システムソフトウェアと
オペレーティング・システム 84－1
（２０００．５．２５）

スレッドレベルと命令レベルの並列度を利用した VLIWプロセッサ

島尻 寛之　　吉田 たけお

琉球大学 工学部 情報工学科

〒 903 - 0213 沖縄県 中頭郡 西原町 字千原 1 番地

E-mail: {shimajiri,tyoshida}@fts.ie.u-ryukyu.ac.jp

あらまし VLIW プロセッサは，命令レベルの並列性を利用して性能を向上させている．しかし，命令レベルの並列度には限界があるため，実行ユニットを多く実装した VLIW プロセッサほど，性能を引き出すことが難しくなる．そこで，本稿ではマルチスレッドの考え方を VLIW に適用する．提案する VLIW プロセッサは，実行するスレッドをメインとサブに分け，メインスレッドでは命令レベルの並列性を利用し，通常の VLIW プロセッサと同様に処理する．さらに，スレッドレベルの並列性を利用して，サブスレッドの命令をメインスレッドの NOP 命令の代わりに実行し，VLIW プロセッサのスループットの向上を実現する．

キーワード　VLIW プロセッサ，スレッド，マルチスレッドプロセッサ，　VIM プロセッサ

A VLIW Pocessor Using Both Thread-level and Instruction-level Parallelism

Hiroyuki SHIMAJIRI　　Takeo YOSHIDA

Department of Information Engineering, Faculty of Engineering,
University of the Ryukyus

1, Senbaru, Nishihara, Nakagami, Okinawa, 903 - 0213 JAPAN

E-mail: {shimajiri,tyoshida}@fts.ie.u-ryukyu.ac.jp

VLIW processors' performance has been improved by using instruction-level parallelism. But it is difficult to achieve improvement in the performance of VLIW processor which implemented many execution units. Because the instruction-level parallelism has a limitation of control dependencies. In this paper, we propose a VLIW based instruction merge processor (VIM processor) which improves throughput. VIM processor uses not only instruction-level parallelism but also thread-level parallelism. We show the effectiveness of VIM processor by presenting experimental result.

Key wards VLIW Processor, Thread, Multi-threaded Processor, VIM Processor

1 はじめに

現在，プロセッサのほとんどは，その処理能力を向上させるために，パイプラインやスーパースカラ，VLIW 等のアーキテクチャを採用している．これらのアーキテクチャでは，複数の実行ユニットを用いて複数の命令を並列処理することで，プロセッサの処理能力を向上させている．特に VLIW プロセッサは，コンパイラの性能向上により，最近，汎用プロセッサのアーキテクチャとして採用され注目を集めている [1, 2]．また，VLIW プロセッサは構造が単純で，アプリケーションや目的に合わせて，実行ユニットの数や種類などの構成を自由に決めることができることも，注目を集める要因となっている．

VLIW プロセッサは命令レベルの並列性を利用して，単一スレッド内の命令を並列に処理している．しかし，VLIW プロセッサは多くの実行ユニットを持つため，並列度の低いスレッドを実行した場合，全ての実行ユニットに命令を割り当てることができず，NOP(No Operation) 命令を実行することが多くなる [3, 4]．この傾向は，実行ユニットの数が増えるほど顕著になる．そのため，実行ユニットを多く実装した VLIW プロセッサほど，実行ユニットの数に見合った性能を引き出すことが難しくなる．

本稿では，VLIW プロセッサのスループットを向上させるために，マルチスレッドの考え方を VLIW プロセッサに適用することについて検討する．ここでマルチスレッドとは，複数のスレッドを並列に処理することにより，プロセッサのスループットを向上させる技術である．またスレッドとは，一本の命令ストリームであり，異なるスレッド間には命令の依存関係は存在しないものをいう．

通常のプロセッサでは命令レベルの並列度を利用して，単一スレッド内に存在する，依存関係のない命令を並列に処理する．しかし，命令レベルでは同時に実行できる命令の数はそれほど多くないため，スループットを向上させることができない．これに対しマルチスレッドでは，スレッドレベルの並列度を利用して，複数のスレッド間の命令を並列に処理するため，高い並列度を得ることができる．また，同時に実行するスレッドの数を増やすことで，より高い並列度を得ることができるため，スループットの大幅な向上が望める．このマルチスレッドの考え方をプロセッサに応用した，マルチスレッドプロセッサも数多く提案されている [5–7]．

