マイクロコンピュータ 48－4
計算機アーキテクチャ 69－4
（1988. 1. 21）

SIMP（単一命令流／多重命令パイプライン）方式の構想

村上和彰‡ 福田 晃† 末吉敏則† 富田眞治†
†：九州大学大学院 総合理工学研究科 情報システム学専攻
‡：九州大学 工学部 情報工学科

新しいプロセッサ・アーキテクチャとして，命令パイプライン処理方式と低レベル並列処理方式とを融合した『SIMP（単一命令流／多重命令パイプライン）方式』を提案する．SIMP方式では，従来のSISD方式（単一命令パイプライン）の汎用プロセッサとの間で機械命令レベルの互換性を保ちつつ，命令パイプラインを多重化することで応答速度の向上を目指している．SIMP方式のVLSIプロセッサ構成としては，命令パイプライン単位でVLSI化を行い，それを速度要求に見合った数だけ装備する『パイプラインスライス・マイクロプロセッサ』構成法を考えている．また，SIMP方式に向いた命令セット・アーキテクチャとしては，『BISC（均衡命令セット・コンピュータ）』が有力である．本稿では，SIMP方式の実現に際しての課題を中心に述べている．

SIMP : Single Instruction stream／Multiple instruction Pipelining (in Japanese)

Kazuaki MURAKAMI‡, Akira FUKUDA†, Toshinori SUEYOSHI† and Shinji TOMITA†
†: Dept.of Information Systems
‡: Dept.of Computer Science and Communication Eng.
Kyushu University
Kasuga, Fukuoka, 816 Japan

We propose a new processor architecture : SIMP (Single Instruction stream／Multiple instruction Pipelining). SIMP is a hybrid processor architecture that combines instruction pipelining with low－level parallelism. Thus SIMP can realize both temporal and spatial parallelism. An SIMP processor is the same as an SISD processor from a program viewpoint, but it offers higher response speed by exploiting multiple instruction pipelines. We also propose the pipeline－sliced microprocessor organization as an actual way to achive a VLSI implementation of SIMP. We consider that a computer with an instruction set suitable for SIMP should be a balanced instruction set computer (BISC), neither RISC nor CISC.

This paper discusses the issues involved in the design of an SIMP processor.

1. はじめに

VLSI時代と呼ばれる今日，高性能かつ高機能なマイクロプロセッサが量販され，様々な計算機システムが構築されている．これらマイクロプロセッサをベースとするシステムにおいて処理性能をさらに向上させようとする場合，個々のプロセッサの高速化ないしプロセッサ数の多重化といった手法を用いている．前者の例が命令パイプラインで，従来は汎用大型計算機のお家芸だったものをマイクロプロセッサにも適用したものである．RISC／CISC論争も，命令パイプラインを如何にVLSI化するかという点に事を発している．一方，後者の例がいま関心を集めているマルチマイクロプロセッサであり，数台から数万台規模のものまで種々の特徴あるシステムが開発されている．

しかし，これらの手法も万全なものではない．命令パイプラインによる高速化はサイクル・タイムの短縮化と命令当りサイクル数の縮小化に係っているが，将来シリコン素子技術が限界に達した場合それ以上の速度向上は望めなくなる．また，マルチマイクロプロセッサによる高並列処理方式では，その性能向上率は並列処理プログラムの品質に依存しており一定ではない．よって，実装したマイクロプロセッサ台数分だけの性能が常に得られるという保証がない．

そこで，筆者らは，現在の32ビット・マイクロプロセッサの後継プロセッサとして，低レベル並列処理方式を採用した超高速プロセッサの開発を行っている．低レベル並列処理方式の実現方法としては，京都大学で開発されたQA－2[1]に代表される超長形式機械命令（VLIW：Very Long Instruction Word）方式が一般的であるが，我々は単一命令流／多重命令パイプライン（SIMP：Single Instruction stream／Multiple instruction Pipelining）方式と呼ぶ新しいアーキテクチャを検討している．[2][3] 本稿では，SIMP方式の目的，特長および，実現に際しての課題を述べる．

2. 目的

SIMP方式は，VLSI時代における高性能計算機システムの心臓部となり得る超高速プロセッサのアーキテクチャであり，このアーキテクチャを有するプロセッサをSIMP型プロセッサと呼ぶ．SIMP方式およびSIMP型プロセッサは，以下の点を特に目的としている．

