計算機アーキテクチャ　86－4
（1991　1．24）

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

納富 昭　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）

DSN型，すなわち，動的ハザード解消，静的コード・スケジューリング，非均質型機能ユニットといった特長を備えたスーパースカラ・プロセッサの試作機を現在開発している．スーパースカラ・プロセッサにおいては，十分なデータ供給能力を確保するため，ロード／ストア・パイプラインの多重化が必要不可欠である．

本稿では，ロード／ストア・パイプラインを多重化する際の問題点について述べた後，その対処法として，実行順序を指定可能なロード／ストア命令アーキテクチャ，および，2つのデータ・アクセスを並列に実行可能なデュアルポート・データキャッシュの構成について述べる．

Load／Store Pipelines of the DSNS Processor Prototype
(in Japanese)

Akira NODOMI, Morihiro KUGA, Kazuaki MURAKAMI, and Shinji TOMITA

Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
6-1 Kasuga-koen, Kasuga-shi, Fukuoka, 816 Japan
e-mail : noudomi@is.kyushu-u.ac.jp

We are developing DSNS (Dynamically-hazard-resolved, Statically-code-schduled, Nonuniform Superscalar) processor prototype. Load／store Pipelines must be multiplied to provide sufficient data bandwidth for superscalar processors.

This paper discusses the issues on multipling load／store pipelines. The following two methods are presented : Load／store-instruction architecture which allows compilers to designate the execution order of load／store instructions, and dual-port data-cache which can accept two load／store instructions simultaneously.

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［1］．

(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ってしまい，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，動的コード・スケジューリングより高い並列性を得ると同時に，ハードウェアの増大を回避している．

(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応じ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する．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，非均質型機能ユニットにおいては，個々の機能ユニットの多重度をどのように設定するかが課題となる．特に，データのロード／ストア・パイプラインはボトルネックとなる危険性があると同時に，いたずらに多重度を大きくすると非常にコストが高くつくユニットである．ロード／ストア・パイプラインを2本以上設けた場合，以下の点を考慮する必要がある．

①ロード／ストア命令間依存関係の保証：ロード／ストア・パイプラインが1本しかない場合には，例えば，in-orderにロード／ストア命令の実行を行うことによりメモリ・アクセスに関するデータ依存関係を保証することができる．しかし，ロード／ストア・パイプラインを2多重以上の構成にした場合には，同一サイクルに多重度分のロード／ストア命令を実行できなければハードウェア量に見合った性能向上は望めない．そこで，複数のロード／ストア命令を同時に実行するための依存関係対策が必要となる．

②データ・アクセスの多重化：通常のデータキャッシュでは1サイクルに1回のデータ・アクセスを処理できれば良いが，ロード／ストア・パイプラインを2多重以上の構成にした場合には，キャッシュへのアクセスも多重化しなければならない．これに対処し得るデータキャッシュの構成を新たに考案する必要がある．

本稿では，DSNSプロセッサの概要（2章）を述べた後，DSNSプロセッサで採用したロード／ストア命令アーキテクチャ（3章），データキャッシュの構成（4章），およびロード／ストア・パイプライン処理過程（5章）について述べる．

2. DSNSプロセッサの概要［2］

現在我々が開発を行っているDSNSプロセッサ・プロトタイプ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．

2.1 プロセッサ構成

DSNSプロセッサは以下の主要ユニットから構成される．

(1) 命令キャッシュ（IC：Instruction Cache）

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と命令キャッシュとは，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．

(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，出力依存の検出
・先行命令ブロックに対するフロー依存，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)［3］

早期分岐解消（early branch resolution）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．

(6) 機能ユニット（FUs：Functional Units）

図1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／ストア系ユニット：2本の独立したロード／ストア・ユニット
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（advanced conditioning）方式のための2つのユニット

浮動小数点の除算器を除くすべての機能ユニットはパイプライン化されており，1サイクル毎（ただし，倍精度の浮動小数点演算は2サイクル毎）に実行結果を得ることができる．また，実行結果を次サイクルの演算のオペランドとして使用可能とするために，バイパスパスを備える．

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

図1に示すように，以下の3系統のレジスタ・ファイルを備える．条件付実行モードと組み合わせて正確な分岐を保証するために，レジスタの各エントリは2重化されている．以下のレジスタ個数は論理的なものである．

①汎用レジスタ（GR）：32ビット長レジスタ32個から成る．読出しポート8，書込みポート2を備える．各レジスタには，コンディションを格納する4ビットのタグがある．
②浮動小数点レジスタ（FPR）：64ビット長レジスタ32個から成る．読出しポート4，書込みポート2を備える．GR同様，

