スーパースカラ・プロセッサの構成方式に関する研究

久我, 守弘
九州大学総合理工学研究科情報システム学専攻

https://doi.org/10.11501/3060381

出版情報：九州大学, 1991, 博士（工学）, 課程博士
バージョン：
権利関係：

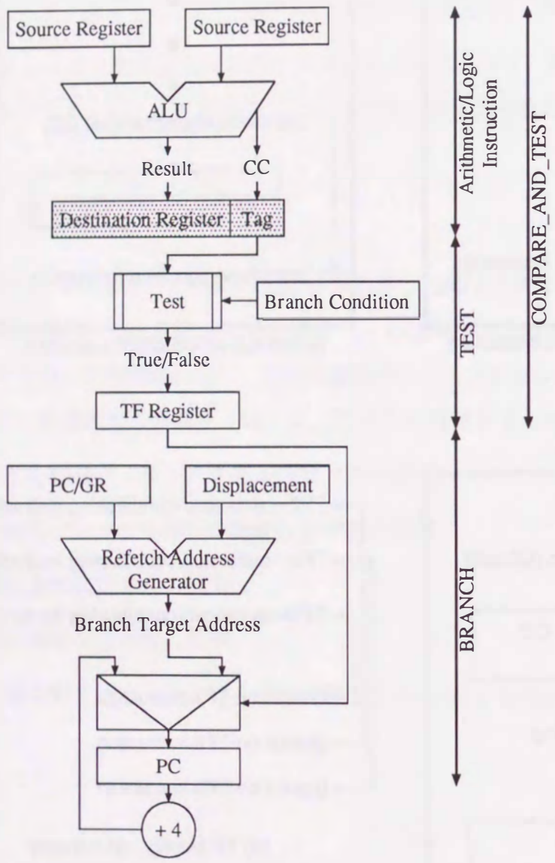


Figure 5.10 Advanced Conditioning.

第 6 章

DD 型スーパースカラ・プロセッサの構成と性能評価

本章では，DD 型スーパースカラ・プロセッサ (以後 DDUS と記す) のハードウェア構成について述べた後，ソフトウェア・シミュレータによる性能評価結果を示し，構成上の問題点を指摘する．

6.1 DDUS プロセッサの概要

DDUS プロセッサの構成を Figure 6.1 に，その諸元を Table 6.1 に示す． DDUS は 5 ステージからなる 1 本当たりピーク性能 8MIPS の命令パイプラインを 4 本備えており，スーパースカラ度 4 のプロセッサである．プロセッサは以下の主要ユニットから構成される．

(a) マルチバンク命令キャッシュ(MBIC : Multiple-Bank Instruction Cache) : 4 つの命令を同時にフェッチ出来るように，4 バンク (4 バイト/バンク) 構成で，64 バイト (=16 命令)/ラインである．命令ブロック単位の分岐予測を可能にするため，ライン当たり 4 つまでの分岐先アドレスを保持する分岐ターゲット・バッファ(BTB : Branch Target Buffer) を有する．

(b) 命令ブロック供給ユニット (IBSU : Instruction-Block Supply Unit) : 5 ステージ命令パイプラインにおける最初の命令ブロック・フェッチ (IF : Instruction-block Fetch) ステージに相当する．MBIC から連続する 4 個の命令を命令ブロックとしてフェッチし，4 本の命令パイプライン・ユニット (IPUs) にそれぞれ投入する．IPU に投入す

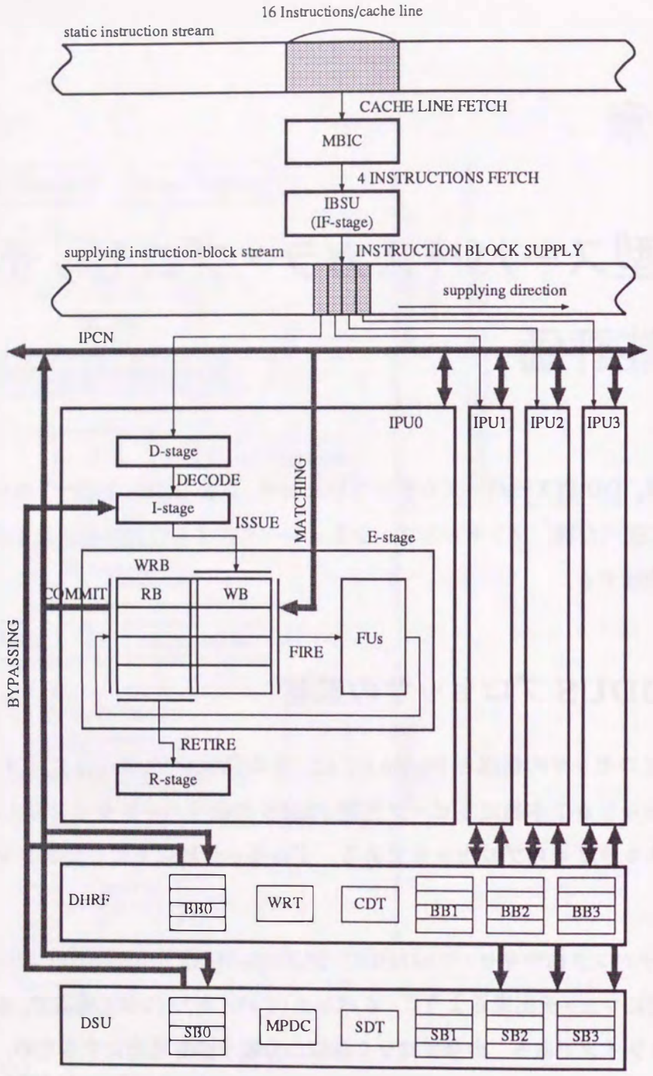


Figure 6.1 The Outline of DDU Superscalar Processor.

Table 6.1 The Specifications of DDU Superscalar Processor.

Instruction Length	32bit fixed length
Register	General Register : 48
	Floating-point Register : 48
	True/False Register : 32
	Supervisor Register : 16
Machine Cycle	60nsec (16.7MHz)
Pipeline	5 stages (2 cycle pipeline)
ALUs	Integer AMD Am29332 (Add/Subtract/Shift) AMD Am29C323(Multiply)
	Floating-point Weitek WTL2265(Add/Subtract/Conversion) WTL2264(Multiply)
Peak Performance	Integer : 48MIPS
	Single Floating-point : 32MFLOPS
	Double Floating-point : 16MFLOPS
Cache†	Instruction Cache : 512K bytes Data Cache : 512K byte

†Direct mapping and virtual Address Cache

る際は，IPU の使用効率を上げるために命令のアラインメントを行う．

(c) 命令パイプライン・ユニット (IPUs : Instruction Pipeline Units) : 4 本の同一構成の命令パイプライン・ユニットが備えられており，各々，命令パイプラインの 5 ステージのうちの残り 4 ステージを担当する．

- デコード (D : Decode) ステージ : 命令内にある種々のフィールドの解読を行う．
- 命令発行 (I : register-read and Issue) ステージ : 命令間の依存関係を解析して，ソース・レジスタの現在値を添えて実行ステージに対して命令の発行 (issue) を行う．
- 実行 (E : Execute) ステージ : 実行ステージには，整数 ALU(IALU : Integer ALU)，整数乗算器 (IMUL : Integer MULtiplier)，浮動小数点数 ALU*(FALU : Floating-point ALU)*，浮動小数点数乗算器 (FMUL : Floating-point MULtiplier)，および，データ・キャッシュ・アクセス・リクエスタ (DCAR : Data Cache Access Requester) と呼ぶ 5 つの機能ユニットを持つ．それぞれの機能ユニットはさら

に演算パイプライン化されており，全体として並列演算パイプラインを構成する．これらには，AMD 社の Am29323/32，Weitek 社の WTL2264/65 などのビルディング・ブロックを用いている．

また，機能ユニットの前後には 4 エントリの供給バッファ(WB : Waiting Buffer) および格納バッファ(RB : Reorder Buffer) と呼ぶ，対になって動作するバッファ (WRB : Waiting and Reorder Buffer) を備える．供給および格納バッファは命令パイプラインの乱れを緩衝すると共に，out-of-order 実行を行う上で重要な役割を担う．

- リタイア (R : register-write and Retire) ステージ : 命令ブロック内の全ての命令実行が終了したとき，それらの結果をレジスタやデータ・キャッシュに対して格納する．

(d) 依存解析機能付きレジスタ・ファイル (DHRF : Dependency-Handling Register File) : 読み出しポート 8，書き込みポート 4 のレジスタ・ファイルである．DDUS はロード/ストア・アーキテクチャであり，また命令パイプラインに投入される命令数が多いため，同時に多くのレジスタを使用すると考えられる．したがって，DDUS では比較的大容量な以下のレジスタ・ファイルを備える．

- 汎用レジスタ (GR : General Register)[48 本] : 整数論理演算やアドレッシングなどに用いる 32 ビット長のレジスタ．
- 浮動小数点レジスタ (FR : Floating-point Register)[48 本] : 浮動小数点数演算用の 32 ビット長レジスタ．
- True/False レジスタ (TF)[32 本] : 先行条件決定による条件分岐処理を実現するための 1 ビット長レジスタ．
- スーパバイザ・レジスタ (SR : Supervisor Register)[16 本] : スーパバイザ・モードにおいてのみ使用できる 32 ビット長レジスタ．

GR，FR，および，SR は，偶数番号と奇数番号のレジスタ 2 本を組み合わせて 64 ビット長レジスタとして使用できる．また，各レジスタは，それをデスティネーションとする命令の実行結果状態を保持する 4 ビットのコンディション・フィールドを備える．したがって，一般のプロセッサにおけるコンディション・コード (Condition Code) は存在しない．

依存解析機能とは，命令間の制御依存やデータ依存関係を検出して，各命令の実行制御情報 (局所データフローグラフ) を作成することであり，I ステージにおける処理の大部分はここで行う．データ依存関係を検出するために，レジスタ対応に書込み予約テーブル (WRT : Write Reservation Table) を設ける．また，制御依存関係や LOAD-After-STORE 依存関係を検出するために，制御依存テーブル (CDT : Control Dependency Table) およびストア予約テーブル (SRT : Store Reservation Table) をそれぞれ設ける．また，各 IPU 内の WRB 中の命令実行結果をソース・オペランドとしてバイパスできるように，バイパス・バッファ(BB : Bypass Buffer) と呼ぶ RB のコピーを設ける．

(e) データ供給機構 (DSU : Data Supply Unit) : データ・キャッシュへのアクセスを処理する．最大 4 つのロードおよびストア要求を受け付けられるように，4 ポートのマルチポート・データ・キャッシュ(MPDC : Multiple-Port Data Cache) を備える．MPDC はロード・ポート 4，ストア・ポート 4，4 バンク (4 バイト/バンク)，64 バイト/ライン構成である．DHRF 同様，各 IPU 内の WRB 中のストア・オペランドをロード・オペランドとしてバイパスできるように，ストア・バッファ(SB : Store Buffer) と呼ぶ RB のコピーを設ける．

(f) 命令パイプライン・チェイニング網 (IPCN : IPU Chaining Network) : IPCN は WRB 間を接続するための相互結合網であり，各 IPU がバス・マスタとなるバス 4 本から成る．各 IPU での命令実行結果は，種々のトークンとして全ての主要ユニット (IPUs，DHRF，DSU および IBSU) に送られる．

6.2 命令パイプライン処理過程

命令パイプラインの処理ステージは，Figure 6.1 に示すように，

- 命令ブロック・フェッチ (IF : Instruction-block Fetch)
- デコード (D : Decode)
- 命令発行 (I : register-read and Issue)
- 実行 (E : Execute)
- リタイア (R : register-write and Retire)

の5ステージから成る．DDUSは1サイクル60nsecの2サイクル・パイプラインであり，1ステージは120nsec周期で動作する．ただし，IF，D，IおよびRステージは2サイクル周期，発火制御，Eステージおよびコミット制御は1サイクル周期で処理する．

以下，各ステージにおける処理について述べる．

6.2.1 命令ブロック・フェッチ (IF) ステージ

マルチバンク命令キャッシュ(MBIC)および命令ブロック供給ユニット(IBSU)から構成される．命令ブロックを命令パイプライン・ユニット(IPU)へ投入する際，命令ブロックを識別するための命令ブロック番号(IBN : Instruction Block Number)とIPUを識別するためのIPU番号(IPN : IPU Number)とから成る命令識別子(IID : Instruction IDentifier)を各命令に付加し，以後この命令識別子により命令を管理する．

IBSUは，以下の3つの機構から成る[原89, 原90a]．

(a) 命令プリフェッチ機構(PIF : Pre-I-Fetch : Instruction prefetch)：命令ブロックのフェッチを行い，Dステージがインターロックしていなければ各命令パイプライン(IPU)に命令を投入する．IPUへ命令を投入する際，4.1節(1)で述べたミス・アラインメントの問題がある．DDUSプロセッサでは，可変フェッチ・バウンダリを採用し命令の並べ替えを行う．しかし，キャッシュ・ラインをまたぐ並べ替えは行わない．

(b) 分岐予測機構(BTP : Branch Target Prediction)：分岐命令の履歴に従って，次にフェッチすべき命令ブロックを決定する．分岐命令の過去の履歴は，分岐先アドレスと共にMBIC内部に設ける分岐先バッファ(BTB : Branch Target Buffer)に登録される．動的分岐予測と分岐先バッファの採用により分岐命令に起因するパイプラインの乱れを抑える．

(c) 命令再フェッチ機構(RIF : Re-I-Fetch : Instruction refetch)：分岐命令が終了後に行うパイプライン復元処理(repair)を行う．復元処理は真に無効にすべき命令のみを選んで無効にする選択的命令無効化を行う．選択的命令無効化を実現するために，パイプライン内にフェッチされた命令をすべて管理する命令ブロック・アドレス・テーブル(IBAT : Instruction Block Address Table)を備える．IBATの情報および分岐命令の実行結果である分岐トークンの情報から，命令無効化情報である制御トークンを作成する(5.3参照)．

6.2.2 デコード (D) ステージ

Dステージに投入された命令について，各IPUで同時かつ独立に以下の処理を行う．

(1) 命令の解読

(a) ステージ開始時に次ステージであるIステージからインターロックされていなければ，IBSUから命令を受け取りデコードを開始する．また，インターロックされている場合にはIFステージに対し命令が受け取れないことを知らせる．もし，命令が供給されない場合は，idle状態となる．

(b) ステージ終了時，もしIステージからインターロックされていなければ，デコード結果を送る．もし，インターロックされていれば送らない．

(2) 命令無効化チェック

選択的命令無効化を行う．分岐命令終了後に送られてくる制御トークンとデコード中の命令識別子(IID)を入力として，命令を無効にするか否かを判定する．もし，命令を無効にする場合は，Dステージの命令が無効化されたことを示す，命令有効ビットをリセット(=無効)する．

6.2.3 命令発行 (I) ステージ

Iステージの処理は依存解析機能付きレジスタファイル(DHRF)で行う．Dステージでの解読結果に従って，依存関係の解析を行う．そして，ソース・レジスタの現在値をレジスタ・ファイルあるいはバイパス・バッファ(BB)から読み出し，データおよび制御依存関係を表わす実行制御情報(5.3参照)を付加して，Eステージの供給バッファ(WB)に対し命令の“発行 *(issue)*”を行う．