(1) 高速性の追求

SIMP方式が目指す『高速性』とは，命令バウンドな処理分野における応答速度（response speed）の向上である．

計算機システムの性能を左右する要因としては，まず行おうとする処理が，

① CPUバウンド

② I／Oバウンド

のいずれの処理分野に属しているかがある．さらに，CPUバウンドな処理分野は，その原因となるのが処理すべきデータ量かあるいは処理の操作量自体かで，

① データ・バウンド

② 命令バウンド

の2つに大別できる．それぞれの分野に対しては，

① I／Oバウンド→I／Oプロセッサ

② データ・バウンド→ベクトル・プロセッサ

③ 命令バウンド→命令パイプライン・プロセッサ

などのプロセッサ・アーキテクチャが既に存在する．SIMP方式が対象とするのは命令バウンドな処理分野であり，命令パイプライン・プロセッサの置き換えを狙ったものである．

一方，計算機システムの性能指標としては，

① 応答時間（response time）

② スループット

の2つがよく用いられる．マルチマイクロプロセッサは，スループットの向上という点では確かに効果がある．しかし，応答時間を短縮するには負荷分散方法など，まだ解決すべき問題が多い．そこで，SIMP方式では，スループットよりむしろ応答時間の面でその真価を発揮させるようにする．

(2) 機械命令の汎用化

現在のマイクロプロセッサ・アーキテクチャはSISD（Single Instruction stream／Single Data stream）方式に基づいている．SIMP型プロセッサの機械命令は，これらSISD型プロセッサの機械命令に対して互換性を保つようにする．

従来の低レベル並列処理方式の実現方法ではVLIW方式に見られるように，機械命令のフォーマットが実装された機能ユニット（ALUなど）数を反映している．したがって，VLIW型プロセッサにおいては，機械命令レベルで機能ユニットの多重度つまり処理の並列度に対する自由度がない．つまり，機械命令そのものが特定のシステムに専用化されたものであると言える．

一方，同じく低レベル並列処理方式の一実現方法であるSIMP方式では，機械命令レベルでは機能ユニット（具体的には，命令パイプライン）数を透過にし，機械命令の汎用化を図るようにする．これにより，SISD方式はSIMP方式の特殊な形式という位置づけになる．すなわち，1本の命令パイプラインを持つ従来のSISD型プロセッサのオブジェクト・プログラムがそのまま，S本の命令パイプラインを持つSIMP型プロセッサ上で実行可能となるからである．

(3) 新しいプロセッサ構成法の開発

SIMP型プロセッサを構成するには，命令パイプライン単位でVLSI化を行い，そのVLSIチップを速度要求に見合った個数だけ実装することで単一プロセッサとなるようにする．よって，1チップだけで構成されたものが従来のSISD型マイクロプロセッサということになる．

このプロセッサ構成法に似た方法として，ビットスライス・マイクロプロセッサを用いたプロセッサ構成法がある．

ビットスライス・マイクロプロセッサ構成法では，実装するチップ数が必要とするデータ長により決まったのに対し，SIMP型プロセッサでは要求される速度により決まる．

SIMP方式により，さしずめ『パイプラインスライス・マイクロプロセッサ』とでも呼ぶべきプロセッサ構成法を開発する．

3. 特長

SIMP方式は，以下の特長を有する．

(1) 処理形態

SIMP方式では，単一の命令流を時間的かつ空間的に並列処理する．

一般的な命令処理の形態は図1（a）（b）に示すように，

① 単一命令パイプライン処理：命令処理にかかる時間をTステージに分割して時間並列処理する．このときの処理の並列度は，時間多重度Tに等しい．

② 低レベル並列処理：S個の機能ユニットを装備し，個々のユニットで1つの命令（あるいは命令フィールド）を担当して空間並列処理する．このときの処理の並列度は，空間多重度Sに等しい．

のいずれかである．SIMP方式では，図1（c）に示す多重命令パイプライン処理の形態をとる．すなわち，Tステージの命令パイプラインをS本装備して，処理の並列度を空間多重度×時間多重度（S×T）に引き上げようとするものである．

