名古屋大学大学院工学研究科博士前期課程
修士学位論文

容量効率を意識したソース・タグ値に基づくセグメント化による発行キューの電力削減

令和　3年 3月
情報・通信工学専攻

森　健一郎

概要

発行キューは電力密度の大きいホット・スポットとして知られている．ホット・スポットは，デバイスの摩耗故障を引き起こし，誤動作やタイミング・エラーを引き起こす．発行キューが大きな電力を消費する原因は，ウェイクアップ論理のタグ比較回路である．この回路は CAM で構成されており，全てのデスティネーション・タグと発行キュー内の全てのソース・タグとの多数の比較を一斉に行うため，非常に大きな電力を消費する．そこで本論文では，CAM の分野で提案されている手法を応用し，タグ比較による消費電力を削減する手法を提案する．本手法では，発行キューを複数のセグメントに分割する．命令は，ソース・タグの下位ビットがセグメント番号と一致するセグメントにディスパッチする．そして，ウェイクアップ時には，ディスティネーション・タグの下位ビットが一致するセグメントにあるタグ比較器のみを動作させる．一致しないセグメントの比較器は動作しないため，タグ比較器の動作回数を削減できる．

本手法では，命令がディスパッチされるセグメントに空きがない場合，他のセグメントに空きがあってもディスパッチできないためストールする．この結果，発行キューの容量効率が低下するという問題が生じる．この問題は，発行キューの容量効率が重要なプログラムにおいて性能低下を引き起こす．そこで本論文では，容量効率を重視したディスパッチ・アルゴリズムと，タグ比較の積極的な削減を重視したディスパッチ・アルゴリズムを動的に切り替える手法を提案する．本手法は，発行キューの容量効率が重要な場合は容量効率の低下による性能低下を抑制し，そうでない場合は積極的にタグ比較器の動作回数を削減することを可能とする．提案手法を SPEC CPU 2017 を用いて評価を行った．結果，性能低下を最大で 5% 以下（平均 -1%）に抑えつつ，タグ比較器の動作回数を平均で 85% 削減できることを確認した．

目 次

第1章　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の問題が深刻になっている [1]．微細化は，経年劣化や摩耗故障を加速し，その結果，タイミング・エラーや誤動作を引き起こし，デバイスの寿命を縮める．経年劣化や摩耗故障は温度に関して指数関数的に加速し [2, 3, 4]，温度 10〜15 ℃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・チップ上には，ホット・スポットと呼ばれる単位面積あたりの電力が大きい場所が存在する．ホット・スポットは，そうでない場所と比べて温度上昇が激しいため，上述した故障を引き起こす確率が高くなる．従って，ホット・スポットを生成する回路の消費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，発行キューがある．発行キューのサイズはプロセッサの世代が進むごとに大きくなっており，より深刻なホット・スポットとなっている．従って，発行キューの電力削減に対する要求は非常に大きい．

発行キューの中で最も電力を消費する回路は，タグ比較の回路である．タグ比較は，発行幅分のディスティネーション・タグとすべてのソース・タグとの間で行われるため，非常に多くの電力を消費する．そこで本論文では，タグ比較器が動作する回数を削減する以下のような手法を提案する．

- 発行キューを複数の**セグメント**に分割する．命令を発行キューにディスパッチする際，第 1 ソース・タグの下位ビットが n である命令は，第 n 番目のセグメントに書き込む．タグ比較時には，ディスティネーション・タグの下位ビットがセグメント番号と一致するセグメントでのみ，第 1 ソース・タグの比較を行う．一致しないセグメントでは比較が行われない．これによりタグ比較回数が削減される．

- 上記の方法では，第 2 ソース・タグの比較回数は削減されない．そこで提案手法では**スワップ**と**サブ・セグメント**と呼ぶ 2 つの方法を導入し，第 2 ソース・タグの比較回数も削減する．スワップは，ディスパッチ時に第 1 ソース・オペランドがレディで，第 2 ソース・オペランドがレディでない命令において，第 1 ソース・タグと第 2 ソース・タグを格納するフィールドを交換し，第 2 ソース・タグの下位ビットを用いてディスパッチするセグメントを決定する手法である．サブ・セグメントは，各セグメントを第 2 ソース・タグにもとづきさらに分割する手法である．

- セグメント化によりディスパッチできるエントリが制限されるため，発行キューの容量効率が低下し，容量に敏感なプログラムにおいて性能が低下するという問題が存在する．この問題に対応するため，本論文では **SWITCH** という手法を提案する．SWITCH では，容量効率を重視したディスパッチ・アルゴリズムと，タグ比較回数の削減を重視したディスパッチ・アルゴリズムを，容量効率の重要性に応じて切り替えて使用することにより，性能低下を抑制する．

提案手法を SPEC CPU 2017 ベンチマークを用いて評価し，性能低下を 最大でも 5% 以下（平均 -1%）に抑えつつ，タグ比較の回数を平均で 85% 削減できることを確認した．

本論文の残りの構成は次の通りである．まず，**??**節で発行キューの基本的な事項を説明する．そして，4.1 節で提案手法の基本となるアイデアに関して説明した後，4.2 節で提案手法における第 2 ソース・タグのタグ比較回数削減方法に関して述べる．その後，5.1 節で提案手法の問題点である発行キューの容量効率の低下に関して説明した後，5.2 節で容量効率の低下に対する対策方法を説明する．6 節で評価を行い，7 節でまとめる．

第2章　発行キュー (IQ：Issue Queue)

本章では，本研究の研究対象である、IQ に関して説明する．まず，IQ の概要と動作を 2.1 節で説明したあと，IQ の回路構成を 2.2 節で述べる．その後，2.3 節で IQ の方式に関して説明する．

2.1　概要と動作

IQ はアウト・オブ・オーダ実行を行うプロセッサにおいて，リネームされた命令を保持し，実行順序をスケジューリングして，機能ユニットへ発行する回路である．IQ は，ディスパッチ，発行，ウェイクアップと呼ばれる 3 種類の動作を行う．以下でそれぞれの動作に関して説明する．

- ディスパッチ：リネームされた命令は，IQ にエントリが割り当てられ，命令の情報が格納される．この動作をディスパッチと呼ぶ．ディスパッチの動作は，IQ の方式により異なる．IQ の方式に関しては，2.3 節で詳しく説明する．

- 発行：IQ 内の命令のうち，ソース・オペランドが両方共レディとなった命令は，依存関係が解消し，実行が可能となる．このような命令を実行ユニットに送出する動作を発行と呼ぶ．なお，発行可能な命令が機能ユニットの数を超える場合 (このような場合を発行コンフリクトと呼ぶ) は，各命令の発行優先度に基づき命令を選択して発行する．発行された命令のエントリは IQ より削除される．

- ウェイクアップ：命令が発行されると，その命令のディスティネーション・オペランドのタグと発行キュー内にある全命令のソース・オペランドのタグの比較が行われる．比較が一致した場合には, 対応するソース・オペランドのレディ・ビットをセットす

る．この動作をウェイクアップと呼ぶ．両方のオペランドがレディとなった命令は，依存が解消したため発行可能となる．

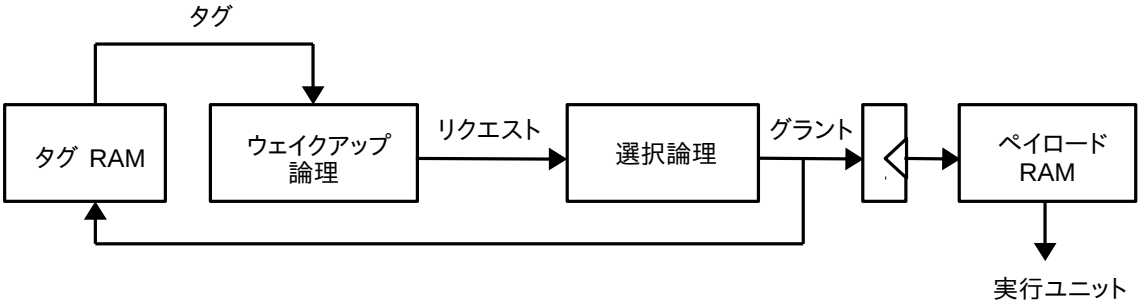

図 2.1: IQ の回路構成

2.2　回路構成

図 2.1 に IQ の回路構成を示す．IQ はウェイクアップ論理，選択論理，タグ RAM，ペイロード RAM と呼ばれる 4 つの回路より構成される．以下で各回路に関して説明する．また，IQ の回路のうちウェイクアップ論理は提案手法に関わる重要な回路であるため，2.2.1 節にて詳細に説明する．

- ウェイクアップ論理 : 命令感の依存関係を管理し，他の命令との依存関係が解消された命令に対して発行要求 (リクエスト信号) を出す．
- 選択論理 : 資源制約を考慮して，発行を要求された命令の中からそれを許可する命令を選択肢，発行許可信号 (グラント信号) を出力する．この選択においては，回路構成の単純化のために IQ の先頭のエントリの命令をより優先する．

- タグ RAM : 発行待機中の命令のディスティネーション・タグを保持する回路で，選択論理から発行許可信号が送られると，対応する命令のタグを読み出し，それをウェイクアップ論理へ送る．

- ペイロード RAM : 発行待機中の命令の命令のコードを保持する．選択論理から発行許可信号が送られると，対応する命令のコードを実行ユニットに送出する．

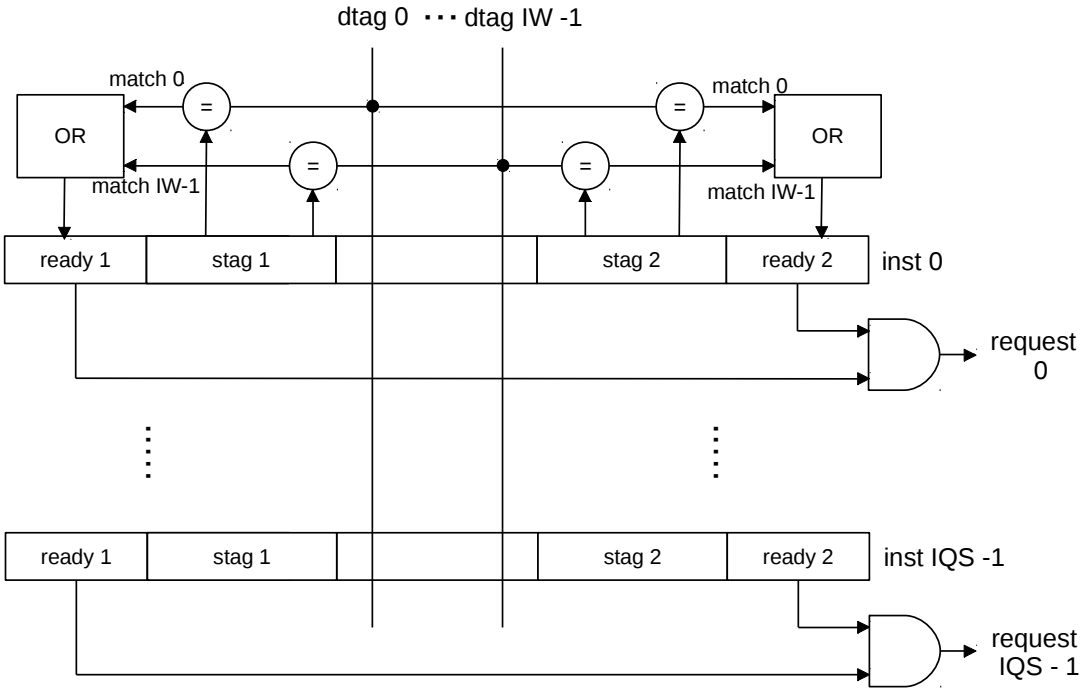

図 2.2: ウェイクアップ論理

2.2.1 ウェイクアップ論理

図 2.2 に，ウェイクアップの回路を示す．図中の IW は発行幅を，IQS は発行キューのエントリ数を表す．ウェイクアップでは，IW 個のディスティネーション・タグ（dtag）が

