計算機アーキテクチャ　86－3
（1991　1．24）

DSN型スーパースカラ・プロセッサ・プロトタイプの分岐パイプライン

原　哲也　久我守弘　村上和彰　富田眞治
（九州大学大学院総合理工学研究科）

我々は現在，DSNS（Dynamically-hazard-resolved,Statically-code-scheduled,Nonuniform Superscalar）アーキテクチャに基づくスーパースカラ・プロセッサの開発を行っている．スーパースカラ・プロセッサにおいては，分岐命令およびそれに起因する制御依存の存在によって，命令フェッチの阻害，後続命令実行の阻害，分岐遅延によるパイプラインの乱れ，および，命令レベル並列性の低下といった分岐ペナルティが生じ，性能が著しく低下する．このような分岐ペナルティの影響を緩和するため，DSNSプロセッサでは，①静的分岐予測＋分岐先バッファ，②投機的命令実行，③先行条件決定方式，および，④早期分岐解消といった手法を用いた分岐アーキテクチャを採用している．

本稿では，分岐アーキテクチャ，および，分岐命令のパイプライン処理過程ついて述べる．

Branch Pipeline of the DSNS Processor Prototype
(in Japanese)

Tetsuya HARA, Morihiro KUGA, Kazuaki MURAKAMI, and Shinji TOMITA
Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyusyu University
6-1 Kasuga-koen, Kasuga-shi, Fukuoka, 816 Japan
e-mail : hara@is.kyushu-u.ac.jp

A DSNS (Dynamically-hazard-resolved,Statically-code-scheduled,Nonuniform Superscalar) processor prototype, has been being built at Kyushu University.

Control hazards due to branches cause a severe performance loss for superscalar processors. The DSNS processor prototype alleviates these effects with ①static branch prediction with branch-target-buffer, ②speculaitve execution, ③advanced conditioning, and ④early branch resolution.

This paper presents the branch architecture and the branch pipeline of the DSNS processor prototype.

1.はじめに

我々は，命令レベルの並列性を活用するプロセッサ・アーキテクチャとしてスーパースカラ方式を採用したプロセッサを開発中である．本プロセッサは以下のような特徴を持つ [3].

(1) 動的ハザード解消

命令間依存関係および資源競合に起因するハザードを実行時にハードウェアで検出し，解消する．これにより，オブジェクト・コードの互換性を保証する．

(2) 静的コード・スケジューリング

資源の競合や命令間の依存関係が存在すると，同時に実行できる命令数は減り，命令レベル並列性が減少する．パイプラインが滞ることなく流れるためには，命令（発行）順序の並べ換え（コード・スケジューリング）を行い同時に実行できる命令数を増す必要がある．このコード・スケジューリングをコンパイル時に静的に行う．これにより並列性の増大を図ると同時に，動的コード・スケジューリングにより生じるハードウェアの増大という問題を回避している．

(3) 非均質型機能ユニット

命令フェッチ機構，デコード機構，レジスタ・ポート，バスなどは，スーパースカラの多重度に従って多重化しているが，機能ユニットについては必要なものだけを多重化している．使用頻度の高い機能ユニットのみを多重化しているので，コスト／パフォーマンスが良い．

上記の特徴に基づき，現在開発中のプロセッサを DSNS (Dynamically-hazard-resolved, Statically-code-scheduled, Nonuniform Superscalar) プロセッサと呼んでいる．

さて，命令パイプライン・プロセッサにおいては，分岐命令およびそれに起因する制御依存の存在によって性能が低下する．スーパースカラ・プロセッサにおいては，これら分岐ペナルティの影響が著しい．分岐ペナルティの主な原因には以下のものがある．

①命令フェッチの阻害：分岐するか否か（条件分岐の場合)，および，分岐先アドレスが確定するまで次にフェッチすべき後続命令が決まらない．

②制御依存による後続命令実行の阻害：もし次にフェッチすべき後続命令を予測してフェッチしても，制御依存が解消するまでこれら後続命令の実行は開始あるいは終了できない．これは，制御依存関係にある命令（つまり，実行するか否かが未定の命令）がレジスタ内容等を更新してしまうと正確なマシン状態が保証できないことによる．

③分岐遅延によるパイプラインの乱れ：しかも，分岐予測が外れた場合は，パイプラインをフラッシュして正しい命令を再フェッチしなければならない．よって，分岐命令実行の遅延時間が長くなると，それだけパイプラインの乱れは増大する．

④命令ミスアラインメント [1] [15]：スーパースカラ・プロセッサでのみ問題となる．すなわち，分岐先アドレスが命令フェッチ・バウンダリに一致していない場合，フェッチした命令中に無駄な命令が含まれる．これにより，命令レベル並列性が低下する．

上記①②③の分岐ペナルティの影響を緩和するために，DSNS プロセッサでは以下の手法を採用している．

ⓐ静的分岐予測＋分岐先バッファ

ⓑ投機的命令実行

ⓒ先行条件決定方式

ⓓ早期分岐解消

なお，上記④の命令ミスアラインメントに関しては，ハードウェアによる対処も可能ではあるが [1]，DSNS プロセッサでは全面的にコンパイラに頼っている．

本稿では，分岐ペナルティに対する対処法（2章）を整理した後，DSNS プロセッサで採用した分岐アーキテクチャ（3章)，DSNS プロセッサの概要（4章)，および，分岐命令のパイプライン処理過程（5章）について述べる．

2. 分岐ペナルティへの一般的な対処法

2.1 分岐命令への対処

分岐命令の存在により生じる分岐ペナルティの影響を軽減する方法として，以下の多くのものが提案されている．なお，下記 (1) 〜 (4) の方法はほぼ直交関係にあり，実現に際しては様々な組合せが可能である．