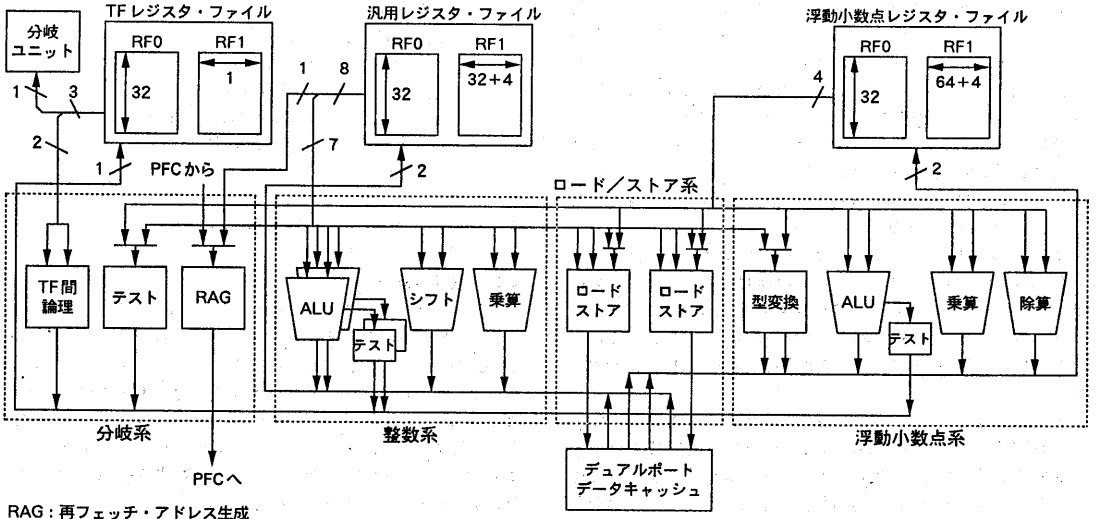


図1. DSNSプロセッサのデータ・パス

各レジスタは4ビットのタグを持つ.

③TFレジスタ（TFR）：1ビット長レジスタ32個から成る. 読出しポート3, 書込みポート1を備える.

また, 2重化レジスタ・ファイルは, その構造上サイクル前半で書込みを行い, 後半で読出しを行う（図2参照）.

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

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. ポートサイズは最長データである倍精度浮動小数点データに備えて8バイトである. また, ダイレクト・マッピング方式の仮想アドレス・キャッシュであり, 主記憶への書込みにはコピーバック方式を採用している. 構成の詳細については4章で述べる.

2.2 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

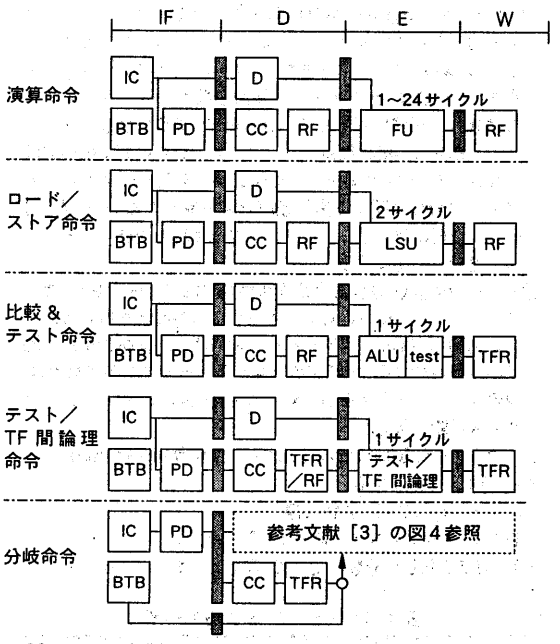
②D：コンフリクト・チェック＋デコード＋オペランド・フェッチ

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2に, 命令の種類毎の命令パイプライン処理過程を示す. ロード／ストア命令処理過程については5章で詳述する.



CC：コンフリクト・チェッカ
D：デコーダ
FU：機能ユニット
IC：命令キャッシュ
LSU：ロード／ストア・ユニット
PD：プリデコーダ
RF：レジスタ・ファイル
TFR：TFレジスタ・ファイル

図2. パイプライン処理過程

3. ロード／ストア命令アーキテクチャ

3.1 ロード／ストア命令間依存関係

DSNSプロセッサでは, データ依存, 制御依存, 資源の競合といったハザードの検出・解消をハードウェアが実行時に行う. これらのハザードの中でデータ依存は, レジスタ・アクセスに関するデータ依存, および, メモリ・アクセスに関するデータ依存に大別できる. そして, レジスタ・アクセスに関するデータ依存については, レジスタ・スコアボードによりその検出・解消を行っている［2］. しかし, メモリ・アクセスに関するデータ依存への対処をレジスタと同様な方法で処理することは不可能である. したがって, 別の対策を講じなければならない.

ここで, メモリ・アクセスに関するデータ依存関係, すなわち, ロード／ストア命令間依存関係として問題になるのは, 次の3種類である.