同時に命令パイプラインに投入されるS個の命令の集合をブロックと呼ぶことにする．

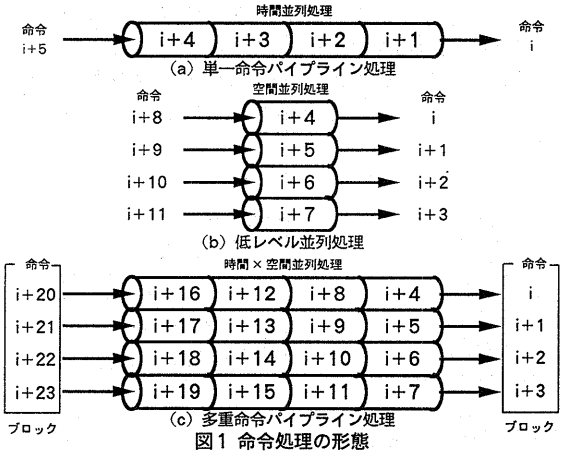


図1 命令処理の形態

(2) システム構成

SIMP方式によるシステム構成および他の有名なシステム構成法を図2に示す．

図からも分かるように，SIMP方式（図2（a）参照）はSISD方式（図2（b）参照）における処理ユニット（PU）

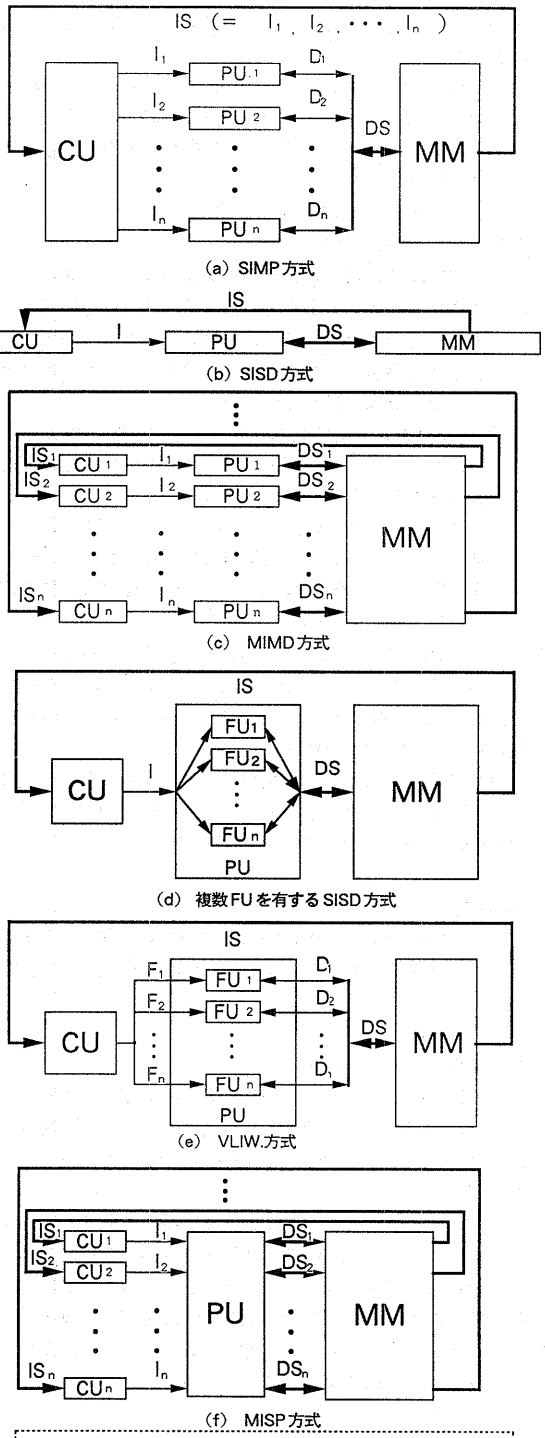


図2 システム構成

を多重化した構成と等価である．SIMP方式によく似た構成として，

① MIMD（Multiple Instruction stream／Multiple Data stream）方式（図2（c）参照）

② 複数の演算ユニット（FU）を有するSISD方式（図2（d）参照）

③ VLIW方式（図2（e）参照）

があるが，次の点で異なっている．

① MIMD方式では複数のプログラムカウンタが存在して各々が対応する命令流および処理ユニットを制御する．一方，SIMP方式においては処理ユニットの数に係わらず命令流そのものは単一であり，従ってプログラムカウンタは1個だけ存在する．