(1) 分岐方式

まずは，通常の compare-and-branch 方式や branch-on-condition 方式とは異なる方式として，次のものがある．

①先行条件決定方式 (advanced conditioning) [2] [12]：条件決定（分岐するか否か）を分岐に先行して行い，分岐命令実行に伴う分岐遅延の低減を図る．

②分岐予告方式 [9]：分岐予告命令 (prepare-to-branch instruction) により分岐先命令を予めフェッチしてバッファリングする．これにより，分岐命令が TAKEN の場合のパイプラインの乱れを抑える．

(2) 分岐命令

分岐命令の機能として，以下のものを加える．

①遅延分岐 [11]：分岐命令の遅延スロットを定義し，遅延スロット内の命令は制御依存を受けないようにする．つまり，分岐結果 (TAKEN／NOT-TAKEN) に関係なく必ず実行される．もし，遅延スロットをすべて埋めることが出来たら，分岐ペナルティは生じない．しかし，制御依存を受けないような命令を発見するのは難しい．そこで，分岐結果によっては遅延スロット内の命令を無効化するような無効化付遅延分岐 (delayed branch with squashing) も提案されている [10].

②静的分岐予測 [10]：コンパイラが意味解析結果およびプロファイル情報に基づいて，個々の条件分岐命令について分岐予測を行う．予測結果は，OP コードに反映される．これにより，命令フェッチが阻害されるのを防ぐ．

(3) 命令フェッチ

命令フェッチを中断することなく連続して行う方法として，以下のものがある．

①動的分岐予測 [8] [13]：静的分岐予測とは異なり，ハードウェアが実行時に分岐予測を行う．これにはさらに，次の手法がある．

ⓐnot-taken 予測：not-taken（分岐しない）と予測して，非分岐先の命令流をフェッチし続ける．

ⓑtaken 予測：taken（分岐する）と予測して，分岐先の命令流をフェッチするようにする．

ⓒ分岐予測バッファ (BPB : Branch Prediction Buffer)：個々の分岐命令の実行時の履歴をバッファリングしておき，それに基づいて分岐予測する．

ⓓ分岐先バッファ (BTB : Branch Target Buffer) [13]：BPB にさらに分岐先アドレスまたは分岐先命令をバッファリングする．これにより，taken 予測の場合，アドレス計算を行わずに直ちに分岐先命令の供給が可能となる．

②複数命令流フェッチ [8] [9]：非分岐先と分岐先の両方の命令流をフェッチする．

(4) 分岐命令実行

分岐命令の実行に当り，次のような配慮を行う．

①早期分岐解消 (early branch resolution) [9]：命令パイプラインの早いステージで分岐命令を実行する．これにより，制御依存の存在時間，および，分岐命令の実行遅延時間の低減を図る．

②分岐命令畳込み (branch folding) [6]：通常の命令パイプ

ラインの前に命令プリフェッチ・ステージを設け，無条件分岐命令はそこで実行を済ませる．よって，命令パイプラインには無条件分岐命令は入って行かない．しかし，条件分岐命令は命令パイプラインで実行される．無条件分岐命令に関しては，分岐ペナルティは生じない．

2.2 制御依存への対処

分岐命令の後続命令は一般的に，当該分岐命令に対して制御依存関係にある．すなわち，分岐命令の後続命令をフェッチできたとしてもこれらを実行するか否かは，制御依存が解消するまで定まらない．したがって，このような制御依存関係にある命令の命令パイプライン内での処置については，特別な配慮が必要である．これに関しては，少なくとも以下の2方法がある．

(1) 非投機的実行

制御依存が解消するまで，当該制御依存関係にある命令の実行を開始しない．実現は容易だが，分岐命令を越えた命令の実行が出来ない．つまり，最大同時に実行可能な命令が，1個の基本ブロックに限定される．よって，基本ブロック内の命令数が少ない場合，命令レベル並列性が落ちる．

(2) 投機的実行（speculative execution）

制御依存が解消しなくても，当該制御依存関係にある命令の実行を開始する．ただし，制御依存が解消しないうちは，これらの命令の実行結果がレジスタ内容等を変更することを禁止する．もし，制御依存が解消した結果これらの命令の実行が不要と判明した場合は，命令自身および実行結果を無効化しなければならない．よって，実現は後述するようにやや複雑になる．しかし，分岐命令を越えた命令の実行が可能となり，命令レベル並列性の向上が期待できる．

さて，投機的実行の具体的な実現方式には，次の2つがある．

①条件付実行モード（conditional mode）：分岐予測によりフェッチされた命令は，対応する分岐命令に起因する制御依存が解消するまで条件付実行モード下に置かれる．条件付実行モード下にある命令は実行を行うことは可能だが，実行結果でレジスタ内容等を更新することは出来ない．

②ブースティング（boosting）[15]：個々の命令に対して，条件付実行モード下に置くか否かをコンパイラが決定する．条件付実行モード下に置かれた命令をブースト（boost）命令と呼び，その指定はOPコードにより行う．ブースト命令はオブジェクト・コード上，対応する分岐命令より前に位置する．すなわち，上記①とは逆に，条件付実行モード下の命令が対応する分岐命令より先にフェッチされ実行を開始する．よって，静的コード・スケジューリング次第では，上記①より投機的実行の効果が期待できる．

また，条件付実行モード下にある命令（ブースト命令も含む）が制御依存の解消を待たずにレジスタ内容等を変更するのを防ぐ手段として，以下のものがある．

ⓐバッファ方式[14]：レジスタ・ファイルにバッファを設ける．このバッファの使用方法には，次の2種類がある．

(i) reorder buffer [16] [14]：条件付実行モード下にある命令の実行結果をバッファリングする．そして，制御依存が解消した時点で分岐予測が当たっていれば，その内容をレジスタに反映する．ただし，本バッファからもオペランド・フェッチが出来るよう，バイパス機構が必要となる．

