RC-001

Dual Renaming を用いた Control Independence アーキテクチャ

孟　林[†]　西川直樹[††]　西岡拓生[†]
上原子正利[††]　小柳滋[††]

現在のスーパースカラプロセッサでは，分岐予測ミスが発生した時に，ミスした分岐命令以降のすべての命令を取り消す．このペナルティを緩和するため Control Independence(CI) を利用する手法が提案されてきた．これらの手法は分岐ミス発生時に分岐の成否に依存しない命令 (control independent instructions : CI 命令) を取り消さずに，再利用する．しかし，データ依存性のある命令は依存関係が更新されるため，そのまま再利用することができず，レジスタ再リネーミング (Re-Renaming) などの手法で対応する．このため既存手法には，メカニズムが複雑化したり，これを避けるため限定されたパターンしか扱えないなどの難点がある．本論文では，簡単なメカニズムで一般的なパターンに対応する Dual Renaming 方式を提案する．これは，CI 命令のソースオペランドに 2 つのタグを割り当て，分岐ミス時にチェックポイントを利用してデータ依存関係を回復する，本提案を SimpleScalar 上に実装し，シミュレーションにより性能評価を行った．15 段のパイプラインステージにおいて，最大 16.4%，平均 6.24% の性能向上が達成された．

Control Independence Architecture Using Dual Renaming

Lin Meng,[†] Naoki Nishikawa,[††] Takuo Nishioka,[†]
Masatoshi Kamiharako[††] and Shigeru Oyanagi[††]

Modern Superscalar Processor squashes up all of wrong-path instructions when the branch prediction misses. In deeper pipelines, branch miss prediction penalty increases seriously owing to large number of squashed instructions. Exploiting control independence has been proposed for reducing this penalty. Control Independence method reuses control independent instructions (CI instructions) without squashing when branch prediction misses. Reusing CI instructions at branch miss prediction is not easy because of changing data dependency between squashed instructions and CI instructions. Conventional researches of CI architecture require complex re-renaming mechanism, or with a limited applicability. This paper proposes a new simple mechanism named Dual Renaming for reusing CI instructions. It utilizes current renaming mechanism with checkpointing by assigning two tags for each source register of CI instruction. The simulation result shows that dual renaming mechanism increases IPCs by maximum 16.4% and average 6.24%.

1. はじめに

アウトオブオーダのスーパースカラプロセッサの性能を向上させるためには，命令レベルの並列性を最大限抽出するため大きな命令ウィンドウが必要である．一方，スーパースカラプロセッサの性能を阻害する要因として，分岐予測ミスによるペナルティが挙げられる．現在のスーパースカラプロセッサでは分岐予測ミスが発生した場合，命令ウィンドウにおける分岐命令以降の命令をすべて取り消し，正しい分岐先命令を新たにフェッチするため，大きな命令ウィンドウを持つことが分岐予測ミスのペナルティを増大させ，性能向上にとって大きな阻害となる．

分岐予測ミスペナルティの減少を目指したアプローチとして Control Independence(CI)[1,2,3,4,5] を利用する技術が注目されている．CI を利用することにより，分岐予測ミスの場合でも分岐の成否に依存しない命令 (Cntrol Independent 命令:以下 CI 命令) を取り消さず再利用することにより分岐予測ミスペナルティの減少を図っている．

分岐の合流点以降の命令は分岐の成否に依存しないので CI 命令である．しかし，CI 命令でも合流点以

† 立命館大学理工学研究科
Graduate School of Science and Engineering, Ritsumeikan University
†† 立命館大学情報理工学部
College of Information Science and Engineering, Ritsumeikan University

前の命令とデータ依存の関係にある命令は，そのままに再利用することができない．これに対処するために，既存手法には，複雑なメカニズムでレジスタの Re-Renaming をしたり，これを避けるため限定されたパターンしか扱えないなどの難点がある [4].

本研究では，できるだけ簡単なハードウェア機構かつ一般的なパターンでこの問題を解決することを目指す．その中心となるアイデアは，CI 命令の Dual Renaming という考え方である．これにより，CI の効果を生かしたアーキテクチャを提案する．

2. Control Independence の概要

ここでは，図 1 に示すような分岐命令を含む命令列を例として CI を説明する．図 1 の○で囲まれた部分は命令ブロックである．ブロック 1 の分岐命令により，ブロック 2 あるいはブロック 3 に分岐する．また，それらはブロック 4 で合流する．合流する命令 D は収束ポイント (Convergence Point) と呼ぶ．合流する前のブロック 2，ブロック 3 は分岐命令の結果に依存するため，ブロック 2，3 の命令は Control Dependent(CD) 命令と呼ぶ．ブロック 1 は分岐ブロックと定義され，ブロック 2 とブロック 3 は CD ブロックと定義され，ブロック 4 は CI ブロックと定義される．