しかし，マルチスレッドプロセッサでは，プロセッサの処理能力が複数のスレッドの処理に分散されるため，一つあたりのスレッドの実行時間は通常のプロセッサに比べて必然的に長くなる．VLIW プロセッサのように多くの実行ユニットを実装するプロセッサにおいては，プロセッサのスループットよりもレスポンスタイムを重視する場合が多い．そのため，VLIW プロセッサをマルチスレッド化する場合，マルチスレッド化によるスループットの向上よりも，レスポンスタイムの低下が大きな問題となる．

そこで本稿では，VLIW プロセッサをマルチスレッド化する際に，この問題点を考慮して，実行するスレッドをメインスレッドとサブスレッドに分けて実行する．メインスレッドは，命令レベルの並列性を利用して通常の VLIW プロセッサと同様に処理する．このとき，スレッドレベルの並列性を利用して，メインスレッドに含まれている NOP 命令の代わりに，サブスレッドの命令を実行する．これにより，メインスレッドのレスポンスタイムの低下を招くことなく，VLIW プロセッサのスループットを向上できると考えられる．

本稿では，**2** でマルチスレッドプロセッサと通常のプロセッサの違いについて述べる．**3** では，VLIW プロセッサをマルチスレッド化する方法について説明し，**4** でマルチスレッド化の実現方法を提案する．**5** では，マルチスレッド VLIW プロセッサと通常の VLIW プロセッサの比較を行う．

2 マルチスレッドプロセッサ

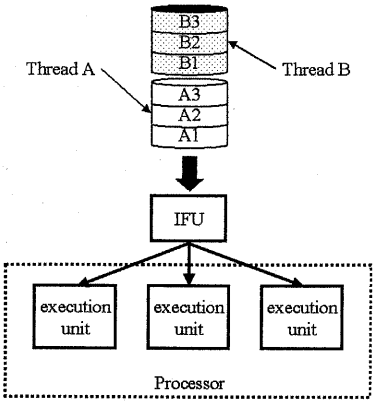
ここでは，通常のプロセッサとマルチスレッドプロセッサの違いについて説明する．通常のプロセッサとマルチスレッドプロセッサでは，複数のスレッドを実行する際の命令の供給方法が異なる．図 1に，それぞれのプロセッサの命

令供給方法を示す．

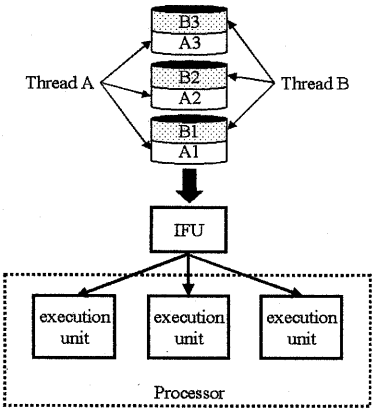
図 1 (a) の通常のプロセッサの命令供給方法では，一つのスレッドの命令が全て実行された後に，次のスレッドの命令が供給される．この場合，スレッドは逐次的に処理され，実行中のスレッド以外のスレッドが同時に実行されることはない．

これに対し，図 1 (b) のマルチスレッドプロセッサの命令供給方法では，異なるスレッドの命令が交互に供給される．これは特にパイプライン化されたプロセッサに有効である．パイプライン化されたプロセッサで一つのスレッドを実行した場合，命令間の依存関係や分岐命令などの影響により連続して命令を実行することができず，代わりに NOP 命令を実行することになる．しかし，図 1 (b) の方法を採用することで，別のスレッドの命令を実行している間に，前に実行されたスレッドの依存関係や分岐命令の問題が解消される．このため，プロセッサの実行ユニットは絶え間なく命令を処理し続けることができる．この場合，各スレッドの実行時間は長くなるが，プロセッサ全体のスループットは増大する．