(1) 依存解析機能付きレジスタ・ファイル

依存解析機能付きレジスタ・ファイル(DHRF)は，依存解析機構，バイパス・バッファ(BB)，レジスタ・ファイル(RF)およびオペランド・マッチング機構(MAT)から成る．また，依存解析機構には命令間の依存解析を行うために，書き込み予約テーブル(WRT)，

制御依存テーブル(CDT)およびストア命令予約テーブル(SRT)と言った依存解析テーブル(RT : Reservation Table)を備える．

DHRFの動作に対するトリガとしては，以下の4種類がある(5.3参照)．

(a) データ・トークン：各IPUの実行ステージ部において，命令がコミットされると，データ・トークンがIPCNを通じて到着する．これは，DDUSが2サイクルパイプラインのため，実行ステージ(E)における演算結果が1ステージ中に2回得られるからである．データ・トークンが到着したら，その中の演算結果(コミット・データ)をBBに書き込む．

(b) 制御トークン：毎サイクルの始め最大1個の制御トークンがIBSUから到着する．この中の命令無効化リスト(ISL : Instruction Squash List)に従って，BB内の該当する命令を無効化する．

(c) オペランド要求：各IPUのDステージ部からでデコード結果が毎サイクルの始め，最大4個到着する．これはソース・レジスタの読出しを要求する．さらに，デスティネーション・レジスタに値を書き込む命令の場合はWRT，分岐命令の場合はCDT，ストア命令の場合はSRTに対しそれぞれ予約を行う．

(d) リタイア要求：命令ブロック内の全ての命令がコミットしたとき，その命令ブロックは発行順にリタイアする．リタイア要求は，データ・トークンのうちステージの始めに到着する前着データ・トークンおよび制御トークンに基づいて，DHRF自身がリタイア可能な命令ブロックが存在するか判定して発生させる．

(2) バイパス・バッファ

バイパス・バッファ(BB)は各IPU対応に1個ずつあり，命令ブロック番号(IBN)をインデクスとするバッファである．エントリ内の各フィールドは格納バッファ(RB)内のフィールドとほぼ同じ構成をとる(Figure5.5参照)．BBに対する操作を以下に示す．

(a) 命令無効化：制御トークン内のISLにより，全エントリのU(Undo)ビットを更新する．

(b) コミット･データの格納：データ･トークン内のコミット命令識別子(CID)のIBNが指すエントリに対してコミット･データを書き込む．同時に該当エントリのX(eXecuted)およびC(Commited)ビットを設定する．

(c) オペランド要求：ソース・レジスタの現在値はRF，BBまたはこのステージに届くデータ・トークン内に存在する．BBからのソース・レジスタの読み出しは，現在有効な4エントリからデスティネーション・レジスタ番号と8個のソース・レジスタ番号とを総当たりで比較する．一致した命令のうち最も新しく発行されている命令の演算結果がソース・オペランドとなる．

(d) リタイア命令ブロックの決定および読み出し：最も先行する命令ブロックを管理するエントリのCあるいはUビットがすべてセットされていれば，当該命令ブロックをリタイアする．このとき，当該エントリの内容を読み出しておき，次サイクルでRFに反映する．

(e) 発行命令ブロックのBBへの登録：新たに発行される命令ブロックを先頭エントリに登録する．

(3) 依存関係解析テーブル

依存解析テーブル(RT)は1IPU当たり7エントリのリングテーブルであり(Figure 5.2参照)，4IPU分で1組である．WRTはレジスタ番号でアドレッシングされ、同時に4命令の依存関係を解析できるように1命令当たり2本の読み出しポート，計8本の読み出しポートを備える．書き込みポートは1ポートである．また，CDTおよびSRTは1本の書き込みおよび4本の読み出しポートを備える．RTへのアクセスは3種類ある．

(a) 発行する命令の予約：発行する命令ブロック内の命令について予約を行う．登録はテイル・ポインタで示されるエントリに対し行う．

(b) 依存関係表現リストの作成：指定されたソース・レジスタ番号によってアドレッシングされる全エントリを読み出したのち，命令発行順に従って並び替えを行う．

(c) リタイア命令に該当するエントリの消去：リタイア要求が発生した命令ブロックの命令に対応するエントリを消去する．

(4) レジスタ・ファイル

AMDのAm29334を16個使用し，64ビットデータの書き込み2，読み出し8の計12ポート・レジスタ・ファイルを構成する．64ビットデータを扱う場合には64個，32ビットデータとして扱う場合には128個のレジスタとなる．なお，レジスタには32ビットに

つき4ビットのタグが付随しており，演算結果のフラグ情報が記憶される．レジスタへのアクセスは1ステージ内に3回のアクセスが起こる．

(a) レジスタ・オペランドの読み出し：命令発行を行うソース・レジスタオペランドを読み出す．BBの場合と同様に，1命令当たり2ソース，4命令で8つの値の読み出しが必要なため8本の読み出しポートを備える．このアクセスは第2サイクルで行なわれる．

(b) リタイア・データのライト：リタイアする命令ブロック内の命令の結果は，レジスタ・ファイルに書き込まれる．レジスタ・ファイルのライト・ポートは2ポートしかないため2回に分けてライトしなければならない．このアクセスは第3，4サイクルで行う．

レジスタ・ファイルのバスは64ビットであるため，32ビットデータのアクセスの際はレジスタ・アドレスの奇遇番号によってアラインされる．

(5) オペランド・マッチング機構

新規に登録する命令ブロックの命令が必要とするレジスタ・オペランドはBB，RFおよびデータ・トークン中から最新値をフェッチすることは述べた．データ・トークンのうち，ステージ開始と同時に到着する前着データ・トークンの結果はすぐにBBに書き込まれ，その後オペランドのアクセスを行うので問題はない．ステージ途中に到着する後着のデータ・トークンの結果は，このオペランド・マッチング機構によって捕まえる．Iステージで作成したSSLおよびCDLを用いて，データ・トークン中の結果が最新値かどうかをチェックする．オペランド待ちにある命令側の命令識別子とデータ・トークン中の命令識別子との差によりインデクスされるSSLのビット位置が最尤フロー依存関係を表わしているとき，目的とするデータであることを示す．

以上のIF，D，Iステージは命令ブロック単位で120nsec毎にin-orderに処理される．

6.2.4 実行ステージ

発行された命令は，まず供給格納バッファ(WRB)に登録される．WRB中の命令は，Iステージで作成された実行制御情報に基づいて発火条件が整い次第，“発火*(fire)*”される．この発火順序はout-of-orderとなる．命令実行結果はWRBに一旦格納される．WRB

中の命令はコミット条件が整い次第，その実行結果をIPCN経由で全IPUおよびDHRFに送ることで“コミット*(commit)*”される．このコミット順序もout-of-orderである(5.3節参照)．命令の発火は60nsec毎に処理される．

(1) 供給・格納バッファ

供給・格納バッファ(WRB)は命令ブロック番号(BNO)をアドレスとする4エントリのリングバッファである(Figure 5.5参照)．WRBに対するアクセス要求は以下に挙げる6種類がある．

(a) 命令発行要求：Iステージから新しく発行される命令をWRBに登録する．登録はWRBの最尾エントリに空きがあるとき，テイル・ポインタが指すエントリに対し行うとともにテイル・ポインタを更新する．

(b) 制御トークン：制御トークンに従って，無効になった命令のエントリの無効化を行う．無効化の方法はDステージに用いられている無効化チェック回路と同じ方法により当該エントリを無効にする．

(c) データ・トークン：オペランド・マッチング機構において60nsec毎に送られてくるデータ・トークン中に目的とする先行命令から結果が含まれているかどうかを判別し，実フロー依存関係を解決する．オペランド・マッチング機構は前節のDHRF内にあるものと同じである．2つのソース・オペランドを捕まえるために，WBの1エントリについて2つのマッチング機構が存在する．

(d) 発火要求：命令が発火可能であるかどうかのチェックを行い，発火可能であれば機能ユニットに対して発火を行う．発火はWRBのソース・オペランドの実フロー依存が解決し，かつ，WRBのオペレーションフィールドと機能ユニットの現在の状態によって決まる．

(e) コミット要求：コミット機構によりコミット・チェックが行われる．コミットが決定した命令は直ちにIPCNを通じて全IPUに伝えられる．コミットには2つの方法がある．

(i) 演算の終了を待つ事なくすでにコミットすることがわかっている場合，その演算が終了後コミットすることが可能であれば，その結果はRB内のリザルト・フィールドに書き込むと共に，IPCNにも出力される．

(ii) (i) が行われない場合には，結果はRB内に保持される．コミット条件を満たした後，コミット可能であればIPCN上にトークンを出力する．

コミットした命令はトークンをIPCNに乗せることにより、命令の終了を知らせる．IPCNにはデータ・トークン，ストア・トークンおよび分岐トークンが流れるため，一番バス幅が大きいストア・トークン(命令識別子：5ビット，ストア・アドレス：32ビット，ストア・データ：32ビット，有効ビット：1ビット，計70ビット)に合わせてある．

(f) リタイア要求：リタイア要求はDHRFからステージの真ん中で送られてくる．このリタイア要求信号により先頭エントリに登録してある命令を抹消する．リタイアはステージの第4サイクルで行われ，操作としてはトップ・ポインタを更新するだけでよい．

(2) 機能ユニット

機能ユニットは，i) 整数ALU，ii) 整数乗算器，iii) 浮動小数点数ALU，iv) 浮動小数点数乗算器，および，v) データ・キャッシュ・アクセス・リクエスタから成る．命令発火のバスはバス構成により接続されており1サイクルには1つの命令のみが発火できる．これにより命令の発行がパイプライン・ピッチで行われること，WBの読み出しポート構造を簡単にする．機能ユニットからRBへのバスはマルチプレクスされているため，2つの演算器から同時期に演算終了した場合は固定優先順位により，優先度の高い演算を先に書き込む．

(3) データ供給機構

データ供給機構(DSU)はストア・バッファ(SB：Store Buffer)およびマルチポート・データ・キャッシュ(MPDC)から構成される．ロード/ストア命令は5.3.4節で述べたように，LOAD-after-STORE依存関係を保証するために，ロード命令のDSUへのアクセスは先行するストア命令がすべてコミットした後で要求される．SBおよびMPDCの動作の概要は以下の通りである[納富90]．

(a) ストア・バッファ：DHRFにおけるBBに相当する．SBの構成はBBにおいてのデスティネーション・レジスタ番号フィールドがストア・アドレスとなっている点が異

なる．すなわち，ストア命令がコミットしたとき，その情報がストア・トークンにより伝えられたストア・データおよびストア・アドレスがリタイアするまでSB内に保持される．後続のロード命令がSB内にあるデータをアクセスしようとした場合このSB内のデータをバイパスすることで高速なデータ受け渡しを可能にする．このためにSBはストア・アドレスによる連想メモリとなっており，同時に4つのロード命令のマッチングを取ることができる．SB内に最新値がない場合は，MPDC内からデータをロードする必要がある．

(b) マルチ・ポート・データ・キャッシュ(MPDC)：

MPDCには，1サイクル毎に最大4つのロード要求が，また，2サイクル毎にリタイア要求が到着する．このため，1サイクルの間に最大4つのライト要求(SBからリタイアされてきたストア命令)と，最大4つのリード要求(IPU内のコミット機構から送られてきたロード要求)に対処する必要がある．

SBにおいて，ストア・データのバイパスは行うが，必ずロード・オペランドのバイパスができるわけではなく，MPDCへのアクセスも必要となる．ロード要求が到着するとMPDCはそれをリード要求として取り扱い，データを読み出す．キャッシュのタグがヒットした場合にはデータ・アレイから読み出したデータをロード・オペランドとしてデータ・トークンにより送出する．ミス・ヒットした場合にはミス・ヒット処理を行った後にロード・オペランドを送出する．SBからのリタイア要求についてもライト要求として受け付け，同様に処理する．

ミス・ヒット処理には主記憶へのアクセスが必要であるが，データ供給機構内ではMPDCが主記憶へのアクセス(実際にはキャッシュが仮想アドレス・キャッシュでありアドレス変換が必要であるのでMMUへのアクセス)を受け持っている．

MPDCは2ステージ・パイプライン構成であり第1サイクルでタグ・アクセスとバンクの競合チェック，第2サイクルでバンクへのアクセスを行う．アクセス要求に対し，もしタグがヒットし，かつ，バンクの競合が起こらなければ，4つのロードあるいはストアを同時に処理可能である．

6.2.5 リタイア・ステージ

命令ブロックを構成する4命令がすべてコミットされたら，DHRF および MPDC において実行結果の格納を行い，その命令ブロックを命令パイプラインから“リタイア *(retire)*”させる．命令ブロック内の出力依存については，レジスタなどの更新を in-order とすることで保証する．また，リタイアの順序は命令ブロック単位で in-order に行う．

6.3 性能評価

6.3.1 目的

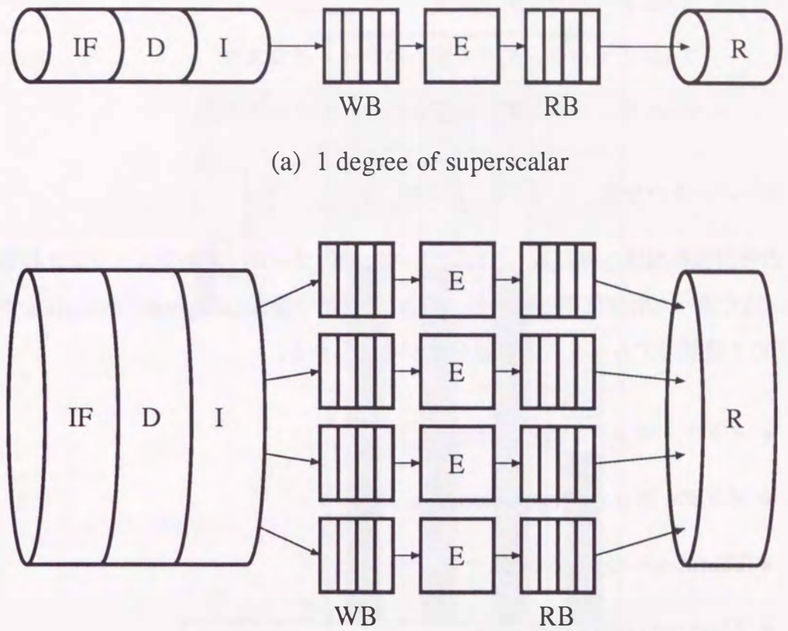
この評価の目的は，設計した DDUS プロセッサの性能評価にある．DDUS を設計するに当たっては，構成あるいは動的コード・スケジューリングなど多くの選択肢があるため，種々の選択肢をパラメータとして変化させたとき，処理性能はどのような変化を見せるかについても調べる．

6.3.2 評価方法

評価に使用したシミュレータは，ワークステーション上において C 言語で記述した．このシミュレータは DDUS のアセンブリ言語のソースを入力して1命令ずつ実行する．DDUS のシミュレーションだけでなく，パイプライン本数，動的コード・スケジューリング・アルゴリズム，分岐予測アルゴリズム，および，均質/非均質などをパラメータとして与えることが可能であり，種々のプロセッサ・モデルをシミュレーション可能である．

(1) 評価モデル

評価するマシンモデルは Figure 6.2に示すように，基本的に DDUS に基づいたモデルであり，命令フェッチ (IF)，デコード (D)，発行 (I)，実行 (E) およびリタイア (R) の5ステージから成る．実行ステージの前側にはオペランドの待ち合わせを行う供給バッファ (WB)，後側には reorder を行う格納バッファ(RB) を備える．WB および RB は各々が対になって動作する4段のバッファである．1パイプライン当たりの機能ユニットは，6.1節で述べたように，IALU，IMUL，FALU および FMUL の4つの演算器から成る．パイプラインは2サイクル・パイプラインとして動作するが，E ステージは1サイクル毎に動作



(a) 1 degree of superscalar

(b) 4 degrees of superscalar

Figure 6.2 Simulation Models.