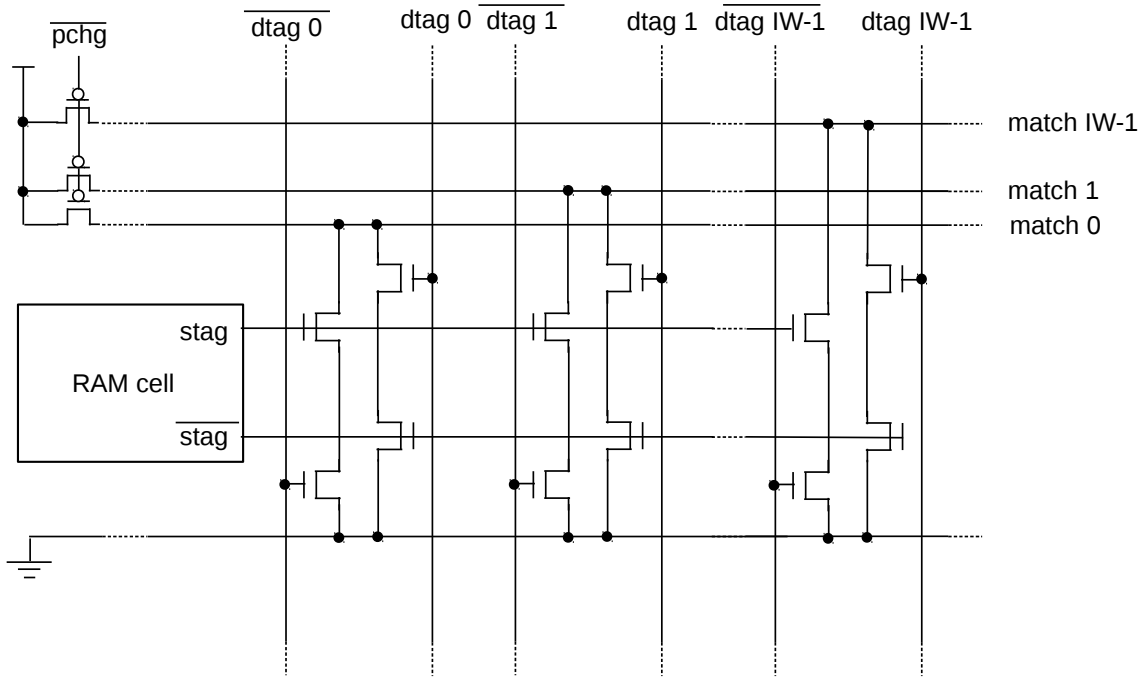

図 2.3: タグ比較器の CAM 回路

発行キュー内の全命令に放送される．各命令は 2 つのソース・タグ (stag) を保持しており，放送されたディスティネーション・タグと比較が行われる．いずれかのディスティネーション・タグとソース・タグが一致した場合，そのソース・オペランドのレディ・ビットがセットされる．2 つのレディ・ビットがセットされた命令は発行が可能となるため発行要求が出力される．

図 2.3 に発行キューに使用されるタグ比較器の CAM 回路を示す．同図は，ソース・タグ 1 ビット分の比較回路を表す．同図に示すように，高速化のため通常ダイナミック論理によって構成される．比較の動作は，次のように行われる．まず，マッチ線がプリチャージされる．次にデスティネーション・タグが放送され，比較が行われる．タグが不一致であれば，直列に接続された 2 つのプルダウン・トランジスタが両方とも ON となり，マッ

チ線がディスチャージされる．タグが一致する場合，マッチ線は H の状態が維持される．比較器はマッチ線のディスチャージ時に電力を消費する．

2.3 IQ の方式

これまで，IQ の方式としてシフト・キュー，サーキュラ・キュー，ランダム・キューの 3 つの方式が提案されている．各方式に関して説明したのち，現在主流な方式であるエイジ論理付きのランダム・キューに関して説明する．

2.3.1 シフト・キュー

シフト・キューは，最も古くに提案され，商用プロセッサに使用された IQ の方式である [6]．シフト・キューでは，IQ の先頭のエントリより順に命令をディスパッチする．これにより，古い命令に高い発行優先度を与えることができる．[1]

また，シフト・キューでは命令を発行したエントリの空きを詰めるコンパクションを行うことにより，高い容量効率も達成することができる．正しい発行優先度と，高い容量効率を同時に達成するため，シフト・キューは IQ の方式の中で最も高い性能を得ることができる．

一方でシフト・キューには，コンパクションの回路が非常に複雑で，また消費電力が非常に大きいという欠点がある．そのため，シフト・キューはスケーリングが困難となっており，現在のプロセッサには使用されていない．

2.3.2 サーキュラー・キュー

サーキュラ・キューは，シフト・キューにおいて問題であったコンパクションを行わない方式である [7]．IQ は，ヘッド・ポインタとテール・ポインタを用いてサーキュラー・バッファとして管理される．

[1] 一般に，古い命令から優先的に発行すると，性能がより高くなることが知られている．

サーキュラー・キューでは，既に空いているが，命令をディスパッチできないエントリが発生し，IQ の容量効率がシフト・キューと比較して低下してしまう．また，ヘッド・ポインタとテール・ポインタの位置が逆転するラップ・アラウンドが生じた際には，新しい命令に高い優先度が与えられる優先度逆転が起き，選択論理が正しい優先度で命令を選択できない．これらの理由から，サーキュラー・キューはシフト・キューと比較して性能が低下する．

特に，容量効率が低下する影響は大きく，現在のプロセッサには使用されていない．

2.3.3　ランダム・キュー

近年は，回路の単純化や電力削減のため空いているエントリに単純にディスパッチするランダム・キューが使用されている．ランダム・キューでは IQ の容量を無駄にすることがなく，高い容量効率を達成する．その一方で，命令が年齢とは無関係にランダムに並ぶため，正しい優先度で命令を発行することが出来ない．

ランダム・キューでは，発行キューの空きエントリのインデクスを保持するフリー・リストを用意する．ディスパッチ時には，フリー・リストから読み出したインデクスが指す発行キューのエントリに命令を書き込む．発行キューから命令が発行されエントリが無効化されると，そのインデクスをフリー・リストへ返す．フリー・リストは FIFO バッファで管理される．

2.3.4　エイジ論理付きランダム・キュー

ランダム・キューにおける発行優先度の欠点を緩和するため，ランダム．キューは一般にエイジ論理と併用される．エイジ論理は選択論理と並列に動作する回路で，発行要求が出された命令の中で最も古い 1 命令を選ぶ．最も古い命令はクリティカル・パス上の命令である可能性が高いため，これを優先して発行することができ，結果としてエイジ論理付きランダム・キューは通常のランダム・キュート比較して性能が大きく向上する．

本研究における IQ は，エイジ論理付きのランダム・キューを使用する．

2.4 IQ の問題点

IQ は電力密度の大きいホットスポットとして知られている．この主な原因は，ウェイクアップ論理での消費電力である．論文によると，ウェイクアップ論理の消費電力は IQ 全体の 20% であり，削減の必要がある．

ウェイクアップ論理での消費電力が大きい理由として，タグ比較回路が挙げられる．タグ比較に使用する CAM は，$IQS \times IW \times 2$ 個必要である．ここで，IQS は IQ のエントリ数， IW は発行幅 を表す．現在想定しているプロセッサ構成では，IQS は 128 エントリ，IQ は 8 命令であるため，CAM の総数は 2048 個となる．

これだけ多くの CAM が毎サイクル動作するため，ウェイクアップ論理の電力密度は大きくなる．本研究では，ウェイクアップ時のタグ比較によって生じる消費電力を削減することを目的とする．

第3章　関連研究

本章では，IQ に関連する研究について述べる．3.1 節で IQ に関する一般的な関連研究に関して説明し，3.1 節 で IQ の研究のうち，電力に関係する研究を述べる．

3.1　IQ に関する関連研究

Palacharla らは，命令発行幅と IQ のサイズを変化させた時の，ウェイクアップ論理と選択論理の遅延を評価した [8]．また，遅延を小さくするために，IQ を複数の FIFO バッファで構成し，依存する命令を同じ FIFO バッファに割り当てる依存ベースの IQ を提案した．この手法では，各バッファの先頭の命令のみ発行可能かチェックすれば良いので，回路が単純化され遅延が減少する．

Stark らは，IPC をほとんど低下させずに，ウェイクアップ論理と選択論理をパイプライン化する手法を提案した [9]．この手法では，投機的にウェイクアップを行うことで，依存する命令を連続するサイクルで発行できるようにした．

五島らは，ウェイクアップ論理を従来の CAM ではなく，依存行列と呼ぶ RAM で構成する手法を提案した [10]．これによって比較器を用いずに依存する命令をウェイクアップすることが可能で，ウェイクアップの遅延を短縮できる．

Sassone らは，依存行列の遅延と電力をより小さくするための手法を提案した [11]．具体的には，従来はすべての命令について，その古さを完全に追跡していたのに対して，命令をグループ化してグループ単位で古いものを選択する．これにより，性能低下を最小限に抑えながら，回路の規模を小さくできる．

Lebeck らは．キャッシュ・ミスするロードのような長いレイテンシの命令に依存する命令を，IQ とは別の待機用バッファに入れ，その長いレイテンシの処理が完了するまで IQ

に挿入しないという方式を提案した [12]．これによって，IQ が待機する命令で埋ることによって起こるストールの頻度が減り，性能が向上する．

Raasch らは，IQ をいくつかのセグメントに分割する方式を提案した [13]．この方式では，各命令の依存命令チェーンのレイテンシを元に割り当てるセグメントが決定される．そして，発行可能になる直前に最下位セグメントである発行バッファに命令を移動する．この発行バッファでのみ発行を行うことで，すべてのエントリから発行できる通常の IQ と比較して遅延を短縮できる．

Kim らは，レイテンシが互いに 1 サイクルの依存関係のある 2 つの命令をグループ化し，1 つの命令として IQ のエントリでスケジューリングすることで，依存グラフのエッジのレイテンシ短縮とキューの容量効率を上げる手法を提案した [14]．

Gibson らは，依存する命令をポインタでつなぎ，ポインタをたどることでウェイクアップを行う手法を提案した [15]．この方式により CAM が不要になり，電力を削減できる．

安藤らは，実行プログラムの命令レベル並列性 (ILP) とメモリ・レベル並列性 (MLP) に応じて IQ の方式を切り替える手法を実装した [16]．ILP と MLP のいずれかが高い場合は IQ の容量効率が重要であるため，ランダム・キューで実行する．ILP も MLP もいずれも低い場合には，容量効率よりも正しい発行優先度のほうが重要であるためサーキュラー・キューで実行する．

甲良らは，実行プログラムの ILP と MLP に応じて IQ のサイズを変化させる手法を提案した [17]．本手法では，いずれかが高い場合には，IQ の容量が重要となるため IQ のエントリ数を増加し，どちらも低い場合には IQ のエントリ数を減少させる．

3.2 IQ の電力削減に関する関連研究

Folegnani らは，空のエントリの比較器や既にレディなオペランドを持つ比較器など，タグを比較する必要がない比較器を動作させないことで，消費エネルギーを削減する手法を提案した [18]．

Ponomarev らは，リソース要求に応じて 発行キューのサイズをリサイズすることにより，消費エネルギーを削減する手法を提案した [19] .

Ernst らは，IQ に入ってくる命令のうちのほとんどが，はじめから少なくとも 1 つのソース・オペランドがレディであると指摘した [20]．そして IQ に，2 つのソース・オペランドを保持できるエントリに加えて，1 つのソース・オペランドのみ保持できるエントリと，ソース・オペランドを保持しないエントリを用意し，レディでないソース・オペランドの数に応じていずれかにディスパッチする手法を提案した．さらにこの手法を実現するために，命令の 2 つのオペランドの内，あとにレディになるオペランドを予測する手法である Last Tag Prediction も提案した．

Sembrant らは，クリティカル・パス上にない命令を発行キューとは別のバッファに入れ，ディスパッチを遅延させることによって，性能を低下させずに発行キューのサイズを小さくする手法を提案した [21].

