計算機アーキテクチャ　63－5
（1986・11・20）

Prolog指向RISCプロセッサ"Pegasus"

Pegasus: RISCy Processor for Prolog Execution

瀬尾 和男　横田 隆史
Kazuo SEO Takashi YOKOTA

三菱電機株式会社　中央研究所
Central Research Lab.
Mitsubishi Electric Corp.

1. はじめに

Prologは第５世代プロジェクトを中心として普及しつつある論理型プログラミング言語であり、人工知能応用、特に自然言語処理やプロダクション・システムの構築などにおいて威力を発揮する。Prologにおける処理は、Unificationと呼ばれるパターンマッチングの繰り返しと、それに失敗したときに起こるBacktrackingと呼ばれる状態の復旧とに集約される。この様にPrologの処理自体は比較的単純ではあるが、その実現には複数のスタックを対象とした非数値処理が中心となるため、汎用計算機による処理では効率良く実行できず専用のアーキテクチャの構築が必要となる。

こういった理由で現在各所においてPrologマシンの開発が進められている[1-5]。それらの多くはD.H.D.Warrenにより提案されたPrologの抽象化命令セット[6]に基づくものであり、μプログラムによる実現が主流となっている。この様なμプログラムによる実現に対し、Prologの均質化された処理形態を前提にカスタムVLSI技術を活用したRISC(Reduced Instruction Set Computers)方式による実現が考えられる。

Riscアーキテクチャは、VLSI時代におけるシステム構築のありかたについて新たな視点からの提案を行ったものであり、総合的なパフォーマンスの向上とシステム開発期間の短縮化とを目指したものである。すなわち、応用プログラムの解析結果に基づいて命令セットを少数の単純なものに絞り込み、それによって従来チップ面積の大部分を占めていた制御回路の縮小化を行う。この結果、命令実行サイクルが短縮され全体としての処理効率が向上すると共に、チップの設計に要する時間も大幅に短縮される。

Riscチップの特長としては、

1.シングル・サイクル演算操作
2.ロード／ストア命令
3.ハードワイヤード制御回路
4.比較的少数の命令とアドレッシング・モード
5.固定長の命令形式
6.コンパイル時における最適化

等が挙げられる[7]。制御系が簡単化され、命令形式も均一化されているため、処理のパイプライン化が容易になる。また、余ったシリコン領域を用いてレジスタ・ファイルやオンチップ・キャッシュを実装することができ、それらを活用した高速化が可能となる。

Risc方式のアーキテクチャは、それがサポートする高級言語によって異なってくる。IBM801、バークレイ大学のRISC I・II、スタンフォード大学のMIPSでは、PL/1、C言語、FORTRAN等を対象としてアーキテクチャを構築しているのに対し[7-10]、LispやSmalltalkを対象としたRiscプロセッサも研究されている[11,12]。Prologに対するRiscアーキテクチャの構築する上で考慮すべき点としては、

＊スタック（メモリ）へのアクセス
＊タグによるデータタイプのサポート
＊Backtrackの為の状態の退避・復旧の効率化

等が挙げられる。

本報告では、我々の研究所において現在開発を行っているμプロセッサ"Pegasus"について述べる。Pegasusは、Prolog処理の効率化を目的としたRisc方式のμプロセッサであり、命令形式や制御

方式を簡単化することによって制御ロジックを小し、余ったシリコン領域をProlog処理の効率化に用いるという設計方針に基づいて設計されている。チップ全体の制御方式としては６ステージ・３ウェイのパイプライン制御を採用しており、Prolog処理の効率化の為に、スタック操作、タグ操作、Backtrack用のレジスタの退避・復旧等を効率良く処理できる命令セットを備えている。これらによって、プロセッサの性能としては、10MHzクロックで動作させた場合決定的なAppendの実行において約240KLIPS(Logical Inferences Per Second)、非決定的な処理を含むQuick Sortにおいても約150KLIPSを達成できることがシミュレーションによって確かめられている。

2. 基本アーキテクチャ

PegasusのProlog処理方式は、WAM(Warren Abstract Machine)[6]に基づくものである。本節では、WAMおよびPrologのプログラムの解析に基づいて決定されたPegasusの基本アーキテクチャについて述べる。

2.1 スタック

WAMで定義されているProlog処理用のデータ領域としては、

Heap(Global Stack):
構造体やグローバル変数が格納される
Local Stack:
制御フレーム（Choice_Point, Environment）が格納される
Trail Stack:
変数への値の格納の履歴が保持される
Push Down List:
構造体のUnificationに用いられる