①LAS（Load After Store）依存関係：フロー依存

②SAL（Store After Load）依存関係：逆依存

③SAS（Store After Store）依存関係：出力依存

これらの依存関係を実行時に検出するためには，これから実行するロード／ストア命令のアドレスと，実行を終了していないすべての先行ロード／ストア命令のアドレスとを比較しなければならない．これをハードウェアで実現するのは非常に困難である．しかも，2.1節で述べたように，DSNSプロセッサでは同一構成のロード／ストア・パイプラインを2本備えている．1章で課題として取り上げたように，2本のロード／ストア・パイプラインに対し同時に命令の実行を行えるような方式を採用しなければ2重化の効果がない．

そこで，DSNSプロセッサではハードウェアによる依存関係の検出は一切行わず，コンパイル時の静的な検出にすべてを任せている．コンパイラはデータ依存解析結果に基づいて，個々のロード／ストア命令に実行順序を指定する情報を付加する．この情報により，ロード／ストア命令は，以下の3種類に分類される．

(1) Strongly Ordered (SO)：完全逐次実行すべき命令である．依存関係の存在が明らかな場合，または，依存関係の有無が判断できない場合，当該ロード／ストア命令を完全逐次実行するよう指定する．この種類に属するロード／ストア命令同士のデータ・アクセスはin-orderに行う．

(2) Weakly Ordered (WO)：同一データ型のロード／ストア命令同士では逐次実行する．すなわち，メモリ・アクセスに関して，2種類のデータ型（整数および浮動小数点データ）を区別する．一般に，整数データと浮動小数点データはメモリ上では異なる領域に割り当てられるので，通常は整数ロード／ストア命令と浮動小数点ロード／ストア命令の間に依存関係は存在しない．したがって，異なるデータ型に対するロード／ストア命令に関する実行順序は何ら制限しない．一方，同一データ型に対するロード／ストア命令同士のデータ・アクセスはin-orderに行う．

(3) Unordered (UO)：完全非逐次実行可能な命令である．他のロード／ストア命令に対してまったく依存関係がないと断定できる場合，当該ロード／ストア命令は完全非逐次実行可能であると指定する．この種類に属するロード／ストア命令は，他のロード／ストア命令に対してout-of-orderにデータ・アクセス可能となる．

以上のように分類されたロード／ストア命令の実行制御については，5.1節で詳述する．

3.2 ロード／ストア命令仕様

表1に，ロード／ストア命令の一覧を示す．

3.1節で定義した分類に従って，表1のニモニックの“＊”には以下の拡張子が入る．

①'so'：完全逐次実行すべき命令
②'wo'：同一データ型同士で逐次実行すべき命令
③'uo'：完全非逐次実行可能な命令

アドレッシング・モードは，次の2種類である．

ⓐベース＋オフセット（符号付き13ビット）
ⓑベース＋インデックス

表1．ロード／ストア命令一覧

		ニモニック	意味
整数	ロード	ld.sb.＊	符号付き整数バイトロード
		ld.sh.＊	符号付き整数ハーフワードロード
		ld.w.＊	整数ワードロード
		ld.ub.＊	符号なし整数バイトロード
		ld.uh.＊	符号なし整数ハーフワードロード
	ストア	st.b.＊	バイトストア
		st.h.＊	ハーフワードストア
		st.w.＊	ワードストア
浮動小数点	ロード	fld.s.＊	単精度浮動小数点ロード
		fld.d.＊	倍精度浮動小数点ロード
	ストア	fst.s.＊	単精度浮動小数点ストア
		fst.d.＊	倍精度浮動小数点ストア

＊：SO／WO／UO

4. デュアルポート・データキャッシュ

4.1 マルチポート化

4.1.1 設計方針

DSNSプロセッサには2本のロード／ストア・パイプラインが存在し，各々が独立にデータキャッシュへのアクセスを要求する．複数のアクセスを同時に処理するデータキャッシュの構成として，マルチポートのメモリチップを使用する方法が考えられる．しかし，高速かつ大容量のマルチポート・メモリチップは入手が困難である．そこで，シングルポートのメモリチップを用いるという前提の下，複数のアクセス要求を高速かつ並列に処理するため，以下の設計方針を立てた．

【設計方針1】複数のアクセス要求を同時に受け付け可能とするため，データキャッシュとして複数のポートを設ける．

【設計方針2】複数のタグ検索（ヒット／ミスヒット判定）を同時に行うことを可能とするため，タグアレイの構成を工夫する．

【設計方針3】複数のデータ・アクセスを同時に行うことを可能とするため，データアレイの構成を工夫する．

方針1に関しては，ロード／ストアパイプラインが2本なので，2個のポートを用意すればよい．方針2および方針3に関しては，キャッシュ構成に関する選択肢が考えられる．

4.1.2 選択肢

図3に示す4つの構成方式が存在する．まず，データアレイをインタリーブするか否かで，次の2つの選択肢に分かれる．

(1) D (Duplicated) 型（図3①）

データアレイおよびタグアレイの完全なコピーをポート毎に設ける．

(2) I (Interleaved) 型

データアレイをインタリーブする．これはさらに，インタリーブされたデータアレイの各バンクをポート間で共有するか否かで，次の2つの選択肢に分かれる．

(i) IS (Interleaved, Shared) 型（図3②）[4]