する．すなわち，WB において発火制御のために1サイクル，RB においてコミット処理のために1サイクル必要とし，各演算器のコストは Table 5.1 に示すだけのサイクルを必要とする．パイプライン本数が4本の場合が DDUS に相当する．

(2) 評価パラメータ

シミュレーションのパラメータは以下の通りとした．

(a) 分岐命令への対処：

- 分岐予測：行う vs. 行わない
- 復元処理：フラッシュvs. 選択的命令無効化

また，4.1節で述べた，フェッチ・バウンダリに関して評価を行うために，以下の4種類について比較を行う [原 90a]．

– フェッチ・バウンダリ固定

– フェッチ・バウンダリ可変かつラインクロス可

– フェッチ・バウンダリ可変かつラインクロス不可

(b) スーパースカラ度：1，2，4，および，8

(c) 命令間依存関係への対処：3.1節で述べた動的コード・スケジューリング技術および4.4節で述べた制御依存への対処，および5.3節で提案したgreedy executionから，以下の7種類のアルゴリズムの組合せが考えられる．

- インターロック+lazy execution
- インターロック+eager execution
- Thornton+lazy execution
- Thornton+eager execution
- Tomasulo+lazy execution
- Tomasulo+eager execution
- Tomasulo+greedy execution

これらアルゴリズムの違いによる性能向上の差異を比較する．

(d) 供給格納バッファ(WRB)コスト：WRBにおける発火およびコミット遅延が発生しない場合vs.発生する(各1サイクル)場合．

(e) 使用ベンチマーク：bsort(バブルソート)，div(引き戻し法による除算)，max(最大値検索)，lfk5〜7(リバモアループ#5〜7番：単精度)，sieve(素数計算)，vsum(配列要素の加算：単精度)の8種類を用いる．いずれも，静的コード・スケジューリングは施していない．

上記以外の選択肢については，付録Aに示すDDUSの諸元どおりである．なお，命令およびデータ・キャッシュのヒット率は100%を仮定した．

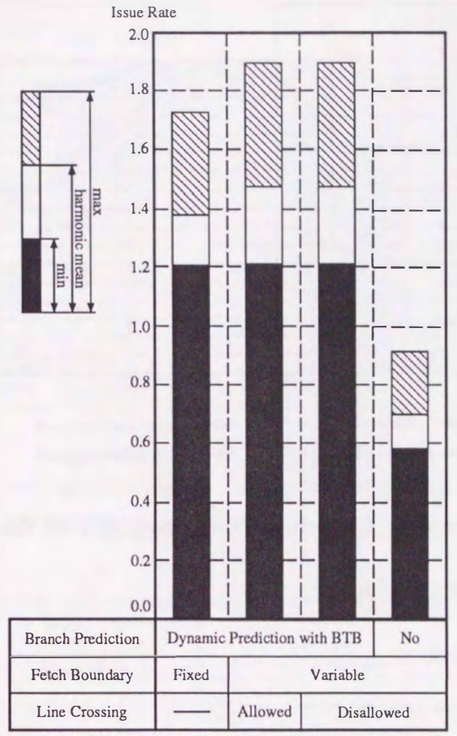


Figure 6.3　Issue rates of various fetch alternatives.

6.3.3　評価結果および考察

(1)　シミュレーション結果

Figure 6.3にフェッチ・バウンダリに関するシミュレーション結果を命令発行率 *(issue rate)*[1]で示す．また，Figure 6.4およびFigure 6.5に動的コード・スケジュールに関するシミュレーション結果を示す．これらは以下の評価項目と命令発行率または性能向上比との関係を表す．

- 多重命令供給への対処
- 分岐命令への対処

1 1サイクル当たりいくつの命令を発行できたかを示す．

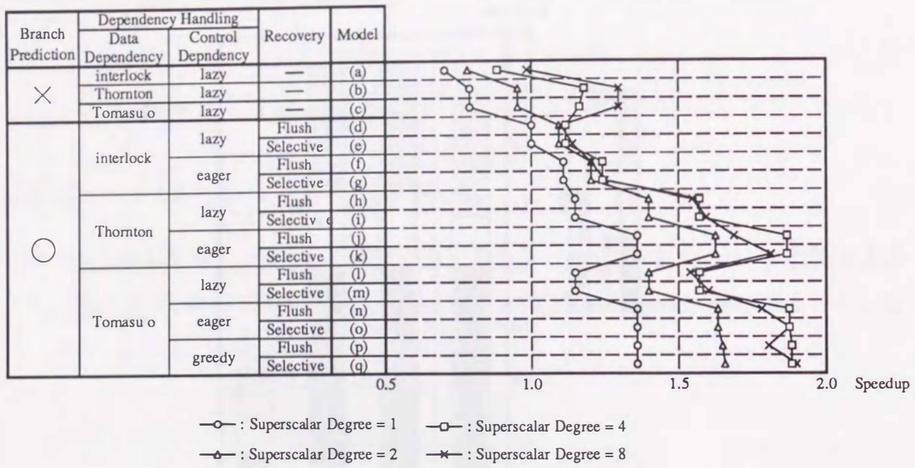


Figure 6.4　Speedups (incl. 1-cycle cost for WRB).

- 命令間依存関係への対処
- スーパースカラ度
- 供給格納バッファ(WRB)コストの有無

性能向上比については，標準的な単一命令パイプライン・プロセッサであるスーパースカラ度1のモデル(d)(Figure6.2(a)参照)を基準とした性能向上比で，全ベンチマーク・プログラムの相乗平均値(**geometric mean**)[2]である．

DDSUプロセッサはスーパースカラ度4のモデル(q)に相当する．Figure 6.4より，単一パイプライン・プロセッサ(モデル(d))に比べて1.88倍の性能向上が可能である．

(2)　パラメータに関する解析

(a) 多重命令供給への対処(Figure6.4参照)

この結果は，スーパースカラ度4，WRBコスト有りにおける評価であり，分岐予測を行っていないものはモデル(p)，行っているものはモデル(q)である．各方式に

2 正規化された実行時間の平均値は相乗平均= $\sqrt[n]{\prod_{i=1}^{n} ratio_i}$ で示す[**HePa90**].

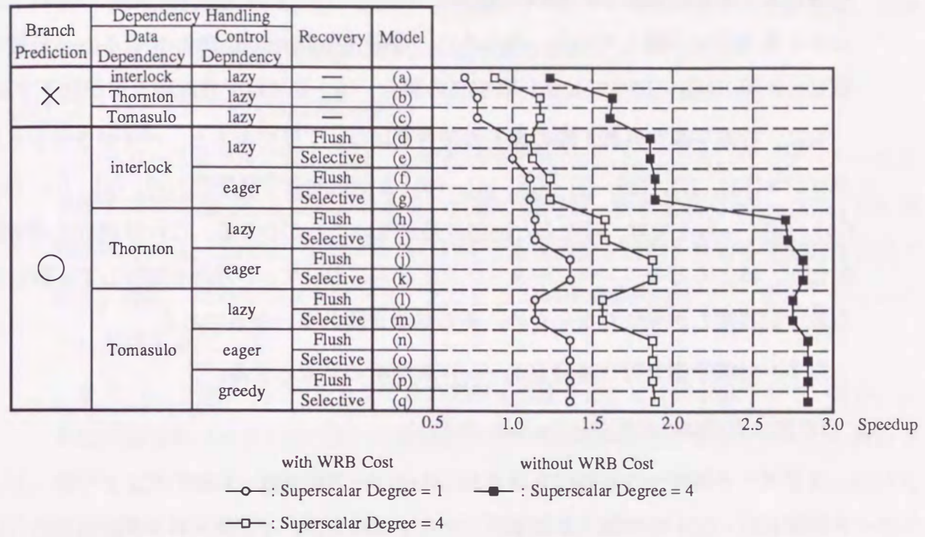


Figure 6.5　Speedups (excl. 1-cycle cost for WRB).

おいてベンチマークの結果の中から，最小値，最大値および調和平均値(harmonic mean)[3]で示す．

まず分岐予測を行わない場合，フェッチ・バウンダリ可変かつラインクロス不可であっても，分岐予測を行う場合に比べ命令発行率は50%弱低下する．フェッチ・バウンダリに関して固定の場合と可変の場合とでは，可変バウンダリの方が約9%性能が良い．またキャッシュのラインクロスの可，不可については，ほとんど性能差がないことがわかる．

この結果から，フェッチ・バウンダリを可変にする場合は，キャッシュのラインクロスが不可であっても十分な効果が得られることがわかる．

(b) 分岐命令への対処(Figure6.4参照)

まず，分岐予測の効果について見てみると，分岐予測を行う場合((d)〜(q))と行わない場合((a)〜(c))とでは，分岐予測を行う方が常に高い性能向上比を得ている．分

3 実行時間の逆数の平均値は調和平均= $\frac{n}{\sum_{i=1}^{n} \frac{1}{Rate_i}}$ で示す[**HePa90**].

岐予測ヒット率自体は90〜98%の範囲にあり，これは参考文献 [原 90a] の分岐予測ヒット率90%に一致している．ちなみに，参考文献 [SmithM89] によると，分岐予測ヒット率100%に比べてヒット率85%の場合，10〜45%程度性能向上比が低下する．

次に，分岐予測が外れた際の復元処理方式について見てみると，パイプライン・フラッシュ((d)，(f)，(h)，(j)，(l)，(n)，(p)) と選択的命令無効化 ((e)，(g)，(i)，(k)，(m)，(o)，(q)) とでは，ほとんど性能に差がないことがわかる．これは選択的命令無効化が，ほとんどのベンチマーク・プログラムにおいて，1万命令実行しても数回から数十回程度しか発生しないため，その効果が現れないからである．

以下，分岐予測を行う場合 ((d)〜(q)) を解析の対象とする．

(c) 命令間依存関係への対処 (Figure6.4参照)

まずデータ依存への対処法を見ると，スーパースカラ度4において，インターロック制御 ((d)〜(g)) の性能向上比は約1.1〜1.2であるが，インターロック制御以外 ((h)〜(q)) の性能向上比は1.5〜1.9である．つまり，後者は前者に比べて25〜70%程度性能がよい．参考文献 [SmithM89] には，20〜65%程度の差が見られる．

しかし，インターロック制御以外の方式 ((h)〜(q)) に関して，Thornton のアルゴリズム ((h)〜(k)) と Tomasulo のアルゴリズム ((l)〜(q)) を比較してみても，その差はほとんどない．一方，参考文献 [Weiss84] は，Cray-1 スカラ・ユニットにおいて，Tomasulo のアルゴリズムは Thornton のアルゴリズムに比べて 20%程度性能がよいと報告している．この違いはレジスタ数の違い[4]，すなわち，逆依存と出力依存の発生頻度の違いに起因するものと考えられる．

次に，インターロック制御以外の方式 ((h)〜(q)) に関して，制御依存への対処法の効果をスーパースカラ度4において見てみる．このとき，データ依存解消アルゴリズムに依らず，lazy execution((h)，(i)，(l)，(m)) の性能向上比は約1.5，eager/greedy execution((j)，(k)，(n)，(o)，(p)，(q)) の性能向上比は約1.8となっている．つまり，後者は前者に比べて20%程度性能がよい．しかし，eager execution((n)，(o)) と greedy execution((p)，(q)) との差はほとんどない．これは，greedy execution におけるコミット条件に問題があったためと考えている．すなわち，設定したコミット条件

4 DDUS は汎用および浮動小数点レジスタが各48個，Cray-1 はアドレスおよびスカラ・レジスタが各8個

がフーロ依存関係にある先行命令から後続命令へのデータ・バイパスを阻害し，後続命令の投機的実行の効果が得られなかったためである．

(d) スーパースカラ度 (Figure6.4参照)

スーパースカラ度に関しては，いずれのモデル ((a)〜(q)) においてもスーパースカラ度4でほぼ飽和することがわかる．同様の傾向が，参考文献 [Jouppi89b] でも報告されている．これは，プログラムに内在する並列度が性能を決める支配的な要因であり，特に，用いたベンチマーク・プログラムの平均並列度が2〜3程度しかないことに起因する．

また，6.4ではスーパースカラ度8の方が4の場合よりも性能が低下しているが，これは供給格納バッファ(WRB) の段数とリタイア処理に原因がある．スーパースカラ度が8にもなると，実行終了が遅い命令によって命令ブロックのリタイアが妨げられ，4段の WRB はすぐにオーバーフローを起こす．したがって，パイプラインがインターロックするため命令の実行が妨げられ性能が低下している．

(3) 考察

スーパースカラ・プロセッサの性能を評価したものには，参考文献 [SmithM89, Jouppi89b] などがある．前節で述べた DDUS の予測性能は，これらに比べて低い．その原因は，ベースモデル (6.3.2項参照) の設定そのものにあると考えている．これまでに明らかになったボトルネックは，以下のとおりである．

(a) リタイア・ボトルネック：正確な割込み/分岐を保証するために，RB (reorder buffer) を用いてレジスタ内容などの更新を in-order に行う．このレジスタ内容などの更新処理は命令パイプラインの最終段の R(リタイア) ステージに相当し，これをもって命令は命令パイプラインから抜け出ると同時に供給格納バッファ(WRB) を解放する．しかしながら，キュー動作を行う WRB の獲得・解放を容易に制御するため，リタイアは命令ブロック単位という制限が付く．よって，命令ブロック中に他よりも実行の遅い命令が存在すると WRB がなかなか解放されず，結局 WRB が一杯になって I(発行) ステージがインターロックされることになる．

(b) コミット・ボトルネック：greedy execution におけるコミット条件が，フロー依存関係にある先行命令から後続命令へのデータ・バイパスを阻害している．通常の eager

execution においては，先行命令はその実行結果を直ちに後続命令にバイパスできる．しかし，greedy execution においては，制御依存が解消されてからでないとバイパスできない．分岐命令が投機的実行される場合は問題ないが，6.3.2節で述べたようにベースモデルでは分岐命令の演算遅延が3サイクルもある．なお，今回の性能評価においては，eager execution にも greedy execution と同じコミット条件を適用している．よって，実際には eager execution の方が性能がよいと考えられる．