の４つのスタック領域があり、それらの領域とプログラム領域とが一般に主記憶上に実現される。

Pegasusでは、Riscの設計思想にそって、メモリへのアクセスはロード／ストア命令に限られている。これらのスタックへのアクセス形態を調べた結果、次の２種類のメモリ参照命令によって効率的にサポートできることがわかった。

＊Local Stack上の制御フレームへのアクセス
　レジスタ<−>メモリ（ｐ±ｎ）
＊通常のスタック・アクセス
　レジスタ<−>メモリ（ｐ），　ｐ＝ｐ±１
（ただし、ｐ＝ポインタ値、ｎ＝変位）

後述のようにレジスタはすべてレジスタ・ファイル上に実現される為に、これらの命令を１サイクルで実行するには、メモリ・アドレスの生成パス上への加算器の導入とレジスタ・ファイルの２ポート化が必要となる。

2.2 レジスタ・ファイル

WAMでは、以下のようなレジスタが定義されている。

E: 現在のenvironmentへのポインタ
B: 現在のchoice pointへのポインタ
H: Heapの先頭アドレス
HB: HeapのBacktrack-point
TR: Trail Stackの先頭アドレス
P: 現在実行中の手続き（節）のアドレス
CP: 次に実行すべき手続きのアドレス
S: Heap上の構造体へのポインタ
A_0-A_n: 手続き呼出しの引数レジスタ

Pegasusでは、これらのレジスタはすべてレジスタ・ファイル上に実現される。

Prologでは、複数のUnification可能な節がある場合、Backtrack処理に備えて状態すなわちレジスタのLocal Stackへの退避が必要となる。退避されるのは、上記レジスタの内の６つの状態レジスタと節の引数分だけの引数レジスタである。また、実際にBacktrackが起こった場合には、退避したレジスタをすべて復旧しなければならない。これら

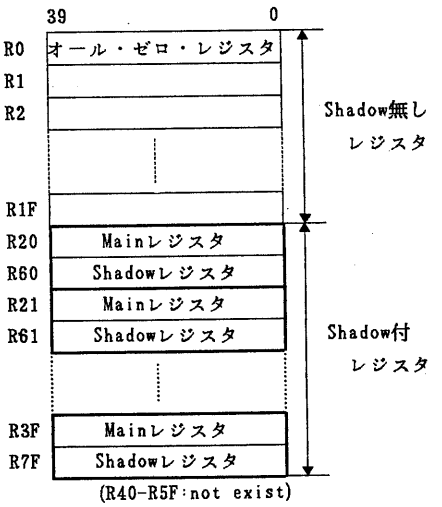


図１　レジスタ・ファイル構成

の退避・復旧は連続するメモリ・アクセスを必要とし、その間次の処理には進めない。

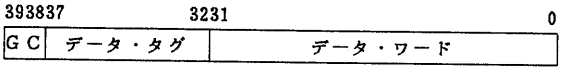
このBacktrackを効率良くサポートするために、レジスタ・ファイルの一部は互いにコピー可能なワードの組（Main レジスタとShadow レジスタ）から構成されている。Backtrackに備えて状態を退避する場合にはMain レジスタからShadow レジスタへと、Backtrackが起こった場合にはShadow レジスタからMain レジスタへとコピーが行われる。さらに、データ・アクセスの為のメモリ・サイクルを使用しない命令の実行と並行に、次のShadow操作に備えてShadow レジスタとメモリ間でデータの転送が行われる。

図１にレジスタ・ファイルの構成を示す。このレジスタ・ファイルはスタティックＲＡＭによって実現され、前述のスタック・アクセスの効率化の為に２ポート化されている。さらに、R20からR3Fまでは互いにコピー可能なShadow レジスタを備えた構成となっている。

２．３ タグ

Prologではデータ型として変数、定数、構造体等をサポートしており、その識別にはタグを用いるのが有効である。また、Heap上のデータに対してはGC(Gabage Collection)を行う為にマーク用のタグも必要となる。これらを考慮して、Pegasusでは、図２に示すように、タグ８ビット、データ３２ビットの４０ビット長ワードを採用している。

Pegasusでは、SOAR[12]において採用されているような算術演算を実行する際にオペランド・データのタイプ・チェックを行うといったタグ操作は導入されていない。その代わりにPrologの処理形



－－－－００ ： 変数
－－－－０１ ： 定数
－－－－１０ ： リスト
－－－－１１ ： 構造体

図２ Pegasusのワード構成

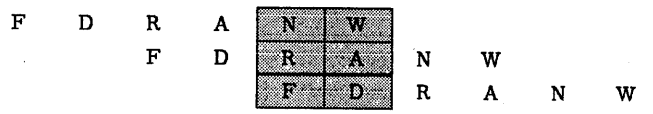
態に即した基本データ・タイプ（変数、定数、リスト、構造体）によるタグ・ディスパッチングがサポートされている。