分岐の合流点はブロック 4 であり，ブロック 4 内の命令は分岐の成否に依存しないため CI 命令である．しかし，ブロック 4 内の命令には前のブロック内の命令とデータ依存関係にあるものが存在する．すなわち，CI 命令はデータ依存のある CIDD(Control Independent - Data Dependent) 命令と，データ依存のない CIDI(Control Independent - Data Independent) 命令に分けられる．ブロック 4 の r3 を用いる命令 D はブロック 2 に対して CIDD であり，r1 を用いる命令 F はブロック 3 に対して CIDD である．

CI 命令のソースオペランド中に，分岐の成否が分かるまで生成先が確定できないものがある．すなわち，ソースオペラントが収束ポイント前の命令により生成された場合である．本論文では，これらのソースオペランドを CIDDSop(Control Indenpendent - Data Dependent Source Operand) と呼ぶ．図 1 の CI 命令 D の r3，E の r2，F の r1 は CIDDSop である．一方，命令 E のソースオペランド r3 は CI ブロックの E により生成されたものなので，CIDDSop ではない．

ここで，このプログラムの実行において，ブロック 1 の分岐でブロック 2 を予測したとする．この予測に従い，ブロック 2，ブロック 4 を実行中に分岐予測ミスが発覚し，ブロック 2 の実行を取り消してブロック

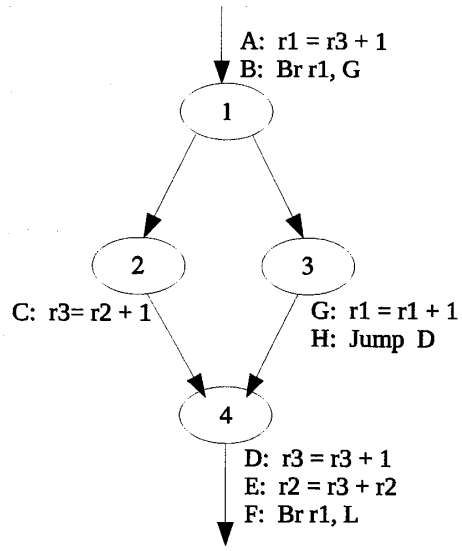


図 1　Control independence の例
Fig. 1　An example of control independece

3 の実行を開始する場面を想定する．この場面では，① ブロック 2 の無効化，② ブロック 3 のフェッチと実行，③ ブロック 4 の中の命令の選択的再発行が必要となる．① の操作では，命令ウィンドウからブロック 2 の命令を削除し，使用しているリソースを解放することが必要である．また，レジスタのマッピングを分岐以前の状態に戻す必要がある．このためには，分岐命令の実行時点でのレジスタのマッピング情報をチェックポイントとして保存し，分岐予測のミス時に復帰させることが必要となる．

② の操作では，ブロック 3 の命令をフェッチして命令ウィンドウに格納することが必要である．また，① と ② に関して，命令の実行順序を保持しているリオーダバッファ (ROB) 内で命令の適切な順序保持のための操作が必要となる．

③ に関しては，ブロック 4 内で実行中の命令がブロック 2，ブロック 3 に関して CIDD であるかどうかを判別し，CIDD ならばレジスタの Re-Renaming を行い，命令の再発行が必要となる．

以上の操作を行うためのメカニズムについて，先行研究で提案されている方法を紹介する．

Walker [1]：CI 研究の先駆けとなる研究である．分岐予測ミスが発覚したとき，実行中の CI 命令すべてについてデータ依存性の判定を行い，CIDD 命令についてはレジスタの Re-Renaming を行い，命令の再発行を行う．CI 利用の効果を最大限に生かすことを目指しているが，データ依存性の判定の回路規模が大きく，またレジスタ Re-Renaming，ROB 内の命令の実行順序の保持のハードウェアコストが高いことが欠

点である．

Skipper [2]：Skipper では分岐予測ミスによる命令実行の取り消しをなくすことを目指している．そのため，分岐先が決まるまで CD 命令を実行せず，CI ブロックの中の CIDI 命令だけを実行する．この方式は，分岐が正しく予測されても分岐結果の判定が終了するまで CI ブロックの中の CIDD 命令を実行できないという欠点がある．