② SISD方式で複数演算ユニットを並列動作させても，処理ユニットとしてはあくまでも1個であり，最高速度も高々1命令／τしか得られない．[4]

③ VLIW方式では機械命令の各フィールドで制御する機能ユニットは必ずしも均質である必要はない．一方，SIMP方式では各処理ユニットが均一な処理機能を持たなければならない．

一方，SIMP方式とは全く正反対の構成を採るものとして，MISP（Multiple Instruction stream／Single instruction Pipelining）方式（図2（f）参照）がある．MISP方式では，1個の処理ユニット（命令パイプライン）が複数の命令流に共有されており，これらの命令を交互に処理していく．一般に，共有パイプライン[5]ないし循環（cyclic）パイプライン[6]と呼ばれる．

(3) 性能

S本の命令パイプラインが装備されている場合，最高速度は1ブロック／τつまりS命令／τとなる（図1（c）参照）．

(4) 命令アーキテクチャ

SIMP方式の機械命令は，SISD方式の機械命令と本質的に違わない．但し，各命令パイプラインで同時に命令解読を行うため，固定長かつ同一長の機械命令形式である必要がある．

SIMP方式とVLIW方式における機械命令形式の相違を図3に示す．VLIW方式では，機械命令レベルで機能ユニット（FU）数が見えるので，コンパイラはそのことを意識する必要がある．一方，SIMP方式では，命令パイプラインの本数が機械命令レベルで透過なため，コンパイラは命令パイプライン本数に依存せずにコードを生成できる．

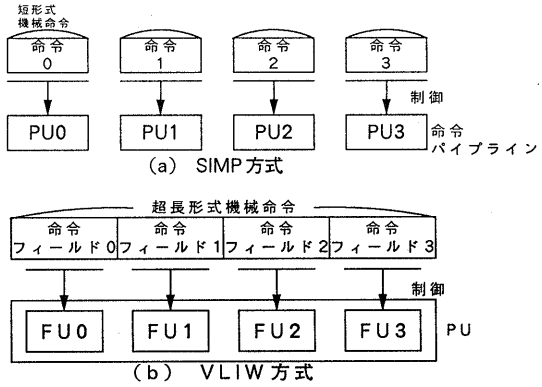


図3 機械命令形式

4. 課題

SIMP方式を実現する場合，最高速度を達成するには，単一命令パイプラインの時にも増して，パイプラインの乱れ（パイプライン・インターロック）を小さく抑える必要がある．さもないと，最悪の場合，単一命令パイプラインと同等の速度しか得られないことになる．命令パイプラインを多重化したことにより，単一命令パイプラインの時に比べて，新たな技術的課題が生じている．

また，命令レベルの並列性を如何に引き出すかという問題は，VLIW方式やSIMP方式といった低レベル並列処理方式には本質的な課題である．VLIW方式のシステムでは，コンパイラの全責任で並列性の抽出およびそれの超長形式機械命令への埋め込みが行われる．一方，SIMP方式のシステムでは，コンパイラは並列性の抽出は行えるが，それを機械命令の中に明示的に反映することはしない．よって，ここでまた，VLIW方式の場合とは異なる技術的課題が生じる．

以下，SIMP方式の実現に関する諸課題について議論する．

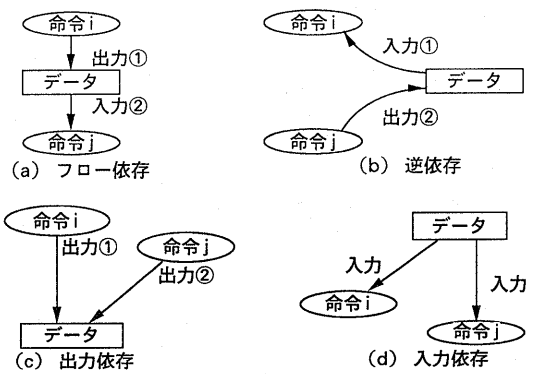


図4 データ依存

4.1 命令間の依存関係

一般に，命令間には，

① 制御依存（control dependence）

② データ依存（data dependence）

の2種類の依存関係が存在する．[7]