２．４ パイプライン処理

Pegasusでは６ステージのパイプライン処理方式を採用している。命令は２ステージ毎にパイプに投入され、３つの命令が同時実行される。パイプラインの各ステージと実行形態を図３に示す。

パイプライン・マシンでは、前の命令によって計算された結果を次の命令で使うといった場合に誤動作をおこす可能性がある。例えば、Pegasusのパイプライン処理の場合には、ある命令のＲステージは直前の命令のＷステージよりも先に実行されるために、レジスタ・ファイルに格納された以前の値が読み出される可能性がある。これを防ぐには、インタロックと呼ばれる機構が必要である。しかし、インタロックをハードウェアで実現するとかなり大がかりなものとなり、また性能的にもなんら改善されない。

これに対し、スタンフォード大学のMIPSプロジェクト[9,10]ではコンパイラにすべて責任を持たせたソフトウェア・インタロックと呼ばれる方式



Ｆステージ：命令のフェッチ
Ｄステージ：命令のデコード
Ｒステージ：レジスタ・ファイルの読出し／メモリのアクセス番地の計算
Ａステージ：ＡＬＵ演算／メモリ・アクセス／Ｊｕｍｐ先番地の設定
Ｎステージ：No-operation
Ｗステージ：レジスタ・ファイルへの書込み

図３ パイプライン動作と各ステージ

を採用している。コンパイラは、まずプログラムの必要な箇所にNo-op命令を入れることによってインタロック機構を実現すると共に、このNo-op命令と入れ替えの可能な命令を探し出してプログラムを再編成することによって効率を向上させる。また、バイパス・レジスタと呼ばれる機構により、演算結果をレジスタ・ファイルに書き込む前に次の命令で参照できるようになっている。Pegasusにおいても、コンパイラによるインタロック機構の実現やバイパス・レジスタを採用している。

3. Pegasusの命令セット

3.1 命令語の構成

Pegasusの命令は、メモリとのインタフェースを簡単化することを目的に、データと同じくTAG部8ビット、VALUE部32ビットの計40ビット固定長の構成となっている。

TAG部には、命令とデータとを区別する為のビットと、Shadow操作に関連して命令の実行を動的に制御する為のフラグ・ビットが保持されている。すなわち、命令のデコード時にTAG部の内容とその時点のプロセッサの状態（例えば、Shadow操作を継続中等）によって、Shadow操作に伴うメモリ・アクセスの埋め込みや待ち合わせが行われる。フラグ・ビットは通常アセンブラによって自動的に設定されるが、必要に応じて変更することも可能である。

以下にTAG部の構成を示す。

39	38	37	36	35	34	33	32
--	--	IN	SE	WS	RS	IM	--

IN: 命令タグ、1…命令、0…データ
SE: Shadowアクセスの可否
WS: Shadow-writeの終了の待ち合わせ
RS: Shadow-readの終了の待ち合わせ
IM: 割込みの禁止

Pegasusの命令は基本的に3オペランド構成となっており、VALUE部には各々8ビット長のオペコードと3つのオペランドが置かれる。オペコードによってプロセッサ内のデータの流れが決定されるが、具体的な演算内容等の指定は修飾子によって行われる。例えば、レジスタ間の演算を行う命令はすべて共通のオペコードを持つが、加算、減算等の演算内容は修飾子によって指定される。演算内容等のヴァリエーションがない命令は修飾子を持たない。

修飾子を持つ命令と持たない命令の2つの場合について、VALUE部の構成を以下に示す。

［修飾子のある場合］

31‥28	27‥24	23‥‥16	15‥‥8	7‥‥‥0
修飾子	ｵﾍﾟｺｰﾄﾞ	ｵﾍﾟﾗﾝﾄﾞ1	ｵﾍﾟﾗﾝﾄﾞ2	ｵﾍﾟﾗﾝﾄﾞ3

（ただし、修飾子の値は0000～1100）

［修飾子のない場合］

31‥29	28‥24	23‥‥16	15‥‥8	7‥‥‥0
1 1 1	ｵﾍﾟｺｰﾄﾞ	ｵﾍﾟﾗﾝﾄﾞ1	ｵﾍﾟﾗﾝﾄﾞ2	ｵﾍﾟﾗﾝﾄﾞ3

3.2 命令セット

表1にPegasusの命令セットの一覧表を示す。命令は次の6つのグループに大別される。

＊ロード／ストア
＊レジスタ転送
＊算術／論理演算
＊ジャンプ／コール
＊Shadow操作
＊その他

前述のようにメモリ・アクセスはすべてロード／ストア命令によって行われる。ロード／ストア命令に付けられた修飾子は、メモリ・アクセスに用いたアドレス（ポインタ）を同時に更新するかどうかを指定する。これによってスタック操作の効率化が可能となる。