SBR [3]：分岐予測ミスを簡単なハードウェア機構で減少させることを目指して，対象とする分岐を Exact Convergence に限定した方式を提案している．Exact Convergence とは else のない if-then 型の分岐命令である．これに限定することにより，分岐予測ミスの場合の処理 (then の予測がミスした場合) において命令フェッチが不要となり，CIDD 命令に関しても Re-Renaming するのではなく，レジスタ間の転送命令を挿入することにより行い，ハードウェア機構を簡単化している．この方式の欠点は対応できる分岐予測が 1 方向のみであり，また対象とする分岐が限定されていることが挙げられる．

Ginger [4]：Ginger は一般的なケースに対応できる CI アーキテクチャである．tag-rewriting 手法を用いて，レジスタの Re-Renaming を行い．map-table のチェックポイントを用いて“search-and replace”を行う．また必要なバンド幅の追加も行っている．欠点としてはハードウェアが複雑であることがあげられる．

TCI [5]：Walker の発展研究である．Walker と同様に CI 利用の効果を最大限に生かすことを目指している．分岐予測ミス時に使用する命令ウィンドウを別に用意することにより，命令の実行順序保持のメカニズムを提案している．

以上のように，CI の研究は複雑なハードウェアにより一般的な分岐パターンを対象とする方法 (Walker, Ginger, TCI) と，比較的簡単なメカニズムにより対象を限定する方法 (Skipper,SBR) に分けられる．我々は，ハードウェアを複雑化することなく一般的な分岐パターンを対象とする方法として，Dual Renaming を提案する．

3. Dual Renaming を用いた Control Independence

3.1 Dual Renaming とは

Dual Renaming は ROB を用いたリネーミングの資源を利用し，CIDDSop に対して，分岐先に対応した 2 つのタグを与え，分岐予測ミスが発生した後にタグを入れ替えることにより，CIDD 命令を再利用する手法である．

最初のタグは，従来の ROB を用いたリネーミングと同じように，ROB から探索することにより得られる．このタグは，分岐予測されたパスにより生成されることから，predicted path tag(P-Tag) と呼ぶ．もうひとつのタグは，分岐ミス発生後に，CI 命令の再利用のために準備した Reserved Tag(R-Tag) である．R-Tag は，ソースオペランドが CIDDSop であると判定された場合に与えられる．

分岐予測ミスが発生した場合，R-Tag を P-Tag に上書きする．そして，R-Tag の依存先を探して CI 命令を再利用する．しかし，R-Tag の依存先は指定されていない．そこで，予測ミス発生後にそれらの依存先を探すために，いくつかのチェックポイントを設ける．

3.2 動作概要

プログラム実行中に分岐命令が出現したとき，その収束ポイントを取得する．本稿では，文献 [11] に示されるような収束ポイントを動的に判定するハードウェア機構を備えているとする．そして，収束ポイントの後にフェッチされた CI 命令について Dual Renaming を行う．図 2 は，図 1 の例を用いて，本提案による CI の実行例を示している．

図 2A ではフェッチされた命令 (A〜F) を命令ウィンドウに保存する．ここで分岐命令 B は，分岐先が C と予測されたとし，その収束ポイントは D と判定されたとする．

図 2B では収束ポイント D に到着後に，命令のリネーミングと CI 命令の Dual Renaming を行う．ここでは，P-Tag と CIDDSop の R-Tag を作成し，チェックポイントも作成する．なお，R-Tag を保存するために命令ウィンドウを拡張し，各命令ごとにソースレジスタに対応して 2 つの R-Tag 欄を追加する．

まず，デスティネーションレジスタのリネーミングを行い，C,D,E,F のソースオペランドのタグを検索し，CI 命令については CIDDSop の識別も行う．ソースオペランドの探索では，CI 命令 D の r3，E の r2,r3，F の r1 に対して，ROB からタグを探索し，P-Tag(D の r3 → P4，E の r2 → P2，r3 → P5，F の r1 → P1) を得る．また，CI 命令 D の r3，E の r2，F の r1 が CIDDSop であることも分かる．次に，CIDDSop に R-Tag(D の r3 → P6,E の r2 → P8，F の r1 → P9) を与える．

リネーミングと同時に，予測ミス発生後に用いられる R-Tag の依存先を探すために以下に示すチェックポイントを用意し、Check Point Map に保存する．チェックポイントの内容を図 2 下に示す．Check Point