インタリーブされたデータアレイの各バンクをすべてのポートが共有する．タグアレイについては，ポート毎に完全なコピーを設ける．バンク数はポート数とは無関係に決まる．

(ii) ID (Interleaved, Dedicated) 型

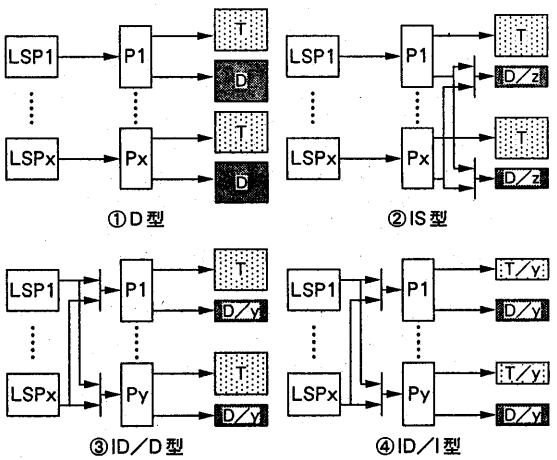
インタリーブされたデータアレイの各バンクはいずれかのポートに専有される．よって，バンク数はポート数に等しい．各ポートはプロセッサのすべてのロード／ストア・パイプラインに共有される必要がある（図3③④参照）．これはさらに，タグアレイをインタリーブするか否かで，次の2つの選択肢に分かれる．

①ID／D (ID／Duplicated) 型（図3③）

タグアレイの完全なコピーをポート毎に設ける．インタリーブ幅はラインサイズとは無関係に決まる．

②ID／I (ID／Interleaved) 型（図3④）[5]

タグアレイをインタリーブする．インタリーブされたタグアレイの各バンクは，対応するデータアレイ・バンクを



LSP：ロード/ストア・パイプライン　　x：LSP数
P：ポート　　y：ポート数
T：タグアレイ（面積はハードウェア量を表わす）　　z：バンク数
D：データアレイ（面積はハードウェア量を表わす）

図3. マルチポート・データキャッシュ構成方式

専有しているポートに専有される．よって，インタリーブ幅はラインサイズに等しくなる．

4.1.3 考察

以上の4方式について，表2に示す項目に関する比較および考察を行った．なお，以下の記述では，次の表記を用いる．

- t：同一サイズのシングルポート・データキャッシュを構成するのに要するタグアレイのハードウェア量
- d：同一サイズのシングルポート・データキャッシュを構成するのに要するデータアレイのハードウェア量
- x：ロード/ストア・パイプライン本数
- y：データキャッシュのポート数（D型およびIS型では，y=x）
- z：データアレイのバンク数（ID型では，z=y）

(1) タグアレイおよびデータアレイのハードウェア量

D型は，タグアレイおよびデータアレイのコピーをポート毎に設けるので，ハードウェア量がO（yt+yd）と最も大きい．一方，ID/I型のハードウェア量はO（t+d）と最も小さく，シングルポート・データキャッシュと大差ない．

(2) 制御

D型は基本的にマルチプロセッサにおけるマルチキャッシュ構成と等価なので，ストア・アクセスの際，コヒーレンス制御が必要となる．また，IS型は，各ポートでロード/ストア命令を受け付けた後にバンク・コンフリクトの調停を行うため，データアレイへのアクセスのタイミング制御が難しい［4］．これらに対し，両ID型は，ポート・コンフリクト調停後の制御はシングルポート・データキャッシュ並みと容易である．

(3) バンク・コンフリクトの影響

D型では，コンフリクトは発生しない．しかし，他の3方式はデータアレイをインタリーブしているので，同一バンクに対するアクセスによりコンフリクトが生じる．

バンク・コンフリクトの発生頻度を抑えるためには，

ⓐバンク数を増やす

ⓑインタリーブ幅を小さくする

といった手法がある．IS型はポートとバンクの間に対応関係がないので，ポート数を増やすことなく手法ⓐを適用できる．一方，両ID型はポートとバンクとの間に1対1の対応関係があり，手法ⓐを適用するにはポート数を増やさなければならない．これは，ハードウェア量の増加を招く．そこで，手法ⓑの適用を考える．ID/D型にはインタリーブ幅に関する制約がないので，任意のインタリーブ幅を採ることができる．たとえば，最大データ・サイズに合わせて，8バイト・インタリーブとすることが可能である．一方，ID/I型では，インタリーブ幅がラインサイズに等しい．よって，手法ⓑを適用するには，ラインサイズを小さくしなければならない．ところが，ある程度の容量を備えたデータ・キャッシュにおいては，ラインサイズに比例してヒット率も上昇する傾向がある［6］．よって，ラインサイズの決定にあたっては，ミスヒットとバンク・コンフリクトとの両ペナルティ間のトレードオフをとる必要があり，むやみに小さくできない．

(4) 空振りアクセス

空振りアクセスとは，ミスヒットとなるアクセス要求がデータアレイのバンクのアクセス権を獲得してしまい，ヒットとなるアクセス要求が待たされる場合をいう．バンク・コンフリクトが生じないD型では，空振りアクセスも生じない．しかし，他の3方式では，ヒット/ミスヒット判定以前にバンク・コンフリクトの調停を行うので，空振りアクセスが生じ得る．

