並列推論マシンＰＩＭ／ｉの命令レベル並列性の評価

大原輝彦　　武田浩一　　佐藤正俊
沖電気工業株式会社

概要

マイクロプロセッサの高速化技術として、命令レベル並列処理が注目されている。我々は、KL1向きアーキテクチャとしてRISCアーキテクチャ、パイプラインアーキテクチャ、タグアーキテクチャ、LIWアーキテクチャを融合したアーキテクチャを提案し、命令レベル並列性の抽出を試みた。設計したプロセッサは、３段パイプライン、分岐遅延１、ロード遅延０という命令パイプラインである。分岐命令やメモリアクセス命令の出現頻度が高いベンチマークプログラムに対して、１命令語あたり平均1.44命令の並列性を抽出できた。

1　はじめに

我々は、第五世代コンピュータプロジェクトの一環として並列推論マシンPIM/iの研究開発を行っている。PIM/iは並列論理型言語（KL1）で記述された知識処理プログラムを高速に実行することを目的として設計されたマシンである。

知識処理の高速化において、我々が最も重視していることは並列性の抽出である。並列性の抽出を行うアプローチとして、言語の処理方式レベルで並列処理を実現する方式と、プロセッサの命令レベルで並列処理を実現する２つの方式が考えられる。PIMの研究において、前者は、KL1のゴールリダクションに基づいたリダクション単位の並列実行であり、これを密結合マルチプロセッサシステム上でコースグレインの並列抽出を行って処理しており、KL1の共有メモリ上の実装方式、ロック付きメモリアクセス、プロセッサ間通信、負荷分散等の研究を行っている。

一方後者の、命令レベルの並列処理は、KL1向きアーキテクチャとコンパイラの実現方法に課題がある。命令レベルの並列処理は、機械語の並列実行で、プロセッサにおけるファイングレインの並列性抽出である。

本稿では、後者の命令レベルの並列性抽出法としてRISCアーキテクチャ、タグアーキテクチャ、パイプラインアーキテクチャ、LIWアーキテクチャを融合したアーキテクチャを示し、実際に設計を行ったPIM/iプロセッサの設計上の選択肢と評価について述べる。さらに、得られた結果をもとに今後の課題についても述べる。

Evaluation of Instruction Level Parallelism on Parallel Inference Machine PIM/i
Teruhiko OOHARA, Koichi TAKEDA, Masatoshi SATO
Oki Electric Industry Co., Ltd.

2　ＰＩＭ／ｉプロセッサのアーキテクチャ

知識処理向き高級言語で記述されたプログラムを高速に処理するマシンを実現するためのアプローチとして、マイクロプログラム制御による専用マシン（以下、専用マシンと略す）がある。その特徴は、

・高機能な機械命令のマイクロプログラムによるインタプリト
・マイクロ命令レベルの並列処理
・タグサポート
・大容量メモリの実装

である。これに対して、低機能な機械命令をワイヤードロジック制御で高速実行するアプローチがある。コンパイラによる命令最適化とハードウェアリソースの効率的使用が特徴である。そこで、マイクロプログラム制御方式の特徴として示した命令レベル並列処理、タグサポートをRISCマシンのアーキテクチャでサポートすることにより、さらに性能のよいアーキテクチャを実現できる可能性がある。

そこで、我々は、PIM/iプロセッサのアーキテクチャとして、RISCアーキテクチャ、パイプラインアーキテクチャ、タグアーキテクチャ、LIWアーキテクチャの４つのアーキテクチャを融合することを検討した。マイクロプログラム制御の専用マシンのアーキテクチャをRISCアーキテクチャの観点からもう一度見直すという方針である。

ここで、今述べた４つのアーキテクチャについて概観し、これらのアーキテクチャで性能の向上をはかるための方法と問題点について整理する。

(1)RISCアーキテクチャ

コンパイラにとって必要最小限な命令セットと機能を用意し、命令を処理するためのマシンサイクルをできるだけ短くすることによって高速化することを目指したアーキテクチャである。垂直型マイクロプログラムを見なおしたアーキテクチャとも考えることができる。命令の処理は、マ