制御依存とは，条件分岐命令の後に続く命令の実行が分岐条件の真偽に左右される状況を指す．

また，データ依存には図4に示すように，ある1つのデータに対する入出力の組合せにより，

① フロー依存（flow dependence）　：出力→入力

② 逆依存（anti−dependence）　：入力→出力

③ 出力依存（output dependence）　：出力→出力

④ 入力依存（input dependence）　：入力→入力

の4通りの依存関係がある．これらのうち，入力依存関係にある命令同士は並列処理しても構わないが，残りの依存関係にある命令の処理は逐次的になる．

これらの依存関係がパイプライン・インターロックの大きな原因となっており，単一命令パイプラインでも次のような対策が講じられている．[8]

① 制御依存への対策：分岐するかしないかを予測して，予測した側の命令を実行する（図5参照）．もし予測が誤っていたら，その命令を無効化する（no−op命令とする）．

② 逆依存および出力依存への対策：この2つは本質的なデータ依存関係ではなく，データ名を変更することで削除可能である．[9]

③ フロー依存への対策：これは本質的なデータ依存関係であり，削除することができない．そこで，演算が完了したら出力を待たずに直ちに入力側ステージへデータをバイパスするような機構を設ける（図6参照）．

④ コンパイラによる最適化：逆依存および出力依存はコンパイル時にも削除できる．また，フロー依存関係にある命令同士は出来るだけ遠ざけるようにする．

以上は，単一命令パイプラインによる時間並列処理における命令間の依存関係，すなわち時間的な依存関係（図7(a)参照）に対する対策であった．多重命令パイプラインにおいては，さらに命令の空間並列処理も行う．したがって，命令間の依存関係としては，時間的依存関係に加えて，同一ステージ上を並行して流れている命令間での空間的な依存関係も生じ得る（図7(b)参照）．SIMP方式の実現に当たっては，この時間的および空間的依存という2次元の依存関係にたいしてどのように対処するかが最大の課題となる．

空間的依存関係が加わったことにより，以下の点に留意する必要がある．

(1) 依存関係の解析

多重命令パイプラインでは同一ブロック内のS個の命令を同一時間に並行して解読するので，

① 依存関係の解析自体をどのように行うか

② 解析の結果，逆依存や出力依存をどのように削除するか

が問題となる（単一命令パイプラインではこれらを逐次的に行っている）．

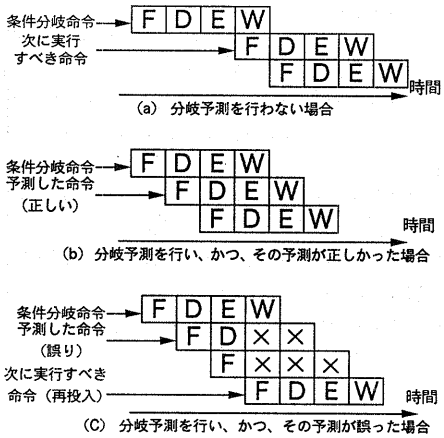


図5 分岐予測

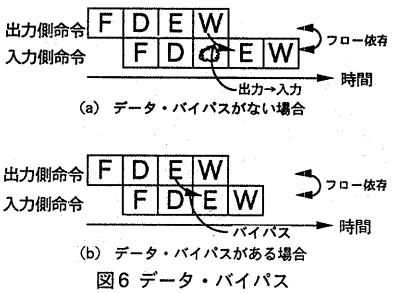


図6 データ・バイパス

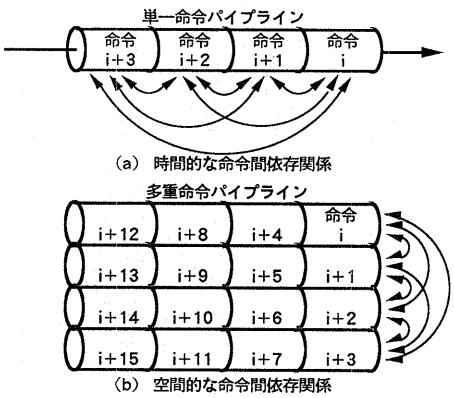


図7 命令間依存関係

(2) 制御依存

多重命令パイプラインでは，ブロック単位でS個の命令が命令パイプラインに投入されるので，次の点が問題となる．

① 分岐予測の単位が命令ではなくブロックとなる．

② 同一ブロック内に条件分岐命令がある場合，制御依存関係にある命令を条件分岐命令と並行して実行しなければならない．また，分岐が確定した時点で，実行してはいけない命令を無効化する必要がある（図8（b）参照）．