(5) ミスヒット処理の範囲

タグアレイの完全なコピーをポート毎に設けるD型，IS型およびID/D型では，あるポートで生じたミスヒットの影響をすべてのポートが受ける．すなわち，ミスヒットに伴うライン・リプレースメントにより，タグアレイのすべてのコピーを更新する．さらに，D型では，データアレイのすべてのコピーも更新する必要がある．これらに対して，ID/I型では，データアレイお

表2. マルチポート・データキャッシュ構成方式の比較

構成方式＼比較項目	ハードウェア量	制御	バンク・コンフリクトの影響	空振りアクセス	ミスヒット処理の範囲
D型	O（yt+yd）	コヒーレンス制御が必要	なし	発生しない	タグアレイ，データアレイの全コピー
IS型	O（yt+d）	タイミング制御が難しい	あり，手法ⓐⓑで対処可	発生し得る	タグアレイの全コピー
ID/D型	O（yt+d）	シングルポート・キャッシュ並み	あり，手法ⓑで対処可	発生し得る	タグアレイの全コピー
ID/I型	O（t+d）	シングルポート・キャッシュ並み	あり	発生し得る	ミスヒットを起したバンクのみ

t：同一サイズのシングルポート・データキャッシュを構成するのに要するタグアレイのハードウェア量
d：同一サイズのシングルポート・データキャッシュを構成するのに要するデータアレイのハードウェア量
y：データキャッシュのポート数（D型およびIS型では，ロード/ストア・パイプライン数に等しい）

よびタグアレイともにインタリーブしている．よって，ミスヒットを起こしたバンク内に閉じてライン・リプレースメントを行うことができる．

4.2 ノンブロッキング化

並列アクセスを可能にするとともに，キャッシュ本来の性能向上も図る必要がある．そのためには，

ⓐヒット率を向上させる

ⓑミスヒット処理の遅延を抑える

という大きく2つのアプローチがある．以下，ⓑについて，キャッシュのノンブロッキングについて検討する．

ノンブロッキング・キャッシュとは，「ミスヒット処理を行っている最中でも，後続のアクセス要求を受け付け得るキャッシュ」であり，これによりミスヒット処理による遅延を見かけ上小さくすることができる．

ここで，ノンブロッキング・キャッシュを構成する際の課題を以下に示す．

①最大許容ミスヒット数（ノンブロッキングを保ったまま保持しておけるミスヒット処理中またはミスヒット処理待ちのアクセス要求の最大数）の設定

②後続ミスヒット（ミスヒット処理中またはミスヒット処理待ちのアクセス要求が残っている段階で生じた新たなミスヒット）の処理方法

③ミスヒット処理のためのデータアレイ・アクセス（ライン・リプレースメント）と，ヒットした後続アクセス要求によるデータアレイ・アクセスとの間の調停方法

ノンブロッキング・キャッシュの実現方式としては，文献［7］のMSHR（Miss information／Status Holding Register）を用いたKroftの方式がある．このKroftの方式について上記の課題を検討すると，以下のようになる．なお，ダイレクト・マッピング方式を仮定する．

①最大許容ミスヒット数：「最大許容ミスヒット数＝（MSHRの数－1）」となる．つまり，MSHRを1個だけ用意した場合，1回ミスヒットが起こると後続のアクセス要求をブロックしなければならない．また，2個のMSHRを用意した場合，1個のミスヒット処理を行っている間はノンブロッキングであるが，後続のアクセス要求がミスヒットを起こすとそれ以後のアクセス要求をブロックする．

②後続ミスヒットの処理方法：図4に示すキャッシュ上で，メモリの第mS＋nブロックに対するミスヒットが発生した場合，当該ミスヒットを以下の3種類に区分する．

・PM（Primary Miss）：第jS＋nブロックに対するミスヒット（すなわち，同一セットのライン上でのミスヒット）が先に発生していない場合．

・SLM（Secondary, same-Line Miss）：第mS＋nブロックに対するミスヒット（すなわち，同一ライン上でのミスヒット）が先に発生しており，当該ミスヒットがミスヒット処理待ちまたはミスヒット処理中である場合．

・SSM（Secondary, same-Set Miss）：第jS＋nブロック（j≠m）に対するミスヒット（すなわち，同一セットで同一ライン以外のライン上でのミスヒット）が先に発生しており，当該ミスヒットがミスヒット処理待ち，または，ミスヒット処理中である場合．

Kroftの方式では，PMに対しては無条件に新たなMSHRを割り当てる．また，SSMに対しても同様に新たなMSHRを割当てる．ただし，同一セットに対する先行ミスヒットのためのライン・リプレースメントは行わないようにする．そして，SLMに対してはMSHRの割当てを行わず，先行する同一ラインへのミスヒットに対して割り当てられているMSHRの内容を部分的に書き換えることにより，ミスヒット処理を1つにまとめている．