イクロプログラム制御ではなく、ワイヤードロジック制御とする。設計が容易であり、最新のデバイステクノロジーをいち早く取入れ高速なマシンを実現することができる。

しかし、命令の機能を縮小することだけにたよってマシンサイクルを短くするには限度がある。また自明であるが、命令の機能を縮小するとそれに比例して、命令実行数の増加が起こる。

(2)パイプラインアーキテクチャ

命令間の時間的な並列性を抽出し処理することにより高速化することを目指したアーキテクチャで一般に広く普及している技術である。命令の処理過程を細分化しマシンサイクルを短縮することにより高速化する。

パイプライン処理を行うプロセッサでは分岐命令が実行されるまで、シーケンシャルに命令のプリフェッチを続ける。分岐が実際に起こると遅延スロット中の命令の実行をキャンセルする。これによって生ずるサイクルの無駄を分岐ペナルティという。高速化のためには、パイプライン段数を深くするとよいが、分岐ペナルティが大きくなるという問題がある。パイプラインの段数と分岐方式は設計上の大きなトレードオフである。

(3)タグアーキテクチャ

知識処理において、動的データ型を効率よく表現し、処理するために有用なアーキテクチャである。ターゲットとする言語に最適なタグ表現、処理機能を充実させることにより高速化する。この場合の高速化とは、高機能な命令の実装による命令実行数の削減である。

(4)LIWアーキテクチャ

複数の命令を同時に実行することにより処理の高速化を目指したアーキテクチャである。水平型マイクロプログラムを見直したアーキテクチャとも考えられる。

しかし、同時に実行可能なユニットを増やしてもそれらを効率よく動作させることは一般に難しいとされる。使用されない命令フィールドにはNOP命令を埋めておく必要がある。人手でプログラミングするのは一般的でなく、コンパイラの性能がそのままシステム性能となる。

これらのアーキテクチャを単純に融合しても効果はその和にはならない。その間には相反する問題があるからである。(1)と(2)はマシンサイクルを短縮することにより高速化するアプローチであるが、(3)と(4)はそうではない。したがって

- LIWアーキテクチャは遅延スロットの実質的な増加をもたらし、分岐ペナルティを大きくする。
- LIWアーキテクチャは分岐命令の出現間隔を短くし、分岐によるパイプラインの乱れを大きくする。
- タグ操作のための高機能命令の実装はクリティカルパス遅延を大きくする。

という問題がある。これらの問題をどう考えるかについて次の章で述べる。

3 設計方針と選択肢

3.1 命令クラス

プロセッサ内部に存在するプログラム可視な実行ユニットとして、一般に(1)分岐(2)メモリ操作(3)演算の3つがある。PIM/iプロセッサにはそれらに加えて、(4)タグ操作の実行ユニットを用意する。ここで、同じ実行ユニットを使用する命令をまとめて命令クラスと呼ぶ。

一般に、知識処理のような非数値計算プログラムに内在する命令の並列度はあまり高くないと考えられている。文献[1]によると、ある命令の実行中に、同じ命令クラスに属する命令が実行可能な確率は、0.1から0.25ということである。もう一つの特徴として、分岐命令の出現頻度が多いという点があげられる。知識処理プログラムではデータ型を判定するという処理が多く、また、データ処理自体も単純な処理が多い。そのため、分岐命令が出現する間隔が短くなると考えられる。そこで、PIM/iプロセッサでは同じ命令クラスの命令を並列実行することよりも、異なった命令クラスの命令の並列実行に重点をおいた設計とするのがよいと考えた。この方式は同じ実行ユニットを使用する命令を複数同時実行する方式に比べリソース間競合制御を容易に行えるという利点がある。