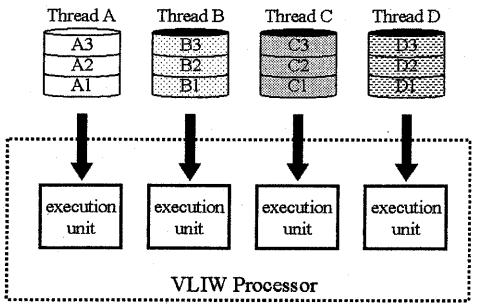
また，VLIW プロセッサをマルチスレッド化する場合，図 1 (b) とは別の命令供給方法をとることもある．これを図 1 (c) に示す．図 1 (c) の方法では，VLIW プロセッサの各実行ユニットに対し，それぞれ異なるスレッドの命令を供給している．図 1 (c) は，4 個のプロセッサコアを持つマルチプロセッサと考えることができる．この場合の実行ユニットは，通常のプロセッサと同じ処理能力を持つことが前提となるため，回路面積が大きくなるという問題点がある．また，各スレッドは一つのプロセッサコア上で処理されることになるため，命令レベルの並列性を利用することができなくなる．



(a) nomal processor



(b) multi thread processor



(c) multi thread VLIW processor

図 1　各プロセッサの命令供給方法の比較

3　VLIW のマルチスレッド化

次に，本稿で提案するマルチスレッド化のための命令供給方法を図 2 に示す．提案方法では，メインスレッドは通常の VLIW プロセッサと同様に実行される．メインスレッドの VLIW 命令に NOP 命令が含まれている場合は，NOP 命令の代わりにサブスレッドの命令を実行する．これにより，実行される NOP 命令の数を

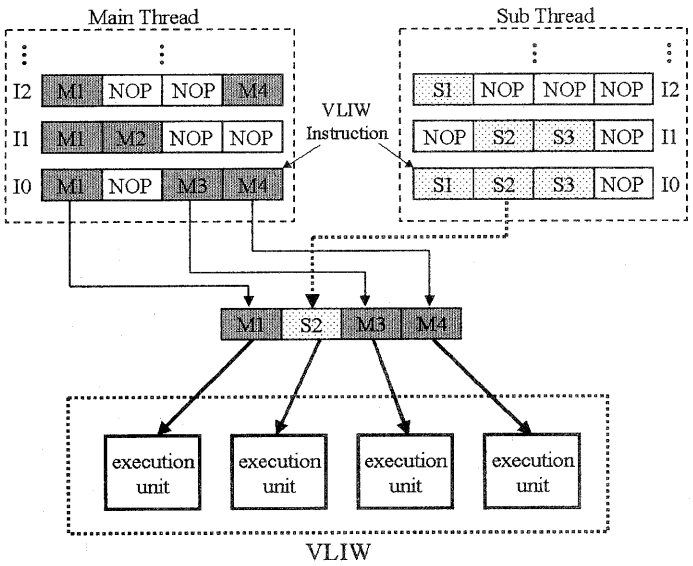


図 2　VLIW 命令を合成する命令供給方法

削減することができる．ここで，VLIW 命令とは，コンパイラによって生成された VLIW プロセッサ専用の命令のことをいう．

VLIW 命令は，コンパイラによって同時に実行可能な複数の命令が一つにまとめられているため，同一の VLIW 命令内の命令ならば，どの命令から実行してもスレッドの実行結果に影響を及ぼすことはない．VLIW プロセッサにはこのような特性があるため，サブスレッドの VLIW 命令内の命令を部分的に発行することが可能となる．

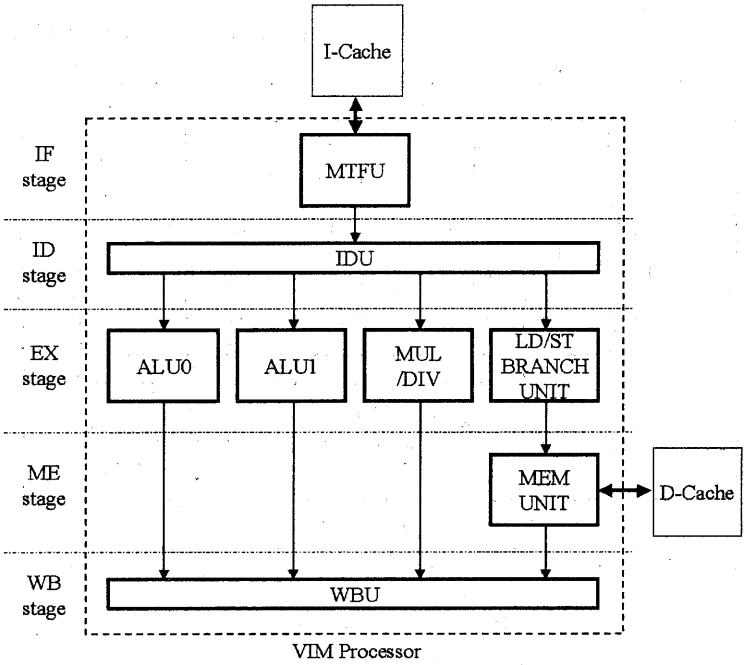
なお，メインスレッドとサブスレッドの VLIW 命令が，同じ実行ユニットに NOP 命令を割り当てていた場合は，そのまま NOP 命令を発行することになる．しかしこれは，同時に実行するサブスレッドの数を増やすことにより，高い確率で回避することができる．また，メインスレッドの VLIW 命令は，通常の VLIW プロセッサと同様にフェッチされる．一方，サブスレッドの場合は，実行中の VLIW 命令の全ての部分命令を発行してから，次の VLIW 命令がフェッチされる．