Homayoun らは，キャッシュ・ミスの処理中に発行幅を半減させることで，IQ の消費電力を削減する手法を提案した [22]．発行幅半減中に元の発行幅の半分以上の命令が発行される場合，一時的にその命令を小さなバッファに移動させることで対応している．

松田らはウェイクアップ時のタグ比較を 2 段階に分割することによりエネルギー削減を行う方法を提案した [23, 24]．この方法では，タグの比較を高位ビットと低位ビットに分割し，低位ビットの比較を最初のサイクルで行う．そして低ビットが一致していた場合のみ，次のサイクルで高位ビットの比較を行うことによってエネルギーを削減する．また，タグの 2 段階比較には，ウェイクアップに 2 サイクル必要であるため性能が低下するという欠点が存在する．これに対し本手法では，クリティカルパス上にある命令のみ 1 サイクルで比較を行い性能低下の軽減を行う．

第4章　提案手法：セグメント化した IQ

4.1　発行キューのセグメント化

本論文では，発行キューのタグ比較器の動作回数を削減するための手法として，発行キューをセグメント化する手法を提案する．本節では，まず提案手法の概要を説明した後，提案手法におけるウェイクアップとディスパッチに関して詳しく説明する．

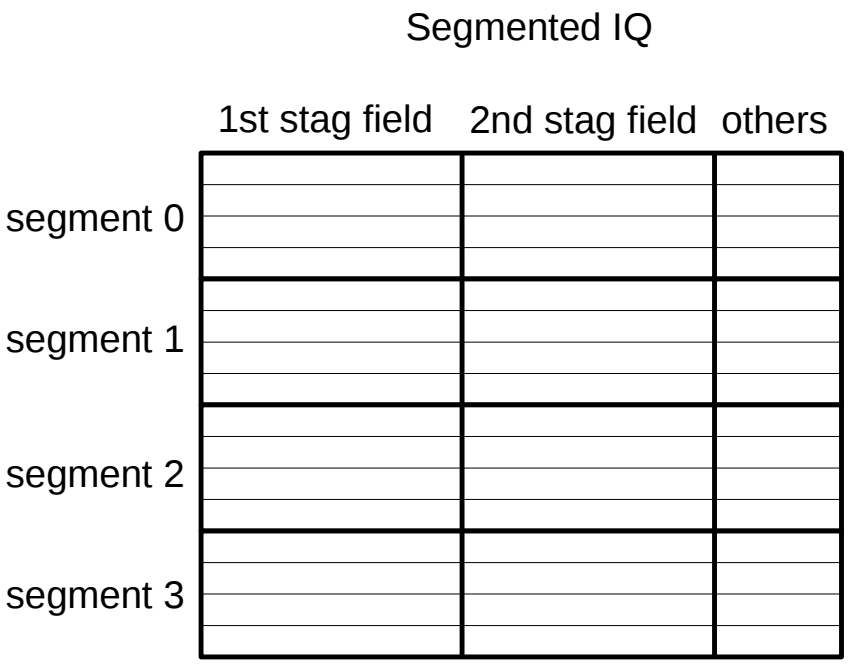

図 4.1: セグメント化した発行キュー

4.1.1　提案手法の概要

提案手法の基本アイデアは，大容量 CAM の電力削減に関する研究 [25, 26] から着想を得ている．この研究において提案されている手法では，CAM を複数の**セグメント** [1]に分割する．各セグメントには下位ビットが同一のデータのみを記録する．そして，比較が行われる際には，比較対象のデータの下位ビットと，記録されているデータの下位ビットが一致するセグメントのみで比較を行う．これによって，比較器が動作する回数を「1/セグメント数」まで削減することができ，消費電力が削減できる．

本手法においても，図 4.1 に示すように発行キューを複数のセグメントに分割する．各セグメントには，第 1 ソース・タグの下位ビットがセグメントの番号と一致する命令をディスパッチする．ウェイクアップ時の第 1 ソース・タグのタグ比較では，ディスティネーション・タグの下位ビットとセグメントの番号が一致するセグメントのみでタグ比較を行う．これによって，第 1 ソース・タグのタグ比較回数を「1/セグメント数」に削減できる．

提案手法におけるディスパッチとウェイクアップに関して詳しく説明する．

4.1.2　提案手法におけるディスパッチ

ディスパッチする発行キューのエントリを決定する回路を図 4.2 に示す．本手法では，フリー・リストをセグメントと同じ数だけ用意する．各フリー・リストは，対応するセグメントの空きエントリのインデクスを FIFO バッファで管理する．各フリー・リストからは発行キューのインデクスが出力され，その中の 1 つを選択してディスパッチするエントリを決定する．どのフリー・リストからの出力を選択するかは，セグメント選択回路（図中の segment select logic）によって決定される．

セグメント選択回路の選択アルゴリズムについて説明する．セグメントの選択方法は，ディスパッチ時に第 1 ソース・オペランドがレディであるかによって異なるため，それぞれの場合に関して説明する．

[1] 文献 [25, 26] ではバンクと呼ばれている

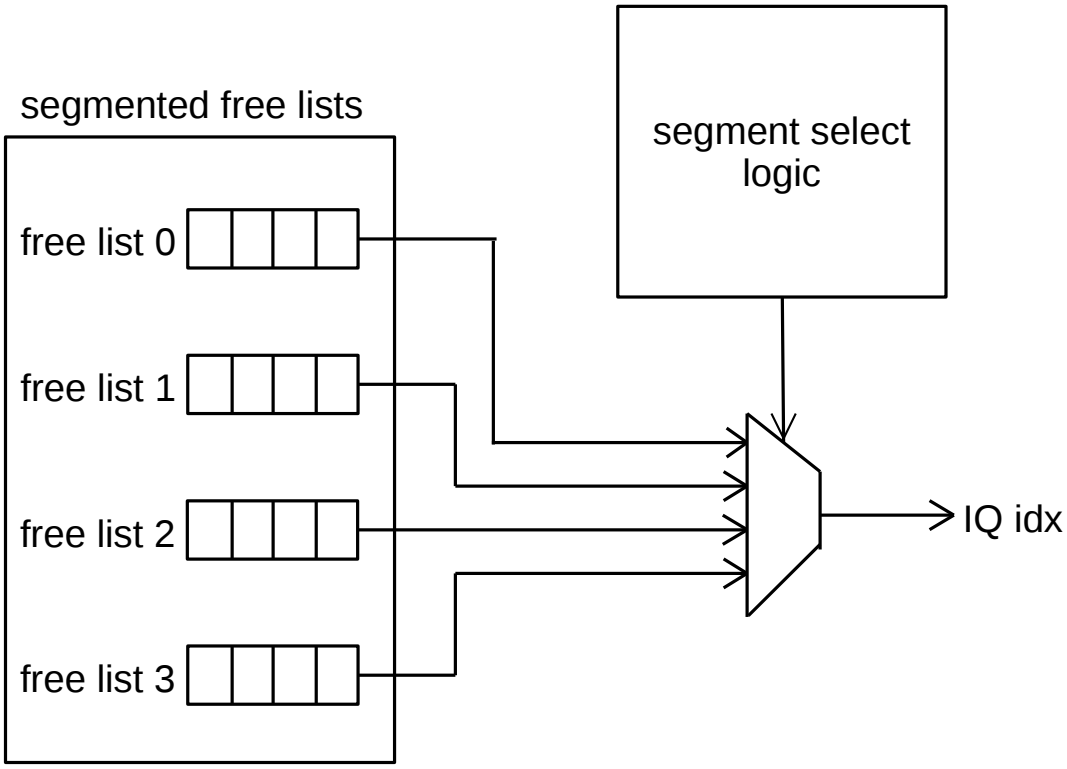

図 4.2: 提案手法におけるディスパッチエントリの決定回路

- **第 1 ソース・オペランドがレディでない場合：** 第 1 ソース・タグの下位ビットと番号が同じセグメントを選択する．選択されたセグメントに空きエントリがある場合，ディスパッチ可能であるため，対応するフリー・リストから読み出したエントリにディスパッチを行う．対応するセグメントに空きがない場合は，セグメントに空きが出るまでディスパッチをストールさせる．

- **第 1 ソース・オペランドがレディである場合：**この場合，第 1 ソース・タグの比較は行われないため，どのセグメントにディスパッチしても問題ない．このような場合を**セグメント・インディペンデント**と呼ぶ．セグメント・インディペンデントの場合，空きエントリのあるセグメントから，ラウンドロビンでディスパッチするセグメントを選択しディスパッチする．

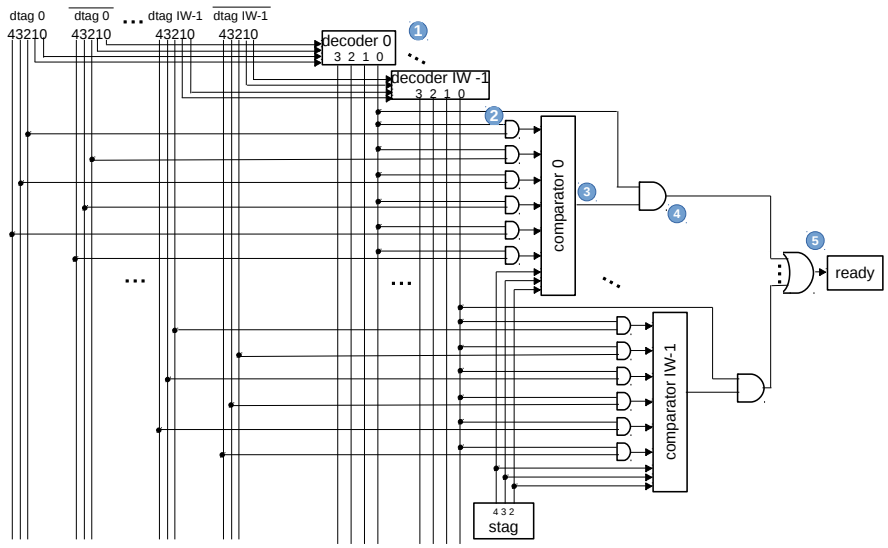

図 4.3: 提案手法におけるタグ比較回路（第 0 セグメント）

例として，第 1 ソース・オペランドがレディでなく，タグが 15（1111_2）である命令を，図 4.1 に示す 4 つに分割された発行キューにディスパッチする場合を考える．第 1 ソース・タグの下位 2 ビットが 3（11_2）であるので，この命令は第 3 セグメントにディスパッチされる．

なお，ソース・オペランドを使用しない命令も存在するが，そのような命令はディスパッチ時にソース・オペランドがレディであるものとして扱う．

4.1.3　提案手法におけるウェイクアップ

提案手法におけるウェイクアップでは，ディスティネーション・タグの下位ビットがセグメント番号と一致するセグメントでのみ，第 1 ソース・タグのタグ比較器を動作させ比較を行う．一致しないセグメントはタグ比較器を動作させない．これは，ディスティネーション・タグの下位ビットと番号が一致しないセグメントには，第 1 ソース・タグの下位ビットがディスティネーション・タグの下位ビットと異なる命令しか入っておらず，タグは必ず不一致となるためである．

例として，放送されたディスティネーション・タグが 6（110_2）で，発行キューが図 4.1 のように 4 つのセグメントに分割されている場合を考える．この場合，下位ビットは 2（10_2）であるため，第 2 セグメントでのみ，第 1 ソース・タグのタグ比較を行う．

なお，第 2 ソース・タグのタグ比較に関しては，セグメントの番号とタグの下位ビットに関係性はないため，すべてのセグメントでタグ比較を行う必要がある．

提案手法におけるタグ比較の回路を図 4.3 に示す．同図は 4 つのセグメントに分割された発行キューのうち，第 0 セグメントのエントリにおける，第 1 ソース・タグの比較回路を示している．タグ・ビット数は 5 とし、発行幅を IW とする．

タグ比較の動作を図中の番号を用いて説明する．①放送されるディスティネーション・タグの下位 2 ビットはデコーダへ送られる．デコーダはセグメント数だけ信号線を出力する．第 n 番目の信号線は，第 n セグメントでのタグ比較が有効であることを示す．つまり，ディスティネーション・タグの下位ビットが n の場合，n 番目の出力線のみ H を出力し，残りはすべて L を出力する．

②AND ゲートによって，デコーダからの信号線が H の場合にのみ，ディスティネーション・タグの高位ビット及びその反転信号がタグ比較器へ入力される．図 4.3 に示す回路は第 0 セグメントのタグ比較回路であるため，デコーダの 0 番目の信号線が AND ゲートに入力されている．

デコーダからの信号線が H の場合，つまり，ディスティネーション・タグの下位ビットとセグメント番号が一致していた場合のみ，比較器に有効なディスティネーション・タグの高位ビットとその反転信号が送られ，ソース・タグの高位ビットと比較が行われる．デコーダからの信号線が L の場合，ディスティネーション・タグとその反転信号がどちらも L としてタグ比較器へ入力される．この場合，ディスティネーション・タグとその反転信号に接続されたプルダウン・トランジスタがすべて OFF となるため，マッチ線はディスチャージされず，電力を消費しない．

③タグ比較の結果，タグの高位ビットが一致した場合は，比較器から H が出力される．

④タグ比較器が H を出力し，かつデコーダからの信号が H である場合に，タグ比較は一致となる．

⑤いずれかのディスティネーション・タグがソース・タグと一致した場合に，ソース・オペランドのレディ・ビットがセットされる．

4.2　第 2 ソース・タグ比較の削減

4.1 節で述べた手法では，命令の第 2 ソース・タグのタグ比較回数は削減できない．そこで本節では，第 2 ソース・タグの比較回数の削減を可能とする**スワップ**と**サブ・セグメント**という 2 つの手法を提案する．

4.2.1　スワップ

スワップは，第 1 ソース・タグと第 2 ソース・タグを格納するフィールドを交換し，第 2 ソース・タグの下位ビットをもとにディスパッチするセグメントを決定する手法である．以下で詳しく説明する．