さらに、限られた命令フィールドを有効に利用するために4つの命令クラスから最大3つを選んで指定できる方式とした。

3.2 分岐方式

分岐ペナルティを軽減するために様々な方式の提案がなされている[2][3][4][5][6]。以下にそれらの方式について簡単に整理する。

(1)遅延分岐

遅延分岐は、分岐命令実行後、遅延スロット内の命令を分岐命令実行の結果の如何によらず実行する方式である。遅延スロット内の命令をいつも有効な命令で埋めておくことは、一般には難しい。

(2)静的予測分岐

前述した遅延分岐では、遅延スロット内を有効な命令で埋めておけない場合に、NOP命令を入れておく。しかし、ある条件が成立したときに遅延スロットの命令の実行をキャンセルする機構を設けておくと分岐先の命令で遅延スロットを埋めておける。すなわち、分岐が成立したとき遅延スロットの命令を実行し、そうでないときキャンセルするこ

とにより遅延スロットを有効に利用できる。これを、中和分岐と呼ぶ。中和分岐とは逆に分岐が成功したとき遅延スロット内の命令をキャンセルし、そうでないとき実行する命令も考えられる。これを通常分岐と呼ぶ。以上のように、分岐する可能性の高い方向をあらかじめ予測しておき、2つの命令を使い分ける。

(3)動的予測分岐

過去に実行した分岐命令の履歴情報を保存しておき、次にその分岐命令が実行される時、その情報をもとに分岐予測を行う。

(4)分岐フォールディング

命令プリフェッチの段階で無条件分岐命令の実行を完了させ、演算パイプラインから分岐命令を排除する方式である。CRISPプロセッサでは無条件分岐命令の分岐遅延を0にしている。

(5)複数命令プリフェッチ

分岐し得る方向のすべての命令をプリフェッチしておき、分岐方向が決定した時点で、正しくない方向の処理を放棄する方式である。一般に2方向に分岐し得る命令がパイプラインにn個同時に存在すると2^n個の命令をプリフェッチする必要がある。

(6)早期分岐決定

分岐の決定をパイプラインの早期で行うことにより分岐ペナルティを減らし、処理の効率化を計る方式である。ハードウェア規模とクリティカルパス遅延が問題であるがその効果はこれを相殺するという報告もある。

低コストで実現可能な方式は(1)-(2)である。その他の方式、特に(4)、(6)は分岐命令の出現率が高い実行環境においてよい性能を発揮すると思われるが、PIM/iプロセッサは1チップ化することを前提としているので、(1)-(2)の方式を採用するにとどめた。

3.3 コンディションコード

分岐方式の決定に際してもう一つ考慮しておくべき事項としてコンディションコードがある。ここで、条件分岐のメカニズムを整理してみると、(a)コンディションの生成(演算)、(b)コンディションの決定、(c)分岐アドレスの計算、(d)分岐となる。(a)-(d)の4つの処理を何サイクルで実現するかにより、次の3つの方式がある。

(1)1サイクル分岐方式

1サイクルで上記の(a)-(d)の処理を行う。(c)は(a)と並列に処理できるとしても3つの方式のうちでクリティカルパス遅延が最大となる。

(2)2サイクル分岐方式

最も一般に見られる方式である。コンディションコードを導入し、2サイクルで分岐する方式である。最近ではコンディションコードをPSWに格納する伝統的な方式だけでなく比較結果は比較命令で指定されるレジスタに格納するプロセッサも現れている。2サイクル分岐方式の問題点は演算命令と分岐命令の依存関係が存在するという点である。さらに、複数命令の同時実行を行う際には、コンディションコードに影響を与える命令が複数存在し得るという点も問題である。

(3)3サイクル分岐方式

コンディションの生成、決定の後、その結果で1ビットのブーリアンフラグをセットし、ブーリアンフラグの値により分岐する方式である。分岐に関するクリティカルパス遅延は最も少ない方式であるが分岐に3サイクルを要する。