(c) WRB ボトルネック：Figure 6.4の結果は，供給バッファ(WB) における発火遅延1サイクル，および，格納バッファ(RB) におけるコミット遅延の1サイクルを演算遅延に加味した場合の値である．Figure 6.5に，これら WRB コストを無視した際のシミュレーション結果を示す．これから，発火遅延およびコミット遅延が演算遅延に隠蔽可能な場合，50〜80%程度の性能向上が期待できる．

(d) 分岐ボトルネック：(b) でも述べたように，分岐命令の投機的実行の可否が性能を左右する．DDUS では，その条件分岐方式として先行条件決定方式を採用しているので，分岐するか否かは早期に決定可能である．しかし，選択的命令無効化を採用しているので，分岐先アドレスを生成し無効化すべき命令を決めるまで，制御依存が解消されない．結局，これに3サイクルかかることになる．

6.4 まとめ

以上，DD 型スーパースカラ方式に基づき設計を行った，DDUS プロセッサについて述べた．DDUS のソフトウエア・シミュレータによる性能評価の結果，動的コード・スケジューリングを行うかなり重いハードウェアを導入しているにもかかわらず，性能向上はあまり大きくないことが判明した．これは特に reorder buffer による in-order なレジスタやメモリの更新方式，および，動的コード・スケジューリング・アルゴリズムの一部に問題があるためである．性能評価結果を踏まえ，これらの問題点に対処したスーパースカラ・プロセッサを次章以降に述べる．

第 7 章

DS 型スーパースカラ・プロセッサの設計

本章では，DS 型スーパースカラ方式を採用した試作プロセッサの開発および設計方針について，特に DD 型から DS 型へ設計を変更した背景を踏まえながら述べる．

7.1 DS 型スーパースカラの開発方針

7.1.1 DDUS プロセッサの開発方針

DDUS プロセッサの開発を始めた当初，以下の3点をその開発方針としたことは5.1項で述べた．

【DDUS の開発方針 1】 オブジェクト・コードの互換性

→動的ハザード解消スーパースカラとする．

【DDUS の開発方針 2】 ハードウェアによる高速化

→高度な動的コード・スケジューリング・アルゴリズムを実装する．

【DDUS の開発方針 3】 スーパースカラ度のスケーラビリティ

→同一構成の命令パイプラインを複数備えた均質型機能ユニットとする．

しかし，DDUS をソフトウェア・シミュレーションにより評価した結果，参考文献[SmithM89, Jouppi89b] において評価されたスーパースカラ・プロセッサの性能に比べて，DDUS の予測性能は低いものであることが判明した．その原因としては，動的コード・スケジューリングを行うためにハードウェアが複雑になりいくつかのボトルネックが生じたことが挙げられる [久我 90]．

7.1.2 開発方針の再検討

そこで，DDUS でのボトルネックを考慮し，開発方針自身から再検討を行った．

開発方針 1 まず，方針 1 のオブジェクト・コードの互換性は，1 章で述べたように，スーパースカラの対 VLIW 存在意義の 2 つの内の 1 つである．**これを放棄することは，現時点では考えない．**

開発方針 2 次に，方針 2 については，これを放棄する．すなわち，最適化コンパイラの存在を前提とし，最適化コードのみを対象とするようにする．

そもそも，静的および動的コード・スケジューリングは相互排他関係にないことから，両者を相補わせる目的で両アルゴリズムの開発を行ってきた．しかしながら，結果的には，コンパイル時に可能なコード・スケジューリングをハードウェアで行わせる必要がないことが判明した．たとえば，Tomasulo のアルゴリズム+投機的実行でもたらされる速度向上の大部分は，ループ・アンローリングにより達成可能である．

さらには，動的コード・スケジューリングを行うプロセッサを対象に静的コード・スケジューリングを行うことは，逆に最適化アルゴリズムを複雑にしてしまう弊害がある．プロセッサの動的振舞いがわからない，あるいは，動的振舞いがわかっていてもそれを考慮しなければならないからである．

よって，コンパイラによる静的コード・スケジューリングのみを用い，ハードウェアによる動的コード・スケジューリングは行わない方針を採る．例外的に，同時にフェッチされる命令ブロック内部でデータ依存関係のない命令については，その命令の発行をブロックしない．

静的コード・スケジューリングのみに頼った場合，スーパースカラ度 **n** 用の最適化コードは当然ながら多重度 **n'**($\neq$**n**) のプロセッサ上では最適に実行されない．もちろん，開発方針 1 により正常に実行できる．しかし，最適ではないので性能の保証は出来ない．性能を追求する場合，スーパースカラ度 **n'** 用に再最適化コンパイルするしかない．

開発方針 3 方針 3 については，次の 2 つの問題点がある．

- 命令レベル並列度は 2〜10 程度とあまり幅がなく，スケーラビリティがさほど要求されない．

- 均質型機能ユニットは非均質型機能ユニットに比べて，コスト / パフォーマンスが悪い．

いまや 100 万トランジスタの時代となり，将来の数千万トランジスタの時代を考えると，1 チップに複数のスーパースカラ・プロセッサ搭載も可能であり，DDUS プロセッサ開発当時考慮したパイプラインスライス・マイクロプロセッサという考え方は現実にそぐわなくなってきた．しかも，スーパースカラ度に比例して命令パイプライン間相互結合量が増加するので，命令パイプライン間相互結合をチップ間で行うのはきわめて困難でもある．

また，スーパースカラのアプリケーションを考えた場合，例えば数値計算の分野であれば，浮動小数点数演算器を重点的に多重化することで性能向上を図れる．つまり，スーパースカラの使用目的に最適な構成を決定することが，性能向上のために重要となる．

したがって，**開発方針 3 を放棄する．**

7.1.3 DSNS プロセッサの開発方針

以上をまとめて，DDUS を抜本的に改良した DSNS プロセッサの開発方針は次のように決定した．

【DSNS の開発方針 1】 オブジェクト・コードの互換性

当初の開発方針通り，スーパースカラ度の異なるプロセッサ間でもオブジェクト・コードが正常に動作するよう，コードの互換性を保証する．

【DSNS の開発方針 2】 最適化コンパイラによる高速化

コンパイル時の静的コード・スケジューリングにより，スーパースカラ度およびプロセッサ構成に応じた最適化コードを生成する．それには，基本ブロック内の局所コード・スケジューリングはもちろん，基本ブロックを越えた広域コード移動も行う．大前提として，コンパイラで対処できることはハードウェアで行わないこととする．その代わりハードウェアを軽量化してバランスのとれた構成にすることによって最適化コードの高速処理を実現する．このように，ソフト (コンパイラ) とハード (プロセッサ) を合わせたトータルでの性能向上を目指す．

【DSNS の開発方針 3】 シーズとニーズに応じたプロセッサ構成

使用できるVLSI技術，および，対象とするアプリケーションの性質に応じて，コスト/パフォーマンスが最良となるようにスーパースカラ度および機能ユニット構成を決定する．

これらの新開発方針に基づいて現在設計したのが，DSNSプロセッサである[原 90b]．すなわち，

- 新開発方針1に対しては，動的ハザード解消スーパースカラのままとする．
- 新開発方針2に対しては，静的コード・スケジューリング・アルゴリズムを最適化コンパイラに実装し，ハードウェアでは動的コード・スケジューリングを行わない．
- 新開発方針3に対しては，アプリケーション指向の機能ユニット構成，コスト/パフォーマンスなどを重視した非均質型機能ユニット構成とする．

となる．

以上の開発方針に基づいて試作するプロセッサを，以後，DSNS*(Dynamically-hazard-resolved, Statically-code-scheduled, Nonuniform Superscalar)*プロセッサと呼ぶことにする．

7.2 DSNSプロセッサの設計方針

7.1節で述べたDSNSの開発方針に従って，DDUSからDSNSプロセッサへ以下のように設計方針の変更を行った．

(1) 命令間依存関係への対処

動的コード・スケジューリング (Tomasuloのアルゴリズム+投機的実行)
→静的コード・スケジューリング (Thornton スコアボード+投機的実行)

DDUSでは，Tomasuloのアルゴリズムに基づくout-of-order実行制御を行っていた[久我 89a]．その実現のためには，

- WB(Waiting Buffer)：フロー依存解消のための待ち合わせ場所(reservation station)
- RB(Reorder Buffer)：正確な割込み/分岐を保証すると同時に，出力/逆依存解消の際のregister renamingにも用いるバッファ

が必要であった．WBとRBとを対にした供給格納バッファ(WRB : Waiting and Reorder Buffer)はキュー構造となっており，そのエントリは命令発行の対象となる．

しかし，高度なout-of-order実行制御を行うために備えたこのWRBの存在が，逆に命令実行を阻害する原因となっていた[久我 90]．さらに，WBはマッチング・メモリとして動作することから，ハードウェア構成が複雑となる．また，データ・バイパスのために，RBのコピーをレジスタ・ファイルおよびデータ・キャッシュに設けたが，やはりこれらもマッチング・メモリとして動作することから，WBと同様の問題が生じる．そこで，このWRB機能の削除を改良の第1目標として，データ依存への対処法をTomasuloのアルゴリズムからThorntonスコーアボードへと変更した．Thorntonスコアボードを採用する理由としては，フロー依存をできる限り早く解消したいからである．したがって，WRBに関しては以下のようにRBの機能のみ削除する．

- WB：フロー依存解消のための待ち合わせ場所：これについてはデータバイパスにおけるマッチング処理のために必要であるため削除しない．ただし，ハードウェア・コストを抑えるためにバッファ段数は0段，すなわち命令発行機構がWBの機能を備えるようにする．
- RB：出力/逆依存解消の際のregister renamingのためのバッファは削除する．

さて，“正確な分岐/割込みを保証する”ためのRBの機能の処置については，(5)(7)で述べる．

(2) 機能ユニットの均質性

均質型 (汎用多機能ユニット 4 個)→非均質型 (専用単機能ユニット 12 個)

新開発方針3で述べたように，コスト/パフォーマンス重視の非均質型機能ユニット構成にする．なお，スーパースカラ度は4のままとする．

(3) 多重命令供給

可変フェッチ・バウンダリ+ラインクロス不可→固定フェッチ・バウンダリ

分岐による命令ミスアラインメントは，コンパイラが分岐先を命令バウンダリに一致させることによって対処可能であるのでハードウェアでは対処しない．

(4)　分岐予測

動的分岐予測+BTB→静的分岐予測+BTB

数値計算の分野のように分岐に偏りがあるようなアプリケーションでは，コンパイラがプロファイルに基づいて行う静的分岐予測は，動的分岐予測の性能に比べ何ら遜色がない[SmithM90]．したがって，分岐予測もコンパイラで行う．また，taken 予測の場合，直ちに分岐先命令の供給を可能とするために分岐先バッファ(BTB ： Branch Target Buffer)を設けている．

(5)　正確な分岐機構

conditional mode+reorder buffer→conditional mode+多重化レジスタ・ファイル

DDUS では，分岐命令以降の命令も分岐予測パスからフェッチして，条件付実行モード(conditional mode) で実行することを可能としていた．ただし，その実行結果でレジスタないしメモリを直ちに更新せず，当該制御依存が解消するまで一旦 RB(reorder buffer) に実行結果を格納していた [久我 89a]．これにより，正確な分岐を保証していた．しかし，正確な割込みも同時に保証する必要から，RB からレジスタないしメモリへの実行結果の反映は，制御依存が解消された時点では直ちに行えない [Sohi87]．すなわち，reorder されて in-order となった命令実行終了順序に従って，レジスタないしメモリを更新する．したがって，最新のデータはレジスタやメモリではなく RB に存在する．フロー依存を早期に解消するためには，RB からの複雑なデータ・バイパス機構が必要であった．

DSNS プロセッサでは RB を削除したことにより，正確な分岐/割込みへの保証を新たに考える必要が生じた．この際に，正確な分岐を保証する機構と正確な割込みを保証する機構とを分離した．後者については，(7) で後述する．

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機構として，reorder buffer の代わりに同じバッファ方式の history buffer も考えられるが，条件付実行モード下の命令のオペランド・フェッチを行う場合の制御が複雑になりパイプラインの動作速度を落とす原因と成りかねないのでこれは採用しない．DSNS プロセッサでは，ハードウェア量は増えるが制御が比較的簡単な多重化レジスタ・ファイルを提案し採用する．これは，future file[SmithJ88] のレジスタ・アクセスパスの切り替えによる一実現方法である．

(6)　パイプライン復元処理

選択的命令無効化→パイプライン・フラッシュ

DDUS では，投機的命令実行の一種である greedy execution 方式を採用しているため，当該分岐命令以降に命令パイプラインに投入され得る命令数が最大 23 命令と多く，分岐予測が外れても正しい命令流が含まれている可能性があると考え，選択的命令無効化を採用していた [原 90a]．ところがシミュレーションの結果，実際には選択的命令無効化が有効である場合はほとんどないことが判明した [久我 90]．したがって，選択的命令無効化をやめパイプライン・フラッシュを採用する．

(7)　正確な割込み機構

reorder buffer→weakly-precise-interrupt

DSNS プロセッサも命令実行終了順序は out-of-order であるから，本質的に割込みが不正確になる可能性がある．DDUS では，reorder buffer による正確な割込みの保証方式 [SmithJ88] を採用していた [久我 89a]．しかし，DSNS プロセッサでは，既に (1)(5) で述べた改良により，reorder buffer に代わる保証方式が必要である．そこで，割込み方式自身を変更することにした．すなわち，不正確な割込みである (つまり，レジスタ内容などの out-of-order 更新を許す) が，プログラム再開を可能とする割込み (weakly-precise-interrupt)[HePa90] アーキテクチャを採用し，1 実現方式を提案する．

7.3　アーキテクチャ上の特長

7.3.1　動的ハザード解消

DSNS プロセッサは，動的ハザード解消・非均質型機能ユニット・スーパースカラ・プロセッサであるので，命令を発行する際に資源の競合およびデータ依存に起因するハザードの検出を行い解消する必要がある．本節では，資源の競合への対処と，データ依存への対処について述べる．データ依存に関してはレジスタ・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分けて述べる．また，制御依存に起因するハザードの解決については 7.3.2項で述べる．

(1) 資源の競合への対処

資源の競合によるハザードは，機能ユニット，レジスタ読み出しポート，および，レジスタ書き込みポートへの競合によって生じる．各命令は発行する前に使用する機能ユニットとレジスタ読み出しポートについて，他命令との間に競合が生じていないかを検出する．競合が生じている場合には先行命令を優先して発行し，後続命令はインターロック制御に基づきブロックされる．レジスタ書き込みポートについては命令の発行を行った後に割り当てを行う．

資源の競合に関して，同時に発行可能な命令の組合せは機能ユニットとレジスタ・ファイル間のデータ・パスの構成によって決定される．DSNS プロセッサのデータ・パス構成およびレジスタ・ファイルのポート割り当ては8章で述べる．