(ii) history buffer [14]：条件付実行モード下にある命令の実行結果でレジスタ内容を変更するのを許す．ただし，そのとき元のレジスタ内容を本バッファにバッファリングする．そして，制御依存が解消した時点で分岐予測が外れていれば，その内容でレジスタ内容を復元する．

ⓑfuture file [14]：同一レジスタ・ファイルを2組設け，一方をアーキテクチャ上定義されたレジスタ・ファイル（architectural file），他方を条件付実行モード下にある命令の実行結果書込み用ファイル（future file）とする．そして，制御依存が解消した時点で分岐予測が当たっていれば，future fileの内容をarchitectural fileに反映する．この反映方法には，ファイル間転送による方法[14] [15]，および，3.2節で述べるレジスタ・アクセスパスの切り換えによる方法[4]がある．

ⓒcheckpoint repair [7]：制御依存が発生した時点（チェックポイント）におけるレジスタ内容を保存しておく．そして，当該制御依存が解消した時点で分岐予測が外れていれば，その内容でレジスタ内容を復元する．

3. 分岐アーキテクチャ

2章で挙げた対処法のうち，DSNSプロセッサでは以下の手法を採用している．

①静的分岐予測＋分岐先バッファ
②投機的実行：条件付実行モードおよびブースティング
③先行条件決定方式
④早期分岐解消

3.1 静的分岐予測＋分岐先バッファ

従来は動的分岐予測と組み合せて用いていた分岐先バッファ（BTB）を，DSNSプロセッサでは静的分岐予測と組み合わせる．

コンパイラは静的分岐予測結果に基づいて，個々の分岐命令に対して以下のいずれかの型を指定する．

①BTB登録型：taken予測，かつ，分岐先アドレスが不変であると判断された分岐命令はこの型になる．実行時に生成した分岐先アドレスをBTBに登録する．

②BTB非登録型：not-taken予測，あるいは，taken予測だが分岐先アドレスが変化すると判断された分岐命令はこの型になる．分岐先アドレスはBTBに登録されない．

BTBに分岐先アドレスが登録されると，対応する分岐命令はtakenと予測され，その登録分岐先アドレスが次サイクルの命令フェッチに使用される．

BTBの構成については4.1節で述べる．また，BTBへの分岐先アドレスの登録処理，および，BTBを用いた命令フェッチ処理については5.2節で述べる．

3.2 投機的実行

投機的実行方式として，2.2節で述べた条件付実行モードおよびブースティングの双方を採用する．両方式のハードウェア上の実現方法は基本的に等価であり，条件付実行モードのためのハードウェア機構がブースティングにも流用できる．

さて，条件付実行モード下にある命令（ブースト命令も含む）が制御依存の解消を待たずにレジスタ内容等を変更するのを防ぐ手段として，DSNSプロセッサでは多重化レジスタ・ファイル[4]を採用している．これは，future file [14]の1実現方法である．ここで，多重化レジスタ・ファイルの多重度は，「条件付実行モードのレベル＋1」となる．DSNSプロセッサの条件付実行モードのレベル数は1であるので，多重度は2となる（3.4節参照）．これから，2重化レジスタ・ファイル（DRF：Dual Register File）と呼ぶ．

2重化レジスタ・ファイルにおいては，各レジスタは論理的には1個であるが，物理的には2個の実体（物理レジスタ）を有する．そして，一時には，一方がカレント（current）状態，他方がオルタネート（alternate）状態となる．オルタネート状態にはさらに，有効／無効の2状態が存在する．また，カレント状態およびオルタネート状態にある物理レジスタを便宜上，カレント・レジスタおよびオルタネート・レジスタとそれぞれ呼ぶ．

2重化レジスタ・ファイルの動作の概要を以下に述べる．

(1) レジスタ読出し

無条件実行モード（unconditional mode：制御依存関係に

ない状態）下の命令は，そのソース・オペランドをカレント・レジスタから読み出す．一方，条件付実行モード下にある命令は，そのソース・オペランドを次のようにして得る．

①オルタネート・レジスタが有効な場合：オルタネート・レジスタから読み出す．

②オルタネート・レジスタが無効な場合：カレント・レジスタから読み出す．

(2) レジスタ書込み

無条件実行モードで終了した命令の実行結果は，カレント・レジスタに書き込む．一方，条件付実行モードで終了した命令の実行結果は，オルタネート・レジスタに書込み当該オルタネート・レジスタを有効状態とする．

(3) 分岐命令の実行終了

制御依存が解消されると，個々のレジスタについて次の状態遷移が起きる．

①分岐予測が当たった場合：有効なオルタネート・レジスタがカレント状態となり，対を成すカレント・レジスタはオルタネート無効状態になる．それ以外のレジスタは，状態に変化なし．

②分岐予測が外れた場合：すべてのオルタネート・レジスタは無効化状態となる．カレント・レジスタは，状態に変化なし．

3.3 先行条件決定方式

3.3.1 従来の条件分岐方式

条件分岐は，一般に以下の処理から成る．

①コンディション生成：条件分岐の元になるコンディション（状態）を生成する．このコンディションは，算術・論理演算等を実行することにより生成される．

②条件決定（conditioning)：①で生成したコンディションを分岐条件でテストし，分岐するか否かを決定する．

上記①②は分岐するしないに関わらず行う．分岐する場合は，さらに以下の処理が必要である．

③分岐先アドレス生成：分岐先の命令アドレスをアドレッシング・モードに従って計算する．なお，アドレッシング・モード次第で，この処理は省ける．

④分岐処理：③で生成した分岐先アドレスをプログラム・カウンタ（PC）に設定する．

さて，従来の一般的な条件分岐方式には，以下の２つがある．

(1) compare-and-branch方式

表1に示すように，条件分岐処理過程の①②③④のすべてを1個のcompare-and-branch命令で行う．クリティカルパスが①→②→④と長いため分岐遅延が長くなる欠点がある．

