■6 群（コンピュータ･基礎理論とハードウェア）- 4 編（コンピュータアーキテクチャ（I）標準的）

8 章 ILP プロセッサ

（執筆者：塩谷亮太）［2011 年 9 月 受領］

■概要■

単一プロセッサの性能向上には命令レベル並列性の抽出が重要である．本章では，命令レベル並列性を利用する ILP プロセッサについて述べる．ILPP は大きくスーパスカラプロセッサ（Superscalar Processor）と VLIW プロセッサ（Very Large Instruction Word Processor）に分けることができる．スーパスカラプロセッサは，逐次処理されることを前提としたプログラムを読み込み，実行時にプロセッサ内で同時に実行できる命令を見つけてスケジューリングする．これに対して VLIW は，コンパイル時に同時に実行できる命令を見つけてスケジューリングし，プロセッサではそれをそのまま実行する．本章ではこれらプロセッサの概要と利害得失について述べた後，スーパスカラプロセッサによる動的な命令のスケジューリングについて解説する．

【本章の構成】

本章では，8-1 節で代表的な ILP プロセッサであるスーパスカラプロセッサと VLIW について，その概要と利害得失について述べる．続く 8-2 節では，スーパスカラプロセッサによる動的な命令のスケジューリングについて解説する．

■6 群 - 4 編 - 8 章

8-1 スーパスカラと VLIW

（執筆者：塩谷亮太）[2011 年 9 月 受領]

命令間に存在する並列性を利用し，命令単位で並列実行を行うプロセッサを ILP プロセッサと呼ぶ．この ILP プロセッサは，スーパスカラプロセッサ（Superscalar Processor）と VLIW プロセッサ（Very Large Instruction Word Processor）に大きく分けることができる．両者の大きな違いは，命令のスケジューリングをいつ誰が行うかという点にある．**図 8･1** に，スーパスカラプロセッサと VLIW におけるスケジューリングの担当の違いを示す．

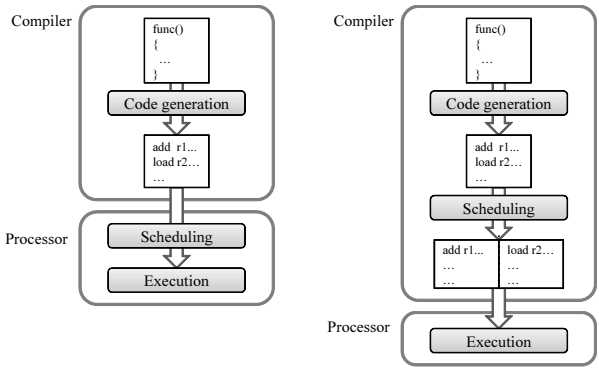

図 8･1　スケジューリングを行う担当の違い

スーパスカラプロセッサでは，プログラム内から同時に処理できる命令をプロセッサが実行時に見つけ，それらのうちのどれを実行するのかを決定する．このような実行時に行う命令のスケジューリングを動的スケジューリングと呼ぶ．

これに対して VLIW では，コンパイラが同時に処理できる命令を見つけ，並列に実行すべき命令の情報を含んだプログラムを出力する．プロセッサは与えられた命令をそのまま実行することで，命令を並列実行する．このようなコンパイラ時に行う命令のスケジューリングを静的スケジューリングと呼ぶ．

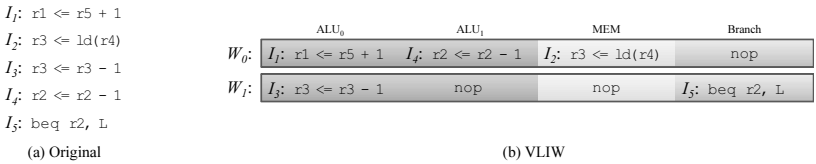

図 8･2　通常の逐次コードと VLIW のコード

図 8･2 に，通常のコードと VLIW のコードの違いを示す．図 8･2（a）は，通常の逐次実行されるコードである．スーパスカラプロセッサでは，これを読み込み，プロセッサ内で動的