(2) レジスタ・アクセスに関するデータ依存への対処

DSNS プロセッサでは，Thornton スコアボードによりデータ依存に対処する．すなわち，先行命令に対して出力依存および逆依存関係にある命令の発行はブロックされる．フロー依存および出力依存はレジスタ・スコアボードにより検出する．逆依存は命令パイプライン構成上命令ブロック内については発生し得るため，レジスタ番号の比較を行うことで検出し解消する．発行をブロックされた命令を含む命令ブロック以降の後続命令ブロックは，インターロックされる．ただし，当該命令ブロック内の後続命令のうちデータ依存関係にない命令は，この限りではない．すなわち，“命令ブロック間は in-order 命令実行開始，命令ブロック内は out-of-order 命令実行開始” となる．命令実行終了順序は out-of-order である．フロー依存関係については，デコード・ステージにおいてバイパスされてくるデータを捕まえられるよう，レジスタ番号のマッチング機構を備える．

(3) メモリ・アクセスに関するデータ依存への対処

メモリ・アクセスを行うロード/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．ところが，ロード/ストア命令の比率は 30%～40% といわれており [HePa90]，DSNS プロセッサではスーパースカラ度 4 に対応して 2 個の互いに非同期動作するロード/ストア・ユニットを備える．したがって，同一サイクルに多重度数分のロード/ストア命令を実行できなければ 2 重化したことによる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．

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/ストア命令間の依存関係として問題になるものに，次の 3 種類がある．

- LAS(Load After Store) 依存関係 : フロー依存
- SAL(Store After Load) 依存関係 : 逆依存
- SAS(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，実行すべきロード/ストア命令の実効アドレスと，実行を終了していないすべての先行ロード/ストア命令の実効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ハードウェア・コスト的に非常に困難である．

そこで，DSNS プロセッサではメモリ・アクセスに関する依存関係の検出は，ハードウェアで一切行わず，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．

(a) Strongly Ordered(SO) : 完全逐次実行すべき命令である．静的な解析により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へのアクセスは in-order に行う．

(b) 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．

(c) Unordered(UO) : 完全非逐次実行可能な命令である．他の任意のロード/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．

7.3.2 分岐アーキテクチャ

4 章で挙げた対処法のうち，DSNS プロセッサでは分岐命令および制御依存への対処として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

分岐命令による命令フェッチの阻害に対処するため，DSNS プロセッサでは静的分岐予測を採用する．また，DDUS で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 : Branch Target Buffer) を，DSNS プロセッサでは静的分岐予測と組み合わせて用いる．BTB が保持する分岐先情報としては，分岐先の命令，あるいは，分岐先アドレスの 2 つの選択肢があるが，これに関しては分岐先アドレスを選択する．これは，命令キャッシュを備えていることから，分岐先アドレスさえ得られれば直ちに分岐先命令をフェッチできるからである．

静的分岐予測はコンパイラがプロファイル情報などに基づいた分岐予測を行い，この結果に分岐先アドレスに関する情報を加味して，個々の分岐命令に対して BTB 登録に関する情報を付加することで行う．この情報により，分岐命令は以下のいずれかの型に分類される．

(a) BTB 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指定する．実行時に生成した分岐先アドレスを BTB に登録する．

(b) BTB 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指定する．分岐先アドレスは BTB に登録しない．

分岐予測では，予測による分岐方向 (not-taken/taken) と分岐先アドレスの両者が分岐命令の実行結果のそれと一致しているときに予測が当たったといえる．分岐予測では分岐方向を予測することは可能であるが，分岐先アドレスが変化する場合にはもはや対応できない．したがって，分岐先アドレスが変化する分岐命令はたとえ taken と予測されても BTB 非登録型に指定する．

またこのことにより，分岐予測と実行結果について分岐方向の一致を調べるだけで分岐予測の当否が分かり，分岐先アドレスの比較を行う必要がない．よって，分岐予測が外れた際のパイプライン復元処理を早く開始できる．

分岐命令は実行されるとその型に従って BTB への登録/非登録を行う．BTB に分岐先アドレスが登録されると，その分岐命令は再度フェッチされた際に taken であると予測し，その登録分岐先アドレスを次サイクルの命令フェッチに使用する．

BTB の構成については 8.1.1 項で述べる．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 8.1.2 項 (1) で詳述する．

(2) 投機的実行 (多重化レジスタ・ファイル)

投機的実行方式として，4.4 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などを変更するのを防ぐ手段として，7.2 節 (5) で述べたように，DSNS プロセッサでは多重化レジスタ・ファイル [原 90b] を採用している．これは，future file[SmithJ88] 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，いくつの分岐命令を越えた命令の実行を許容するか，すなわち，条件付実行モードのレベル数によって，“条件付実行モードのレベル+1” と決定される．したがって，条件付実行モードレベル数の設定が必要である．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ので，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．また，レベル数を 2 以上にすることで大幅な性能向上を期待できるかどうかは疑わしい．したがって，条件付実行モードのレベル数は 1 とする．上記の問題点については早期分岐解消によって対処可能であるのでそちらに任せる．

このように 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．このことより，2 重化レジスタ・ファイル *(DRF : Dual Register File)* と呼ぶ (Figure 8.4 参照)．

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．

(a) 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．

- 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．
- 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．

(b) 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．

(c) 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．

- 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．
- 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) 先行条件決定方式

DDUS と同様，5.4.3項で述べた先行条件決定方式を採用する．これは，先行条件決定方式が5.4.3項 (6)で述べた理由に加えて，

- 分岐命令では TF レジスタ参照と分岐先アドレスの生成を行うだけで，その処理が少ないことから．したがって，“早期分岐解消” が可能となり分岐遅延自体を低減できる．

すなわち，後述する早期分岐解消方式の実現を容易にする特長を持つからである．

(4) 早期分岐解消

DSNS プロセッサでは1レベルの条件付実行モードのみ認識するため，条件付実行モードが解消されない間は，新たな分岐命令およびその後続命令の発行はブロックする．よって，分岐解消が遅れると多数の命令がブロックされ性能が低下する恐れがある．さらに，分岐予測が外れた場合の命令パイプライン復元処理をパイプライン・フラッシュ方式によって行っているため，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．

早期分岐解消は，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，余分なハードウェアが必要となるという問題が生じる．しかし，先行条件決定方式の採用により，分岐命令で行うべき処理が TF レジスタ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く，早期分岐解消を実現するのに向いている．

このような理由から，DSNS プロセッサには早期分岐解消 *(early branch resolution)* を採用している．分岐命令実行専用に分岐ユニットを設け，その命令パイプライン処理過程も他の一般命令とは独立にした．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．分岐命令の詳しい処理過程は 8.1.3 項で述べる．

7.3.3 IPRS(ImPrecise, but ReStartable) 割込み方式

DSNS プロセッサでは，命令実行終了順序は out-of-order であるから，本質的に割込みが不正確になる可能性がある．しかし，仮想記憶や IEEE 浮動小数点フォーマットをサポートしている場合，割り込み処理を行った後，再び割り込みを起こした命令，あるいは割り込みを起こした命令の次の命令から，処理が再開できなくてはならない．したがって，処理が再開できるように正確な割込みを保証する必要がある．DDUS では，reorder

buffer により正確な割込みを保証していた [久我 89a]．しかし，DSNS プロセッサでは，これまでに述べた改良によって reorder buffer を削除したため，これに代わる保証方式が必要である．そこで，割込み方式自身を変更することにした．

まず，通常の内部割込み方式における “正確な割込み (precise interrupt)” とは，次のように定義される [SmithJ88]．

(1) 退避したプログラム・カウンタ (PC) が指している命令より前の命令はすべて，その実行を正常に完了しており，かつ，マシン状態 (レジスタやメモリなど) を正しく更新している．

(2) 退避した PC が指している命令より後の命令はすべて，まだ実行されておらず，かつ，マシン状態を更新していない．

(3) 退避した PC の指している命令が実行を完了しているか否か，および，マシン状態を更新しているか否かは，アーキテクチャに依存する．

上記の定義を満たさないものを “不正確な割込み (imprecise interrupt)” と呼ぶ [SmithJ88]．

DSNS プロセッサでは，IPRS(ImPrecise, but ReStartable) 割込み方式を提案する (Figure 7.1)．これは，“不正確な割込みであるが，プログラム再開を可能とする割込み” 方式であり，out-of-order の命令終了順序でマシン状態を更新する可能性があるスーパースカラ・プロセッサ向きの割込み方式である．まず，退避される PC として，次の 3 種類を定義する．なお，以下の i[PC *] はそれぞれ，各 PC が指している命令を意味する．

- PCR(Program Counter/Restart) : プログラム再開点の命令を指す．
- PCU(Program Counter/Unexecuted) : 割込みの影響で実行を終了出来なかった命令で，i[PCR] 以外の命令を指す．
- PCI(Program Counter/Interrupt) : 内部割込みを起こした命令を指す．

PCR は 1 個，PCU は 0 個以上，そして，PCI は 1 個以上退避される．PCR と PCI が同一命令を指す場合もある．さて，“不正確な割込みであるが，プログラム再開を可能とする割込み” は，次のように定義できる．

(1)’ i[PCR] より前の命令で，i[PCU] および i[PCI] 以外の命令はすべて，その実行を正常に完了しており，かつ，マシン状態を正しく更新している．

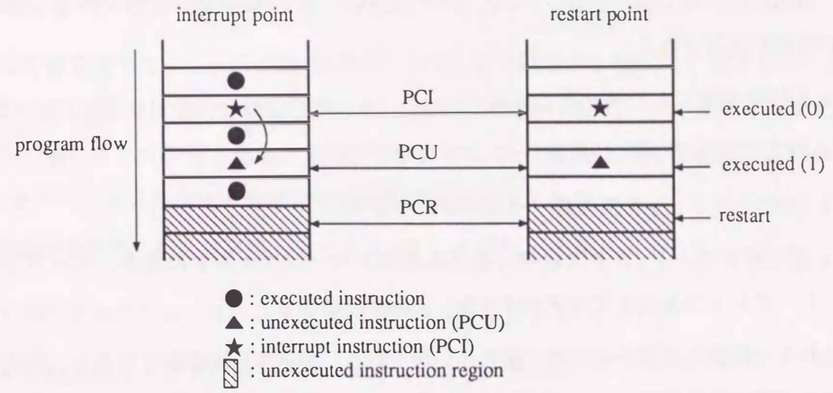


Figure 7.1　IPRS(ImPrecise, but ReStartable) Interruption.

(2)’ i[PCR] より後の命令はすべて，まだ実行されておらず，かつ，マシン状態を更新していない．

(3)’ 各 i[PCU] は実行を完了していない．

(3)” 各 i[PCI] が実行を完了しているか否か，および，マシン状態を更新しているか否かは，アーキテクチャに依存する．

(3)”’ i[PCR] が i[PCI] でない場合，それはまだ実行されておらず，かつ，マシン状態を更新していない．

割込み情報として，PCR，PCU，PCI 以外に，各 i[PCU] および各 i[PCI] に関する次の情報が退避される．

- i[PCI] が起こした割込みの原因
- i[PCU] および i[PCI] 自身
- i[PCU] のフェッチ済みのソース・オペランド
- i[PCI] がロード/ストア命令で割込み原因がページフォルトの場合，ページフォルト・アドレスとストア・データ (ストア命令の場合)

これらの情報を退避させた後，割込み処理を行う．割込み処理が終了すると，割込みハンドラが実行未終了であった i[PCU] および i[PCI] の実行を行った後，i[PCR] の命令アドレスを PC に設定することによって割込み処理後のプログラム再開が可能となる．

7.4 DSNS の ISP(命令セット・プロセッサ) アーキテクチャ

7.4.1 特長

DSNS プロセッサはスーパースカラ・プロセッサであるため，VLIW 方式ほどハードウェア構成を ISP アーキテクチャに反映させる必要はない．しかしながら，4.7節で述べた制約が ISP アーキテクチャに課せられる．

DSNS の命令は以下の特長を持つ．

- 32 ビットの単一命令長
- ロード/ストア・アーキテクチャ
- 固定サイクル演算
- 先行条件決定に基づく条件分岐命令
- コンパイラによる命令への情報付加 (分岐，ロード/ストア命令)

また，分岐およびロード/ストア命令において，コンパイラからの制御情報が付加可能であることが大きな特長である．これにより，静的コード・スケジューリングの恩恵を受けることができる．以下，分岐およびロード/ストア命令の仕様を述べる．なお，DSNS プロセッサの命令フォーマットおよび命令の一覧表を付録 B.4DSNS プロセッサの命令一覧にまとめる．

7.4.2 分岐命令の仕様

7.3.2 項で述べたように，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．

分岐命令のアドレッシング・モードは以下の 3 種類がある．

- PC(プログラム・カウンタ)+オフセット (符号付き 18 ビット)
- GR(汎用レジスタ)+オフセット (符号付き 13 ビット)
- PC+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．

7.4.3 ロード/ストア命令の仕様

DSNS では，i) 符号付き/符号なし 8 ビット整数，ii) 符号付き/符号なし 16 ビット整数，iii) 符号付き/符号なし 32 ビット整数，iv)IEEE フォーマット単精度浮動小数点数 (32 ビット)，v)IEEE フォーマット倍精度浮動小数点数 (64 ビット) のデータ・タイプを取り扱う．

DSNS プロセッサで採用している全ロード/ストア命令のニモニックを Table 7.1に示す．7.3.1 項 (3) で定義した分類に従って，Table 7.1のニモニックの "*" には以下の識別子が入る．

(a) 'so' : 完全逐次実行すべき命令であることを示す．

(b) 'wo' : 同一データ型同士で逐次実行すべき命令であることを示す．

(c) 'uo' : 完全非逐次実行可能な命令であることを示す．

また，ロード命令およびストアのアドレッシング・モードは以下の 2 種類である．

- GR+オフセット (符号付き 13 ビット)
- PC+オフセット (符号付き 13 ビット)

Table 7.1 Load/Store Instructions.