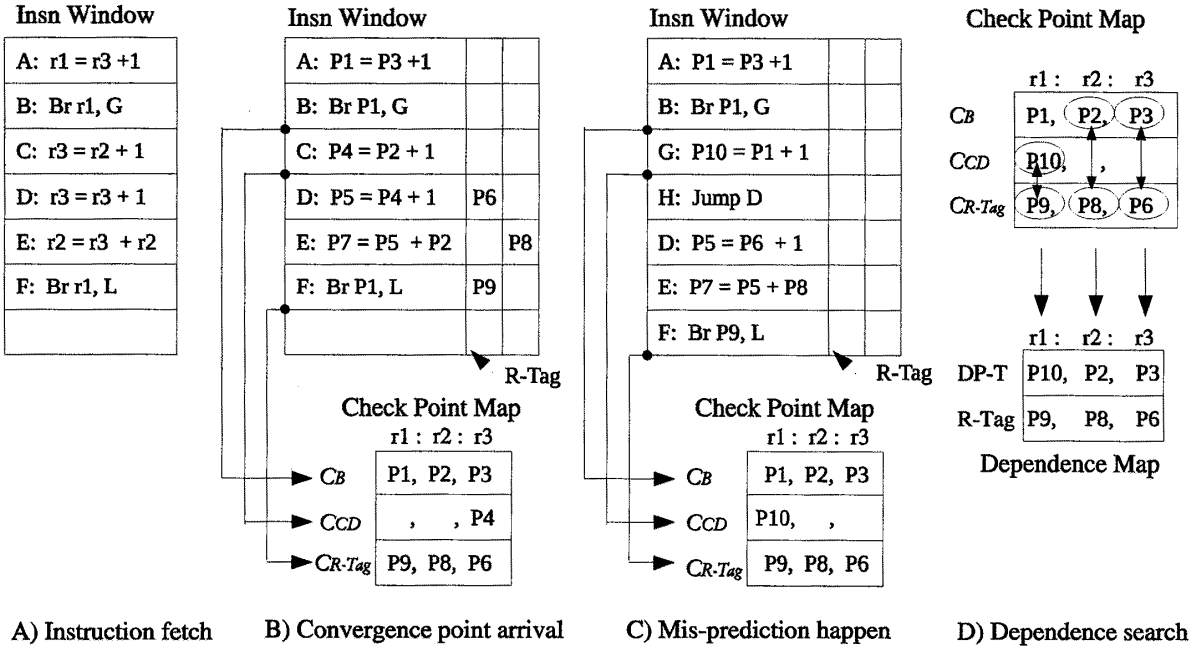


図 2 Dual Renaming 動作例
Fig. 2 Dual Renaming Action Diagrams

Map のインデックスはレジスタ番号である．

Check Point Branch(C_B) :分岐ブロック命令のタグとレジスタの関係を示すチェックポイントである．図 2B では，分岐命令 B について r1 → P1，r2 → P2，r3 → P3 の関係を保持する．

Check Point CD(C_{CD}) :予測パスの CD 命令に生成されたタグとレジスタの関係を示すチェックポイントである．図 2B では，CD 命令 C について r3 → P4 との関係を保持する．

Check Point R-Tag(C_{R-Tag}) :R-tag とレジスタの関係を示すチェックポイントである．図 2B では，CI 命令 D,E,F について r1 → P9，r2 → P8，r3 → P6 の関係を保持する．

図 2C では，予測ミスが発生したときの動作を示す．分岐命令 B は予測ミスが発生したときに，ミスパス命令 (C) を取り消し，正しいパス命令 (G，H) をフェッチしてリネーミングする．それとともに，命令ウィンドウでは，CI の R-Tag を用いて対応する P-Tag に上書きを行う．例では、図 2C のように CI 命令 D では P4 → P6，E では P2 → P8，F では P1 → P9 に上書きする．

しかし，図 2C では，新たに変更された C_{R-Tag} の P6,P8,P9 の生成先が指定されていない．これらの依存先の探索を図 2D に示す．

R-Tag の依存先は，Check Point Map の C_B と C_{CD} から求めることができる．その結果を Dependence Map に保存する．Dependence Map は R-Tag と依存先 (DP-T) で構成され，インデックスがレジスタ番号である．R-Tag の生成先は Dependence Map に示すように，P10 → P9，P2 → P8，P3 → P6 となる．この Dependence Map を用いて C_{R-Tag} の P9,P8,P6 を生成先のタグ P10,P2,P3 と対応付けることにより，CIDD 命令を正しく実行することができる．

以上の例は単純な場合であるが，複数の分岐命令が存在する場合でも各分岐命令に対してチェックポイントを設けることにより対応できる．

4. Dual Renaming の実現

ここでは、前節で説明した Dual Renaming の実現方法について述べる．

4.1 基本アーキテクチャ

本システムの基本アーキテクチャを図 3 に示す．図 3 のように，Decode Queue, Dispatch Queue, Issue Queue の他に，Re-dispatch Queue を備える．Re-dispatch Queue は，分岐先が決定していない CI 命令を格納するためのバッファである．これらは，以下のように動作する．