第 1 ソース・オペランドがレディで，第 2 ソース・オペランドがレディでない場合について説明する．この場合，4.1 節で説明した方法では，命令はセグメント・インディペンデントとしてディスパッチされる．第 1 ソース・オペランドは既にレディであるため，比較は第 2 ソース・タグについてのみ行われるが，第 2 ソース・タグのタグ比較は全てのセグメントで行われるため，タグ比較の回数は削減されない．

そこでこのような場合に，第 1 ソース・タグと第 2 ソース・タグを交換し（スワップ），第 2 ソース・タグの下位ビットを使用してディスパッチするセグメントを選択する．これにより，4.1 節で述べたセグメント化の効果でタグ比較回数が削減される．なお，スワップではタグを交換するが，ペイロード RAM に格納するソース・タグを交換するわけではないので，命令の意味は保持される．

スワップを行う場合のセグメント選択アルゴリズム

セグメント選択回路は，以下に示すアルゴリズムによってディスパッチするセグメントを決定する．

- **両ソース・オペランドともレディでない場合**：第 1 ソース・タグでセグメントを選択する．
- **第 1 ソース・オペランドのみレディである場合**：スワップを行い，第 2 ソース・タグでセグメントを選択する．
- **第 2 ソース・オペランドのみレディである場合**：第 1 ソース・タグでセグメントを選択する．
- **両ソース・オペランドがレディである場合**：セグメント・インディペンデントとしてラウンドロビンでセグメントを選択する．

なお，両ソース・オペランドがレディのとき以外で，選択されたセグメントに空きがない場合は，ディスパッチをストールして当該のセグメントに空きが出るまで待ち合わせる．

4.2.2 サブ・セグメント

サブ・セグメント方式は，第 1 ソース・タグの下位ビットに応じて分割されるセグメントを，第 2 ソース・タグの下位ビットに応じてさらに細かく分割する．第 2 ソース・タグの下位ビットによる分割をサブ・セグメント（S-seg）と呼び，従来の第 1 ソース・タグによる分割をサブ・セグメントに対応してメイン・セグメント（M-seg）と呼ぶこととする．

サブ・セグメントを導入した発行キューの分割を図 4.4 に示す．黒色の枠で示す各メイン・セグメントを，赤色と青色で示すようにさらにサブ・セグメントに分割する．同図は，メイン・セグメント数が 4 ，サブ・セグメント数が 2 の場合の例を表している．各セグメントの左には，(M-seg, S-seg）という形式でメイン及びサブ・セグメントの番号を表している．

図 4.4: サブ・セグメントを実装した発行キュー

サブ・セグメント方式について，ディスパッチとウェイクアップの動作をそれぞれ説明する．

サブ・セグメントにおけるディスパッチ

サブ・セグメント方式におけるディスパッチにおいては，フリー・リストを M-seg × S-seg だけ用意する．図 4.4 に示した例の場合 8 個のフリー・リストが必要となる．

サブ・セグメント方式におけるセグメント選択のアルゴリズムに関して説明する．アルゴリズムはソース・オペランドのレディ状況によって異なるため，以下ですべての場合に関して説明する．説明を簡単にするため，命令 $p5 = p13 + p6$ を，図 4.4 に示す発行キューにディスパッチする場合について例示する．第 1 ソース・タグが 13 で，第 2 ソース・タグが 6 である．

- **両ソース・オペランドともレディでない場合**：第 1 ソース・タグでメイン・セグメントを，第 2 ソース・タグでサブ・セグメントを選択する．例の場合，第 1 ソース・タグ 13（1101_2） の下位ビット 1（01_2）より，メイン・セグメントは 1 となる．また，第 2 ソース・タグ 6（110_2） の下位ビット 0（0_2）より，サブ・セグメントは 0 となる．従って（1, 0）のセグメントを選択する．

- **第 1 ソース・オペランドのみレディである場合**：第 2 ソース・タグでサブ・セグメントを選択する．例の場合，第 2 ソース・タグ 6（110_2） の下位ビット 0（0_2）より，サブ・セグメントは 0 となる．第 1 ソース・オペランドは既にレディであるため，メイン・セグメントの制限はない．従って，(0, 0) , (1, 0) , (2, 0) , (3, 0）のいずれかのセグメントをラウンドロビンで選択する．このように，メイン・セグメントの制限がない場合をメイン・セグメント・インディペンデント（M-seg インディペンデント）と呼ぶこととする．

- **第 2 ソース・オペランドのみレディである場合**：第 1 ソース・タグでメイン・セグメントを選択する．例の場合，第 1 ソース・タグ 13（1101_2） の下位ビット 1（01_2）より，メイン・セグメントは 1 となる．第 2 ソース・オペランドは既にレディであるため，サブ・セグメントの制限はない．従って，(1, 0）または（1, 1）のいずれかのセグメントをラウンドロビンで選択する．このように，サブ・セグメントの制限がない場合をサブ・セグメント・インディペンデント（S-seg インディペンデント）と呼ぶこととする．

- **両ソース・オペランドがレディである場合**：セグメント・インディペンデントとしてラウンドロビンでセグメントを選択する．

サブ・セグメントにおけるウェイクアップ

第 1 ソース・タグの比較は，ディスティネーション・タグの下位ビットがメイン・セグメント番号と一致するセグメントのみで行う．また，第 2 ソース・タグの比較は，ディス

ティネーション・タグの下位ビットがサブ・セグメント番号と一致するセグメントのみで行う．このような比較により，第 1 ソース・タグだけでなく，第 2 ソース・タグの比較に関しても，「1/サブ・セグメント数」まで削減が可能となる．

サブ・セグメントとスワップの併用

サブ・セグメント方式はスワップと併用することが可能である．併用する場合は，ディスパッチ時に第 1 ソース・オペランドのみレディである場合の選択アルゴリズムを，以下のように変更する．

- **第 1 ソース・オペランドのみレディである場合**：スワップを行い，第 2 ソース・タグでメイン・セグメントを選択する．例の場合，第 2 ソース・タグ 6（110_2） の下位ビット 2（10_2）より，メイン・セグメントは 2 となる．第 1 ソース・オペランドは既にレディであるため，S-seg インディペンデントである．従って，(2, 0）または（2, 1）のいずれかのセグメントを選択する．

サブ・セグメント方式とスワップを併用することによって，ディスパッチ時に第 1 ソース・オペランドのみレディである命令におけるタグ比較回数の削減が「1/サブ・セグメント数」から「1/メイン・セグメント数」となる．従って，図 4.4 に示した分割のようにメイン・セグメント数がサブ・セグメント数よりも多い場合に，タグ比較回数をより多く削減できる．

第5章　SWITCH 方式

5.1　容量効率の低下

提案手法には発行キューの容量効率が低下するという問題点がある．この問題点は，発行キューの容量効率が重要なプログラムにおいて，性能低下を引き起こす．本節では，容量効率が低下する原因について説明した後，容量効率の低下により性能低下を引き起こすプログラムの特徴に関して説明する．

5.1.1　提案手法による容量効率低下の原因

発行キューの容量効率の低下に関して，図 5.1 を用いて説明する．図において，灰色のエントリは命令を保持していることを示している．

図の状態の発行キューに，新たに命令 $p2 = p5 + p6$ をディスパッチする場合を考える．この命令のソース・オペランドは両方レディでないとする．この場合，第 1 ソース・タグの下位ビットから第 1 セグメントにディスパッチされることが決定する．しかし，第 1 セグメントに空きエントリはないため，空きが出るまでディスパッチをストールさせ，待ち合わせを行う必要がある．

このように，命令がディスパッチされるセグメントに空きがない場合，他のセグメントに空きがあってもディスパッチをストールする必要があり，その結果，提案手法ではセグメント化されていない発行キューと比較して容量効率が低下する．

5.1.2　容量効率の低下による性能低下

プログラムには，性能が発行キューの容量に敏感なものとそうでないものとがある [16, 17, 21]. 次の 2 つの特徴のうちいずれかに当てはまるプログラムでは，性能が発行キュー

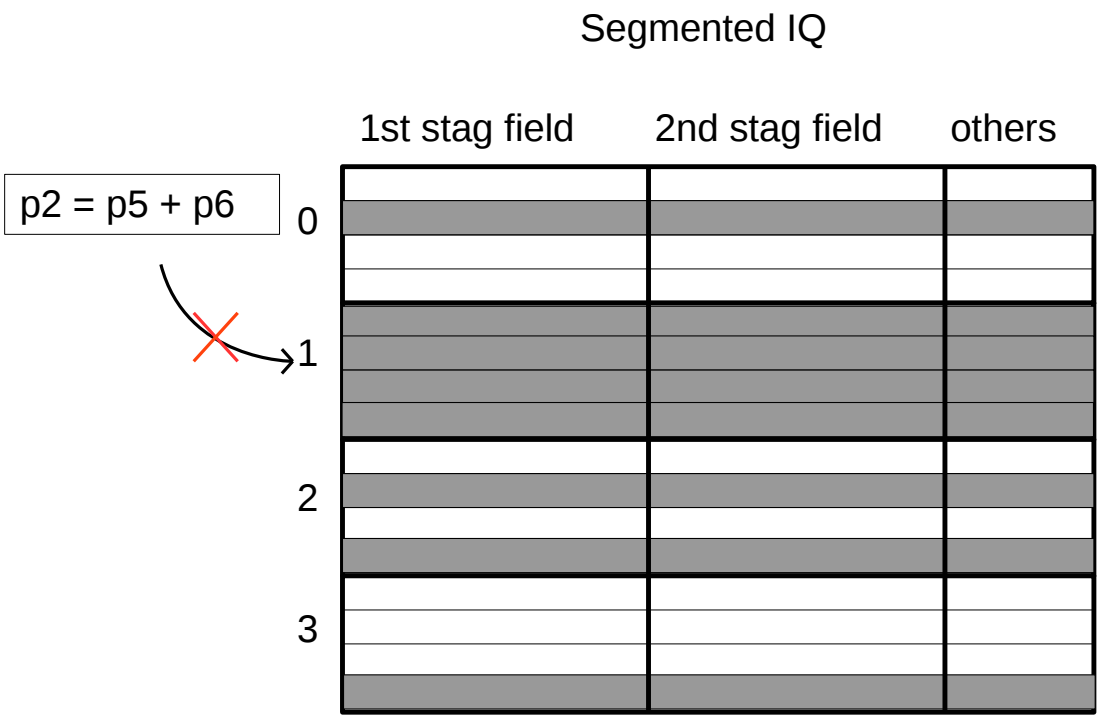

図 5.1: 容量効率が低下する例

の容量に敏感なため，与えられた発行キューの容量においては，その利用効率が重要である．このため，提案手法による容量効率の低下によって性能が低下する．

- 命令レベル並列性（**ILP**：**Instruction Level Parallelism**）が高いプログラム
- メモリ・レベル並列性（**MLP**：**Memory Level Parallelism**）が高いプログラム

ILP が高いプログラムでは，できるだけ発行キューに命令を多く供給し，より多くの命令を並列に発行できるようにすることで高い性能が得られる．発行キューの容量効率が低下すると，並列に発行できる命令数が減少するため，性能が低下する．

MLP が高いプログラムでは，できるだけ多くのキャッシュ・ミスを並列に実行することにより，メモリ・アクセスのレイテンシが実行時間に与える影響を縮小できる．発行キューの容量効率が低下すると，並列に処理できるメモリ・アクセスが減少するため，性能が低

下する．

これらのことから，ILP もしくは MLP が高い場合には，提案手法による容量効率の低下を最小限に抑える工夫が必要となる．

5.2 容量効率低下への対策：SWITCH 方式

発行キューの容量効率低下による性能低下を抑制する方式として，**SWITCH** と呼ぶ方式を提案する．SWITCH 方式では，次のようにして性能低下を抑制する．

- セグメント選択回路の選択アルゴリズムとして，容量効率は低下するが，タグ比較回数を多く削減できるような選択を行う **AGGRESSIVE モード** と，タグ比較回数の削減率は低下するが，容量効率が大きく低下しないような選択を行う **CONSERVATIVE モード** の 2 つを用意する．
- 実行プログラムの ILP 及び MLP を一定のインターバルで監視し，ILP もしくは MLP が高いと判断されたなら次のインターバルでは CONSERVATIVE モードでディスパッチし，そうでないなら AGGRESSIVE モードでディスパッチを行う．

本節では，まず 2 つのセグメント選択のアルゴリズムに関して説明を行う．その後，ILP 及び MLP の評価方法と，切り替えアルゴリズムに関して説明する．

5.2.1 2 つのセグメント選択アルゴリズム

?? SWITCH 方式では，タグ比較回数の削減重視の AGGRESSIVE モードと，容量効率重視の CONSERVATIVE モードの 2 つを適切に切り替えて使用する．各モードには，タグ比較回数の削減と容量効率に関して，表 5.1 に示すトレード・オフの関係がある．それぞれのセグメントの選択方法に関して説明する．