③データアレイ・アクセスの調停方法：特には規定せず，インプット・スタック（ラインフェッチ・データを保持するFIFO）を設定しているのみである．しかし，ハードウェアでの実現に際しては重要な項目であり，ノンブロッキングの効果に直接影響する．

このように，Kroftの方式では，データアレイへのアクセスを最小限にする，MSHRを有効に利用する，といったきめ細かなミスヒット処理を行っている．しかし，ミスヒットが発生した時点でPM, SLM, SSMを識別するには，ミスヒット・アドレスによりMSHRを連想アクセスできなければならない．また，MSHRの部分書換えなどを実現し得るハードウェアは非常に複雑なものとなる．

以上のような点を考慮した上で，ノンブロッキング化に対して次のような実現方針を立てた．

(1) 最大許容ミスヒット数：MSHRを複数設けることにより，最大許容ミスヒット数を1以上とする．

(2) 後続ミスヒットの処理方法：PM, SLM, SSMといったミスヒットの識別を発生時には行わず，すべてのミスヒットに対してMSHRを割り当てる．そして，ミスヒット処理をその発生順に逐次的に行う．しかし，ミスヒット処理待ちからミスヒット処理実行に移る際に再度タグを検索することにより，SLMか否かを識別する．SLMであった場合（タグがヒットした場合），ライン・リプレースメントを行わず，所要データ・アクセスを行う．SLMではなかった場合（タグがミスヒットした場合），ライン・リプレースメントを開始する．

(3) データアレイ・アクセスの調停方法：ミスヒット処理を優先する．これは，ミスヒット処理の遅延を小さくすることにより，MSHRの早期解放が可能となり，また，後続命令の有効なアクセスにもつながるからである．しかも，ハードウェアの実現も比較的容易である．

具体的なミスヒット処理過程については，5.2節で詳述する．

4.3 構成

DPDCのポートサイズは，最大データ長である倍精度浮動小数点データに合わせて8バイトとした．

構成は，D型，IS型，ID／D型，ID／I型の4方式の中から，ID／I型を採用した．これは，

①ヒット処理，ミスヒット処理ともに制御が容易である．

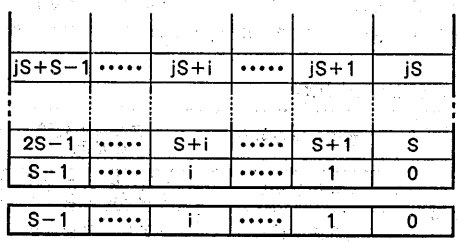


図4. キャッシュとメモリのラインとの対応関係

②全ハードウェア量が最も小さい.

という点を重視した結果である. バンク・コンフリクトが生じやすいという問題点については, 静的コード・スケジューリングによりある程度対処できる.

4.1節で述べたように, ID/I型を採用するにあたっては, キャッシュのラインサイズの設定が問題となる. ラインサイズの決定にあたっては, 同じID/I型を採用したマルチポート・ノンブロッキング・キャッシュに関するシミュレーション結果 [5], および, 文献 [6] より, 32バイトが妥当であると判断した.

図5に示すように, DPDCは各バンク毎に, 以下の主要ユニットを備える.

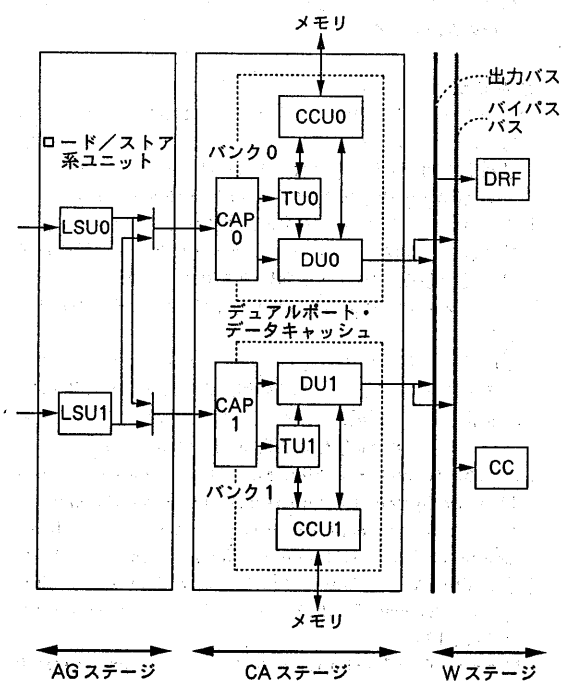
①キャッシュ・アクセス・ポート (CAP) : ロード/ストア・パイプラインからの要求を受け付ける.

②タグ・ユニット (TU) : タグアレイ, タグ制御回路, および, ヒット/ミスヒット判定回路などを含む.

③データ・ユニット (DU) : データアレイ, ダイナミック・サイジング処理回路などを含む.

④キャッシュ・コントロール・ユニット (CCU) : ミスヒット処理やデータキャッシュ制御命令の処理を行う. また, メモリ・インターフェースを備える.