に命令をスケジューリングする．これに対して VLIW では，図 8･2（b）のように，コンパイラが同時に実行する命令を並べてスケジューリングする．プロセッサが同時に実行できる命令の組合せは命令セットで定められ，その情報をもとにしてコンパイラは命令をスケジューリングする．同図では，ALU が 2 つ，メモリユニットが 1 つ，分岐ユニットが 1 つ，この順で並んでいるプロセッサを想定している．各ユニットで実行する命令がない場合は，図のように明示的に nop を挿入する．VLIW という名前は，このように同時に実行される命令のまとまりを 1 つの長い命令と考えたことから付けられている．

8-1-1 スーパスカラプロセッサと VLIW プロセッサの比較

スーパスカラプロセッサと VLIW は，スケジューリングをプロセッサとコンパイラのどちらが行うのかという点の違いにより，以下で述べる得失がある．

（a）ハードウェア量と複雑さ

一般に，スーパスカラプロセッサで必要となる動的なスケジューリングのためのハードウェアは複雑であり，チップ上で大きな面積を占める．これに対して VLIW では，静的にスケジューリングされた命令をそのまま実行するため，このようなハードウェアは不要である．このため，VLIW はスーパスカラプロセッサよりもハードウェア量が少なく単純であることが普通である．

（b）スケジューリング

動的なスケジューリングはハードウェアとして実装されるため，単純なアルゴリズムしか用いることができない．これに対して静的なスケジューリングでは，コンパイラによって十分な資源と時間を使えるため，より高い並列性を取り出すことができる．ただし，このことは必ずしも VLIW の優位さを示すものではない．なぜなら，コンパイラによる静的スケジューリングの効果は，スーパスカラプロセッサでも同様に受けることができるためである．静的スケジューリングと同様にして，並列に実行できる命令を動的スケジューリング可能な場所に置くことにより，スーパスカラプロセッサでもこれらの命令を並列実行することができる．

また，これとは逆に動的スケジューリングの優位な点として，実行時にのみ分かる情報の利用がある．このような例としては，例えばキャッシュミスへの対応がある．動的スケジューリングでは，ミスに応じて命令をスケジューリングし直すことができるため，ペナルティの影響を緩和できる．これに対して静的スケジューリングでは，このような動的に変化するレイテンシに対応することは難しく，ミスによるペナルティを隠蔽することは難しい．

（c）互換性

スーパスカラプロセッサでは，プロセッサの構成が変化した場合でもバイナリを変更することなくプログラムを利用することができる．これに対して VLIW では，プロセッサの構成が変化した場合には，それに合わせて再コンパイルが必要となる．このバイナリ互換性は，商業的に極めて重要であり，スーパスカラプロセッサの非常に有利な点となっている．

より詳細な比較や，VLIW そのもの，そこで用いられる静的スケジューリングの詳細については，例えば文献[1)]を参照されたい．

■参考文献

1) 安藤秀樹：“命令レベル並列処理 – プロセッサアーキテクチャとコンパイラ –,” コロナ社, 2005.

■6 群 - 4 編 - 8 章

8-2 スーパスカラ

（執筆者：塩谷亮太）[2011 年 9 月 受領]

プログラムオーダ（6 群 5 編 1 章）上で先行する命令 I_{pred} と後続の命令 I_{succ} を考える．複数の命令を同時に処理する ILP プロセッサでは，I_{pred} と I_{succ} が同時に命令パイプライン上の同じステージに進むことがある．また，プロセッサによっては，I_{succ} が I_{pred} を追い越して先に下流のステージに進む場合がある．このように I_{succ} が I_{pred} を追い越して下流のステージに進んでいるとき，これらの 2 命令は out-of-order に処理されているという．逆に，I_{succ} が I_{pred} よりも上流か，あるいは同じステージにあって追い越しが起こっていないとき，これらの 2 命令は in-order に処理されているという．