分岐を2サイクル以上で実現する方式では、それらの命令間の依存関係が問題になる。これにより命令の移動によるプログラムの最適化が難しくなる。この点を考慮して、PIM/iプロセッサではコンディションコードを用いない1サイクル分岐方式を採用した。

3.4 タグのサポート

KL1では実行時にデータ型を判定してその結果を元に操作を行うため、これらをハードウェアで支援することは有効であると考えた。

タグのサポートもいくつかの選択肢が存在する。

(1)メモリ2語を使用する方式

値に1語、タグに1語使用する方式である。データの表現に2語必要とするがタグに1語使用できるので多数のデータ型を実現できる。またタグ操作のために特殊な命令を用意しなくて済むという利点がある。しかし、メモリが多数必要であり、アクセスにも特別な工夫をしない限り2サイクルかかる。

(2)メモリ1語にタグ専用メモリを付加する方式

メモリの実装コストは(1)程ではないが少なくはない。また、プロセッサにタグを操作する命令を用意する必要がある。

(3)メモリ1語を値とタグに分けて使用する方式

SPARC[7]のように1語32bitの一部をタグに割り当てる方式がある。メモリの実装コストは最も少なくてすむが、タグ操作を効率よく行うための命令が必要となる。

以上の3方式を検討した結果、我々は(2)の方式を採用した。(3)の方式は魅力的であるが、

・KL1では多くのデータ型を使用する

・タグと値を同時に処理できる確率が高い

などの理由によった。

4 設計

4．1 命令パイプライン

PIM/iプロセッサの命令パイプラインは、

(1)Pステージ：命令フェッチ

(2)Eステージ：レジスタ読み出し、命令のデコード、
命令実行

(3)Wステージ：レジスタへの結果の書込み

の3ステージから成る構成をとっている[8]。各ステージは1マシンサイクルで実行できる。

このような構成は、レジスタ読み出し／命令のデコードと命令実行を別ステージとする命令パイプライン設計に比べてクリティカルパス遅延が大きくなる。しかし、命令パイプライン設計の重要なポイントは、各ステージの処理時間が均等になる様に設計することである。この点を考慮して、Eステージの遅延と、並列キャッシュアクセスとのバランスをとり、ロード遅延を0とする設計とした。ロード命令の直後で、読み出したデータを使った演算が直ちに行える。さらに、Pステージとのバランスもとれるように配慮している。そのため、命令バスとデータバスを分離することは、必須である。

PIM/iにおける命令パイプライン設計をまとめると、浅いパイプライン段数、小さな分岐遅延、ロード遅延を0とする、という組合せで性能を出そうというアプローチとなる。

4．2 命令セット

命令セットはRISC型命令セットとし、以下に示す方針で定めた。それを命令クラス毎について概観する。

(1)分岐

プログラム実行の順序制御を行う。ローカルメモリの実装を考え、絶対分岐命令を用意した。

(2)メモリ操作

汎用レジスタとメモリ間の転送はロード／ストア命令のみとする。ただし、マルチプロセッサ支援のためにロック付きメモリ操作命令を用意した。

(3)演算

演算はレジスタ間の演算だけに限る。オペランドは3つ指定できる。算術、論理、ビットフィールド、アドレス計算等の演算を行う命令を用意した。

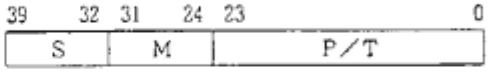
(4)タグ操作

タグ操作として、タグマスク、タグ取り出し、タグのセット等の基本的操作を支援する命令を用意した。タグ操作を支援する命令の機能が十分であれば、タグ操作のための命令数削減の効果が期待できる。