分岐命令が出現したとき分岐予測に従い，予測先の CD 命令，および CI 命令は Dispatch Queue に転送される．このとき，CI 命令は同時に Re-dispatch Queue にも格納する．分岐予測ミスが発生したとき，Dispatch Queue，Issue Queue 上の命令はフラッシュされ，正しい分岐先の CD 命令がフェッチされて，Dispatch Queue に転送される．同時に，Re-dispatch Queue 上

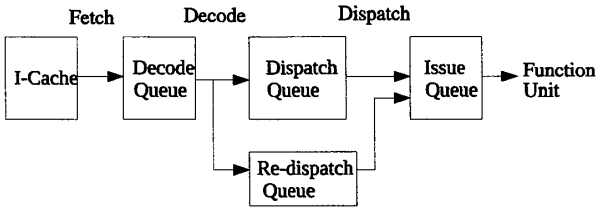


図 3 Dual Renaming の基本アーキテクチャ
Fig. 3 Basic architecture of of dual renaming

の CI 命令についても tag を書き換えて Issue Queue に転送される．また，分岐予測が正しかったとき，Re-dispatch Queue はフラッシュされる．命令の実行制御に関しては従来と同様であり，Re-dispatch Queue により分岐ミスのとき CI 命令のフェッチ，デコードを省略することができる点が特徴である．

4.2 ROB を用いた実現

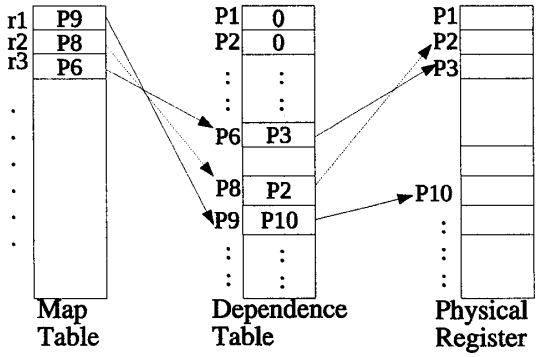
Dual Renaming を従来の ROB を用いて実現する．CIDDSop の生成先として，P-Tag と R-Tag の二つのタグを与え，分岐予測ミスの発生後に R-Tag を用いて正しい依存関係に戻す．そのために，分岐ブロック，CD ブロック，CI ブロックを区別する必要があり，ROB 上では以下の場所でチェックポイントを設置する．

ROB Check Point Branch(ROB C_B) ：分岐命令ブロックの最後のレジスタを生成する命令である．すなわち分岐ブロックと CD ブロックの切れ目である．

ROB Check Point CD(ROB C_{CD}) ：CD 命令の最後のレジスタを生成する命令である．すなわち CI ブロックと CD ブロックの切れ目である．

分岐命令がデコードされたとき，分岐ブロックと CD ブロックの切れ目と判明し，ROB 上にチェックポイント ROB C_B を作成する．また，タグとレジスタの対応関係を Check Point Map の C_B に保存する．次に収束ポイント D により，命令 C は CD ブロックと CI ブロックの切れ目と判明し，ROB 上にチェックポイント ROB C_{CD} を作成する．また，CD 命令内で生成されたタグとレジスタの対応関係 C_{CD} を Check Point Map に保存する．CI 命令のタグを検索するときに，ROB C_{CD} の前で見つかったものは CIDDSop として判別され，R-Tag を与える．それらの情報も Check Point Map の C_{R-Tag} 欄に保存する．分岐予測ミスが発生したとき，ミスパスを取り消し，正しいパスをフェッチし，ROB と Check Point Map を変更する．

このように，提案手法では従来の ROB タグ検索リソースを用いてリネーミングを行うことにより，複雑な Re-Renaming 機構を省略することができる．



Dependence Map

	r1	r2	r3
DP-T	P10	P2	P3
R-Tag	P9	P8	P6

図 4 tag 変換機構
Fig. 4 tag conversion mechanism

4.3 tag の変換

Dual Renaming では，分岐ミスが判明したとき，Re-dispatch Queue 上の CI 命令について tag を書き換える必要がある．また，Dependence Map を用いて，書き換えた tag と依存する tag を関連づけなければならない．このハードウェア機構を図 4 に示す．

前に述べた例では，Dependence Map を用いて C_{R-Tag} の P9,P8,P6 を生成先のタグ P10,P2,P3 に対応づけなければならない．この操作は renaming 機構を少し変更するだけで対応できる．図 4 のように論理レジスタと tag とを対応づける Map Table と物理レジスタの間に Dependence Table を備える．すなわち，論理レジスタと物理レジスタの対応は Dependence Table を介して行われる．Dependence Table のエントリが 0 のとき，論理レジスタは従来と同様に tag の値がそのまま物理レジスタ番号となる．Depen-