③ 分岐予測を誤った場合の後続するブロックの無効化は，基本的には単一命令パイプラインにおける後続命令の無効化と同様の方法で行える．しかし，後述するように，次に実行すべき命令が既に投入されてる可能性が大きいので，そのことを考慮する必要がある（図8（c）参照）．

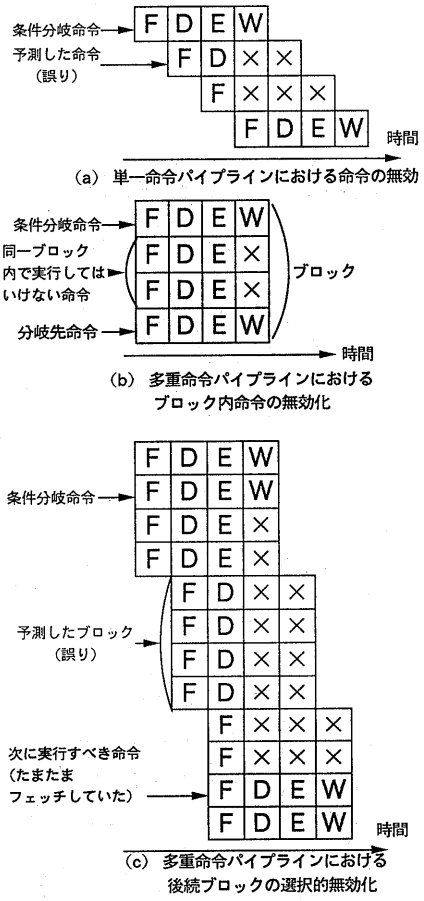


図8 命令の無効化

(3) フロー依存

単一命令パイプラインではステージ間のデータ・バイパスで対処したが，多重命令パイプラインではさらにパイプライン間でのデータ・バイパスが必要となる（図9（b）参照）．このパイプライン間のデータ・バイパスをチェイニングと呼ぶ．しかし，チェイニングにより新たに引き起こされるパイプライン・インターロックへの考慮が必要となる（図9（c）参照）．

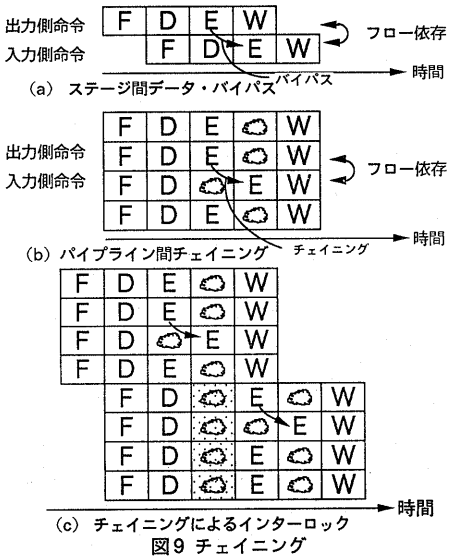


図9 チェイニング

4.2 命令およびデータ供給

命令パイプラインの処理能力に応じて，命令およびデータを供給する必要がある．命令ないしデータの不在がパイプライン・インターロックのもう1つの大きな原因となるからである．

単一命令パイプラインの命令およびデータ供給系では，

① バッファ：アドレス・バッファ，命令バッファ，データ・バッファ

② キャッシュ：命令キャッシュ，データ・キャッシュ

③ メモリ・インタリーブ

などの手法を複合的に用いて，命令パイプラインの処理能力とのバランスをとっている．

多重命令パイプラインにおいては，単一命令パイプラインの命令およびデータ供給系を基にして，命令パイプラインの空間多重度に応じてそれらを多重化する必要がある．特に，キャッシュの多重化に関しては，

① マルチバンク

② マルチポート

③ マルチキャッシュ

などの方法が考えられる．命令キャッシュとしてはマルチバンクが，またデータ・キャッシュとしてはマルチポートが妥当な多重化方法と思われる．

4.3 命令機能の均衡化

命令の演算機能にバラツキがある場合，単一命令パイプラインでは並列演算パイプライン方式を採用することである程度対処できたが，多重命令パイプラインでは他の命令パイプラインに悪影響を及ぼしてしまう（図10参照）．つまり，同一ブロック内のS個の命令の中で最も実行時間の長い命令によって，その他の命令の実行時間が抑えられることになる．これは，多重命令パイプライン独自の問題である．