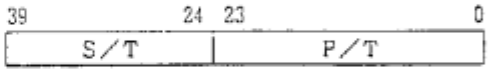
4．3 命令形式

1命令語は40ビット長固定である。1命令語は複数の命令フィールドから構成され、次の3つの型式に分けられる。

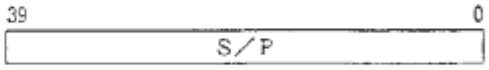
(1)3フィールド型式



(2)2フィールド型式



(3)1フィールド型式



命令フィールドには命令が1つだけ指定できる。Sは分岐命令を指定できる命令フィールドである。以下同様に、Mはメモリ操作命令を、Pは演算命令を、Tはタグ操作命令を指定できる命令フィールドである。P／Tは演算命令かタグ操作命令のいずれか1つを指定できることを意味している。

分岐命令にはオフセットの大きさにより3つのバリエーションがある。PIM/iプロセッサのアドレス空間は1G語であり、この範囲内に分岐するために30ビットオフセットが必要である。ごく近くに分岐する場合には6ビットオフセットの命令で分岐する。サブルーチンリターン命令のようにオフセットを必要としない命令もある。

以上のようにPIM/iプロセッサは、複数命令を静的スケジュールにより同時にディスパッチできる。動的に命令をディスパッチするマシンに対して、

・命令デコード回路の実装が楽になる。

・ハードウェアのリソース間統合制御が容易に行える。

という利点がある。同じ命令クラスの命令を並列実行するマシンに対しては、

・命令キャッシュの実装が楽になる。

・命令フィールドを有効に使用できる。

という利点がある。

5 評価

評価に先立ち、PIM/iにおけるKL1プログラムの実行について簡単に説明する。KL1プログラムは、KL1コンパイラにより、抽象機械命令KL1bに変換される[8]。変換されたKL1b命令はさらに、KL1bポストコンパイラによりPIM/iプロセッサの機械語に変換される。

5．1 評価方式

評価データの測定は、PIM/iプロセッサの機械語に変換されたKL1プログラムをレジスタトランスファレベルのシミュ

レータで実行することにより行った。プロセッサ数は1台でシミュレートした。使用したベンチマークプログラムはクイーン問題（qk8、qu8）、構文解析（bup）、ハノイの塔の問題（han）、素数を求める問題（pri）である。表1にベンチマークの諸元を示す。

表1　ベンチマークプログラム諸元

	qk8	qu8	bup	han	pri
命令サイズ（KW）	1.7	2.5	10.5	0.7	0.9
実行サイクル（MC）	3.2	5.8	4.3	2.7	3.8

（KW：Kilo Words、MC:Mega Cycle）

これらのベンチマークプログラムにおける命令出現率を表2に示す。この表では、各ベンチマークプログラムにおける総処理命令語数に対する百分率が示されている。総和が100%を超えるのは、1命令語で命令が複数指定できるためである。

表2　命令出現頻度

	qk8	qu8	bup	han	pri	平均
S:分岐	35.9	28.0	36.2	31.8	58.1	38.0
条件分岐	14.0	13.3	14.6	19.5	44.7	21.2
無条件分岐	5.9	4.9	6.2	8.1	3.7	5.8
タグ分岐	15.9	9.7	15.4	4.2	9.8	11.0
M:メモリ操作	25.6	22.2	22.5	19.8	9.3	19.9
リード	14.9	12.8	11.7	10.2	5.0	10.9
ライト	10.0	9.2	9.8	8.2	3.7	8.2
ロック	0.7	0.2	1.0	1.4	0.6	0.8
P:演算	72.3	81.6	70.5	68.6	79.4	74.4
address	27.3	22.3	26.0	28.5	10.4	22.9
move	13.7	11.4	14.4	5.7	12.8	11.6
immediate	13.4	20.1	15.5	14.4	8.8	16.0
その他	17.8	19.7	14.6	29.8	47.4	23.9
T:タグ操作	9.1	13.7	12.5	18.8	4.9	11.8