表 1 プロセッサの構成
Table 1 Processor configuration

Pipeline	15 stages: 1 Predict, 3 Fetch, 4 Decode, 1 Rename, 1 Dispatch, 1 schedule, 1 Issue, register read/bypass, 1 Execute 1 memory access/register write, 1 commit
Fetch, Decode	4 instructions
Issue	Int: 4, fp: 2, mem: 2
Window	Instruction window: 64, Re-dispatch Queue: 64, LSQ :256
Branch predictor	10-branch history 32K-entry gShare, 4K-entry 4-way associative BTB, 32-entry RAS

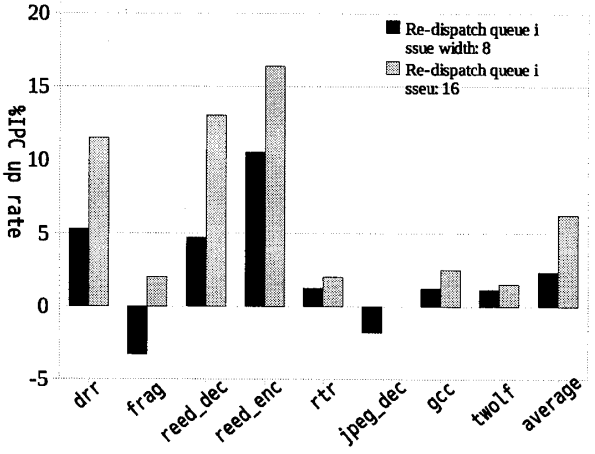


図 5 IPC 向上率
Fig. 5 Enhancement of IPC

dence Table に Dependence Map で示される生成先のタグを書き込むことにより，論理レジスタと物理レジスタの対応を変更することができる．すなわち，分岐予測ミスが発生したとき生成先のタグを検出して依存表に書込み，物理レジスタの使用が終了したとき依存表をクリアすればよい．

5. 評　　価

提案手法を，SimpleScalar Tool Set[9] の上に実装し，シミュレーションにより評価を行った．パイプラインステージは 15 ステージに拡張し，命令セットは PISA を用いた．ベンチマークには SPECint2000 と CommBench[12] をあわせて 8 本を用いた．SPECint2000 では gcc,twolf を使用し，CommBench では drr,frag,reed_dec,reed_enc,rtr,jped_dec を使用した．シミュレーションで仮定するプロセッサの仕様を表 1 に示す．なお，Re-dispatch Queue の発行幅が 8 の場合と 16 の場合について実験を行った．

5.1 考　　察

5.2 評 価 結 果

表 2〜表 4 は，各ベンチマークの実行結果を詳細に示している．表中の各行は以下のデータを示している．

Mis/Bra ：分岐命令の予測ミス率である．

Bra/Ins ：実行命令の中の分岐命令の割合である．

CI/Mis ：予測ミスされた分岐命令の中で CI を利用したものの割合である (CI 利用率)．

SBR/CI ：CI を利用した分岐命令の中で SBR タイプ [3] の割合である．

IPCup ：ベースプロセッサと比較した IPC の向上率である．

表 2 は CI を適用しないベースプロセッサの結果を示す．表 3,4 は CI を適用した場合の結果である．表 3 は Re-dispatch Queue の発行幅が 8，表 4 は 16 のときの結果である．

図 5 は Re-dispatch Queue が 8 のときと 16 のときの IPC の向上率を比較したものである．図 6(A) は各ベンチマークにおいて，3 つの場合の分岐予測ミス率 (Mis/Bra) を比較したものである．図 6(B) は Re-dispatch Queue が 8 のときと 16 のときの CI を利用した分岐命令の割合 (CI/Mis) を比較したものである．

実験結果について考察を加える．図 5 より，CI 利用における IPC の向上率は Re-dispatch Queue の発行幅が 16 の場合，最大 16.4%，平均 6.24% であった．Re-dispatch Queue の発行幅が 8 の場合，最大 10.5%，平均 2.36%であった．図 5 より，drr, read_dec, read_enc では IPC の向上率が 10% を超えている．これより，ベンチマークの種類によっては CI の効果が十分にあることが示された．また，Re-dispatch Queue の発行幅は性能向上に重要であることが示された．

次に，IPC 向上の要因について考察する．CI は分岐がミスしたときに動作するため，分岐ミス率は重要なパラメータである．また，CI は命令ウィンドウ内に収束ポイントが見つからないと適用できないため，分岐ミスの中でどの程度の割合で CI が適用できるかも重要なパラメータである．

