploiting Basic Block Value Locality with Block Reuse, Proc, 5th HPCA, pp.106-114（1999）.

[16] Connors, D.A. and Hwu, W.W.: Compiler-Directed Dynamic Computation Reuse: Rationale and Initial Results, 32nd MICRO（1999）.

[17] Connors, D.A., Hunter, H.C., Cheng, B. and Hwu, W.W.: Hardware Support for Dynamic Activation of Compiler-Directed Computation Reuse, 9th International Conference on Architectural Support for Programming Languages and Operating Systems（ASPLOS）, pp.222-233（2000）.

[18] Huang, J. and Lilja, D.J.: Exploring Sub-Block Value Reuse for Superscalar Processors, PACT（2000）.

[19] Alvarez, C., Corbal, J., Salam´i, E. and Valero, M.: On the Potential of Tolerant Region Reuse for Multimedia Applications, ICS '01（2001）.

[20] Yang, J. and Gupta, R.: Load Redundancy Removalthrough Instruction Reuse, International Conference on ParALLel Processing（ICPP）, pp.61-68 （2000）.

[21] Önder, S, and Gupta, R.: Load and Store Reuse Using Register File Contents, ICS '01, pp.289-302（2001）.

[22] Roth, A. and Sohi, G.S.: Register Integration: A Simple and Efficient Implementation of Squash Reuse, 33rd MICRO（2000）.

[23] Roth, A. and Sohi, G.S.: Speculative Data-Driven Multithreading, 7th HPCA, pp.37-50（2001）.

[24] Wu, Y., Chen, D. and Fang, J.: Better Exploration of Region-Level Value Locality with Integrated Computation Reuse and Value Prediction, 28th ISCA, pp.98-108（2001）.

[25] Molina, C., Gonzálex, A. and Tubella, J.: Trace-Level Speculative Multithreaded Architecture, International Conference on Computer Design（ICCD）'02（2002）.

[26] MUSIC SEMICONDUCTORS Inc.: Feature Sheet: MOSAID Class-IC DC18288, 1.3 edition（2003）.

[27] 中島康彦，津邑公暁，五島正裕，森眞一郎，富田眞治: 動的命令解析に基

づく多重再利用および並列事前実行，情報処理学会論文誌コンピューティングシステム，Vol.44，No.SIG10(ACS2)，pp.1-16（2003）.

[28] 津邑公暁，中島康彦，五島正裕，森眞一郎，富田眞治: 並列事前実行機構における主記憶値テストの高速化，情報処理学会論文誌コンピューティングシステム，Vol.44，No.SIG10(ACS4)，pp.31-42（2004）.

[29] 高洪波，李森，中島康彦，嶋田創，森眞一郎，富田眞治: 並列事前実行における再利用バッファの高速化， 情報処理学会研究報告，Vol.2005-ARC-165, pp.27-32 (2005) .

[30] 高洪波，李森，中島康彦，嶋田創，森眞一郎，富田眞治: 並列事前実行における連想検索装置の設計， 平成 17 年度情報処理学会関西支部大会講演論文集，pp.183-186（2005）.

[31] 李森，高洪波，中島康彦，富田眞治: 区間再利用バッファの分割とその高速化，平成17年度情報処理学会関西支部大会講演論文集，pp.181-182（2005）.