この問題を解決するには，命令の機能のバラツキを小さくするように命令セットを定義すればよい．このように機能が均衡化された命令セットを有する計算機をここではBISC（Balanced Instruction Set Computer：均衡命令セット・コンピュータ）と呼ぶ．BISCにおいては，機能が複雑（complex）かあるいは縮小されてる（reduced）かは議論の対象とはならない．

高性能のSIMP型プロセッサは，必然的にBISCになると思われる．

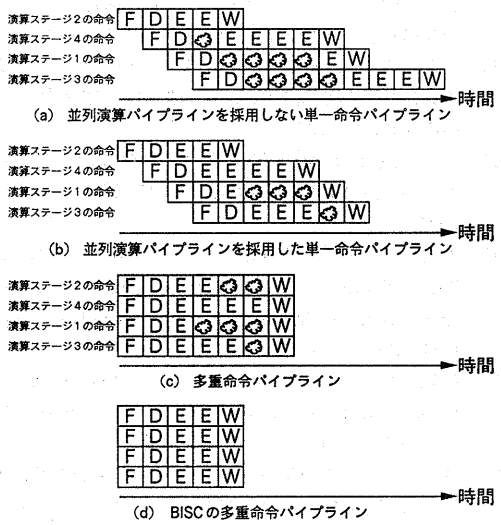


図10 命令の機能のバラツキ

4.4 命令パイプラインの使用率

分岐により，命令パイプラインの使用率が低下する可能性がある．命令パイプラインの使用率低下については，

① ブロック内における使用率低下

② 複数ブロックをまたいでの使用率低下

の2つの側面がある．

(1) ブロック内における使用率低下

SIMP方式では，ブロック単位でS個の命令を命令パイプラインに同時に投入する．このブロック内における分岐命令および分岐先命令の相対位置が，命令パイプラインの使用率に大きな影響を与える（図11参照）．このような命令パイプライン使用率の低下に対する対策としては，

① コンパイラによる最適化：分岐命令はブロックの最後尾に，また，分岐先命令はブロックの先頭にそれぞれ配置する（図11（a）参照）．

② ブロック・アライメント：S個の命令をフェッチする際，分岐先命令がブロックの先頭になるように，ブロック境界を調整する（図12参照）．

などの方法がある．但し，コンパイラによる最適化は，コンパイラが命令パイプラインの本数を知っている必要があること，また，ある一定の命令パイプライン数のシステムにしか有効でないことから，あまり好ましくない．