図 6(A) より，gcc は分岐ミス率が高いが，CI 利用率が低い．このため，CI による IPC 向上率は低い．一方，read_enc, read_dec は分岐ミス率が高く，CI 利用率も高い．このため，CI による IPC 向上率は高い．jpeg_dec は分岐ミス率が低く，CI 利用率も低い．このため，CI による効果はほとんどない．drr は分岐ミス率が低いが，CI 利用率は高い．この場合，高い IPC 向上率が得られた．以上より，分岐ミス率よりも，CI 利用率が高い方が CI にとって有効であることが分かる．

図 6(A) より，frag, rtr, jpeg_dec, gcc, twolf では CI を用いると分岐ミス率が増加している．これは，CI を利用することにより，分岐予測器の更新が遅くなるためと考えられる．この影響により，これらのベンチマークの性能向上は抑えられていると考えられる．

なお，表 3,4 より，CI の中の SBR タイプの比率は低いことが分かる．これより，SBR により CI の適用範囲を狭めることは有効ではないと考えられる．

6. おわりに

本論文では，Dual Renaming を用いた Control Independence の利用手法を提案した．分岐予測ミスが

表 2　ベースプロセッサの実行結果
Table 2　base processor execution result

	drr	frag	reed_dec	reed_enc	rtr	jped_dec	gcc	twolf
Mis/Bra	0.022160	0.030284	0.041209	0.062034	0.026466	0.018410	0.059365	0.028395
Bra/Ins	0.212873	0.182125	0.182566	0.221351	0.238614	0.049040	0.200361	0.360214
IPC	2.315856	2.368210	2.314176	1.639078	2.213610	2.988334	1.252837	1.541280

表 3　命令発行幅が 8 のときの実行結果
Table 3　8-issue execution result

	drr	frag	reed_dec	reed_enc	rtr	jped_dec	gcc	twolf
SBR/CI	0.001697	0.332493	0.012001	0.004881	0.490738	0.258621	0.299124	0.187852
CI/Mis	0.856013	0.820182	0.792901	0.863223	0.414370	0.306554	0.308871	0.498528
Mis/Bra	0.021207	0.041316	0.041234	0.058032	0.028760	0.018621	0.060373	0.030852
Bra/Ins	0.214518	0.176070	0.182133	0.225198	0.239469	0.048711	0.199737	0.349753
IPC	2.439589	2.289485	2.423250	1.811398	2.240280	2.934444	1.267285	1.558641
IPCup (%)	5.3	-3.3	4.7	10.5	1.2	-1.8	1.2	1.1

表 4　命令発行幅が 16 のときの実行結果
Table 4　16-issue execution result

	drr	frag	reed_dec	reed_enc	rtr	jped_dec	gcc	twolf
SBR/CI	0.001488	0.332263	0.011879	0.004705	0.441444	0.275168	0.289574	0.189107
CI/Mis	0.865165	0.500511	0.795329	0.849946	0.362979	0.299197	0.306678	0.492126
Mis/Bra	0.021207	0.033701	0.039697	0.057700	0.029137	0.019280	0.060143	0.030286
Bra/Ins	0.214518	0.183423	0.181835	0.225582	0.238933	0.049103	0.199853	0.356924
IPC	2.583082	2.415559	2.615696	1.90767	2.257680	2.988951	1.285155	1.579275
IPCup(%)	11.5	2.0	13.0	16.4	2	0	2.5	2.5