タグに対する操作はデータに対する操作と同じくプリミティヴなものがほとんどであるが、前述のようにProlog処理に即したタイプ・チェックの手段としてタグ・ディスパッチ命令（JTD）が用意されている。この命令は、2つのオペランド・レジスタのTAG部2ビット（図2参照）を連結した4ビットによってこの命令に続いて格納されている表を引き、そこに置かれたディスパッチング・アドレスにジャンプするものである。これによって、16ウェイの分岐が実現され、Unificationルーチン等のプログラミングにおいて有効である。また、オペランド・レジスタの片方にゼロ・レジスタ（R0）を指定することにより4ウェイ分岐を実現することができ、WAMのIndexing命令等をプログラムするのに活用できる。

Pegasusはパイプライン処理を行っている為、ジャンプ命令の後には1スロット（1命令）のペナ

表1．Pegasusの命令セット

OPcode	Modifier	OP1	OP2	OP3	Meaning
Load/Store Group					
LDP	.	R_{DST}	C_{OFF}		R_{DST}← M[PC+ C_{OFF}]
LDR	<mdf>	R_{DST}	R_{IDX}	C_{OFF}	R_{DST}← M[R_{IDX}+ C_{OFF}]
STP	.	R_{SRC}	C_{OFF}		R_{SRC}→ M[PC+ C_{OFF}]
STR	<mdf>	R_{SRC}	R_{IDX}	C_{OFF}	R_{SRC}→ M[R_{IDX}+ C_{OFF}]
SCP	.	C_{TAG}	R_{SRC}	C_{OFF}	$C_{TAG}$$R_{SRC}$→ M[PC+ C_{OFF}]
SCR	<mdf>	C_{TAG}	R_{SRC}	R_{IDX}	$C_{TAG}$$R_{SRC}$→ M[$R_{IDX}$]
Register Transfer Group					
MCV	.	R_{DST}	C_{SRC}	R_{SRC}	R_{DST}← $C_{SRC}$$R_{SRC(VAL)}$
MTV	.	R_{DST}	R_{S1}	R_{S2}	R_{DST}← $R_{S1(TAG)}$$R_{S2(VAL)}$
MTT	.	R_{DST}	R_{S1}	R_{S2}	R_{DST}← $R_{S1(TAG)}$$R_{S2(TAG)}$
MVV	.	R_{DST}	R_{S1}	R_{S2}	R_{DST}← $R_{S1(VAL)}$$R_{S2(VAL)}$
MAD	.	R_{DST}	C_{OFF}		R_{DST}← PC+ C_{OFF}
Arithmetic/Logical Operation Group					
OVR	<mdf>	R_{DST}	R_{S1}	R_{S2}	$R_{DST(VAL)}$← $R_{S1(VAL)}$<mdf> $R_{S2(VAL)}$
OVC	<mdf>	R_{DST}	R_{SRC}	C_{SRC}	$R_{DST(VAL)}$← $R_{SRC(VAL)}$<mdf> C_{SRC}
OTR	<mdf>	R_{DST}	R_{S1}	R_{S2}	$R_{DST(TAG)}$← $R_{S1(TAG)}$<mdf> $R_{S2(TAG)}$
OTC	<mdf>	R_{DST}	R_{SRC}	C_{SRC}	$R_{DST(TAG)}$← $R_{SRC(TAG)}$<mdf> C_{SRC}
Jump/Call Group					
JPP	.	C_{OFF}			PC← PC+ C_{OFF}
JPR	.	R_{IDX}	C_{OFF}		PC← R_{IDX}+ C_{OFF}
JIP	.	C_{OFF}			PC← M[PC+ C_{OFF}]
JIR	.	R_{IDX}	C_{OFF}		PC← M[R_{IDX}+ C_{OFF}]
CAP	.	R_{DST}	C_{OFF}		R_{DST}← PC; PC← PC+ C_{OFF}
CAR	.	R_{DST}	R_{IDX}	C_{OFF}	R_{DST}← PC; PC← R_{IDX}+ C_{OFF}
CIP	.	R_{DST}	C_{OFF}		R_{DST}← PC; PC← M[PC+ C_{OFF}]
CIR	.	R_{DST}	R_{IDX}	C_{OFF}	R_{DST}← PC; PC← M[R_{IDX}+ C_{OFF}]
JCR JUR JSR	<mdf>	R_{rf1}	R_{rf2}	C_{OFF}	*if* R_{rf1} <mdf> R_{rf2} *then* PC← PC+ C_{OFF}
JCC JUC JSC	<mdf>	R_{ref}	C_{ref}	C_{OFF}	*if* R_{ref} <mdf> C_{ref} *then* PC← PC+ C_{OFF}
	Compare-and-Jump JCx for TAG part (as an unsigned integer) JUx for VAL part as an unsigned integer JSx for VAL part as a signed integer				
JTD	.	R_{S1}	R_{S2}		PC← M[PC+ *Dispatch*(R_{S1}, R_{S2})]
JTR	<mdf>	R_{S1}	R_{S2}	C_{OFF}	*if Trap*(R_{S1}, R_{S2}, MAR) *then* PC← PC+ C_{OFF}
Shadow Operation Group					
SRR	.	R_N	R_{IDX}	C_{OFF}	ShadowRead $R_N$words
SRC	.	C_N	R_{IDX}	C_{OFF}	$C_N$words
SWR	.	R_N	R_{IDX}	C_{OFF}	ShadowWrite $R_N$words
SWC	.	C_N	R_{IDX}	C_{OFF}	$C_N$words
SGR	.	R_N	R_{IDX}	C_{OFF}	ShadowGet $R_N$words
SGC	.	C_N	R_{IDX}	C_{OFF}	$C_N$words
SPR	.	R_N	R_{IDX}	C_{OFF}	ShadowPut $R_N$words
SPC	.	C_N	R_{IDX}	C_{OFF}	$C_N$words
Miscellaneous Instruction					
NOP	.				No_Operation
HLT	.				Halt
CTT	<mdf>	R_{DST}	R_{S1}	R_{S2}	Compare TAG of R_{S1} and R_{S2}, Cause TRAP