(2) branch-on-condition方式

コンディション・コード（CC：Condition Code）を用いて，①と②③④とを別々の命令で行う．表1に示すように，①は通常の算術・論理演算命令におけるCC設定で，②③④は1個のbranch-on-condition命令で行う．branch-on-condition命令自身のクリティカルパスが②→④ないし③→④と短くなるため，compare-and-branch命令に比べて分岐遅延は短い．しかし，CCを導入したことで，CCに関するフロー依存関係の検出，および，CCへのアクセス競合回避といった課題が生じる．

3.3.2 先行条件決定方式

表1． 条件分岐方式

方式＼処理	①コンディション生成	②条件決定	③分岐先アドレス生成	④分岐
compare-and-branch	compare-and-branch命令			
branch-on-condition	算術・論理演算命令	branch-on-condition命令		
advanced-conditioning	算術・論理演算命令	test命令	branch命令	
	compare-and-test命令			

前項で述べた条件分岐方式以外に，図1に示す先行条件決定方式（advanced-conditioning）がある［12］［2］．DSNSプロセッサでは本方式を採用する．本方式は表1に示すように，①はbranch-on-condition方式同様，通常の算術・論理演算命令におけるCC設定で行うが，②と③④は別々の命令で行う．③④は1個のbranch命令で行い，そのクリティカルパスは③→④と短い．

先行条件決定方式の導入に関連して，次の２種類のレジスタを定義する．

ⓐTFレジスタ（TFR：True／False Register)：32個の1ビット長レジスタで，「分岐が成立するか（True＝TAKEN）／否か（False＝NOT-TAKEN)」を保持する．

ⓑタグ付き汎用／浮動小数点レジスタ：個々の汎用／浮動小数点レジスタに4ビットのタグを付ける．算術・論理演算命令の実行により生成されたCCは，そのデスティネーション・レジスタのタグに格納される（図1参照）．これにより，従来のbranch-on-condition方式におけるCCに関する課題を解決している［2］．

また，関連命令として，次の３種類の命令を定義する（図1参照）．

ⓐテスト（test）命令：ソースレジスタのタグ内のCCに対して分岐条件と合致するか否かのテストを行い，分岐する／しないのtrue／false値をデスティネーションTFレジスタに設定する．条件分岐処理過程の②に相当する．

ⓑ比較＆テスト（compare-and-test）命令：2つのソースオペランド間の算術・論理比較を行い，その結果に対して分岐条件と合致するか否かのテストを行い，true／false値をデスティネーションTFレジスタに設定する．条件分岐処理過程の①②に相当する．

ⓒ分岐（branch）命令：ソースTFレジスタのtrue／false値に

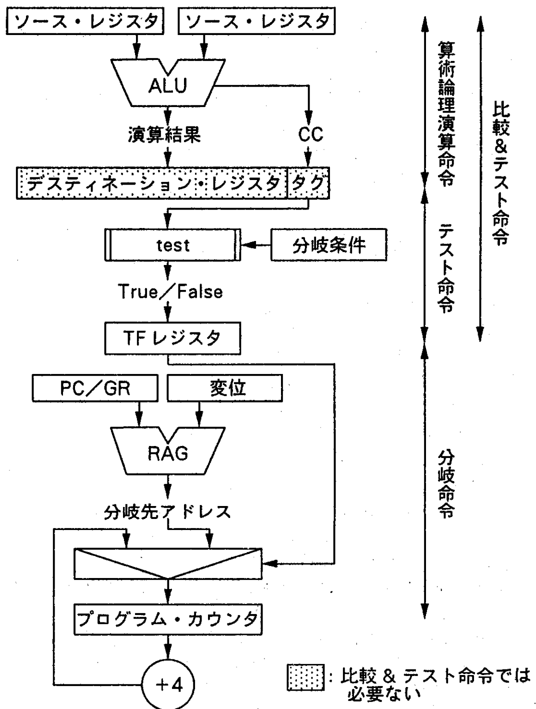

図1．先行条件決定方式

従って分岐結果（TAKEN／NOT-TAKEN）を決定する．TAKENの場合さらに，プログラム・カウンタ（PC）に分岐先アドレスを設定する．分岐先アドレスはアドレッシング・モード（PC相対／GR相対／PC＋GR）に従って計算する．条件分岐処理過程の③④に相当する．

これ以外に，多方向分岐用に，2個のTFレジスタ間の論理演算を行う命令も用意している．

先行条件決定方式により，以下の利点が期待できる．

- テスト命令または比較＆テスト命令と分岐命令との間の距離を大きくようにスケジューリングすることで，条件決定（分岐するか否か）に伴う分岐遅延を隠蔽できる．
- TFレジスタ間の論理演算命令を用いることによって，複数の分岐命令を1個の分岐命令にまとめることができる．すなわち，分岐命令数の削減が可能である．
- 分岐命令で行う処理（TFR参照と分岐先アドレス生成）が少ないことから，3.4節で述べるように早期分岐解消が可能となる．その結果，分岐遅延自体を低減できる．

3.4 早期分岐解消

3.1-3.3節で述べた方法に加えて，分岐命令の実行自身の高速化，つまり，分岐の早期解消が重要である．これは，以下の理由に依る．

- 分岐予測が外れた場合，条件付実行モード下のすべての命令を無効化することで命令パイプラインを復元する（パイプライン・フラッシュ）．したがって，当該分岐命令に起因する制御依存の解消が遅くなるほど，多数の命令が無効化されることになる．当然，分岐ペナルティも増大する．
- 条件付実行モード下で新たな分岐命令をフェッチした場合，1レベルの条件付実行モードでは当該分岐命令ならびに後続命令の実行は開始できない．これは，条件付実行モードのレベルが足りないことに原因がある．しかし，条件付実行モードのレベル数の増加は，多重化レジスタ・ファイル（3.2節参照）の多重度の増加を意味し，ハードウェア量の増大に直接結びつく．よって，条件付実行モードのレベル数を2以上に上げるのは難しい．