		Mnemonic	Type
integer	load	ld.sb.*	signed 8 bits load
		ld.sh.*	signed 16 bits load
		ld.w.*	signed 32 bits load
		ld.ub.*	unsigned 8 bits load
		ld.uh.*	unsigned 16 bits load
	store	st.b.*	8 bits store
		st.h.*	16 bits store
		st.w.*	32 bits store
floating point	load	fld.s.*	single precision load
		fld.d.*	double precision load
	store	fst.s.*	single precision store
		fst.d.*	double precision store

* : 'so' or 'wo' or 'uo'

7.5 まとめ

本節では，DS 型スーパースカラ方式を採用した試作プロセッサの開発方針および設計方針について述べた．試作プロセッサである DSNS プロセッサは，最適化コンパイラで対処できることはハードウェアで行わない代りに，ハードウェアを軽量化し最適化コードの高速処理を実現する．つまり，ソフトウェアとハードウェアとを組み合わせたトータルでの性能向上を目指す．以下に，DSNS プロセッサの特長をまとめる．

(a) 静的コード・スケジューリングの積極的な採用．

(b) 静的分岐予測 + 分岐先バッファの採用．

(c) 先行条件決定方式 + 早期分岐解消の採用．

(d) 2 重化レジスタ・ファイル + ブースティングによる投機的実行の採用．

(e) 静的スケジューリング可能なロード/ストア命令の採用．

(f) 不正確な割り込みであるが，再開可能な割り込み方式の採用．

第 8 章

DS 型スーパースカラ・プロセッサの構成と性能評価

本章では，DS 型スーパースカラ・プロセッサ (以後 DSNS と記す) の構成を述べ，一般演算命令の命令パイプライン処理過程，分岐パイプラインの構成および分岐命令処理過程，ロード/ストア・パイプラインの構成およびロード/ストア命令処理過程について述べる．さらにソフトウェア・シミュレータによる性能評価を行う．

8.1 DSNS プロセッサの構成

8.1.1 全体構成

DSNS プロセッサは Figure 8.1に示すように，以下の主要ユニットから構成される．また，DSNS プロセッサの諸元を Table 8.1に示す．

(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，その容量は 512K バイトである．

命令ブロック毎に 1 エントリ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．BTB の各エントリは以下のフィールドを有する．

- V(Valid) : 静的分岐予測値 (エントリの有効性)

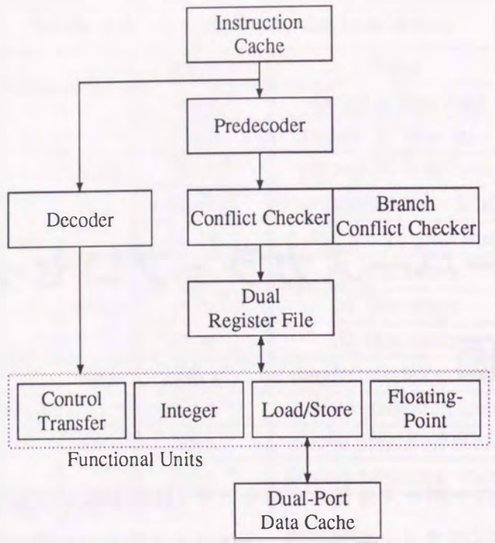


Figure 8.1 The Outline of DSN Superscalar Processor.

- BIA(Branch Instruction Address) : 登録した分岐命令の命令ブロック内アドレス (2bits)
- PTA(Predicted Target Address) : 分岐先命令のアドレス (30bits)

このBTB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．また，BTBは，命令キャッシュと連動しており，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，命令種類，発行対象機能ユニット，ソース・レジスタ番号，デスティネーション・レジスタ番号などコンフリクト・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．

Table 8.1 The Specifications of DSN Superscalar Processor.

Instruction Length	32bit fixed length
Register	General Register : 32
	Floating-point Register : 32
	True/False Register : 32
Machine Cycle	60nsec (16.7MHz)
Pipeline	4 stages (1 cycle pipeline)
ALUs	Integer AMD Am29332 (Add/Subtract/Shift) AMD Am29C323(Multiply)
	Floating-point Weitek WTL2265(Add/Subtract) WTL2264(Multiply/divide)
Peak Performance†	Integer : 48MIPS
	Single Floating-point : 32MFLOPS
	Double Floating-point : 16MFLOPS
Cache‡	Instruction Cache : 512K bytes Data Cache : 512K byte

†The peak performance is limited by the configurations of functional units and register ports.
‡Direct mapping and virtual address cache

(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・ファイルの読み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ステージ・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 エントリの分岐命令バッファ(BIB : Branch Instruction Buffer)を備える．詳細は 8.1.3 項で述べる．

(6) 機能ユニット (FUs : Functional Units)

Figure8.2に示すように，以下の 4 系統，計 12 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．

- 整数系機能ユニット : ALU(× 2)，シフタ，乗算器．
- 浮動小数点系機能ユニット : ALU，乗算器，除算器，型変換器．
- 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．
- 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機能ユニット (TF テスト・ユニット [条件決定] および TF 論理ユニット [TF 間論理演算])．

浮動小数点の除算器を除いて，すべての機能ユニットはパイプライン化されており，毎サイクル (ただし，浮動小数点の倍精度については 2 サイクル毎) 命令発行可能である．また，Figure 8.2には表示していないが，各機能ユニットからレジスタ読み出しポートへのバイパス・バスを設けている．これにより，フロー依存によりブロックされている命令へのソース・オペランドをレジスタを介すことなく (つまり 1 サイクル早く) 供給可能である．

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

Figure 8.2に示すように，以下の 3 系統のレジスタ・ファイルを備えており．いずれも 2 重化している．よって，以下のレジスタ個数は論理的なものである．

- 汎用レジスタ (GR : General Registr file) : 32 ビット長レジスタ 32 個から成る．読み出しポート 8(1 ポートは再フェッチ・アドレス生成器専用)，書き込みポート 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．
- 浮動小数点レジスタ (FR : Floating-point Register file) : 64 ビット長レジスタ 32 個から成る．読き出しポート 4，書み込みポート 2 を備える．GR 同様，各レジスタは 4 ビットのタグを持つ．

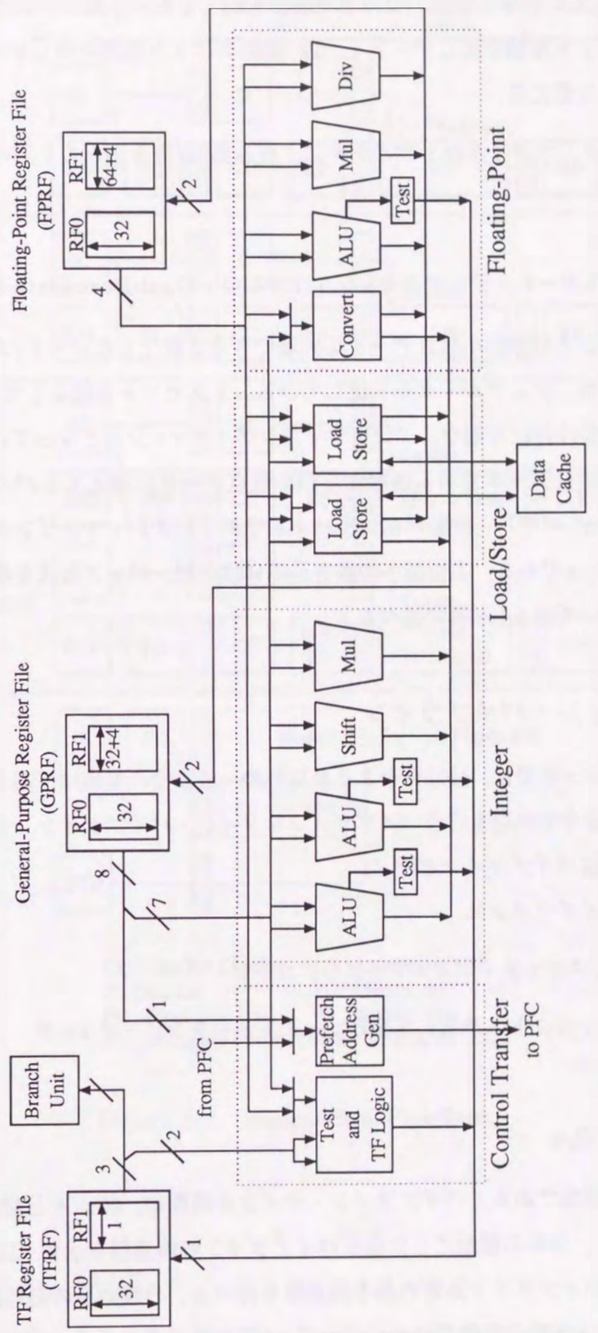


Figure 8.2 The Datapath of DSN Superscalar Processor.

- TF レジスタ (TF : True/False Register file) : 1 ビット長レジスタ 32 個から成る．分岐ユニット用読み出しポート 1，TF 論理ユニット用読み出しポート 2，書き込みポート 1 を備える．

2 重化レジスタ・ファイルは 1 サイクルに，読み出し/書き込みの 2 回のアクセスが可能である．

(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・ユニットに対応するため，デュアル・ポート化している．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．また，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．ライン・サイズが 32 バイト，容量が 512 K バイトでダイレクト・マッピング方式の仮想アドレス・キャッシュである．主記憶への書き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 8.1.4 項で述べる．

8.1.2 メイン・パイプライン

DSNS プロセッサでは，分岐命令とそれ以外の一般命令の処理は，別々のパイプラインで行う．一般命令の処理を行うパイプラインをメイン・パイプライン，分岐命令の処理を行うものを分岐パイプラインと呼ぶ．

メイン・パイプラインは，

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

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．

Figure 8.3に，命令の種類ごとに命令パイプライン処理過程を示す．以下，一般命令に関して，命令パイプライン処理の基本的過程を述べる．分岐命令の処理については 8.1.3 項，ロード/ストア命令の処理については 8.1.4 項で述べる．

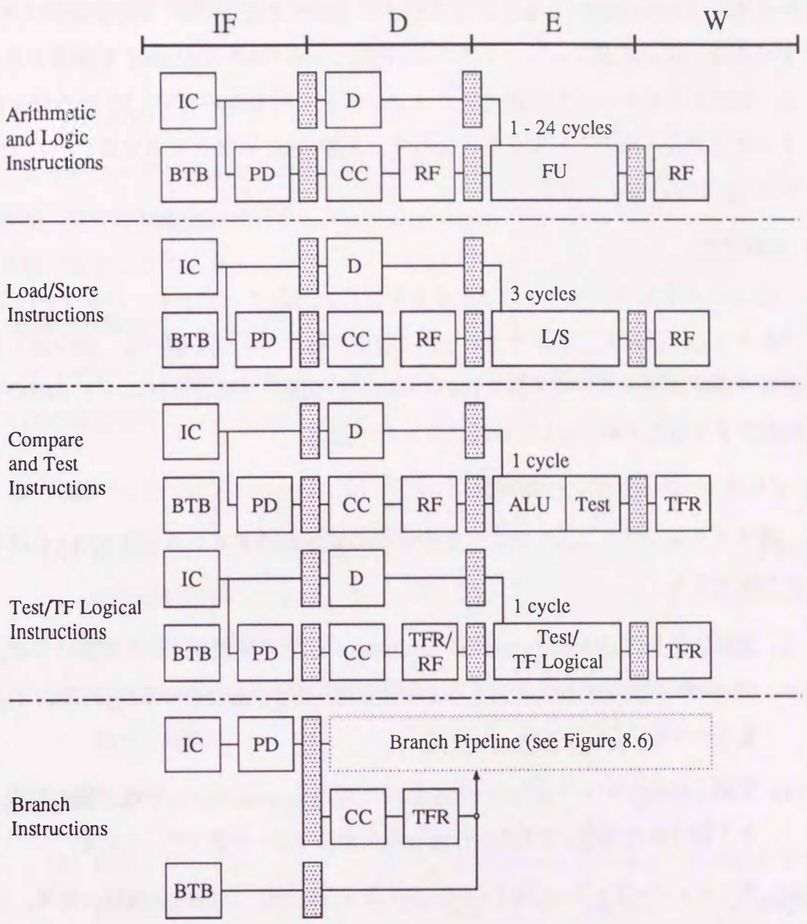


Figure 8.3 Instruction Pipelines.

(1) IF ステージ

(a) 命令ブロック・フェッチ

プリフェッチ・カウンタ (PFC : Pre-Fetch Counter) の値に基づいて，命令キャッ

シュ(IC : Instruction Cache) から連続する 4 命令を読み出す．PFC はプリフェッチすべき命令流の先頭ワード・アドレス (PFA : Pre-Fetch Address) を保持する．しかし，DSNS プロセッサでは命令アラインメントを行わないので，IC からは 4 命令バウンダリで読み出すことになる．よって，実際に IC にアクセスするワード・アドレスは $\lfloor PFC/4 \rfloor \times 4$ となる．

(b) 分岐予測

IC からの命令ブロック・フェッチと同時に，分岐ターゲット・バッファ(BTB) にアクセスし，読み出した命令ブロックに対応するエントリを調べる．読み出した静的分岐予測値に従って分岐予測を行い，taken と予測した場合にはエントリ内の予測分岐先アドレスを PFC および分岐ユニットへ送る．

(c) プリフェッチ・カウンタの更新

次サイクルでプリフェッチすべき命令流の先頭命令アドレス (次 PFA) を以下の中から決定する．

(i) 連続アドレス (SA : Sequential Address) : 通常の連続フェッチにおいては，次のフェッチ･バウンダリに相当する 4 命令を読み出す．したがって，$\lfloor (PFA+4)/4 \rfloor \times 4$ をもって次 PFA とする．

(ii) 予測分岐先アドレス (PTA : Predicted Target Address) : 分岐予測の結果 taken と予測された場合，BTB から送られて来る PTA を次 PFA とする．

(iii) 再フェッチ・アドレス (RFA : Re-Fetch Address) : 分岐命令実行の結果，分岐予測が外れて命令再フェッチが必要となった場合，再フェッチ・アドレス生成器から送られて来る RFA を次 PFA とする．

これらの優先順位は，高い方から (iii)→(ii)→(i) の順である．

(d) NOP 置換

命令アラインメントを行わないので，命令のミスアラインメントが生じる．また，選択的命令無効化を行わないので，分岐予測パスに含まれない命令を除去する必要がある．よって，命令ブロック内の以下の命令を NOP に置換する．

(i) PFA が 4 命令バウンダリにない場合 : 先頭命令から，PFA の指す命令の直前の命令までを NOP とする．

(ii) 静的分岐予測値が taken の場合 : 当該分岐命令より後の命令をすべて NOP とする．

(e) プリデコード

プリデコーダ (PD) は，命令ブロック内の 4 命令を同時にプリデコードして，命令種類，発行対象機能ユニット，ソース・レジスタ番号，デスティネーション・レジスタ番号などコンフリクト・チェック (D ステージ) に必要な情報を得る．

(f) 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．

(i) 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 :

(A) 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，(ii) の処理を行う．

(B) 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．

(ii) 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 :

(A) 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．

(B) 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 (i)(ii) 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．

(2) D ステージ

(a) データ依存検出

コンフリクト・チェッカ (CC) は，PD のプリデコード結果に基づいて，

- 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

を同時に行う．

(i) まず，命令ブロック内の命令間データ依存は，ソースおよびデスティネーション・レジスタ番号の総当りにより検出する．

- フロー依存：各命令のソース・レジスタ番号 (最大 2 個) と，その先行命令 (最大 3 命令) のデスティネーション・レジスタ番号とのマッチングをとる．一致するものがあれば，それがフロー依存となる．
- 出力依存：各命令のデスティネーション・レジスタ番号 (1 個) と，その先行命令 (最大 3 命令) のデスティネーション・レジスタ番号とのマッチングをとる．一致するものがあれば，それが出力依存となる．

(ii) 一方，先行命令ブロックに対するデータ依存は，2 重化レジスタ・ファイルの各々の論理的なレジスタに付加されたスコアボード (Figure 8.4参照) を用いて検出する．スコアボードは，カレント・レジスタおよびオルタネート・レジスタ対応に 1 ビットづつある．それぞれを CS(Current-register Scoreboard) および AS(Alternate-register Scoreboard) と呼ぶ．いずれも以下のように，同じ意味を持つ．

- 0：書き込み予約されていない．
- 1：書き込み予約されており，まだ演算結果が格納されていない．

CS および AS のいずれかが '1'，すなわち，いずれかのレジスタが書き込み予約されている場合，フロー依存ないし出力依存が存在することになる．

データ依存関係にある命令は，その発行をブロックされる．

(b) 機能ユニット競合検出・調停

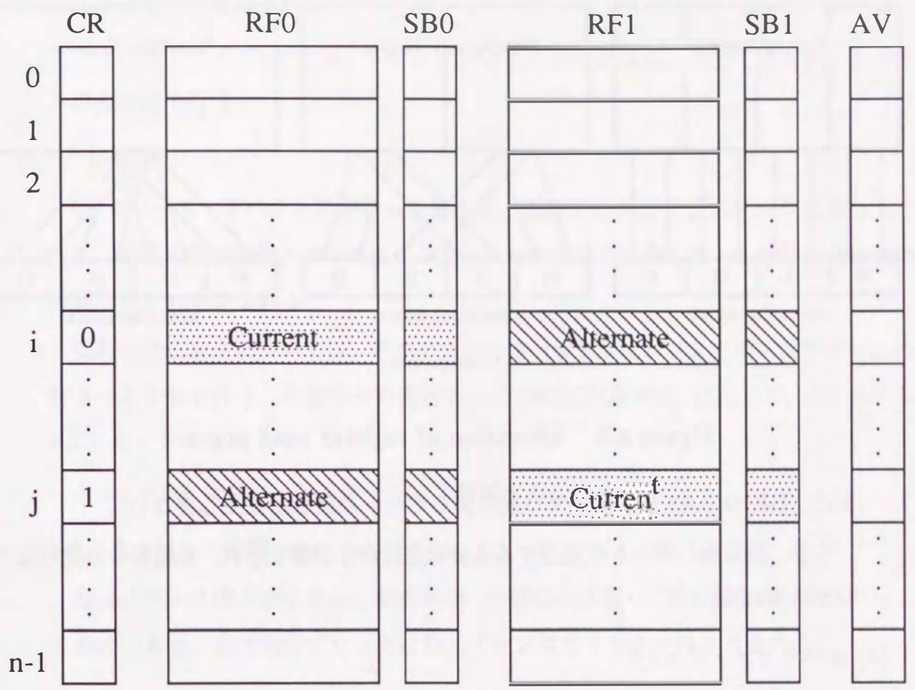


Figure 8.4 Dual Register File.