提案方法を採用することにより，メインスレッドは，通常の VLIW プロセッサで実行したときと同じレスポンスタイムで処理することができる．その上，NOP 命令の代わりにサブスレッドの命令を処理するため，命令レベルの並列性とスレッドレベルの並列性の両方を利用することができる．また，図 1 (b) の命令供給方法のように実行ユニットの制約がないため，どのような構成の VLIW プロセッサでもマルチスレッド化を実現することができる．本稿では，このような方法によって実現したマルチスレッド VLIW プロセッサを，VIM プロセッサ (VLIW based Instruction Merge Processor) と呼ぶことにする．以降では，VIM プロセッサの構成を示す．

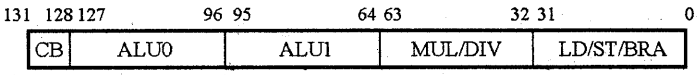
4　VIM プロセッサの構成

4.1　VIM プロセッサの諸元

図 3 に VIM プロセッサの構成を，表 1 に VIM プロセッサの諸元を，それぞれ示す．なお表 1 において，MDU は乗除算器を，LSBU(Load/Store and Branch Unit) はロー



(a) VIM プロセッサのブロック図



(b) VIM プロセッサの命令フォーマット

図 3　VIM プロセッサの構成

表 1　VIM プロセッサの諸元

命令セット	MIPS 社 R3000^{TM} の命令セットに準拠
命令フォーマット	命令語 (32bit) × 4, チェックビット (4bit)　合計 132bit
パイプライン段数	5 段 (IF, ID, EX, MEM, WB)
実行スレッド数	メインスレッド × 1, サブスレッド × 1　合計 2 個
実行ユニット	ALU × 2　MDU × 1　LSBU × 1　合計 4 個

ド・ストア命令および分岐命令を処理するユニットを，それぞれ表す．

図 3 (a) の MTFU(Multi Thread Fetch Unit) は，VLIW プロセッサをマルチスレッド化するのためのユニットで，**3** で提案した命令供給方法を実現している．MTFU の構成と詳細は，**4.2** で紹介する．

図 3 (b) は VIM プロセッサへの入力となる VLIW 命令の命令フォーマットである．命令語の最初の 4bit はチェックビット (CB) であ

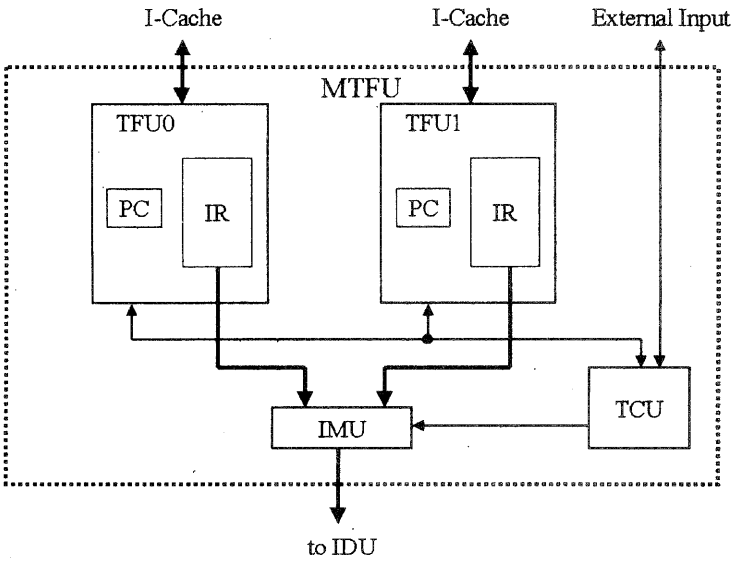


図 4　MTFU のブロック図

り，CB は MTFU で NOP 命令の有無を確認するために用いられる．CB の各ビットは，4 つの実行ユニットへの命令に対応しており，最上位ビットから順番に ALU0，ALU1，MDU，LSBU の命令に対応している．CB の各ビットは，実行ユニットへの命令が NOP 命令のときは 0，実行する命令があるときは 1 となる．