このような背景から，DSNSプロセッサには早期分岐解消（early branch resolution）を採用している．5.2節で述べるように，分岐命令以外の一般命令は命令パイプラインのE（実行）ステージで実行を行うのに対して，分岐命令はそれよりも前のD（解読）ステージで実行を開始する．

しかし，このように一般命令とは異なるステージで分岐命令の実行を行うためには，余分なハードウェアが必要となるという問題が生じる．ところが，DSNSプロセッサでは，3.3節で述べたように，分岐命令で行うべき処理がTFR参照と分岐先アドレス生成だけと少ない．よって，必要とされるハードウェア量もさほど多くはない．この分岐命令専門のハードウェアは，5.1節で述べる分岐ユニットとして設けている．

3.5 分岐命令の仕様

DSNSプロセッサの分岐命令の仕様は以下の通りである．

<u>(1) 種類</u>

3.1節で述べた通り，BTB登録型とBTB非登録型の2種類に分類される．なお，すべての分岐命令は条件分岐命令である．分岐するか否かは，ソース・オペランドであるTFRの値によって決定される．TFR0の値は常にTure（＝TAKEN）であり，これをソース・オペランドに指定する分岐命令が無条件分岐命令となる．

<u>(2) アドレッシング・モード</u>

分岐先アドレス生成の際のアドレッシング・モードとして，以下の3種類を用意する．

①PC（プログラム・カウンタ）＋オフセット（符号付き19ビット）
②GR（汎用レジスタ）＋オフセット（符号付き14ビット）
③PC＋GR

なお，命令長が4バイトであるため，命令アドレスの下位2ビットは常に‘00’である．よって，分岐先アドレス計算では上位30ビットのみを生成する．

4. プロセッサの概要

DSNSアーキテクチャの有効性を検証するため，DSNSアーキテクチャに基づく試作プロセッサ“DSNSプロセッサ”の開発を行っている．この章では，DSNSプロセッサの構成と命令パイプライン処理過程について，その概要を述べる［4］．

4.1 DSNSプロセッサの構成

DSNSプロセッサは，以下の主要ユニットから構成される．

<u>(1) 命令キャッシュ（IC：Instruction Cache)</u>

ポート・サイズ16バイトで，4バイト長命令4個から成る命令ブロックを一時にフェッチする．ライン・サイズ64バイト（＝4

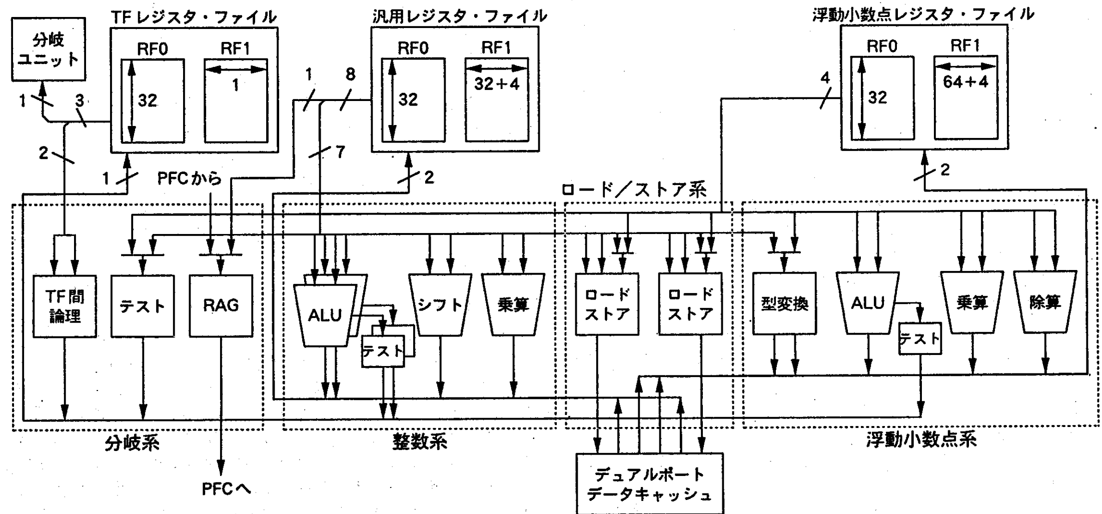

図2. DSNSプロセッサのデータ・パス

命令ブロック）のダイレクト・マッピング方式の仮想アドレス・キャッシュである．

命令ブロック毎に1個の予測分岐先アドレスを保持する分岐先バッファ（BTB：Branch Target Buffer）を備える．これにより，分岐命令の有無に関わらず，命令ブロックの連続フェッチが可能となる．BTBと　命令キャッシュは，タグ・アレイを共用する．

(2) プリデコーダ（PD：PreDecoder）

フェッチした4命令をプリデコードして，コンフリクト・チェックに必要な情報を得る．また，分岐命令実行に関するすべての情報もここで生成する．

(3) デコーダ（D：Decoder）

各命令の実行制御に必要な情報，特に機能ユニットの制御情報をROMから読み出す．

(4) コンフリクト・チェッカ（CC：Conflict Checker）

プリデコード結果に基づいて，最大4命令に対して，
・命令ブロック内の命令間のフロー依存および出力依存の検出
・先行命令ブロックに対するフロー依存および出力依存の検出
・機能ユニット競合の検出と調停
・レジスタ・ファイルの読出しポートの割当て
を行う．