(a) でデータ依存関係にない命令について，使用する機能ユニットに競合があるか否かを調べる．競合がある場合は先行命令が優先され，後続命令の発行はブロックされる．

(c) レジスタ・ファイル読み出しポート割当て

(b) で機能ユニットを確保した命令に対して，レジスタ・ファイルの読み出しポートを割り当てる．各レジスタ・ファイルから一時に読み出し可能な組合せは Figure 8.5に示すように，各レジスタのポートが固定的に命令ブロック内の命令に割り当てられている．例えば FR に注目すると，偶および奇数スロットにある命令は同時に FR をアクセスできるが，偶数あるいは奇数スロット同士の命令は競合が発生する．

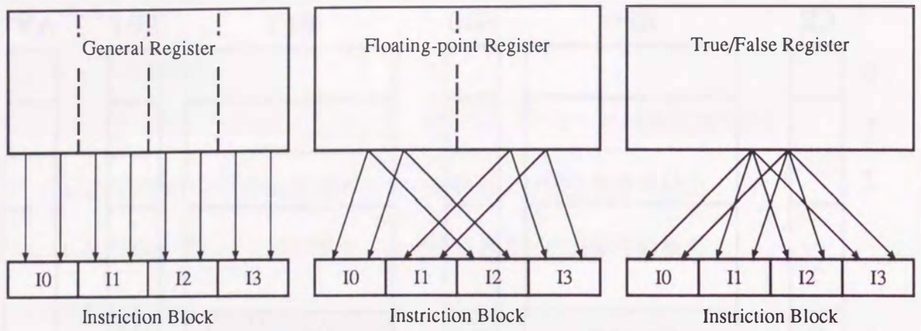


Figure 8.5　Allocation of register read ports.

また，GR のフェッチにおいては読み出しポートの競合が発生し得ない．

なお，読み出しポートが競合する場合は先行命令が優先され，後続命令の発行はブロックされる．

(d) オペランド・フェッチ

上記 (a)〜(c) の結果，命令発行をブロックされなかった命令は，レジスタ・ファイルからソース・オペランドを読み出す．このとき，カレント・レジスタおよびオルタネート・レジスタのどちらから読み出すかを決定する必要がある．

各々の論理的なレジスタには，2 ビットのスコアボード以外に，次の 2 ビットが用意している．

- AV(Alternate-register Valid) : オルタネート・レジスタが有効なデータを格納しているか (=1) 否か (=0) を示す．
- CR(Current Register) : 物理的に 2 個あるレジスタのうち，どちらが現在カレント・レジスタかを示す．

まず，無条件実行モードにある命令は，カレント・レジスタからオペランドを読み出す．一方，条件付実行モードにある命令は，

- AV=1 の場合 : オルタネート・レジスタ
- AV=0 の場合 : カレント・レジスタ

からオペランドを読み出す．

オペランド・フェッチは，レジスタ・ファイルの構造上，パイプライン・サイクルの後半部で行う．

(e) 命令発行

(c) でソース・オペランドが揃った命令は，次のサイクルで E ステージに進む．ただし，発行対象の機能ユニットにインターロック事象が発生している場合，当該命令の発行はブロックされる．

発行可能な命令については，そのデスティネーション・レジスタのスコアボードに書き込み予約を行う．当該命令の実行モード (無条件/条件付) に応じて，CS ないし AS をセットする．

なお，NOP 命令は，D ステージにて消滅する．

(f) インターロック制御

命令ブロック内のいずれか 1 個の命令 (分岐命令は除く) でも命令発行がブロックされていると，当該命令ブロックはDステージにてインターロックされる．

(3)　E ステージ

(a) 命令実行

一旦発行された命令は，各機能ユニットにおいて独立に実行を行う．その実行時間は，機能ユニットおよび命令の種類により 1〜24 サイクルと幅がある．Table 8.2に，各機能ユニットの発行間隔サイクル数と実行結果を得るまでの遅延サイクル数を示す．

(b) 格納バス競合検出・調停

各命令は，E ステージの最終サイクルになると，実行結果をデスティネーション・レジスタへ書き込むための格納バスの使用権を要求する．格納バスは，書込み先レジスタ・ファイルの書き込みポート対応に備える．格納バス数とそれを使用する機能ユニット数との関係は，以下の通りである．

- 汎用レジスタ (GR) : 格納バス 2 本 vs. 機能ユニット 8 個
- 浮動小数点レジスタ (FR) : 格納バス 2 本 vs. 機能ユニット 4 個
- TF レジスタ (TF) : 格納バス 1 本 vs. 機能ユニット 4 個.

Table 8.2 Issue and result latency of functional units.

Operational Unit	Instruction Type	Issue Latency (cycles)	Result Latency (cycles)
Integer Unit	Integer ALU	1	1
	Shifter	1	1
	Multiplier †	2	3
Floating-point Unit	Floating-point ALU	1/2 ‡	4/5 ‡
	Floating-point Multiplier	1/2 ‡	4/7 ‡
	Floating-point Divider †	13/24 ‡	13/24 ‡
	Converter	1/2 ‡	4/5 ‡
Load/Store Unit	Load	1	2
	Store	1	2
Control Transfer Unit	Test/TF logic	1	1
	Refetch Address Generator	1	1

† Not pipelined
‡ Single precision/Double precision

格納バスに競合が発生した場合，固定優先順位によって格納バスの割当を行う．

(c) インターロック制御

格納バスを確保した命令は，次のサイクルでWステージに進む．そうでない命令は，機能ユニットの最終段にてインターロックされる．

(4) Wステージ

Wステージに入った命令は，確保した格納バスを用いて，直ちに実行結果をデスティネーション・レジスタに書き込む．このとき，当該命令の実行モード (無条件/条件付) により，以下のように制御される．

(a) 無条件実行モードの場合：カレント・レジスタに書き込み，CSをクリアする．

(b) 条件付実行モードの場合：オルタネート・レジスタに書き込み，ASをクリアする．さらに，AVをセットして，オルタネート・レジスタを有効化する．

実行結果格納は，レジスタ・ファイルの構造上，パイプライン・サイクルの前半部で行う．

8.1.3 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．

(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．

(a) 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つの分岐処理のみ行う．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命令ずつ順番に実行してゆく．

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

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，分岐ユニットはデータ依存検出のために，分岐命令専用のコンフリクト・チェッカ (BCC : Branch Conflict Checker) を備える．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．

分岐命令のデータ依存の検出に用いるユニットは，以下の2つの場合では異なる．

D–BD ステージ同期：BD ステージで処理中の分岐命令の所属する命令ブロックがDステージに存在する．つまり，当該分岐命令は，一般命令用のDステージにも存在する．よって，データ依存の検出には，BCCではなくCCを用いる．

D–BD ステージ非同期：当該分岐命令の所属する命令ブロックは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．つまり，当該分岐命令は，一般命令用のDステージには存在しない．よって，データ依存の検出には，CCではなくBCCを用いる．

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・ファイル読み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・ファイルの読み出しポートの割当て

を行う必要がない．

(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．

(a) 分岐予測 (taken/not-taken)

(b) アドレッシング・モード (GR アクセスが必要か否か)

(c) 分岐命令の型 (BTB 登録型/非登録型)

(d) 分岐結果 (TAKEN/NOT-TAKEN)

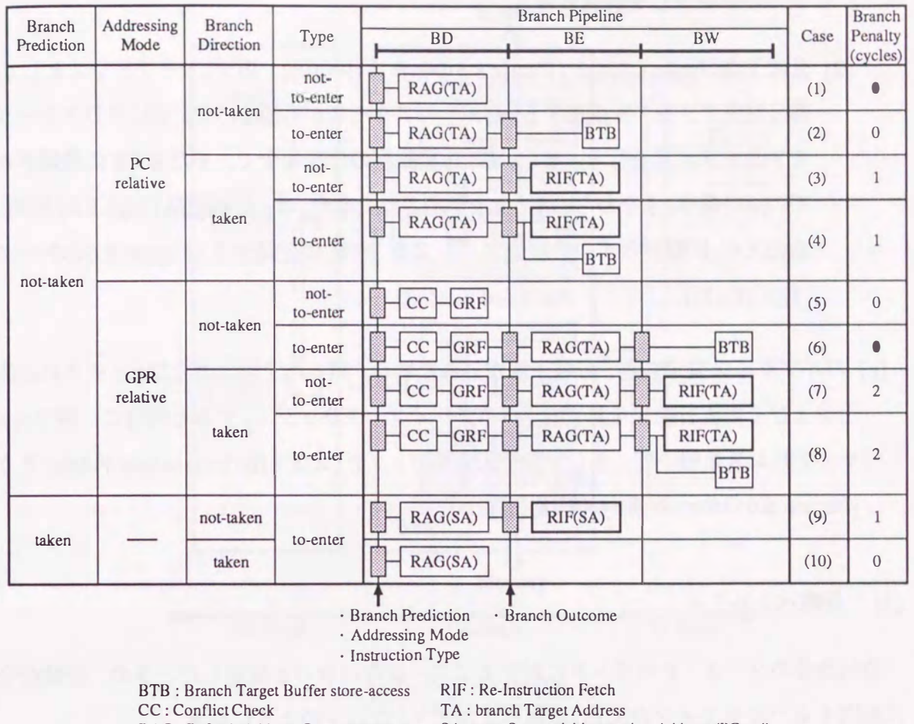
上記 (a)(b)(c) は IF ステージで，(d) は BD ステージで判明する．

Figure 8.6に分岐パイプラインの処理の概要を示す．

(a) 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 TF にフロー依存がない場合，TF 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．

(i) 分岐予測が当たった場合 (Figure 8.6のケース (1)(2)(5)(6)(10))：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，7.3.2 項 (2) 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・レジスタに切り換える．Figure 8.6 のケース (1)(2)(5)(10) は BD ステージで，ケース (6) は BE ステージで分岐ユニットを解放する．

(ii) 分岐予測が外れた場合 (Figure 8.6のケース (3)(4)(7)(8)(9))：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．



BTB : Branch Target Buffer store-access
CC : Conflict Check
RAG : Refetch Address Generate
RIF : Re-Instruction Fetch
TA : branch Target Address
SA : next Sequential instruction Address (PC + 4)

Figure 8.6 Branch Pipeline

(b) 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．

(i) 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．

(A) アドレッシング・モードが PC 相対の場合 (Figure 8.6のケース (1)〜(4))：BD ステージでアドレス計算を行う．

(B) アドレッシング・モードが GR 相対および PC + GR の場合 (Figure 8.6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.

(ii) 分岐予測がtakenの場合(Figure 8.6のケース(9)(10)): 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(Figure 8.6のケース(2)(4)(6)(8)).

(c) 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(Figure 8.6のケース(3)(4)(7)(8)(9)).

(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.

(a) 分岐予測ヒットの場合(Figure 8.6のケース(1)(2)(5)(6)(10)): ペナルティなし.

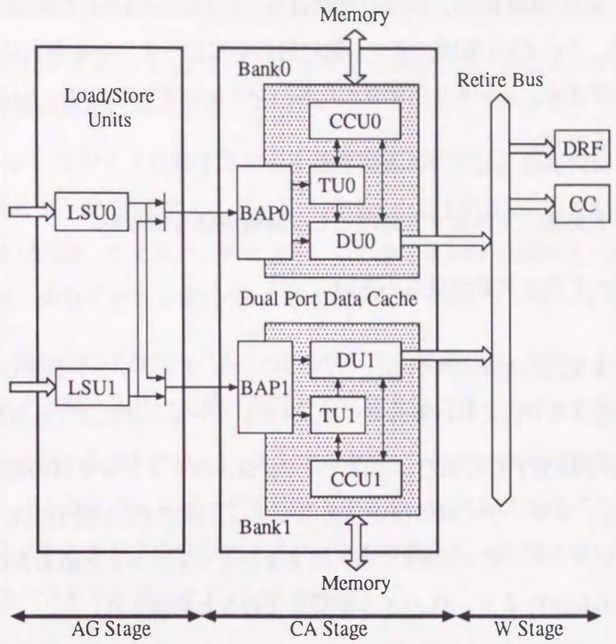
(b) PC相対で分岐予測ミス, あるいは, taken予測で分岐予測ミスの場合(Figure 8.6のケース(3)(4)(9)): 1サイクルのペナルティ.

(c) GR相対またはPC+GR, かつ, not-taken予測で分岐予測ミスの場合(Figure 8.6のケース(7)(8)): 2サイクルのペナルティとなる.

また, 制御依存は, TF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.

8.1.4 ロード/ストア・パイプライン

ロード/ストア命令の処理は, Dステージにおいてロード/ストア・ユニットへのディスパッチを行ったあと, ロード/ストア・ユニットからデュアルポート・データキャッシュへのアクセスを行う.



BAP : Bank Access Port
CC : Conflict Checker
CCU : Cache Control Unit
DRF : Dual Register File
DU : Data Unit
LSU : Load Store Unit
TU : Tag Unit

Figure 8.7 Load/Store Pipeline.

(1) ロード/ストア・パイプラインの構成

Figure 8.7に示すように, ロード/ストア・パイプラインは2つのロード/ストア・ユニットと以下に示すデュアルポート・データキャッシュから構成されており, その処理は他の一般命令のEステージに相当する.