（単位：%、address:ｱﾄﾞﾚｽ計算、move:ﾚｼﾞｽﾀ間転送
immediate:即値のﾚｼﾞｽﾀへの転送）

表3　各命令型式における命令フィールドの出現率

	qk8	qu8	bup	han	pri	平均
3field	56.1	60.0	55.8	60.6	39.8	54.5
－－－	0.0	0.0	0.0	0.0	2.4	0.0
－－P	28.9	36.1	28.5	35.3	24.6	30.7
－－T	0.2	0.1	1.2	2.6	0.6	0.9
－M－	10.1	8.6	8.8	5.6	4.2	7.5
－MP	14.1	13.1	11.9	9.0	4.4	10.5
－MT	0.4	0.3	0.9	3.8	0.0	1.1
S－－	0.9	0.4	2.9	1.4	3.0	1.7
S－P	0.8	1.2	0.7	1.5	0.0	0.8
S－T	0.0	0.0	0.0	0.0	0.0	0.0
SM－	0.7	0.2	1.0	1.4	0.6	0.8
SMP	0.0	0.0	0.0	0.0	0.0	0.0
SMT	0.0	0.0	0.0	0.0	0.0	0.0
2field	37.8	35.1	38.0	31.3	56.4	39.7
S　－	9.1	3.6	7.8	5.0	6.1	6.3
S　P	20.3	18.2	19.7	13.9	46.0	23.6
S　T	0.4	0.3	0.7	3.6	0.0	1.0
T　P	8.0	13.0	9.8	8.8	4.3	8.6
1field	5.9	4.9	6.2	8.1	3.7	5.9
S	5.1	4.5	5.1	5.4	3.7	4.9
P	0.8	0.4	1.1	2.7	0.0	1.0

（単位：%）

表4　各命令型式における平均命令記述度

	qk8	qu8	bup	han	pri	平均
3field	1.29	1.25	1.25	1.26	1.07	1.23
2field	1.76	1.90	1.79	1.84	1.89	1.84
全体	1.42	1.45	1.41	1.39	1.51	1.44

5．2　フィールド特性

各命令型における命令フィールドの使用率を表3に示す。なおSは分岐命令、Mはメモリ操作命令、Pは演算命令、Tはタグ操作命令フィールドである。－は命令フィールドにNOPが指定されていたことを示す。

表3によると、3フィールド型の平均出現率は54.5%であり、2フィールド型は38.0%、1フィールド型は6.2%である。しかし、3フィード型で2フィールドしか指定されていないものは平均で全体の13.3%、1フィールドしか指定されていないものは平均で全体41.2%であった。これを表4に各命

命型における平均命令記述数としてまとめた。3フィールド型の平均命令記述数は大変低い。これに対して2フィールド型では平均で1.84と高い数値を示した。全体では1.44である。3フィールド型の記述数が低い理由は5．4節で考察する。

演算フィールドは処理した命令語数のうち74.4%で有効に稼働していたが、表2のその他の欄で示される算術論理演算器の稼働率はそのうち23.9%と少ない。

メモリ操作は全命令語の19.9%に含まれており、命令パスとデータパスを分離した効果はあったと考えられる。

5．3　分岐特性

各ベンチマークにおいて発生した分岐ペナルティの割合を表5に示す。表2から、分岐命令の出現頻度は平均で38.0%である。その約3分の1で分岐ペナルティが生じている。priは除算ルーチンが多く使用されており、分岐ペナルティが他のベンチマークより大きな値となっている。それを除けば平均は10%以下となり、予測分岐命令を実装した効果があったと言える。

表5　分岐ペナルティ

qk8	qu8	bup	han	pri	平均
13.3	9.8	9.8	5.7	28.7	13.5

（単位：%）

表6．遅延スロットの使用率

	qk8	qu8	bup	han	pri	平均
条件	46.4	49.2	59.5	64.8	51.5	54.3
無条件	10.8	11.4	10.5	16.4	3.2	10.5