なお，SWITCH 方式はサブ･セグメント方式と併用が可能であるが，説明を簡単にするため，サブ･セグメントを使用しない場合に関して説明する．サブ･セグメントと SWITCH 方式を併用するような拡張は容易に行うことができる．

表 5.1: 2 つのセグメント選択モードのトレード・オフ

モード	タグ比較回数の削減	容量効率
AGGRESSIVE	○	×
CONSERVATIVE	×	○

AGGRESSIVE モード

AGGRESSIVE モードは，4.2.1 節で示した選択アルゴリズムを使用してディスパッチするエントリを決定する．このモードでは，選択されたセグメントに空きがない場合，他のセグメントに空きがあってもディスパッチは行わないため，容量効率が低下する．しかし，セグメント化の利益を最大限利用し，タグ比較回数を大幅に削減できる．

CONSERVATIVE モード

AGGRESSIVE モードでは，命令のソース・オペランドが両方レディであり，セグメント・インディペンデントとしてディスパッチできる場合以外では，ソース・タグによって選択されるセグメントに空きがない場合にディスパッチをストールさせる．これに対し，CONSERVATIVE モードでは，以下で説明する工夫を行うことによって，このディスパッチのストールを回避し，容量効率の低下を抑制する．

- 両ソース・オペランドともレディでない場合：

  AGGRESSIVE モードでは第 1 ソース・タグの下位ビットによってセグメントを選択する．選択されたセグメントに空きがない場合，ディスパッチを行わない．これに対して CONSERVATIVE モードでは，第 1 ソース・タグによって選択されたセグメントに空きがない場合には，**スワップ**[1]してディスパッチを試みる．スワップするため，第 2 ソース・タグにより選択されるセグメントに空きがあればディスパッチが可能となる．

[1] 4.2.1 節では，スワップの定義を「第 1 ソース・オペランドのみレディの場合に，第 1 ソース・タグと第 2 ソース・タグを書き込むフィールドを交換する」としていたが，本節以降ではこの定義を拡大し，単に「第 1 ソース・タグと第 2 ソース・タグを書き込むフィールドを交換する」という意味で用いる．

- **第 1 ソース・オペランドのみレディである場合：**

  AGGRESSIVE モードでは，スワップを行い，第 2 ソース・タグでセグメントを選択する．選択されたセグメントに空きがない場合，ディスパッチを行わない．これに対して CONSERVATIVE モードでは，第 2 ソース・タグによって選択されたセグメントに空きがない場合には，**スワップをやめて**ディスパッチする．スワップをやめるため，第 1 ソース・タグによってセグメントが選択されるが，第 1 ソース・オペランドは既にレディであるため，どのセグメントにディスパッチしても良い．従って，セグメント・インディペンデントとしてディスパッチが可能となる．

- **第 2 ソース・オペランドのみレディである場合：**

  AGGRESSIVE モードでは第 1 ソース･タグでセグメントを選択する．選択されたセグメントに空きがない場合，ディスパッチを行わない．これに対して CONSERVATIVE モードでは，第 1 ソース・タグによって選択されたセグメントに空きがない場合には，**スワップして**ディスパッチする．スワップするため，第 2 ソース・タグによってセグメントが選択されるが，第 2 ソース・オペランドは既にレディであるため，どのセグメントにディスパッチしても良い．従って，セグメント・インディペンデントとしてディスパッチが可能となる．

上述の工夫によって，CONSERVATIVE モードでは，どちらかのソース・オペランドがレディである場合は，必ずディスパッチが可能となる．また，両ソース・オペランドともレディでない場合でも，第 1 ソース・タグにより選択されるセグメントと第 2 ソース・タグにより選択されるセグメントのうち，いずれかのセグメントに空きがあればディスパッチが可能となる．従って，ストールする確率は大きく減少する．

以下に，CONSERVATIVE モードにおけるセグメントの選択アルゴリズムをまとめる．

- **両ソース・オペランドともレディでない場合**：第 1 ソース・タグでセグメントを選択する．選択したセグメントに空きがない場合，スワップして第 2 ソース・タグをもとにセグメントを決定する．なおも空きがない場合はストールする．

- **第 1 ソース・オペランドのみレディである場合**：スワップを行い，第 2 ソース・タグでセグメントを選択する．選択したセグメントに空きがない場合は，スワップをやめて，セグメント・インディペンデントとしてラウンドロビンでセグメントを選択する．
- **第 2 ソース・オペランドのみレディである場合**：第 1 ソース・タグでセグメントを選択する．選択したセグメントに空きがない場合はスワップを行い，セグメント・インディペンデントとしてラウンドロビンでセグメントを選択する．
- **両ソース・オペランドがレディである場合**：セグメント・インディペンデントとしてラウンドロビンでセグメントを選択する．

CONSERVATIVE モードにおけるタグ比較回数の削減

CONSERVATIVE モードでは，AGGRESSIVE モードと比較してタグ比較回数の削減率が 低下する可能性がある．この理由について説明する．例として，第 2 ソース・オペランドのみレディである命令をディスパッチする場合について説明する．

CONSERVATIVE モードでは，まず第 1 ソース・タグでセグメントを選択する．選択されたセグメントに空きがあれば，そのセグメントにディスパッチする．この場合，レディでない第 1 ソース・タグが，セグメント化によってタグ比較回数が削減される第 1 ソース・タグのフィールドに書き込まれるため，AGGRESSIVE モードと同様にタグ比較回数が削減される．

第 1 ソース・タグによって選択されたセグメントに空きがなければ，CONSERVATIVE モードではスワップしてセグメント・インディペンデントとしてディスパッチする．この場合，タグ比較回数の削減は行うことができない．これは，既にレディである第 2 ソース・オペランドのタグが，セグメント化によってタグ比較回数を削減できる第 1 ソース・タグのフィールドに書き込まれ，一方で，まだレディでなくタグ比較が行われる第 2 ソース・オペランドのタグが，セグメント化によってタグ比較回数が削減されない第 2 ソース・タグのフィールドに書き込まれるためである．

AGGRESSIVE モードでは，第 1 ソース・タグによって選択されたセグメントに空きがなければ，ストールして空きが出るまで待ち合わせる．このストールにより，容量効率は低下するが，空きが出たあとディスパッチするため，タグ比較回数は削減される．これに対して CONSERVATIVE モードでは，タグ比較回数の削減は行えなくなるが，スワップしてディスパッチすることによってストールを回避し，容量効率の低下を防ぐ．

従って，CONSERVATIVE モードは，タグ比較回数の削減をある程度犠牲にして，発行キューの容量効率の低下を抑制するアルゴリズムであるといえる．

5.2.2 モードの切り替え

SWITCH 方式では， AGGRESSIVE と CONSERVATIVE の 2 つのモードを，実行プログラムの ILP や MLP の量に応じて切り替えて使用する．ここで重要となるのは ILP や MLP の量の評価方法である．本研究では， ILP の評価方法として Issue Stall Rate（ISR）という評価値を使用する．また， MLP の評価方法としては，最終レベル・キャッシュ（LLC: last-level cache）の MPKI（misses per kilo instructions）を使用する．それぞれに関して詳しく説明した後，切り替えアルゴリズムを説明する．

Issue Stall Rate（ISR）

ISR は,「インターバルの全サイクルのうち 1 命令も発行されないサイクルの割合」を表す指標である．ILP が高い場合，多くのサイクルで命令が発行されるため，ISR は低い値を示す．一方， ILP が低い場合には，命令が発行されないサイクルが一定の割合で発生するため，ISR は高くなる．

あらかじめ ISR にしきい値を設け，インターバルでの ISR がしきい値を下回った場合に ILP が高いと判断し，そうでなければ低いと判断する．

LLC MPKI

LLC MPKI は LLC のキャッシュ・ミスの発生頻度を表す指標である．LLC MPKI があらかじめ定めたしきい値を上回った場合に MLP が高いと判断し，そうでなければ低いと判断する．

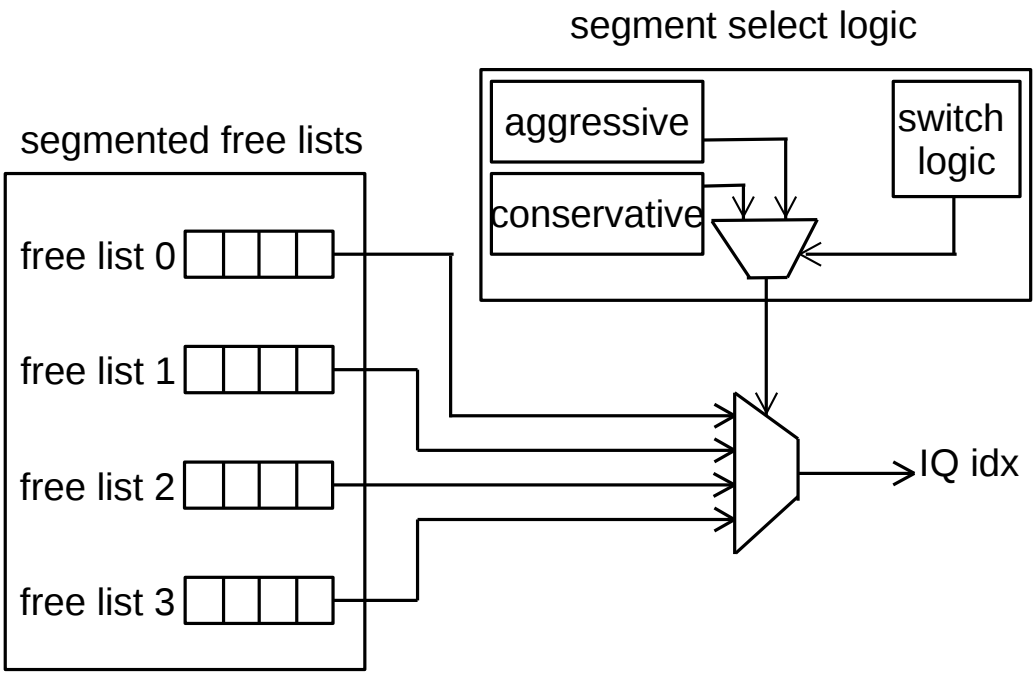

図 5.2: SWITCH 方式におけるディスパッチエントリの決定回路

切り替えアルゴリズム

切り替えアルゴリズムは以下に示すとおりである．一定のインターバルにおいて，ISR と LLC MPKI を測定し，ILP および MLP の高低を判断する．ILP または MLP のいずれかが高いと判定された場合，次のインターバルを CONSERVATIVE モードで実行する．ILP と MLP がどちらも低いと判定された場合，次のインターバルを AGGRESSIVE モー

ドで実行する．

SWITCH 方式におけるディスパッチするエントリの決定回路を図 5.2 に示す．AGGRESSIVE と CONSERVATIVE の 2 つの選択アルゴリズムのうち，どちらを利用するかを SWITCH 回路が選択し，その結果に応じてセグメントが選択される．

第6章　評価

本章では，提案手法の評価を行う．

6.1　評価環境

評価環境について説明する．性能やタグ比較回数を評価するために，SimpleScalar v.3.0a をベースに作成したシュミレータを使用した．評価で仮定したプロセッサ構成を表 6.1 に示す．

提案手法の各種パラメータを，表 6.2 と表 6.3 に示す．これらのパラメータは，5% 程度の性能低下と引き換えにできるだけ多くのタグ比較回数削減を達成できるよう，実験によって求めた最適なパラメータである．表 6.2 は SWITCH 方式をサブ・セグメントと併用する場合に最適なパラメータを，表 6.3 はサブ・セグメントを使用しない場合に最適なパラメータを示している．各パラメータの評価に関しては，で説明する．

測定ベンチマークには，SPEC CPU 2017 ベンチマークのうち，int 系 9 本と fp 系 9 本の計 18 本を使用した．プログラムの入力には ref データ・セットを用いた．ベンチマークの測定区間は，プログラムの先頭から 16B 命令をスキップした後の 100M 命令である．

6.1.1　ベンチマークの分類

提案手法は，プログラムの ILP や MLP に着目した制御を行う．そこで，SPEC CPU 2017 ベンチマークを ILP が高いベンチマーク，MLP が高いベンチマーク，いずれも低いベンチマークの 3 種類に分類する．ここて，ILP が高い及び MLP が高いベンチマークとは，次の条件を満たすベンチマークである．