4.2　マルチスレッドフェッチユニット：MTFU

図 4 に，MTFU のブロック図を示す．MTFU は，各スレッドの VLIW 命令のフェッチを行うスレッドフェッチユニット (TFU : Thread Fetch Unit)，スレッドの制御と各 TFU の状態をプロセッサの外部に出力するスレッド制御ユニット (TCU : Thread Control Unit)，メインスレッドとサブスレッドの VLIW 命令を合成し，命令デコーダに命令を発行する命令合成ユニット (IMU : Instruction Merge Unit) から構成される．なお，今回は VIM プロセッサと MTFU の構成を簡略化するために，同時に実行するスレッドの数を 2 つとした．そのため，TFU の数は 2 つである．また，各 TFU に対応したレジスタファイルを 2 つ用意した．

4.3　スレッドフェッチユニット：TFU

TFU は，TCU によって割り当てられたスレッドの VLIW 命令を命令キャッシュからフェッチするユニットである．TFU は，プログラムカウンタ PC と命令レジスタ IR で構成されている．

IR は，IMU から要求があると VLIW 命令を IMU に渡し，命令キャッシュから命令をフェッチする．サブスレッドの VLIW 命令は，全ての命令を発行するまで保持され続ける．TFU は，スレッド終了の命令をフェッチすると TCU に対して，スレッドが終了したことを伝える．

TFU では，メインスレッドとサブスレッドの区別は行わず，どちらのスレッドも保持することができる．また，今回は 2 つしか用意しなかったが，同時に実行するスレッドの数を増やすときには TFU の数を増やすことで対応する．

4.4　スレッド制御ユニット：TCU

TCU は，TFU, IMU の制御を行う．TCU は，スレッドの生成やスレッドの優先権の変更などの指示を，割り込みなどの方法によりプロ

セッサの外部から与えられる．スレッドの生成は，外部から与えられたスレッドのアドレスをTFUのPCに書き込むことで実行する．スレッドの優先権の変更，すなわち，メインスレッドとサブスレッドの切り替えは，IMUに対してどのTFUがメインスレッドかを知らせることで切り替えを実行する．この優先権の変更は，プロセッサの実行中いつでも変更することが可能である．またTCUは，TFUからスレッドの終了を告げられると，プロセッサの外部にスレッドの終了を知らせる信号を出力する．

4.5 命令合成ユニット：IMU

IMUは，メインスレッドとサブスレッドのVLIW命令を合成し，合成したVLIW命令を命令デコーダに発行するユニットである．IMUはTCUから，どのTFUのVLIW命令がメインスレッドのVLIW命令であるかを伝えられる．その情報をもとにメインスレッドとサブスレッドのVLIW命令の合成を行う．また，VLIW命令を発行すると，次のVLIW命令を各TFUに対して要求する．

IMUは，メインスレッドのCBを調べ，CBに0が含まれている場合，続いてサブスレッドのCBを調べる．このとき，メインスレッドのNOP命令と同じ実行ユニットの命令がサブスレッドのVLIW命令に存在すれば，NOP命令の代わりにサブスレッドの命令を命令デコーダに発行する．命令の一部を発行したサブスレッドのVLIW命令は，発行した部分の命令にはNOP命令を割り当て，CBには0が割り当てられる．このとき，CBが全て0なら，TFUに新たなVLIW命令のフェッチを要求する．

5 性能評価

ここでは，VIMプロセッサと通常のVLIWプロセッサとの比較を行い，提案手法の有効性を示す．比較の対象となる通常のVLIWプロセッサのモデルは，4で示したVIMプロセッサのMTFUを通常のIFU(Instruction Fetch Unit)に置き換えたものであり，実行ユニットの構成はVIMプロセッサと同じである．