1個のアクセス要求に対して, タグ・ユニットとデータ・ユニットは並列に動作し, ロードおよびストアとも1サイクルでアクセスを完了する. ロード命令によるデスティネーション・レジスタへの書込みの際には, 出力パスおよびバイパスパスにデータを乗せる. 出力パスはレジスタ・ファイルの書込みポートに, また, バイパスパスはコンフリクト・チェッカに, それぞれ接続されている.



CAP : キャッシュ・アクセス・ポート
CC : コンフリクト・チェッカ
CCU : キャッシュ・コントロール・ユニット
DRF : 2重化レジスタ・ファイル
DU : データユニット
LSU : ロード/ストア・ユニット
TU : タグユニット

図5. ロード/ストア・パイプライン構成

5. ロード/ストア・パイプライン処理過程

5.1 通常処理過程

2.2節で述べた命令パイプラインの4ステージのうち, 次の2ステージがロード/ストア命令に特有の処理内容となる.

(1) Dステージ : レジスタに関するデータ依存関係の判定後, 依存関係にないロード/ストア命令をロード/ストア・ユニット (LSU0またはLSU1) にディスパッチする. 最大2命令まで同時にディスパッチできる. このディスパッチ制御については, 後述する.

(2) Eステージ : 以下の2ステージにさらにパイプライン化される.

①AG (Address Generate) ステージ : 各LSUにおいて実効アドレスを計算し, LSU間のバンク・コンフリクト調停を行う. バンク・コンフリクトの結果CAPを獲得できなかったLSUは, インターロックされる. バンク・コンフリクト調停の際の優先順位は, インターロックしているLSUの方が高い (両LSUともインターロックしていない場合は, LSU0の方が高い). よって, 現サイクルでインターロックされたLSUは, 次サイクルで必ず当該CAPを獲得できる.

②CA (Cache Access) ステージ : DPDCの各バンクにおいてデータ・アクセスを遂行する. ヒットした場合, ロードおよびストアとも1サイクルでアクセスを完了する. ミスヒットした場合, 5.2節で述べるように, 使用可能なMSHRが存在すればブロックしない. ただし, MSHR設定のため1サイクルのインターロックが必要となる. よって, ストア・アクセスは2サイクルで完了する. 一方, ロードは, メモリからのデータ到着までアクセス完了を待たされる.

さて, Dステージにおけるロード/ストア命令のディスパッチ制御は, 以下の方針に基づく. なお, 少なくとも1個のLSUはインターロックしていないものとする.

【方針1】最も先行するロード/ストア命令はその種類 (SO, WO, UO) に関係なく, レジスタに関するデータ依存関係がない限り常にディスパッチ可能である.

【方針2】先行するSOのロード/ストア命令がディスパッチ不可能なら, 後続のSOのストア命令もディスパッチ不可能とする (SALおよびSASの保証). また, 先行するSOのストア命令がディスパッチ不可能なら, 後続のSOのロード命令もディスパッチ不可能とする (LASの保証).

【方針3】先行するSOまたはWOの整数ロード/ストア命令がディスパッチ不可能なら, 後続のWOの整数ストア命令もディスパッチ不可能とする (SALおよびSASの保証). また, 先行するSOまたはWOの整数ストア命令がディスパッチ不可能なら, 後続のWOの整数ロード命令もディスパッチ不可能とする (LASの保証).

【方針4】先行するSOまたはWOの浮動小数点ロード/ストア命令がディスパッチ不可能なら, 後続のWOの浮動小数点ストア命令もディスパッチ不可能とする (SALおよびSASの保証). また, 先行するSOまたはWOの浮動小数点ストア命令がディスパッチ不可能なら, 後続のWOの浮動小数点ロード命令もディスパッチ不可能とする (LASの保証).

【方針5】UOのロード/ストア命令は, レジスタに関するデータ依存関係がない限り常にディスパッチ可能である.

次に，ディスパッチ先のLSUは，以下のように決定する．

①両LSUともにインターロックしていない場合：ディスパッチ可能なロード／ストア命令のうち，最も先行する命令をLSU0に，また，その次の命令をLSU1に，それぞれディスパッチする．

②いずれかのLSUがインターロックしている場合：ディスパッチ可能なロード／ストア命令のうち最も先行する命令をインターロックしていないLSUにディスパッチする．

5.2 ミスヒット処理過程

4.2節で述べたノンブロッキング・キャッシュを実現するために，各バンクのキャッシュ・コントロール・ユニット（CCU）にMSHRを設ける．MSHRは，ミスヒットを起こしたアクセス要求に関する以下の情報を保持する．