- ILP：IPC が 3.5 以上のベンチマーク

表 6.1: プロセッサの基本構成

Pipeline width	8 instructions wide for each of fetch, decode, issue, and commit
Reorder buffer	300 entries
IQ	128 entries
Load/Store queue	128 entries
Physical registers	300（int） + 300（fp）
Branch prediction	16KB Perceptron predictor [27] 2K-set 4-way BTB 10-cycle misprediction penalty
Function unit	4 iALU, 2 iMULT, 3 FPU, 2 LSU
L1 D-cache	32KB, 8-way, 64B line 2-cycle hit latency
L1 I-cache	32KB, 8-way, 64B line 2-cycle hit latency
L2 cache	2MB, 16-way, 64B line 12-cycle hit latency
Main memory	300-cycle latency 8B/cycle bandwidth
Prefetch	stream-based,32-stream tracked, 16-line distance, 2-line degree, prefetch to L2 cache

- MLP：LLC MPKI が 2 以上のベンチマーク

分類結果を表 6.4 に示す．また，以降に示す図において，ILP(青色) 及び MLP(赤色) の表記は，そのベンチマークが ILP もしくは MLP が高いことを表す．

6.2 評価結果：提案手法の効果

提案手法によるタグ比較回数の削減と性能低下に関して評価を行う．評価モデルは以下の 4 種類である．

- BASE：セグメント化しない通常の IQ を使用するモデル
- AGGRESSIVE：提案手法において常に AGGRESSIVE モードで実行するモデル
- CONSERVATIVE：提案手法において常に CONSERVATIVE モードで実行するモデル

表 6.2: 提案手法のパラメータ構成 (サブ・セグメント使用)

メイン・セグメント数	8
サブ・セグメント数	2
切り替えインターバル	10K instructions
ISR しきい値	15%
LLC MPKI しきい値	2.0

表 6.3: 提案手法のパラメータ構成 (サブ・セグメント不使用)

メイン・セグメント数	8
サブ・セグメント数	2
切り替えインターバル	10K instructions
ISR しきい値	15%
LLC MPKI しきい値	2.0

表 6.4: ベンチマークの分類

high ILP	exchange2, leela, xz, bwaves cactuBSSN, cam4, imagick nab, pop2, roms
high MLP	omnetpp, xalancbmk, lbm
low ILP and low MLP	deepsjeng, mcf perlbench, x264, fotonik3d

- SWITCH：提案手法の SWITCH 方式を使用するモデル

なお，評価に関してはサブ・セグメントを使用しない場合と使用する場合に関してそれぞれ評価を行う．

6.2.1　サブ・セグメントを使用しない場合

サブ･セグメントを使用しない場合の提案手法に関して評価する．提案手法に関するパラメータは表 6.3 に示したものを使用する．セグメント数は 16 であり，(16, 1) と表記する．

タグ比較回数の削減

図 6.1 に，提案手法の BASE モデルに対するタグ比較回数の割合をベンチマークごとに示す．AGGRESSIVE と CONSERVATIVE のタグ比較回数削減に関しては、同図より，いずれのベンチマークにおいても，AGGRESSIVE のほうがタグ比較回数が少ないことがわ

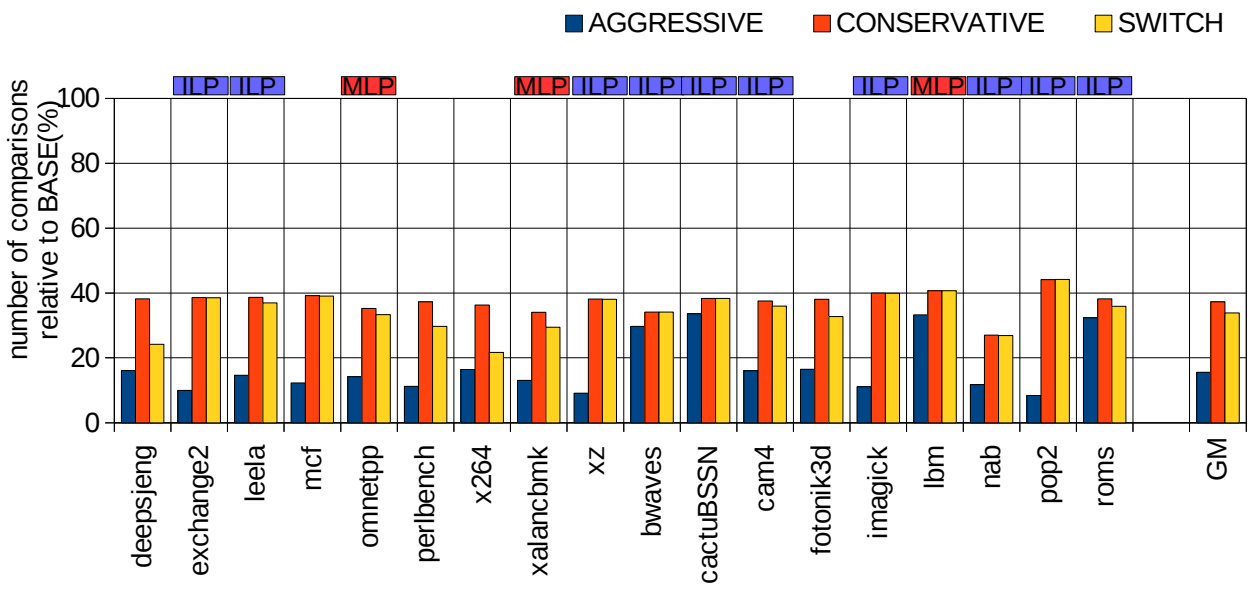

図 6.1: 提案手法によるタグ比較回数の削減 (16, 1)

かる．その差は 平均で 20% 程度となっており，AGGRESSIVE モードのタグ比較回数を積極的に削減できるという性質が確認できる。

同図より SWITCH 方式では，平均で BASE モデルの 30% 程度のタグ比較回数となっており，70% の削減を達成している．

SWITCH 方式では，ILP や MLP の高いベンチマークにおいては CONSERVATIVE と同程度のタグ比較回数であるのに対して，そうでないベンチマーク (deepsjeng，x264 など) においては AGGRESSIVE に近いタグ比較回数となっていることがわかる．したがって，発行キューの容量効率が重要でないベンチマークにおいては，AGGRESSIVE モードを選択して積極的にタグ比較回数の削減が行えていることがわかる．

性能低下

図 6.2 に，BASE に対する提案手法による性能低下をベンチマークごとに示す．同図より，SWITCH 方式による性能低下は最大で 4% 程度であり，多くのベンチマークでは 0% に近く性能はほとんど低下しないということが確認できる．

SWITCH 方式の有効性に関して述べる．同図より，ILP や MLP が高い cactusBSSN

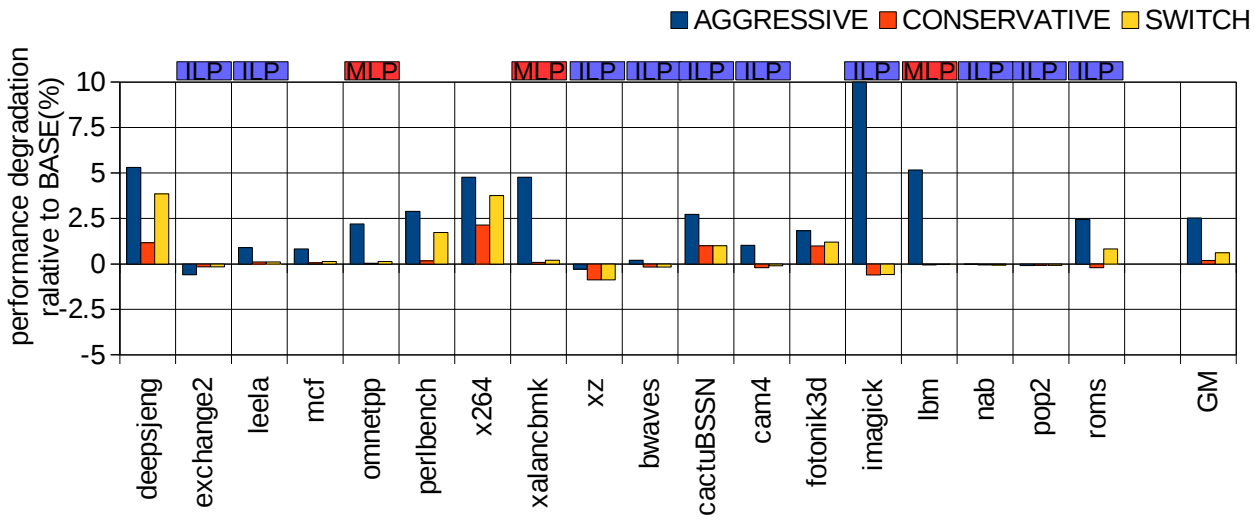

図 6.2: 提案手法による性能低下 (16,1)

や imagick，lbm などのベンチマークにおいて，AGGRESSIVE では大きく性能低下しているのに対して，CONSERVATIVE では性能低下が抑制されていることが分かる．そして SWITCH では，CONSERVATIVE と同程度の性能低下にとどまっている．従って，容量効率が性能にとって重要なプログラムにおいて，SWITCH 方式によって性能低下が抑制できていることが分かる．

図 6.3 に各モデルでの発行キューの占有率を示す．占有率とは，発行キューの全エントリのうち使用された割合であり，この値が BASE のそれに近いほど容量効率が低下していないことを示す．

同図より，AGGRESSIVE では BASE に対して占有率が大きく低下しているのに対して， CONSERVATIVE では占有率の低下がある程度抑制できていることが分かる．そして，ILP や MLP が高いベンチマークでは SWITCH 方式での占有率が CONSERVATIVE と同程度となっていることが分かる．このことからも，SWITCH 方式では，容量効率の性能に対する重要性に応じて適切にモードを選択し，容量効率の低下による性能低下を抑制できており， SWITCH 方式が有効であるとわかる．

図 6.2 より，いくつかのベンチマークでは性能が僅かに向上していることが分かる．こ

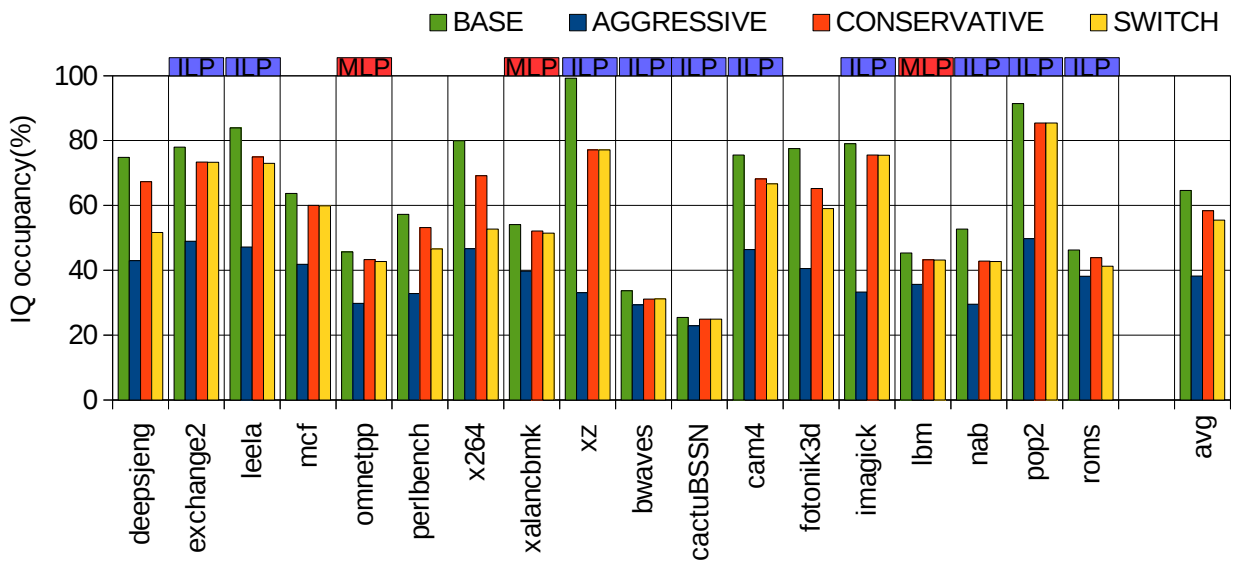

図 6.3: IQ の占有率の変化 (16,1)

の理由に関して説明する．一般にランダム・キュー方式の発行キューには，命令がプログラム順に並んでいないため，最も優先して発行すべき命令の発行が遅れる可能性があるという欠点が存在する．ランダム・キューでは，命令の並びが年齢についてランダムになる一方，選択論理は，下のエントリほど優先して発行命令を選択するため，レディ命令が発行幅以上に存在する発行コンフリクトが生じた場合，誤った優先度で命令を選択することが生じる．

提案手法では発行キューの容量効率が低下するため，発行キュー内の命令数が少なくなり，結果的に発行コンフリクトが生じる確率が低下し，問題が生じにくくなり，僅かに性能が向上する．また，ILP や MLP が高いにもかかわらず，AGGRESSIVE においても性能低下の小さいベンチマークがあることが分かる．こういったベンチマークにおいても，発行コンフリクトの緩和による性能向上が発生しているため，容量効率の低下による性能低下が小さいと考えられる．