処理が out-of-order になることを許すスーパスカラプロセッサを out-of-order スーパスカラプロセッサ，許さないスーパスカラプロセッサを in-order スーパスカラプロセッサという．また，単にスーパスカラプロセッサと言った場合は out-of-order スーパスカラプロセッサを指すことが多い．本節ではこの out-of-order スーパスカラプロセッサの主に命令スケジューリングの方法について解説する．

8-2-1 構成

スーパスカラプロセッサの構造は，命令スケジューリングを行う前と後のどちらでレジスタを読み出すかによって，大きく 2 つに分けられる．本節では，スケジューリング後にレジスタ読み出しを行う方式を前提として解説するが，両者の基本原理は大きくは変わらない*．

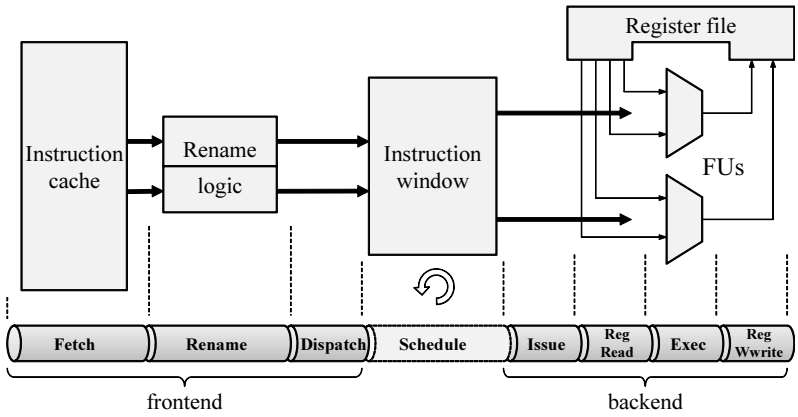

図 8·3　スーパスカラプロセッサの構造と命令パイプライン

スーパスカラプロセッサは命令のスケジューリングを行うための論理的なバッファを持つ．フェッチ（fetch）された命令はいったんこのバッファに格納された後，このバッファに格納さ

* データパスのコンパクトさなどにより，近年では後者の方式が主流となりつつある．スケジューリング前にレジスタを読み出す方式については，文献[1)]を参照されたい．

れている命令がスケジューリングの対象となる．すべての命令の中で現在見えている部分がスケジューリングの対象となるため，このバッファは命令ウィンドウ（Instruction Window）と呼ばれる．

図 8･3 に，スーパスカラプロセッサの基本的な構造と命令パイプラインを示す．スーパスカラプロセッサの命令パイプラインは，命令ウィンドウを境にして上流と下流に分離されている．本節では，命令ウィンドウの上流をフロントエンド（Frontend），下流をバックエンド（Backend）と呼ぶことにする．フロントエンドとバックエンドは，それぞれが独立したパイプラインとなっており，命令ウィンドウによって緩く結合された構造をとる．命令は，フロントエンドにおいてフェッチされ，命令ウィンドウに格納された後，スケジューリングされてバックエンドに送り出される．以降では，これらの各処理について順に説明する．

8-2-2 フロントエンド

フロントエンドは，命令間にある偽の依存を取り除き，それが済んだ命令を命令ウィンドウに供給するためのパイプラインである．フロントエンドの処理は，主に以下の 3 つのフェーズからなる．

1. **フェッチ**（Fetch） まず，命令キャッシュから命令が読み出される．このことをフェッチするという．命令の読み出しは，分岐予測器を用いて投機的に行われる．分岐予測器を用いた投機の詳細については，5 編 1 章 1-1 節を参照されたい．