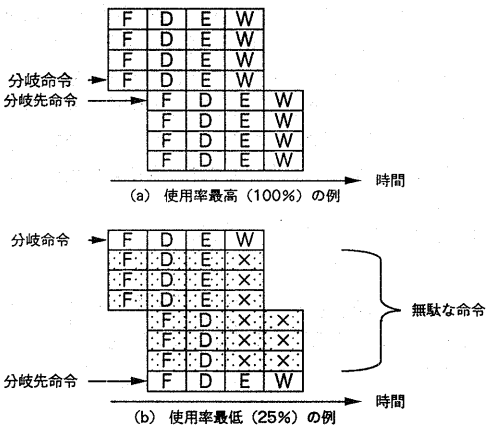


図11 命令パイプラインの使用率

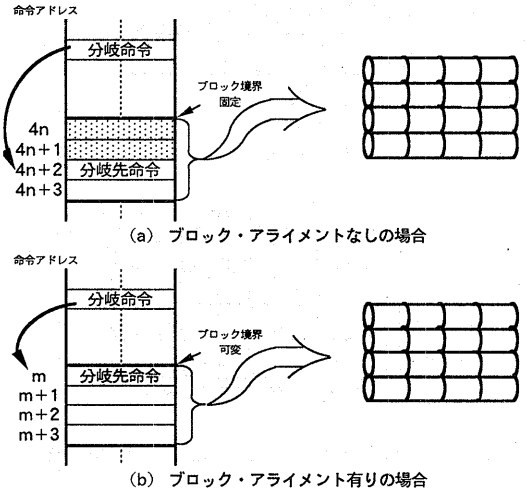


図12 ブロック・アライメント

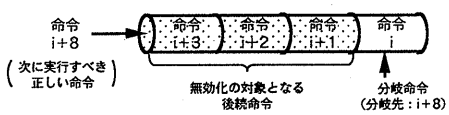
(2) 複数ブロックをまたいでの使用率低下

分岐予測を誤った場合には，命令パイプラインに既に投入してしまった後続ブロックを無効化する必要がある．単一命令パイプラインでは，分岐予測が誤りであることが判明した時点で，パイプライン内のすべての後続命令（T－1）個を単に無効化すればよい．そして，実行すべき命令を命令パイプラインに投入し直す（図13（a）参照）．しかし，多重命令パイプラインでは，後続ブロック内の命令としてS×（T－1）個もの多くの命令がパイプライン内に存在している．分岐予測が誤っていても，これらの命令の中に次に実行すべき正しい命令が含まれている可能性が大きい．したがって，パイプライン内のすべての後続ブロックを単に無効化したのでは効率が悪くなる（図13（b）参照）．このような命令パイプラインの使用率低下に対する対策は，

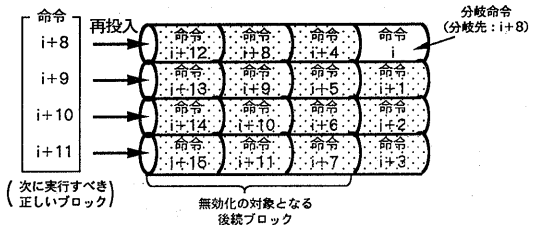
① 命令を選択的に無効化する．選択の方法は，最も古いブロックから命令アドレスを調べて行き，次に実行すべき正しい命令のアドレスと出会うまで続ける．

② 命令供給系は命令パイプライン中に存在するブロックを把握しておき，次に実行すべき正しいブロックが存在する場合にはそれをパイプラインに再投入しないようにする．

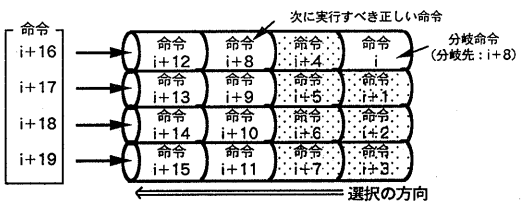
のようになる（図13（c）参照）．



(a) 単一命令パイプラインにおける命令の無効化



(b) 多重命令パイプラインにおける命令の非選択的無効化



(c) 多重命令パイプラインにおける命令の選択的無効化

図13 命令の無効化方法

5. おわりに

以上，筆者らが考案したSIMP方式のプロセッサ・アーキテクチャについて述べた．現在，ここでの論議に基づいて，SIMP型プロセッサのディスクリートICによる試作機を開発している．この試作SIMP型プロッセサに関する報告は，別の機会に譲りたい．

また，SIMP型プロセッサの開発と並行して，シミュレータにより，命令パイプライン，キャッシュ，命令セットなどの各種構成要素をパラメータ化して，SIMP方式の性能評価を行っている．これらの評価についても，結果が得られたら改めて報告したい．

謝辞

我々と共に，設計・開発を行っている五島龍宏，久我守弘，入江直彦の各氏，および，日頃ご討論いただく富田研究室の皆様に感謝いたします．

参考文献

1) 北村 他：ユニバーサル・ホスト計算機QA－2の低レベル並列処理方式，情処論27－4（1986）

2) 村上 他：SIMP（単一命令流／多重命令パイプライン）方式の構想，昭和62年電気関係学会九州支部連合大会，No.1044

2) 五島 他：SIMP方式における命令間依存性の取扱い，昭和62年電気関係学会九州支部連合大会，No.1045

4) W.Hwu and Y.Patt：HPSm，a High Performance Restricted Data Flow Architecture Having Minimal Functionality，13th ISCA（1986）

5) B.Smith：The Architecture of HEP，Parallel MIMD Computation（ed. J.S.Kowalik），MIT－Press，pp. 41－55（1985）

6) 後藤 他：FLATS2のアーキテクチャ，情報処理学会第35回（昭和62年後期）全国大会，6C－4

7) D.J.Kuck：The Structure of Computers and Computations，Vol.1，John Wiley & Sons（1978）

8) 富田眞治：並列計算機構成論，昭晃堂（1986）

9) R.M.Tomasulo：An Efficient Algorithm for Exploiting Multiple Arithmetic Units，IBM J.R&D,Vol.11，No.1，pp.8－24（1967）