評価の方法は，VIMプロセッサとVLIWプロセッサ上で複数のスレッドを実行し，各実行ユニットのスループットと各プロセッサにおけるレスポンスタイムを比較する．各実行ユニットのスループット，すなわち，各実行ユニットの利用率を調べることで，VIMプロセッサがどのくらいNOP命令を削減できたかを知ることができる．

表2 各実行ユニットのスループット

ユニット	VIM プロセッサ	VLIW プロセッサ
ALU0	37.7%	27.7%
ALU1	2.3%	0.2%
MUL	4.3%	3.0%
LSBU	75.1%	52.1%
全体	29.5%	20.8%

表3 各プロセッサのレスポンスタイム

ユニット	VIM プロセッサ	VLIW プロセッサ
スレッド1	1	1
スレッド2	1.44	1
全体	1.44	2

今回は，C言語で組まれた比較的小さなプログラムを用いた．このプログラムは円周率を求めるプログラムで，GNU gccコンパイラでコンパイルを行った．コンパイラから出力されたコードをリストスケジューリングに基づいてスケジューリングを行い，VIMプロセッサとVLIWプロセッサ上で実行した．なお，VIMプロセッサとVLIWプロセッサの実行ユニットの構成は同じであるため，コードのスケジューリング結果は同じものとなる．

表2に，実行ユニットのスループットを示す．表2から，全ての実行ユニットにおいて，スループットが向上していることがわかる．LSBUでは20%以上のスループットの向上が見られ，プロセッサ全体のスループットは約10%の向上が見られる．なお表2において，プロセッ

サ全体の利用率が低いのは，今回評価の対象としたプログラム自身の並列性が低いことが要因であると思われる．

また表 3 に，VLIW プロセッサのスレッドのレスポンスタイムを 1 としたときの，各スレッドのレスポンスタイムを示す．表 3 のスレッド 1 とスレッド 2 は，VIM プロセッサでのメインスレッドとサブスレッドにあたる．表 3 において，スレッド 1 のレスポンスタイムは各プロセッサとも同じである．これは，VIM プロセッサのレスポンスタイムが低下していないことを示している．また，VIM プロセッサでは，スレッド 2 のレスポンスタイムは長くなっているが，プロセッサ全体のレスポンスタイムは短くなっている．これは，スレッド 2 がサブスレッドとして，スレッド 1 と同時に実行されるためである．

6 むすび

本稿では，VLIW プロセッサのスループットを向上させるために，マルチスレッドの考え方を利用し，プロセッサのレスポンスタイムを低下させない命令供給方法を提案した．また，この命令供給方法に基づいた VIM プロセッサの構成を示した．さらに，VIM プロセッサの有効性についても検証した．

今後は，スレッド数を変更した場合の VIM プロセッサの性能評価と VIM プロセッサの実装を行う予定である．

参考文献

[1] “解説 VLIW 技術に新風，米 Sun 社の MAJC 登場,” 日経エレクトロニクス, 1999.10.4 (No.753), pp55-66.

[2] “特集 インテル新生「Itanium」に託す,” 日経エレクトロニクス, 1999.11.29 (No.758), pp119-153.

[3] 中澤喜三郎,“計算機アーキテクチャと構成方式,” 朝倉書店, 1995.

[4] マイク・ジョンソン,“スーパースカラ・プロセッサ,” 日経 BP 社, 1994.

[5] 平田博章, 木村浩三, 永峰聡, 西澤貞次, 鷺島敬之,“多重スレッド・多重命令発行を用いる要素プロセッサ・アーキテクチャ,” 情報学会論文誌, Vol.34, No.4, pp.595-605, Apr 1993.

[6] 平田博章, 奥村晃生, 柴田幸茂, 新實治男, 柴山潔,“マルチスレッドプロセッサおよび 1 チップマルチプロセッサのための命令キャッシュ構成・命令フェッチ方式の性能比較,” 信学会, Vol.J81-D-I, No.6, pp718-727, Jun 1998.

[7] 小林真輔, 武内良典, 北嶋暁, 今井正治,“命令インタリーブ発行機構を有するマルチスレッド向けプロセッサの提案,” 信学技法, ARC-99-135-7.

[8] Gerry Kane 著, 前川守 監訳,“mips RISC アーキテクチャ —R3000/R2000—,” 共立出版,1992 .