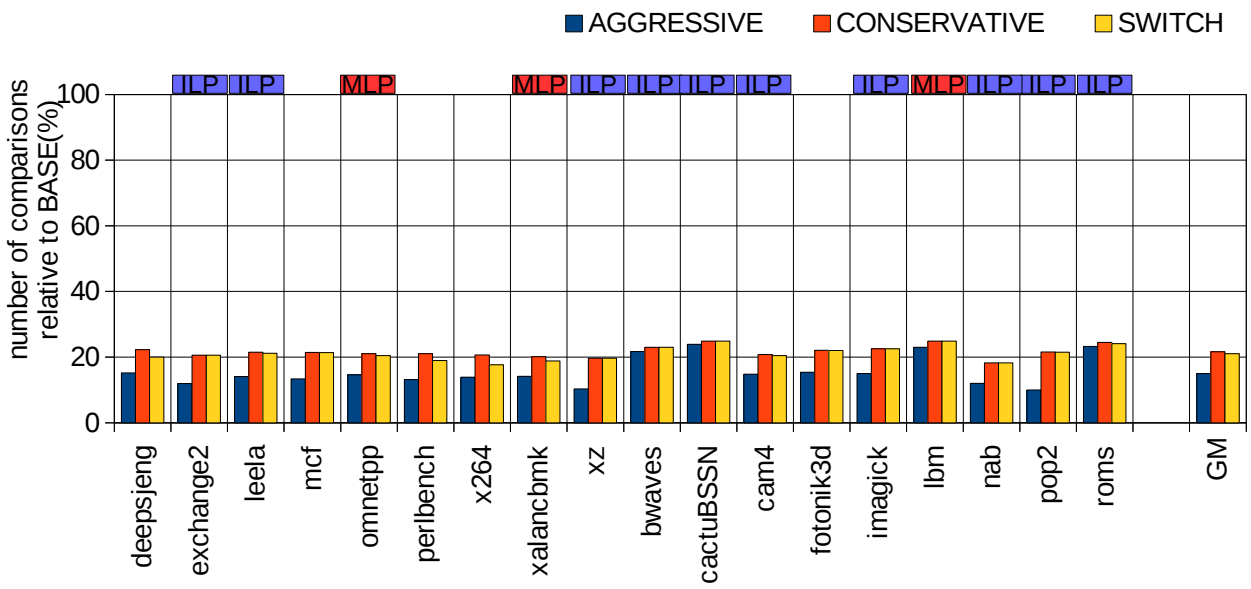

図 6.4: 提案手法によるタグ比較回数の削減 (8, 2)

6.2.2　サブ・セグメントを使用する場合

サブ・セグメントを使用する場合の提案手法に関して評価する．提案手法に関するパラメータは表 6.2 に示したものを使用する．メイン・セグメント数は 8，サブ・セグメント数は 2 であり，(8, 2) と表記する．

タグ比較回数の削減

図 6.4 に，提案手法の BASE モデルに対するタグ比較回数の割合をベンチマークごとに示す．AGGRESSIVE と CONSERVATIVE を比較すると，(16, 1) の場合と同様に，AGGRESSIVE のタグ比較回数が少ないことが分かる．

セグメントの総数が同じである (16, 1) と (8, 2) の CONSERVATIVE を比較すると，(16, 1) の場合が平均で 40% 程度であるのに対して， (8, 2) では 20 % 程度と，(8, 2) のほうがより削減できていることが分かる．これは，サブ・セグメントを使用する場合，?? 節で説明した CONSERVATIVE モードでのストールの回避を行った際にも，タグ比較の削減が可能となるためである．図 6.7 を用いて詳しく説明する．

図に示す命令をディスパッチする場合を考える．サブ・セグメントを使用しない場合 (図

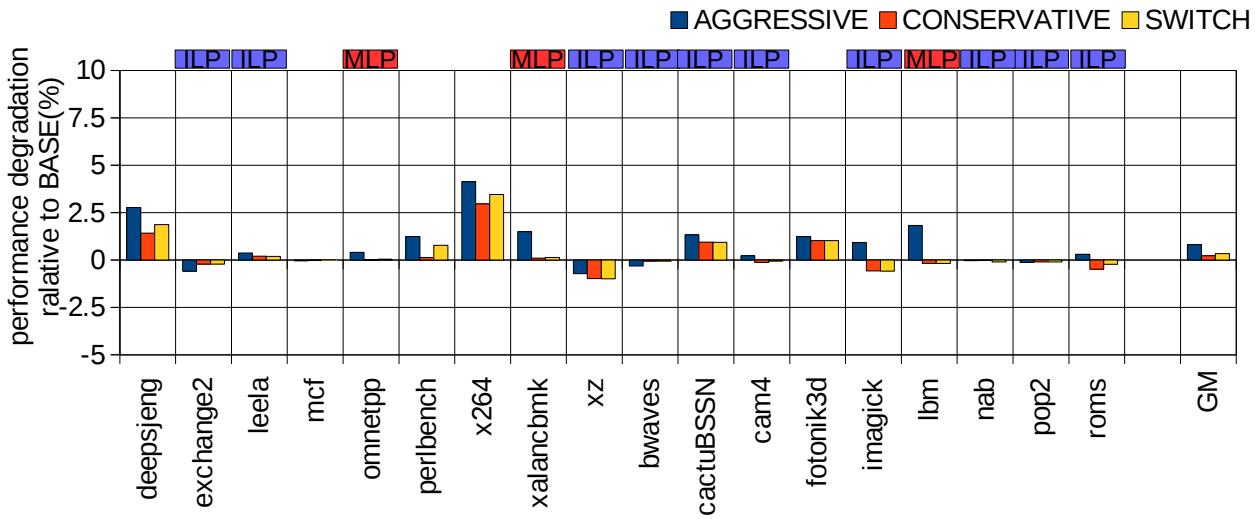

図 6.5: 提案手法による性能低下 (8,2)

左側)，第 1 ソース・タグ $p4$ によって決定されるセグメント (第 4 セグメント) に空きがなければ，CONSERVATIVE ではスワップを行い，セグメント・インディペンデントとしてディスパッチを行う．この場合，第 1 ソース・タグ $p4$ のタグ比較回数は削減されない．

一方で，サブ・セグメントを使用する場合 (図右側)，第 1 ソース・タグ $p4$ によってメインセグメントが決定され，もし空きがなければ，スワップを行う．そして，第 1 ソース・タグ $p4$ によってサブ・セグメント番号が決定され，該当する番号のいずれかに空きがあれば (図中の黄色で示したセグメント)，そのセグメントディスパッチする．このとき，第 1 ソース・タグ $p4$ の比較は，タグの下位ビットがサブ・セグメント番号と一致する場合のみ行われるため，その比較回数は $1/sub\_segment\_num$ だけ削減が可能となる．

以上で説明したように，サブ・セグメントを用いると，CONSERVATIVE モードでストールの回避を行った際にも，タグ比較の削減が可能となる．その結果，図 6.5 で示すような CONSERVATIVE モードでの高いタグ比較削減率を達成することが出来る．

最後に SWITCH 方式に関して評価する，平均で BASE モデルの 20% 程度のタグ比較回数となっており，80% の削減を達成している．これは，セグメントの総数が同じである (16, 1) と比較しても高い削減率であり，サブ・セグメントが有効であると言える．

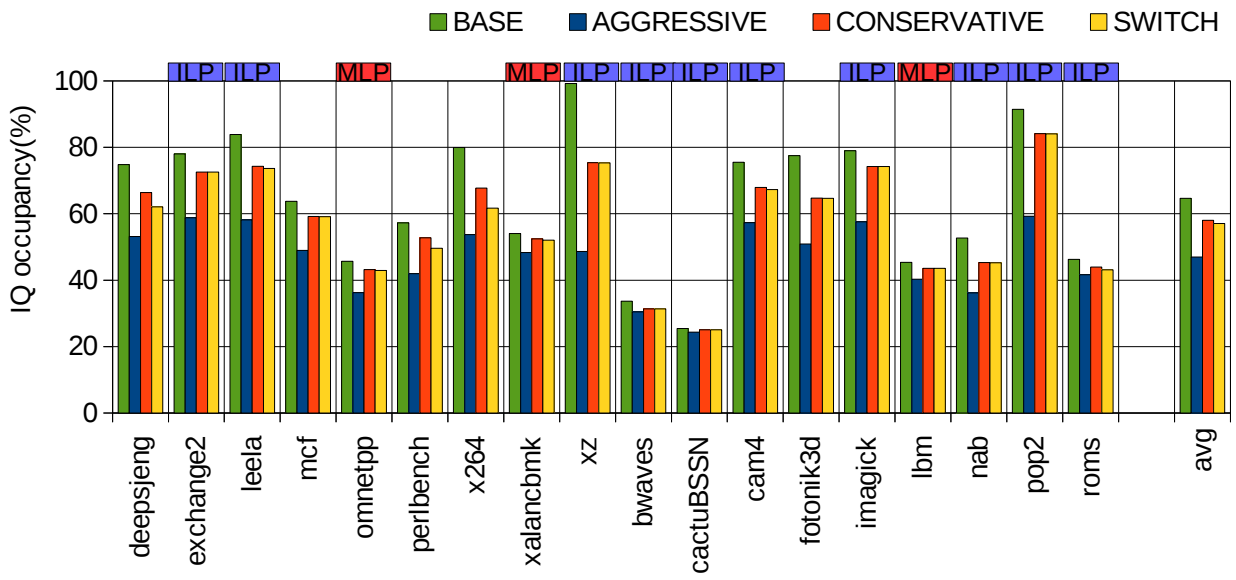

図 6.6: IQ の占有率の変化 (8,2)

性能低下

図 6.5 に，BASE に対する提案手法による性能低下をベンチマークごとに示す．同図より，SWITCH 方式による性能低下は最大で 4% 程度であり，多くのベンチマークでは 0% に近く性能はほとんど低下しないということが確認できる．

AGGRESSIVE の性能低下率に関して考える．(8,2) では，(16,1) と比較して AGGRESSIVE の性能低下率が低いことが分かる．これは，サブ･セグメントによって AGGRESSIVE モードでの容量効率の低下が抑制されているためであると考えられる．図 6.3 と図 6.6 の占有率を比較すると，(16, 1) の場合は平均で 40% 程度であった占有率が (8, 2) では，平均で 50% となっている．また，imagick に関して見てみると，(16,1) の AGGRESSIVE では占有率が 30% 程度で，性能低下が 10% であるのに対して， (8,2) の AGGRESSIVE では占有率が 60% 近くまで上昇しており，その結果性能低下が 2% 以下となっている．

以上の考察から，セグメントの総数が同じである場合，サブ・セグメントを使用することによって，AGGRESSIVE モードでの容量効率の低下による性能低下を抑制できることがわかった．

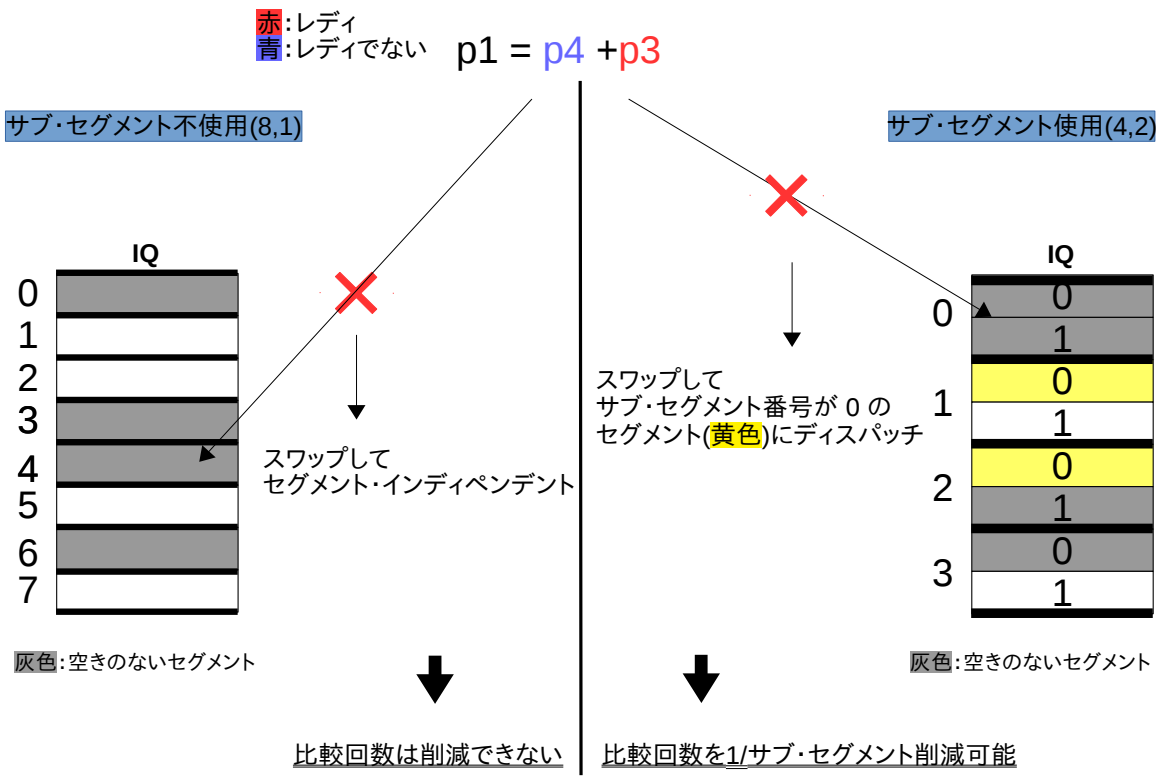

図 6.7: サブ・セグメントと CONSERVATIVE モードの組み合わせ

最後に，SWITCH 方式の有効性に関して説明する．(8, 2) の場合，サブ・セグメントが非常に有効であり，AGGRESSIVE での大幅な性能低下が見られないため，(16,1) の場合と比較して SWITCH 方式の有効性は高くないように見られる．しかし， imagick や lbm などのベンチマークにおいては AGGRESSIVE で発生する性能低下を抑制できている．また，サブ･セグメントを使用しているため，CONSERVATIVE でのタグ比較削減率が高く，その結果 SWITCH 方式自体のタグ比較削減率も高くなっている．