(5) 分岐ユニット（BU：Branch Unit）

早期分岐解消を行うための分岐命令実行専用ユニットで，3段のパイプライン構成となっている．また，分岐命令専用のコンフリクト・チェッカ（BCC）と，逐次的に分岐命令を実行していくための4段の分岐命令バッファ（BIB：Branch Instruction Buffer）を備える．詳細は5章で述べる．

(6) 機能ユニット（FUs：Functional Units）

図2に示すように，以下の4系統，計13個の機能ユニットを備える．最大4命令が毎サイクル，任意の機能ユニットの組合せに対して発行可能である．

①整数系機能ユニット：ALU（×2），シフタ，乗算器
②浮動小数点系機能ユニット：ALU，乗算器，除算器，型変換器
③ロード／ストア機能ユニット：2つの独立したロード／ストア・ユニット [5]
④分岐系機能ユニット：再フェッチ・アドレス生成器，先行条件決定方式のための2つのユニット

(7) 2重化レジスタ・ファイル（DRFs：Dual Register Files）

図2に示すように，①汎用レジスタ（GR），②浮動小数点レジスタ（FPR），③TFレジスタ（TFR）の3系統のレジスタ・ファイルを備える．

(8) デュアルポート・データキャッシュ（DPDC：Dual-Port Data-Cache）[5]

整数データおよび浮動小数点データを処理できる2重化したロード／ストア・パイプラインに対応するため，デュアルポート化している．ポート・サイズ8バイト，ダイレクト・マッピング方式の仮想アドレス・キャッシュである．

また，ミスヒットを起こしても後続のロード／ストア命令を受け付け可能な，ノンブロッキング・キャッシュとなっている．

4.2 命令パイプライン処理過程

命令パイプラインは，

①IF：命令ブロック・フェッチ＋プリデコード
②D：コンフリクト・チェック＋デコード＋オペランド・フェッチ
③E：実行
④W：書込み

の4ステージ構成である．パイプライン・サイクル時間は，60nsを目標にしている．

図3に，命令の種類毎に命令パイプライン処理過程を示す．分岐命令の処理については，5章で述べる．

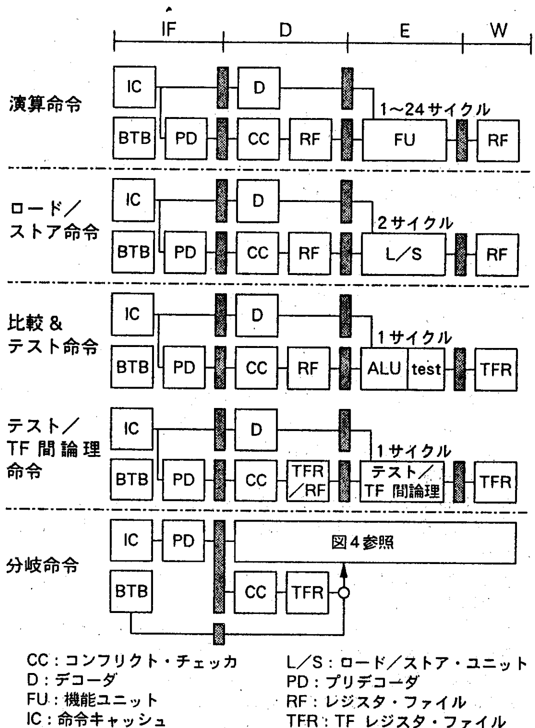

図3．パイプライン処理過程

5. 分岐パイプライン

分岐命令の処理は，IFステージにおける分岐予測，条件付実行モードの設定のあと，分岐パイプラインによって行われる．分岐パイプラインの実行主体は分岐ユニットであり，これはBD，BE，BWの3ステージから構成される．

5.1 分岐ユニット

分岐ユニット（BU：Branch Unit）には，以下の2個のユニット，および，分岐パイプラインのための制御機構を備える．

(1) 分岐命令バッファ（BIB：Branch Instruction Buffer）

分岐命令は基本的に逐次実行しなければならないので，分岐ユニットは一時に高々1つの分岐処理しか行わない．したがって，同一命令ブロックに複数の分岐命令が存在する場合に備えて，分岐ユニットは4命令を保持する分岐命令バッファ（BIB：Branch Instruction Buffer）を設けている．分岐ユニットでは，分岐命令を含む命令ブロックを，一旦BIBにバッファリングしたあと，分岐命令を1命令づつ順番に実行してゆく．

(2) 分岐命令用コンフリクトチェッカ（BCC：Branch Conflict Checker）

メインパイプラインのDステージとBDステージが同期している場合，分岐ユニットで処理中の分岐命令は，メインパイプラインのDステージにも存在する．よって，データ依存の検出にはメインパイプラインのCCを用いる．しかし，分岐ユニットとメインパイプラインの処理は独立しているため，フロー依存でBDステージがインターロックされる場合，分岐命令の所属する命令ブロックがEステージ以降に進み，Dステージには後続命令ブロックが存在する状態が生じる可能性がある．つまり，当該分岐命令は，一般命令用のDステージには存在せず，CCを使用することができない．このため，分岐ユニットには，分岐命令専用のコンフリクトチェッカ（BCC：Branch Conflict

Checker）を備え，上記のようにCCが使用できない場合のデータ依存検出を行う．BCCは，一般命令のCC同様，先行命令ブロックに対するデータ依存の検出を行うが，命令ブロック内の命令間データ依存の検出は行わない．

また，機能ユニット（再フェッチ・アドレス生成器）およびレジスタ・ファイル読出しポートは分岐ユニット専用となっているので，

・機能ユニット競合の検出と調停

・レジスタ・ファイルの読出しポートの割当て

を行う必要がない．

5.2 分岐命令処理過程

5.2.1 IFステージ

IFステージは，分岐命令以外の一般命令と共通のステージである．分岐に関しては，以下の処理を行い，その結果をメインパイプラインのDステージと分岐パイプラインのBDステージへ送る．