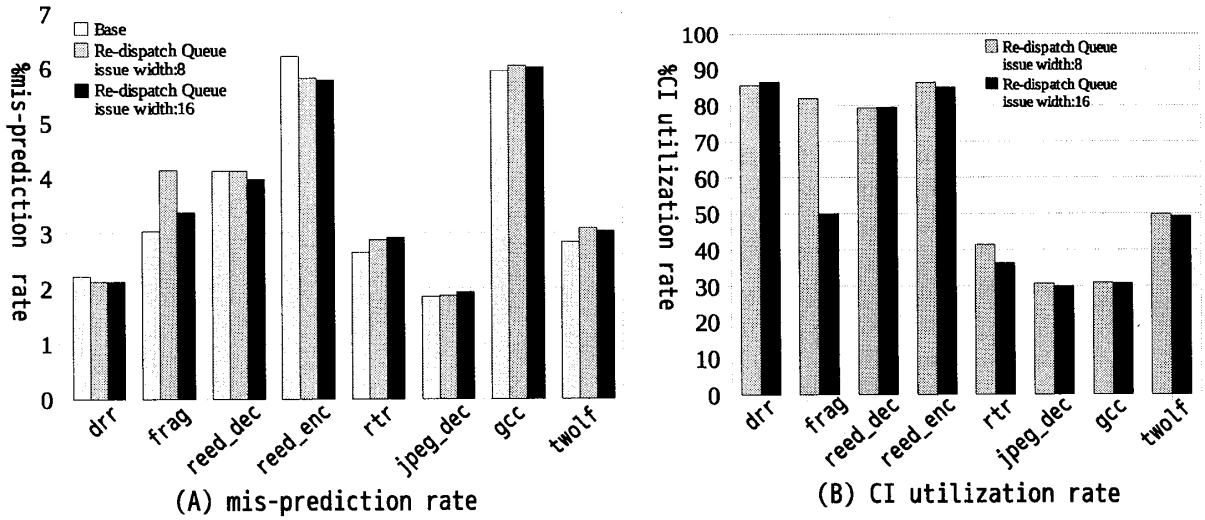


図 6　分岐予測ミス率と CI 利用率
Fig. 6　Misprediction rate and CI utilization rate

発生しても，命令を再利用することにより分岐ミスペナルティを減少させる手法である．再利用しようとしている命令の依存性を回復するために，レジスタに Dual Renaming を行い，二つのタグを与える．予測ミス発生後に，Dual Renaming により生成されたタグを利用し，依存性を回復して CI 命令を実行することができる．提案手法では，先行研究と違い，複雑な Re-Renaming を行わず，分岐パターンを限定した方式ではなく，命令の実行を遅延させない特徴を持つ．シミュレータにより提案手法の評価を行い，15 ステージパイプラインのスーパースカラプロセッサにおいて，最大 16.4%，平均 6.24% の IPC の向上を達成した．

実験の結果，性能向上を阻む要因の一つとして，CI による分岐ミスの増加が判明した．これについて，対策を今後詳細に検討したい．

また，本論文ではすべての分岐命令を対象としてい

るが，分岐予測機構と連動して信頼性の低い分岐命令に限定すればハードウェアリソースを抑えることが可能と考える．さらに，収束ポイントの検出ハードウェアについても検討したい．また，ROB 上での命令順序を維持する機構についてもいくつか提案されているが [1,2,4]，詳細に検討したい．今後の課題としては，これらを取り入れて簡潔なメカニズムで CI の利用効果を高める方法について検討したい．

参 考 文 献

1) E. Rotenberg, Q. Jacobsen, and J. Smith. " A Study of Control Independence in Superscalar Processors."Proc. 5th Int'l Symposium on High Performance Computer Architecture, pp.115-124, Jan. 1999.
2) C. Cher and T. Vijaykumar. " Skipper: A Microarchitecture for Exploiting Control-Flow Independence. " Proc. 34th Int'l Symposium on Microarchitecture, pp.4-15, Dec. 2001.
3) A. Gandhi, H. Akkary, and S. Srinivasan. "Reducing Branch MispredictionPenalty via Selective Branch Recovery. " Proc. 10th Int'l Symposium on High Performance Computer Architecture, pp.254-265, Feb. 2004.
4) A. Hilton, and A. Roth. " Ginger: Control Independence Using Tag Rewriting. " Proc. 35th Int'l Symposium on Computer Architecture, pp.436-447, May 2007.
5) A. S. Al-Zawawi, V. K. Reddy, E. Rotenberg, and H. H. Akkary. " Transparent Control Independence (TCI). " Proc. 35th Int'l Symposium on Computer Architecture, pp.470-481, May 2007.
6) T.H. Heil, and J.E. Simth. "Selective Dual Path Executiong. " Technical Report, University of Wisconsin-Madison. " ECE, 1997.
7) J.L. Aragon, J. Gonzalez, A. Gonzalez, and J.E. Smith. " Dual Path Instruction Processing. " ICS'02, pp.22-26, June 2002.
8) E. Jacobson, E. Rotenberg, and J. Smith. "Assigning Confidence to Conditional Branch Predictions. " Proc. 29th Int'l Symposium on Microarchitecture, pp.142-152, Dec 1996.
9) D. Burger,T.M. Austin. " The SimpleScalar Tool Set ,Version2.0. " Technical Report, University of Wisconsin-Madison Computer Sciences Dept., July 1997.
10) J.E. Smith, A.R. Pleszkun. " Implementation of Precise Interrupts in Pipelined Processors. " 12th Int'l Symposium on Computer Architecture, pp.36-44, 1985.
11) J.D.Collins, D.M.Tullsen, Hong Wang, " Control Flow Optimization Via Dynamic Reconvergence Prediction. " 37th Int ' l Symposium on Microarchitecture, pp.129-140, 2004.
12) T.Wolf and M.A.Franklin, " CommBench - a Telecommunications Benchmark for Network Processors, " Proc. of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Austin, TX, pp.154-162 Apr 2000.