（単位：%）

遅延スロットの使用率を表6に示す。使用率は、実行された分岐命令に対する割合で示されている。タグ分岐を含む条件分岐命令では遅延スロットの使用率は平均で54.3%であった。また表3から、3フィールドともNOP命令である命令語の出現率がほとんどないことから遅延スロットが有効な命令で埋められていると言える。これに比べて無条件分岐命令では平均で10.2%と低い。表2から、無条件分岐命令の出現率は平均で5.8%あり、最適化の余地がある。

表7　中和／通常分岐の効果

	qk8	qu8	bup	han	pri	平均
中和	6.1	8.6	7.3	16.7	7.6	9.3
通常	26.4	26.9	30.8	35.8	32.0	30.4

（単位：%）

予測分岐命令の効果について表7に示す。この表では中和分岐と通常分岐について遅延スロットが有効に利用された割合を示す。KL1ベンチマークでは通常分岐のほうが効果が大きかったことがわかる。

表8　分岐命令出現間隔

間隔	qk8	qu8	bup	han	pri	平均
1	27.5	18.4	24.1	16.4	56.0	28.5
2	10.8	7.6	14.6	17.2	11.6	12.4
3	8.6	12.3	10.5	8.7	6.7	9.4
4	8.0	6.6	5.9	7.0	10.7	7.7
5	6.2	5.2	5.8	1.7	4.7	4.7

（単位：%、間隔は命令語数を表している）

分岐命令の出現間隔について表8に示す。この表によると分岐命令の次が分岐命令である場合が平均で28.5%であり、分岐命令の出現間隔が短いことが確認された。

5．4　タグサポートの効果

タグサポートの効果について表9に示す。この表によるとタグ分岐命令による効果は平均で11.0%である。またタグ操作命令が単独で出現する率は表2から0.5%であり、他の命令と同時実行可能な割合が極めて高いことを示している。

表9　タグ操作

	qk8	qu8	bup	han	pri	平均
タグ分岐	15.9	9.7	15.4	4.2	9.8	11.0
タグ操作	9.1	13.7	12.5	18.8	4.9	11.8

（単位：%）

5．5　レジスタファイルの稼働状況

レジスタファイルのポート稼働状態について表10に示す。PIM/iプロセッサは1サイクルでレジスタファイルに最大5つのアクセスが起こり得る。この結果によるとレジスタファイルの多ポート化による効果は平均で3.5%である。期待した程の効果が出ていない。この原因は、2ポート読み出しの割合が平均で15.8%と低く、1ポート読み出しが平均で66.5%と高いことから明らかなように、演算フィールドに出現する命令の大半が即値、あるいは即値との演算等の2オペランド命令であることによる。

表10　レジスタファイルのポート稼働状況

	qk8	qu8	bup	han	pri	平均
4ポート以上	3.9	5.3	2.8	3.5	1.9	3.5
2ポート読出	13.7	20.4	12.4	21.9	16.4	15.8
1ポート読出	66.6	43.8	66.8	75.9	78.5	66.5

（単位：%）

表11　メモリ操作

	qk8	qu8	bup	han	pri	平均
address-stor	6.7	6.2	7.1	7.2	2.5	5.9
ストア単独	55.1	72.7	54.5	79.6	74.0	67.4
ロード単独	41.7	40.5	41.6	21.7	49.2	38.9

（単位：%）

PIM/iでは並列キャッシュの構成上、ストア命令の次のメモリアクセス命令は1サイクル伸たされる。この影響が処理時間に対してどの位あるか調べた。最初の欄はアドレス演算命令の次にストア命令が実行され、かつ他のフィールドが有効な命令で埋められていない場合の全処理時間に対する割合である。これは平均で全処理時間の5.9%をしめており、影響は大きいと言える。2，3番目の欄はストア命令とロード命令が単独で出現した割合である。メモリ操作命令が他の命令と並列に実行できる割合が少ない。