①アドレス：ミスヒットを起こしたアドレス

②命令付随情報：ロード／ストア・アクセスの区別，デスティネーション・レジスタ番号（ロード・アクセスの場合），など

③ストアデータ：ストア・アクセスの場合，ストアすべきデータ

④Waitフラグ：ミスヒット処理待ちであることを示すフラグ

Kroftの方式におけるMSHRほどではないにしても，上記のMSHR1個当りのデータ量はかなり大きい．よって，1バンク当り何個のMSHRを設けるかが課題となる．Kroft [7] によると，性能向上はMSHR4個でほぼ飽和するとなっている．また，Kroftの方式によるマルチポート・ノンブロッキング・キャッシュに対するシミュレーション結果 [5] を見ると，1バンク当り4個のMSHRを設けた場合ブロッキングがほとんど起きていない．これらから，本DPDCでも，1バンク当り4個のMSHRを設けることにする．ただし，4.2節で述べたように，我々の採用した実現方針はKroftの方式に比べるとレジスタの使用効率が悪くなる可能性がある．したがって，MSHR数については，今後も検討が必要である．

さて，ミスヒットは当該バンクで以下のように処理される．

(1) まず，ミスヒットとなったラインのタグを無効化する．これは，後続のアクセス要求が当該ラインに対してヒットするのを防ぐためである．また，MSHRの割当てを行い，ミスヒット処理に必要な情報を設定する．このとき，すでに先行するミスヒット処理が進行中であれば，Waitフラグを立ててその終了を待つ．そうでなければ，(2) へ進む．なお，このMSHR割当てにより空きのMSHRが無くなった場合，以後のアクセス要求をブロックする．

(2) MSHRに登録しているアドレスを用いて，ライン・リプレースメントを行う．コピーバック方式を採用しているので，当該ラインがdirtyであればラインフェッチに先立ちコピーバックを行う．ラインフェッチには，ラップアラウンド方式のブロック転送を行う．これはライン中の所望のデータから先に転送する方式であり，ミスヒットによる遅延を短縮する [8]．所望のデータをフェッチしたら，次のように処理を行う．

①ロード・アクセスの場合：当該データを出力パスおよびバイパスパスに乗せる．

②ストア・アクセスの場合：当該データをMSHR中のストアデータで書き換えた後，データアレイの当該バンクに書き込む．

ラインフェッチを終了したら，タグを更新し有効にする．そして，MSHRを解放する．

(3) ミスヒット処理待ちのMSHRは，ミスヒットの発生順に処理を行う．その処理内容は上記 (2) に等しいが，処理開始に当りMSHR中のアドレスで再度タグ検索を行う点が異なる．すなわち，このタグ検索により，当該ミスヒットがSLM（4.2節参照）か否かを識別する．SLMである場合（ヒットした場合）は，直ちにアクセス要求を遂行する．このとき，ライン・リプレースメントは行わない．そうでない場合は，上記 (2) の処理を行う．

なお，上記の処理において，データアレイおよびタグアレイへのアクセス，および，MSHR設定を行っているサイクルのみ，当該バンクへのアクセス要求をブロックする．

6. おわりに

以上，現在開発中であるDSNSプロセッサのロード／ストア・パイプラインについて述べた．ロード／ストア・パイプラインの多重化に伴い，①ロード／ストア命令間依存関係の保証，および，②データ・アクセスの多重化，という課題が生じる．これらの課題に対処して，本稿では，①実行順序を指定可能なロード／ストア命令アーキテクチャ，および，②2個のデータ・アクセスが同時に可能なデュアルポート・データキャッシュを提案した．さらに，デュアルポート・データキャッシュは，ミスヒットの遅延を抑えるためにノンブロッキング化を図っている．

今後は，ハードウェアの開発と並行して，シミュレーションにより本ロード／ストア・パイプラインの有効性を評価していく予定である．特に，5.2節で述べたように，MSHRの個数については性能とハードウェア量とのトレードオフをとる必要がある．また，本稿で提案した実行順序を指定可能なロード／ストア命令アーキテクチャを活かすための最適化コンパイラの開発も今後の課題である．

参考文献

[1] 村上ほか："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・プロセッサの改良方針，" 信学技報「1990年並列処理に関する『琉球』サマー・ワークショップ（SWoPP琉球'90）」, CPSY90-54（1990年7月）.

[2] 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「1990年並列処理に関する『琉球』サマー・ワークショップ（SWoPP琉球'90）」, CPSY90-55（1990年7月）.

[3] 原ほか："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，" 情処研報, ARC-86-3（1991年1月）.

[4] 納富ほか："『新風』プロセッサのマルチポート・データキャッシュ，" 情処41全大論文集, 3P-1（1990年9月）

[5] Sohi, G. and Franklin, M. : "High-Bandwidth Data Memory System for Superscalar Processors,"Computer Sciences Department University of Wisconsin-Madison, Computer Sciences Technical Report #968, Sep. 1990.

[6] Smith, A. J. : "Line (Block) Size Choice for CPU Cache Memories," IEEE Trans. Comput., vol.C-36, no. 9, Sep. 1987.

[7] Kroft, D. : "Lockup-free Instruction Fetch／Prefetch Cache Organization," Proc. 8th Int'l. Symp. Comput. Archit., pp.81-87, May. 1981.

[8] Matick, R. E. : "Functional Cache Chip for Improved System Performance," IBM Journal of Research and Development, vol.33, no.1, pp.15-32, Jun. 1989.