2. **リネーム**（Rename） レジスタリネーミングと呼ぶ処理により，偽の依存である逆依存や出力依存を取り除く．詳細については後述する．

3. **ディスパッチ**（Dispatch）
   レジスタリネーミングの済んだ命令を命令ウィンドウに書き込む．このことをディスパッチするという．

命令は，フェッチされてから命令ウィンドウへディスパッチされるまで in-order に処理される．

(1) レジスタリネーミング

スーパスカラプロセッサではレジスタリネーミングと呼ぶ処理により，偽の依存である逆依存や出力依存を取り除く．一般に，プログラム中には多くの逆依存や出力依存が含まれているため，効率良く命令スケジューリングを行うためには，これらをあらかじめ取りのぞいておくことが重要である．

逆依存や出力依存は，プログラム中において同じロケーション，すなわち同じ番号のレジスタを使いまわしているために生じる（4 編 7 章 7-1 節）．これに対し，各命令の実行結果をそれぞれ別のロケーションに保存することで，この問題を解消することができる．この目的のため，スーパスカラプロセッサは物理レジスタと呼ぶ物理的なロケーションを持つ．命令セットによって定められたレジスタは，この物理レジスタと区別して論理レジスタと呼ばれる．

物理レジスタは，各命令のディスティネーションに動的に割り当てられ，各命令はそこに

実行結果を格納する．実行に必要なソースオペランドは依存元命令のディスティネーションオペランドとして割り付けられた物理レジスタを参照することにより得られる．このことは，各命令のオペランドの論理レジスタ番号を物理レジスタ番号に付け換えた（Rename）ようにみなせる．このため，この処理はレジスタリネーミングと呼ばれる．

8-2-3 命令スケジューリング

命令ウィンドウにディスパッチされた命令は，以下の 2 つのフェーズによってスケジューリングされる．

1. **ウェイクアップ（Wakeup）** 命令をスケジューリングするためには，命令ウィンドウの中から実行可能な命令を検出する必要がある．このために，命令ウィンドウ中で “眠っている” 命令を文字通り “起こす” 処理がウェイクアップである．ここで，命令ウィンドウ中で “眠っている” 命令とは，依存関係が満たされていない命令のことである．命令は依存関係が満たされたら，すなわち，依存しているすべてのオペランドが利用可能になったら，ウェイクアップされる．

2. **セレクト（Select）** 実行可能な命令の中から，実際にどの命令を実行するかを選択する．このことをセレクトするという．1 サイクル当たりに実行できる命令の数は演算器などの資源の数によって制約されるため，命令ウィンドウ中で実行可能な命令の中から，この制約を満たすよう命令をセレクトする．

上記のウェイクアップとセレクトの処理はフィードバックループを形成している．命令ウィンドウ中で実行可能な命令の中からセレクトを行い，それによって新たに実行可能になった命令を次のサイクルにウェイクアップする．命令のスケジューリングは，これを毎サイクル繰り返すことによって進行する．

8-2-4 バックエンド

スケジューリングされた命令は命令ウィンドウから読み出され，機能ユニットに送信される．このことを発行（Issue）するという．発行された命令は，通常のプロセッサの場合と同様にしてレジスタの読み出しを行い，実行される．

8-2-5 まとめ

以上が，スーパスカラプロセッサによる動的な命令スケジューリングの基本である．レジスタリネーミングやスケジューリングを実現するための具体的なロジック，各種の投機からの回復方法などの詳細については，文献[1, 2, 3, 4]を参照されたい．

■参考文献

1) マイク・ジョンソン：“スーパスカラ・プロセッサ – マイクロプロセッサ設計における定量的アプローチ,” 日経 BP 出版センター, 1994
2) 安藤秀樹：“命令レベル並列処理 – プロセッサアーキテクチャとコンパイラ –,” コロナ社, 2005.
3) Kessler, R.: “The Alpha 21264 microprocessor,” *IEEE micro*, vol. 19, no. 2, pp. 24–36, 1999.

4) Yeager, K.: “The Mips R10000 Superscalar Microprocessor,” *IEEE micro*, , vol. 16, no. 2, pp. 28–41, 1996.