最後に、コンディションコードを導入しなかったことによる得失について考察する。条件分岐命令はqk8の場合、表2から14.0%出現しており、コンディションコードを導入すると、2サイクルに分けて実行される。しかし条件分岐命令は2フィールド型で指定するためそのまま実行サイクル数が倍になるようなことはない。実行サイクル数がどのくらい増えるかシミュレートしたところ10.9%増加した。コンディションコードを導入することによる得失は以下の式で評価できる。

i1、n1：コンディションコードありの場合の
　　実行サイクル数、マシンサイクル
i2、n2：コンディションコードなしの場合の
　　実行サイクル数、マシンサイクル

$i1*n1-i2*n2>0$　であればコンディションコードを導入しないほうがよいと言える。この場合、コンディションコードを導入してもマシンサイクルを10%以上短縮することは難しく、さらに実装の複雑さを考慮すると、導入するメリットはなかったと言える。

6　課題の整理

我々の設計に関して今回の評価で明かになった課題をまとめ以下に示す。

・40ビット長に3フィールド型命令を割り付けたこと

3フィールド型の出現率は平均で50%以上あるが、3フィールド同時指定がほとんど出現しないことによりこの問題は明かである。とくに条件分岐命令のほとんどが2フィールド型式でしか記述できないことが大きな原因である。これに対して、2フィールド型の平均記述率は平均1.74と有効に記述されている。

・ストア命令実行後のメモリアクセス

ストア命令の2分の1以上が単独で出現している。最適化の余地がまだあるにせよ大きな問題である。ローカルメモリには並列キャッシュに対するメモリアクセス命令の実行制限はないので、ローカルメモリを有効に使用することにより数%の改善が見込まれる。

・フィールドを有効に利用する目的で命令の並列度を可変としたが、命令デコードの遅延時間分だけメモリアクセス時間を損している。

7　おわりに

知識処理プログラムを効率よく実行するためには分岐処理の効率化が大きなウェイトをしめる。本稿では、パイプラインの乱れを少なくし、かつ処理の効率化を目的としてRISC、パイプライン、タグ、LIMという4つのアーキテクチャを融合したPIM/iプロセッサについて述た。さらに評価を行った結果、1命令語あたり平均で1.44命令の並列性抽出が行えた。

謝辞　：
日頃、助言をいただく総合システム研究所 羽下所長、ならびに（財）新世代コンピュータ技術開発機構（ICOT）第1研究室、沖電気のPIM担当諸兵に感謝する。

参考文献：
[1] N.P.Jouppi:"The Nonuniform Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance", IEEE Transaction on Computer, vol.38, no.12, Dec.1989.
[2] S.McFarling and J.Hennessy:"Reducing the Cost of Branches", in Proc.13th Symp Computer Architecture, 1986.
[3] D.J.Lilja:"Reducing the Branch Penalty in Pipelined Processors", IEEE COMPUTER, Vol.21, No7, Jul.1988.
[4] J.A.DeRosa and Henry M.Levy:"An Evaluation of Branch Architectures", in Proc.14th Symp. Computer Architecture, May 1987.
[5] N.P.Jouppi and D.W.Wall:"Available Instruction Level Parallelism for Superscalar and Superpipelined Machines", in Proc. of 3th International Conference on Architectural Support for Programming Language and Operating Systems, IEEE Computer Society Press, Apr.1989.
[6] A.D.Berenbaum et al.:"Architectural Innovation in the CRISP Microprocessor", In Proc. COMPCON, 1987.
[7] CYPRESS SEMICONDUCTOR:"CY7C600 RISC Family Users Guide", 1988.
[8] 武田、大原：「並列推論マシンPIM/iの要素プロセッサのアーキテクチャ」情報処理学会第40回全国大会予稿集
[9] Y.Kimura, T.Chikayama:"An Abstract KL1 Machine and its Instruction Set", In Proc. of Logic Programming, Aug.1987.