第7章　まとめ

LSIの微細化の進展に伴って，経年劣化が加速し摩耗故障が増加する問題が深刻になっている．この故障は，デバイスの温度に関して指数関数的に加速するため，チップ内のホット・スポットの解消が求められている．

発行キューはこのホット・スポットの 1 つとして知られている．この主な原因はウェイクアップ時の多数のタグ比較である．本論文では，ウェイクアップ時のタグ比較回数を削減するために，発行キューをセグメント化，および，それに関わるいくつかの手法を提案した．

提案手法には発行キューの容量効率が低下するという問題点が存在する．この問題点に対して，本論文ではさらに，異なる 2 つのディスパッチ・アルゴリズムを，性能についての容量効率の重要性に応じて切り替えて使用することにより，容量効率の低下による性能低下を抑制する手法を提案した．提案手法を SPEC CPU 2017 を使って評価したところ，性能低下を最大でも 5% 以下（平均 -1%）に抑えつつ，タグ比較回数を 80% 削減できることを確認した．

付 録 A　提案手法のその他の工夫

提案手法によるタグ比較回数をより減らすための工夫として，Last Tag Prediction(LTP) という手法が有効ではないかと考え，シミュレータに実装し評価を行った．本章では，LTP とその評価結果に関して説明する．なお，評価の結果，LTP はあまり効果がないことがわかったため，最終的な提案手法には実装していない．

A.1　LTP : Last Tag Prediction

CONSARVARTIVE 方式 において，第 1 ソース・タグと第 2 ソース・タグがどちらもレディでない命令は，以下のアルゴリズムでセグメントを選択すると述べた．

- サブ・セグメントを使用しない場合 : 第 1 ソース・タグでセグメントを選択する．選択したセグメントに空きがない場合，スワップして第 2 ソース・タグをもとにセグメントを決定する．なおも空きがない場合はストールする．
- サブ・セグメントを使用する場合 : 第 1 ソース・タグでメイン・セグメントを，第 2 ソース・タグでサブ・セグメントを選択する．選択したセグメントに空きがない場合，スワップを行い，第 2 ソース・タグでメイン・セグメントを，第 1 ソース・タグでサブ・セグメントを選択する．スワップしてなおも空きがない場合はストールする．

このアルゴリズムにおいて，スワップする場合としない場合に選択されるセグメントのどちらにも空きがある場合を考える．このような場合，CONSERVATIVE のアルゴリズムでは，スワップを行わずにディスパッチするセグメントを決定する．

このような場合に，レディとなるのがより遅く，比較がより多く行われるソース・オペランドのタグ (ラスト・タグ) を第 1 ソース・タグのフィールドに書き込むようにセグメン

トを選択すれば，タグ比較回数をより多く削減できると考えられる．ただし，ラスト・タグがどちらになるかという情報はデコード時にはわからないため，予測を行う必要がある．

ラスト・タグの予測方法は，論文 [20] で提案されている．この方法を Last Tag Prediction(LTP) と呼ぶ．以下，LTP による予測とセグメントの選択方法に関して説明する．

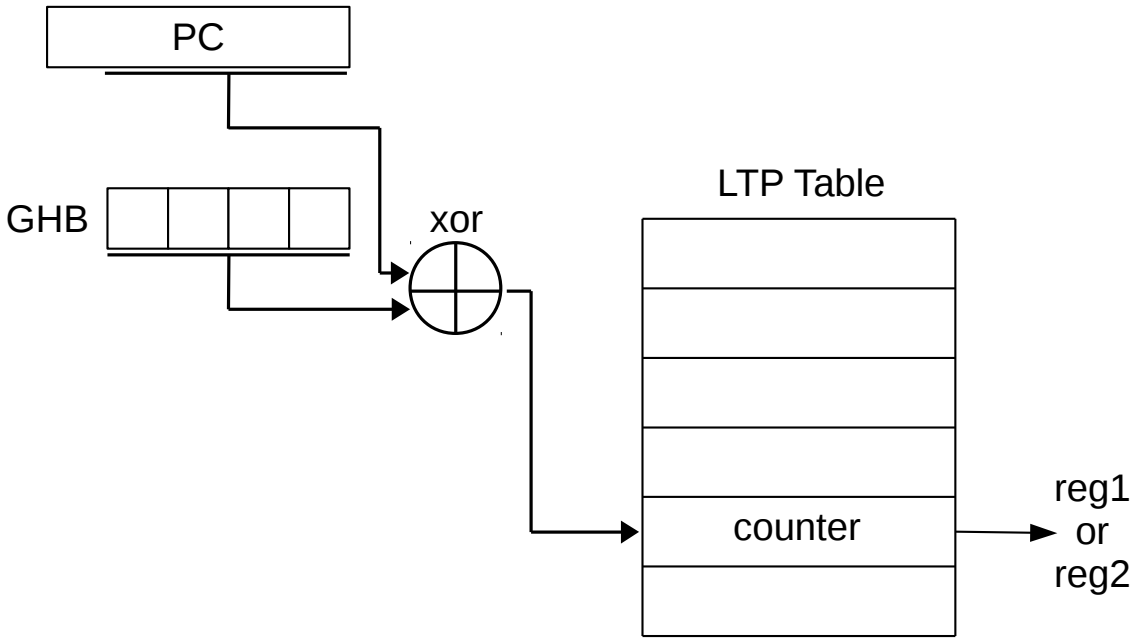

図 A.1: LTP の構成

LTP は，図 A.1 のように．命令の PC の下位ビットビットとグローバル分岐履歴 (GHB : Global History Buffer) のハッシュをインデクスとするテーブル (LTP Table) で構成される．LTP Table の各エントリは　2 ビットの飽和型アップ・ダウン・カウンタで構成され，このカウンタは，値が 0 または 1 の場合に第 1 ソース・タグがラスト・タグであることを示し， 2 または 3 の場合に第 2 ソース・タグがラスト・タグであることを示す．予測と学習の方法について説明する．

A.1.1 予測方法

命令ディスパッチ時に，第 1 ソース・タグと第 2 ソース・タグがともにレディでなく，なおかつスワップした場合としない場合に選択されるセグメントがいずれもディスパッチ可能な場合に予測が行われる．予測の際には，PC と GHB のハッシュを用いてテーブルを検索し，該当するエントリのカウンタ値を読み出す．

第 1 ソース・タグがラスト・タグであると予測された場合には，スワップを行なわずにセグメントを選択しディスパッチする．第 2 ソース・タグがラスト・タグであると予測された場合には，スワップを行いセグメントを選択し，ディスパッチする．

A.1.2 学習方法

学習は命令発行時に，ディスパッチ時に予測を行った命令でのみ行われる．命令発行時に，第 1 ソース・タグと第 2 ソース・タグがレディとなったサイクルを比較する．第 1 ソース・タグ のほうが遅かった場合にはカウンタをデクリメントし，第 2 ソース・タグのほうが遅かった場合にはカウンタをインクリメントする．

A.2 LTP の評価

発表実績

- 森健一郎, 安藤秀樹, “容量効率を意識したソース・タグ値に基づくセグメント化による発行キューのエネルギー削減”, 情報処理学会研究報告, Vol.2020-ARC-241, No.3, pp.1-12, 2020 年 7 月

謝辞

本研究を進めるにあたり，多大なる御指導と御鞭撻を賜わりました名古屋大学大学院工学研究科 情報・通信工学専攻 安藤秀樹教授に心より感謝いたします．また，本研究の遂行を支えてくださいました，名古屋大学大学院工学研究科情報・通信工学専攻安藤研究室の諸氏に深く感謝します．

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*, October 2001, pp. 20–25.

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

[4] J. 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, pp. 1–16, August 2000.

[6] J. A. Farrell and T. C. Fischer, "Issue logic for a 600-mhz out-of-order execution microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 5, pp. 707–712, 1998.

[7] 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.

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

[9] J. Stark, M. D. Brown, and Y. N. Patt, "On pipelining dynamic instruction scheduling logic," in *Proceedings of the 33rd Annual International Symposium on Microarchitecture*, December 2000, pp. 57–66.

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

[11] 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*, June 2007, pp. 335–346.

[12] A. R. Lebeck, J. Koppanalil, T. Li, J. Patwardhan, and E. Rotenberg, "A large, fast instruction window for tolerating cache misses," in *Proceedings of the 29th Annual International Symposium on Computer Architecture*, May 2002, pp. 59–70.

[13] 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*, May 2002, pp. 318–329.

[14] I. Kim and M. H. Lipasti, "Macro-op scheduling: Relaxing scheduling loop constraints," in *Proceedings of the 36th Annual International Symposium on Microarchitecture*, December 2003, pp. 277–289.

[15] D. Gibson and D. A. Wood, “Forwardflow: A scalable core for power-constrained CMPs,” in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, June 2010, pp. 14–25.

[16] H. Ando, “SWQUE: A mode switching issue queue with priority-correcting circular queue,” in *Proceedings of the 52nd Annual International Symposium on Microarchitecture*, October 2019, pp. 506–518.

[17] Y. Kora, K. Yamaguchi, and H. Ando, “MLP-aware dynamic instruction window resizing for adaptively exploiting both ILP and MLP,” in *Proceedings of the 46th Annual International Symposium on Microarchitecture*, December 2013, pp. 37–48.

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

[19] 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*, December 2001, pp. 90–101.

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

[21] A. Sembrant, T. Carlson, E. Hagersten, D. Black-Shaffer, A. Perais, A. Seznec, and P. Michaud, “Long term parking (ltp): Criticality-aware resource allocation in ooo processors,” in *Proceedings of the 48th International Symposium on Microarchitecture*, December 2015, pp. 334–346.

[22] H. Homayoun, A. Sasan, J. Gaudiot, and A. Veidenbaum, “Reducing power in all major cam and sram-based processor units via centralized, dynamic resource size management,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 11, pp. 2081–2094, Nov 2011.

[23] 小林誠弥, “発行キューにおけるタグの 2 段階比較による消費エネルギー削減,” 名古屋大学大学院工学研究科博士課程 (前期課程), 修士学位論文, 2015 年 3 月.

[24] 松田　康誉, “ランダム発行キューにおけるタグの 2 段階比較による電力削減,” 名古屋大学大学院工学研究科博士課程 (前期課程), 修士学位論文, 2020 年 3 月.

[25] M. Motomura, J. Toyoura, K. Hirata, H. Ooka, H. Yamada, and T. Enomoto, “A 1.2-million transistor, 33 mhz, 20-bit dictionary search processor with a 160 kb cam,” in *1990 37th IEEE International Conference on Solid-State Circuits*, 1990, pp. 90–91.

[26] ——, “A 1.2-million transistor, 33-mhz, 20-b dictionary search processor (disp) ulsi with a 160-kb cam,” *IEEE Journal of Solid-State Circuits*, vol. 25, no. 5, pp. 1158–1165, 1990.

[27] D. A. Jimenez and C. Lin, “Dynamic branch prediction with perceptrons,” in *Proceedings of Seventh International Symposium on High-Performance Computer Architecture*, January 2001, pp. 197–206.