(a) ロード/ストア・ユニット: メモリ・アクセスのための実行アドレスの生成を行う, 同一構成のロード/ストア・ユニットを2個備える.

(b) デュアルポート・データキャッシュ(DPDC: Dual-Port Data-Cache): シングル・ポートのメモリチップを用いてマルチポートのキャッシュを構築する構成方式には4つの方式

が存在するが [納富 91a]，DPDC の構成は ID/I(Interleaved, Dedicated/Interleaved) 型とした．この ID/I 型は，ポート数に対応してデータアレイおよびタグアレイをインタリーブさせ，インタリーブされた各バンクはいずれかのポートに占有させる構成である．これは，

(i) ヒット処理，ミスヒット処理ともに制御が容易である．

(ii) 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．一方，バンク・コンフリクトが生じやすいという問題点が存在するが，これは静的コード・スケジューリングによりある程度対処できる．

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 [納富 91a]．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 [Sohi90]，および，文献 [SmithJ87] より，32 バイトが妥当であると判断した．

Figure 8.7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．

(i) バンク・アクセス・ポート (BAP : Bank Access Port) : ロード/ストア・パイプラインからの要求を受け付ける．

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

(iii) データ・ユニット (DU : Data Unit) : 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．

(iv) キャッシュ・コントロール・ユニット (CCU : Cache Control Unit) : 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・レジスタへの書き込みの際には，格納バスにデータを乗せる．

(2) 通常処理過程

8.1.2 項で述べたメイン・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．

(a) 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 SO または 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 SO または 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 SO または 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 SO または 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．

(i) 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．

(ii) 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．

(b) 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．

(i) AG(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BAP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BAPを獲得できる．

(ii) CA(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，(3) ミスヒット処理過程で述べるように，使用可能なミスヒット情報を退避させるレジスタ(MSHR：Miss information/Status Holding Register[**Kroft81**])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．

(3) ミスヒット処理過程

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．

(a) アドレス：ミスヒットを起こしたアドレス

(b) 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

(c) 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

(d) wait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[**Kroft81**]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[**Sohi90**]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．しかし，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ので，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．

(a) 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，(b)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．

(b) 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[**Matick89**]．所望のデータをフェッチしたら，次のように処理を行う．

(i) ロード・アクセスの場合：当該データを格納バスに乗せる．

(ii) 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．

(c) 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(b)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．

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(b)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．

8.2 DSNSプロセッサの評価

本節では，ソフトウェア・シミュレーションによって行ったDSNSプロセッサの性能予測について述べる．

8.2.1 目的

8.2.2 シミュレーション・モデル

本シミュレータは，8.1.2項で述べたDSNSプロセッサの命令パイプライン処理過程を忠実にシミュレート可能である．ただし，以下に挙げる点が実機とは異なる．

- 命令キャッシュ：本シミュレータでは，プロセッサ外部へのアクセスを取り扱っていない．よって，命令コードはあらかじめ命令キャッシュ(容量：512Kバイト)内に存在するものとし，キャッシュのヒット率は100%を仮定している．
- データキャッシュ：DSNSプロセッサでは，最大2個のロード/ストア命令を同時に実行可能とするために，データキャッシュを2バンクにインタリーブしている[**納富91a**]．そして，2個の独立したロード/ストア・ユニット間でバンク・コンフリクトの調停を行う．しかし，本シミュレータは，バンク・コンフリクトは一切起きないものとしている．すなわち，2個のロード/ストア・ユニットに投入された各々のロード/ストア命令は，次のサイクルで必ずデータキャッシュにアクセスできるものと仮定している．なお，7.3.1項および7.4.3で述べたロード/ストア命令に付随する実行順序に関する情報については，8.1.4項(2)(a)で述べた手順に従っている．また，キャッシュのヒット率は，命令キャッシュ同様に100%を仮定している．

また，本シミュレータはDSNSプロセッサで採用したスーパースカラ度(=4)だけでなく，任意のスーパースカラ度での動作が設定可能である．しかし，機能ユニットの構成，

および，レジスタ・ファイルの読み出し/書き込みポート数などのデータ・パスについては，Figure 8.2に示した構成で■定である．機能ユニットの演算時間については，Table 8.2に基づく．

8.2.3 パラメータ

シミュレーションを行う上でのパラメータとして，まず，ハードウェアに関するものとしてスーパースカラ度を用いた．すなわち，次節で述べるベンチマーク・プログラムを，スーパースカラ度1，2，4，8の環境下でシミュレートした．

また，静的コード・スケジューリングの効果を調べる上で，各ベンチマーク・プログラムに対して次のようなスケジューリングを施した．

(a) 局所コード・スケジューリング(LCS)：すべてのベンチマーク・プログラムについて，局所コード・スケジューリングを施したものと，そうでないものの2種類を用意した．その際，ハードウェアのスーパースカラ度に合わせて，スケジューリングを行った．

(b) ループ・アンローリング(LU)：ループ・アンローリング可能なすべてのベンチマーク・プログラムについて，1回(LU1)および3回(LU3)アンローリングを施したものを用意した．

(c) ソフトウェア・パイプライニング(SP)：ソフトウェア・パイプライン化可能なベンチマーク・プログラムについて，ソフトウェア・パイプライニングを施した．さらに，ループ・アンローリング可能なものについては，1回ループを展開した後にさらにソフトウェア・パイプライニングも施したものを用意した．

8.2.4 ベンチマーク・プログラム

max(最大値検索)，2ddda(2次元平面上での直線描画アルゴリズム)，lfk1，3，5，7，12(リバモアループ：単精度)の7種類のベンチマーク・プログラムを使用した．

maxおよび2dddaはループ内部にIF文を含んでおり，ソフトウェア・パイプライン化に際しては3.2.3節で述べたハイアラーキカル・リダクション法を用いた．また，2ddda はループ・アンローリングが適用不可能であり，lfk5はソフトウェア・パイプライニングが適用不可能である．

8.2.5 評価結果および考察

Figure 8.8に，シミュレーション結果を示す．結果は，スーパースカラ度=1，最適化なしのベンチマーク・プログラムの実行に要した総サイクル数を基準とした性能向上比であり，その最小値，最大値，相乗平均値を示す．

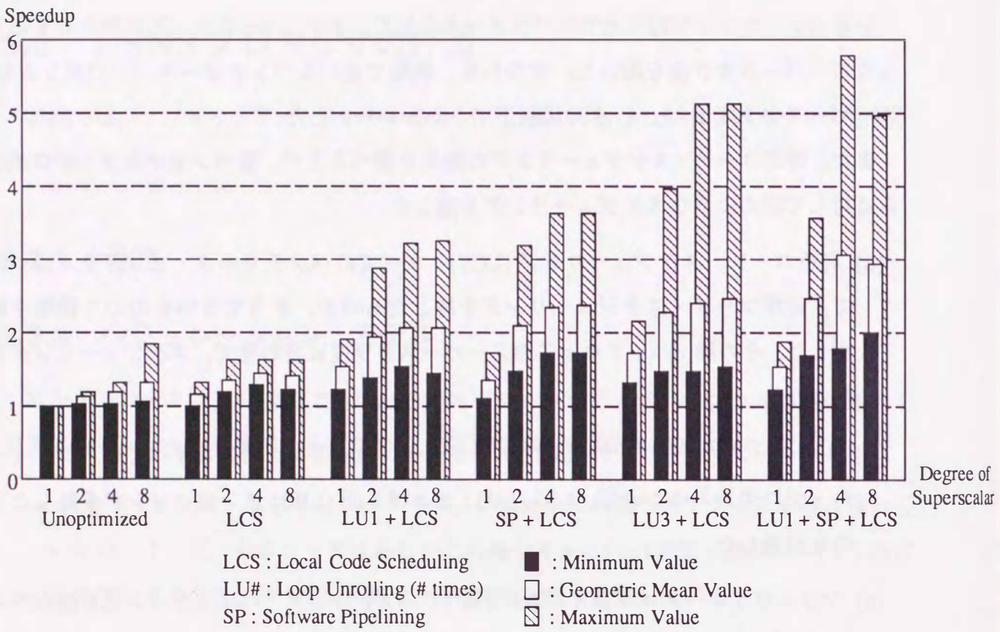


Figure 8.8 Speedups.

(1) スーパースカラ度と静的コード・スケジューリングの関係

まず，最適化なし，および，局所コード・スケジューリングのみを施した場合 (LCS) に注目すると，スーパースカラ度を上げても性能はあまり伸びていない．例えば，スーパースカラ度=8 の場合においても平均で 1.5 倍弱の性能向上しか得られない．これに対し，ループを 3 回アンローリングした場合 (LR3+LCS) には，スーパースカラ度が 1 の場合でも，平均して約 1.7 倍の性能向上が得られる．結局，ループ・アンローリングやソフトウェア・パイプライニングなどの強力なコード・スケジューリングを施さなければ，スーパースカラ度を大きくしても劇的な性能向上は得られないと言える．つまり，ループ最適化を行わないオブジェクト・コードには並列性があまり含まれていないことがわかる．

次に，各スーパースカラ度毎に静的コード・スケジューリングの効果を比較する．スーパースカラ度=1 の場合には，ループ・アンローリング，ソフトウェア・パイプライニングなどの手法を用いても，平均値の比較において最大 1.6 倍程度の伸びしか得られない．一方，スーパースカラ度が 4 の場合には，ループを 1 回アンローリングして，さらにソフトウェア・パイプライニングを施す (UL1+SP+LCS) ことにより，平均的に 3 倍強の性能向上が得られる．つまり，スーパースカラ度が大きいほど最適化の効果は顕著に現れると言える．

ただし，ループ・アンローリングを 1 回行い，ソフトウェア・パイプライニングを施した場合 (UL1+SP+LCS) には，スーパースカラ度が 4 から 8 に上がると逆に性能が低下している．この理由は以下の通りである．

- DSNS プロセッサでは，各機能ユニットからレジスタ・ファイルへの書き込みポートへの格納バスの調停を，機能ユニットに対して固定優先順に行っている．通常のプログラムは，データの読み込み→演算→データの書き込み，と進行することから，ロード/ストア・ユニットからの出力を最優先にしている．しかし，ソフトウェア・パイプライニングを施すと，むしろこの順序は逆になり，演算結果の書き込みが後続のロード命令の書き込みに妨害され，あまり性能向上が得られない．

- ループ本体のみを実行した結果を比較すると，スーパースカラ度が 4 であろうと 8 であろうと実行時間に差はない．しかし，スーパースカラ度が 8 の場合には，プロローグ部の命令の実行により，ループ内のある演算の出力が遅らされ，それがループ内で定常的に繰り返される，といった現象が生じている．格納バスの優先順位についてはさらに検討を行うとともに，局所コード・スケジューリングの際に対処する必要もある．

- DSNS プロセッサは，スーパースカラ度に対応した命令数に 1 つの BTB(分岐先バッファ) を設けている．つまり，スーパースカラ度 8 では，8 命令に 1 つの BTB エントリが割り当てられている．これに対して，分岐命令の存在確率は 20 数%といわれており 4 命令に 1 つが分岐命令となる．よって，BTB の奪い合いが生じて分岐予測のヒット率が低下する場合がある．

(2)　ループ・アンローリング vs. ソフトウェア・パイプライニング

スーパースカラ度=4 の場合に注目すると，ループを 1 回アンローリングしたもの (UL1+LCS) は局所コード・スケジューリングのみを施した場合 (LCS) に比べて，平均で約1.3倍の向上が認められる．しかし，ループを3回アンローリングしたもの (UR3+LCS) は約1.6倍の性能向上に留まっている．このように，ループ・アンローリングは，アンローリング回数に必ずしも性能向上が線形比例しない．ただし，ベンチマーク・プログラムによっては，3回のアンローリングで，局所コード・スケジューリングのみの場合 (LCS) に比べて性能が3倍弱になる例もある．

ソフトウェア・パイプライニングを施した場合 (SP+LCS) は平均的に，ループを3回アンローリングしたもの (UL3+LCS) とほぼ同程度の性能向上を得ている．ソフトウェア・パイプライニングとループ・アンローリングのいずれを適用すべきかは，対象となるループの特長に依存する．今回用いたベンチマーク・プログラムを調べてみると，以下のような特長がある．

- ループ・アンローリングが適しているもの：lfk5，lfk12

  lfk5 は，前のイタレーションの最後に定義した値を次のイタレーションの先頭で参照するようなベンチマークなので，パイプライン化ができない．よって，ループ・アンローリングしか適用できない．また，lfk12 は，イタレーション間に依存関係がないので，ループ・アンローリング，ソフトウェア・パイプライニングともに効果が期待できる．しかし，ループ本体が小さいため，ソフトウェア・パイプライニングを施しても同時に実行可能なイタレーション数が限られている．逆にループ本体が小さいという点を活かすと，各イタレーション毎に異なるレジスタを割り当てることができるので，ループ・アンローリングが適している．

- ソフトウェア・パイプライニングが適しているもの：lfk1，lfk7

  lfk1，lfk7 は，イタレーション間に依存関係がなく，かつ，ループ本体が比較的大きい．よって，ソフトウェア・パイプライニングを施した場合にオーバラップ可能なイタレーションの数は増え，その効果は大きい．逆にループ本体が大きいので，イタレーション毎にすべて異なるレジスタを割り付けることは困難であり，ループ・アンローリングの効果も頭打ちとなる．

8.3　まとめ

以上，DSNS プロセッサの構成および DSNS プロセッサの性能予測について述べた．

機能ユニットを多重化する場合，それに伴ってレジスタ・ポート数も増加させる必要があり，ハードウェア量およびネットワークの複雑化により実現は困難とみられる．また，それだけのハードウェアを用意しても実際に使用できるだけの並列性を引き出すには，高度な最適化コンパイラが必要である．ハードウェアの実現面およびコスト/パフォーマンスの面からみて，スーパースカラ度8は現実的ではなく，スーパースカラ度は4が妥当である．また，ハードウェア構成の観点からみると，DSNS プロセッサがコンパイラによる最適化コードを十分高速に実行できるだけのバランスのとれたハードウェア構成であることを示している．

DSNS プロセッサの特長である2重化レジスタ・ファイルなどについては，今回のシミュレーションではパラメータ化することができず，評価できなかった．早期分岐解消との兼ね合い，および，ハードウェア・コストの点から，2重化レジスタ・ファイルはその効果を正確に検証する必要があり，今後の課題である．

静的コード・スケジューリングについては，局所コード・スケジューリングや，ループ・アンローリング，ソフトウェア・パイプライニングといったループに対する技法の有効性を示すことができた．逆に，DSNS プロセッサの性能を最大限に引き出すためには，これらの技法を効率良く活用しなければならないといえる．DSNS プロセッサに適したループに対するスケジューリング技法の開発を急ぐ必要がある．

また，今回の評価では，ベンチマーク・プログラムとして数値計算を主体としたものを用いたため，ループ再構成によって並列性を引き出せたが，非数値系ベンチマークでは同じ手法で並列性を引き出すのは容易でない．よって，非数値系のプログラムに対してはパーコレーション・スケジューリングによって並列性を引き上げる必要がある．例えば，記号処理などの応用に対しても評価対象とする必要がある．