(1) 分岐予測

PFC（プリフェッチ・カウンタ）値による命令キャッシュからの4命令（命令ブロック）のプリフェッチと同時に，対応するBTBエントリの読み出しを行う．そのエントリが有効であれば当該命令ブロック内に存在する分岐命令をtakenと予測する．この場合，次サイクルのプリフェッチはBTBエントリに登録された分岐先アドレスを用いて行われる．

(2) NOP置換

分岐予測がtakenの場合，当該命令ブロックに存在し，分岐予測に対応する分岐命令の後続命令をすべてNOPとする．

(3) 制御依存検出とインターロック

プリデコードの結果，分岐命令ないしブースト命令の有無が判明すると，以下のように条件付実行モードを設定，および，インターロック制御を行う．

①分岐パイプラインに実行未終了の分岐命令が存在する場合：

(i) 命令ブロック内に分岐命令が存在する場合：分岐パイプライン中の分岐命令が実行終了するまで，IFステージをインターロックする．実行終了したら，②の処理を行う．

(ii) 命令ブロック内に分岐命令が存在しない場合：すべての命令をレベル1の条件付実行モードとして，Dステージへと進める．

②分岐パイプラインに実行未終了の分岐命令が存在しない場合：

(i) 命令ブロック内に分岐命令が存在する場合：分岐命令より後の命令をすべて条件付実行モードとする．複数の分岐命令が同一命令ブロックに存在する場合，条件付実行モードのレベルを分岐命令数に比例して増やす．すなわち，1番目の分岐命令以降はレベル1，2番目の分岐命令以降はレベル2，となる．レベル2以上の条件付実行モード下の命令が存在しても，IFステージにおいてはこれらの命令のインターロックは行わない．しかし，これらの命令は，Dステージにおいて命令の発行をブロックされる．

(ii) 分岐命令が存在しない場合：命令ブロック内の命令はすべて，無条件実行モードとなる．

ブースト命令については，上記①②で設定される条件付実行モードをさらに1レベル上げて設定する．

5.2.2 分岐パイプライン処理過程

分岐パイプラインにおける処理は，以下の要因によって支配される．

ⓐ分岐予測（taken／not-taken）

ⓑアドレッシング・モード（GRアクセスが必要／不要）

ⓒ分岐命令の型（BTB登録型／非登録型）

ⓓ分岐結果（TAKEN／NOT-TAKEN）

上記ⓐⓑⓒはIFステージで，ⓓはBDステージで判明する．

図4に分岐パイプラインの処理の概要を示す．

(1) 制御依存解消

データ依存検出の結果，当該分岐命令のソース・オペランドであるTFRにフロー依存がない場合，TFRを読み出す．読み出したTF値が分岐結果となる．これと予測結果を比較し，分岐予測の当否を判定する．

ⓐ分岐予測が当たった場合（図4のケース①②⑤⑥⑩）：すべての命令の条件付実行モードを1レベル下げる．つまり，レベル1の条件付実行モードで実行されている命令は，無条件実行モードとし，レベル2の条件付実行モードの命令は，レベル1となる．また，2重化レジスタ・ファイルに関しては，3.2節で述べたように，有効なオルタネート・レジスタをカレント・レ

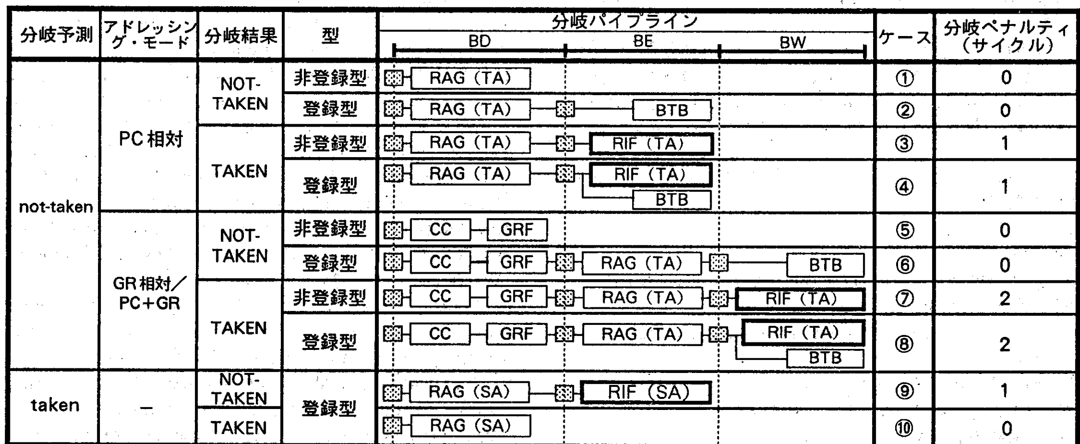

図4． 分岐パイプライン処理過程

ジスタに切り換える．図4のケース①②⑥⑩はBDステージで，ケース⑥はBEステージでそれぞれステージのインターロックを解除する．

ⓑ分岐予測が外れた場合（図4のケース③④⑦⑧⑨）：すべての条件付実行モードの命令を無効化する．また，オルタネート・レジスタも無効化する．

(2) 再フェッチ・アドレス生成

分岐予測が外れた際に必要となる再フェッチ・アドレスの生成を行う．再フェッチ・アドレスは，機能ユニットの1つであるRAG（Re-fetch Address Generator）を用いて計算する．再フェッチアドレス生成処理は，以下のように，予測結果とアドレッシング・モードによって支配される．

ⓐ分岐予測がnot-takenの場合：再フェッチアドレスとして，分岐先アドレスを生成する．

(i) アドレッシング・モードがPC相対の場合（図4のケース①-④）：BDステージでアドレス計算を行う．

(ii) アドレッシング・モードがGR相対およびPC+GRの場合（図4のケース⑤-⑧）：BDステージではGRへのアクセスを行い，BEステージでアドレス計算を行う．

ⓑ分岐予測がtakenの場合（図4のケース⑨⑩）：再フェッチアドレスとして，非分岐先アドレスを生成する．GRへのアクセスが不要なので，常にBDステージでアドレス計算を行う．また，BTB登録型の分岐命令で，まだBTBに登録されていない命令（すなわちnot-taken予測された命令）は，分岐結果（TAKEN／NOT-TAKEN）に関わらず，分岐先アドレスをBTBに登録する（図4のケース②④⑥⑧）．

(3) パイプライン復元処理

分岐予測が外れた場合，誤った予測によりフェッチされた命令およびその実行結果を無効化（パイプライン・フラッシュ）すると同時に，再フェッチ・アドレスを用いて，正しい命令流を再フェッチ（RIF：Re-Instruction Fetch）する（図4のケース③④⑦⑧⑨）．

5.3 分岐ペナルティ

分岐命令のソース・オペランドに対するフロー依存はないと仮定したときの，分岐命令に起因するパイプラインの乱れ（分岐ペナルティ）は以下のようになる．

ⓐ分岐予測ヒットの場合（図4のケース①②⑤⑥⑩）：ペナルティなし．

ⓑ・PC相対で分岐予測ミス，あるいは，taken予測で分岐予測ミスの場合（図4のケース③④⑨）：1サイクルのペナルティ．

ⓒ・GR相対またはPC+GR，かつ，not-taken予測で分岐予測ミスの場合（図4のケース⑦⑧）：2サイクルのペナルティとなる．

制御依存は，TFRのフロー依存がなければ，すべてBDステージで解消される．

6. おわりに

以上，DSNSプロセッサの，分岐アーキテクチャ，および，分岐パイプライン処理について述べた．

分岐ペナルティの影響を軽減するために，以下の手法を採用している．

①静的分岐予測＋分岐先バッファ
②投機的実行：条件付実行モードおよびブースティング
③先行条件決定方式
④早期分岐解消

現在，ハードウェア開発を進めると同時に，ソフトウェア・シミュレータを開発している．採用した上記の4手法は互いに密接に関係し合っているので，これらの間の相互作用について今後評価を行う予定である．

また，本分岐アーキテクチャは，最適化コンパイラの存在を前提としている．静的分岐予測，ブースティング，および，先行条件決定方式については，最適化コンパイラ抜きにしては効果が期待できない．特に，ブースティングおよび先行条件決定方式を有効たらしめるには，高度な静的コード・スケジューリング・アルゴリズムの開発が必須である．これら最適化コンパイラ技術については，別の機会に報告したい．

参考文献

[1] 原ほか："SIMP（単一命令流／多重命令パイプライン）方式に基づくスーパースカラ・プロセッサ『新風』の命令供給機構，" 情処研報，ARC-80-7（1990年1月）．

[2] 原ほか："『新風』プロセッサの条件分岐方式，" 情処40回全大，7L-6（1990年3月）．

[3] 村上ほか："SIMP（単一命令流／多重命令パイプライン）方方式に基づくスーパースカラ・プロセッサの改良方針，" 信学技報，CPSY-90-54（1990年7月）．

[4] 原ほか："SIMP（単一命令流／多重命令パイプライン）方式に基づく改良版スーパースカラ・プロセッサの構成と処理，" 信学技報，CPSY-90-55（1990年7月）．

[5] 納富ほか："DSN型スーパースカラ・プロセッサ・プロトタイプのロード／ストア・パイプライン，" 情処研報，ARC-86-4（1991年1月）．

[6] Ditzel,D.R. and McLellan,H.R.: "Branch Folding in the CRISP Microprocessor: Reducing Branch Delay to Zero," Proc. 14th Ann. Int'l. Symp. Computer Architecture, pp.2-9, June 1987.

[7] Hwu,W.W. and Patt.Y.N.: "Checkpoint Repair for High-Performance Out-of-Order Execution Machines," IEEE Trans. Comput., vol.C-36, no.12, pp.1496-1514, Dec. 1987.

[8] Lee,J.K.F. and Smith,A.J.: "Branch Prediction Strategies and Branch Target Buffer Design," Computer, vol.17, no.1, pp.6-22, Jan. 1984.

[9] Lilja,D.J.: "Reducing the Branch Penalty in Pipelined Processors," Computer, vol.21, no.7, pp.47-55, July 1988.

[10] McFarling,S. and Hennessy,J.: "Reducing the Cost of Branches," Proc. 13th Ann. Int'l. Symp. Computer Architecture, pp.396-403, June 1986.

[11] Pleszkun,A.R. et al.: "WISQ: A Restartable Architecture Using Queues," Proc. 14th Ann. Int'l. Symp. Computer Architecture, pp.290-299, June 1987.

[12] Rosocha,W.G. and Lee,E.S.: "Performance Enhancement of SISD Processors," Proc. 6th Ann. Symp. Computer Architecture, pp.216-231, Apr. 1979.

[13] Smith,J.E.: "A Study of Branch Prediction Strategies," Proc. 8th Ann. Symp. Computer Architecture, pp.135-148, May 1981.

[14] Smith,J.E. and Pleszkun,A.R.: "Implementing Precise Interrupts in Pipelined Processors," IEEE Trans. Comput., vol.37, no.5, pp.562-573, May 1988.

[15] Smith,M.D.,Lam,M.S., and Horowitz,M.A.: "Boosting Beyond Static Scheduling in a Superscalar Processor," Proc. 17th Ann. Int'l. Symp. Computer Architecture, pp.344-354, May 1990.

[16] Sohi,G.S. and Vajapeyam,S.: "Instruction Issue Logic for High-Performance, Interruptable Pipelined Processors," Proc. 14th Ann. Int'l. Symp. Computer Architecture, pp.27-34, June 1987.