ルティを伴う。これに対して、ジャンプの次の命令が必ず実行されることを利用したディレイド・ジャンプ手法によって処理効率を大幅に改善することができる。

Shadow レジスタの存在は命令セットにも反映され、これを扱うShadow操作命令がPegasusの命令セットの大きな特徴となっている。Shadow Read/Write命令では、Shadow レジスタとMain レジスタの間でコピー操作を行った後に、以降の命令の実行と並行してShadow レジスタと主記憶間のデータ転送が行われる。これによって、Choice-pointの生成やBacktrack操作をサポートできる。これに対して、Shadow Get/Put命令ではShadow レジスタと主記憶間のデータ転送だけが行われ、Cut操作をサポートするのに用いられる。

Shadow命令によって起動されたShadow レジスタと主記憶間のデータ転送は、それ以降に実行される命令に埋め込まれた形で処理される。埋め込むことのできる命令としては、主記憶へのデータ・アクセスがなくかつレジスタ・ファイルの１ポートが空いていることが条件となる。あるShadow命令の実行から次のShadow命令の実行までには通常数十命令が実行される為、Shadow操作による主記憶のアクセスは他の命令の実行とほぼ完全にオーバーラップさせることが可能である。

３．３ コンパイラ

PrologをPegasus命令に変換するクロス・コンパイラをDEC2060上のDEC10 Prologで開発中である。コンパイル方法は、まずPrologをwamモデルに近い中間形式に変換した後、それらをマクロ展開することによってPegasusのアセンブラ命令に変換する。このようにして得られたアセンブラ・プログラムには多くのNo-op命令が含まれている為、主にディレイド・ジャンプ手法等による最適化を行うことによって最終的なオブジェクト・コードを生成する。

４．実装およびシステム構成

Pegasusプロセッサは、２μCMOS２層メタルの製造プロセスによる実装を前提として設計を行っている。現在、主要なモジュールのレイアウト設計を行いつつあるが、チップ面積が約９㎜×９㎜、ピン数が９２ピンとなる予定である。レジスタ・ファイルが約５㎜×４㎜と最も大きな面積を占めるが、Shadowの制御等によって制御部も他のRiscプロセッサに比べれば大きくなるものと予想される。

Pegasusプロセッサを用いたシステムの核部分は、図４に示されるように主記憶、キャッシュ・メモリ、クロック・ジェネレータといったモジュールから構成される（ただし、I/Oは含まれていない）。Pegasusではパイプラインの１ステージを最高100 nsecと設定している為、主記憶とPegasusプロセッサ間にはキャッシュ・メモリが不可欠である。主記憶とキャッシュ・ユニットはアドレスおよびデータ・バスを共有し、メモリ・アクセスの制御はキャッシュ・ユニットが行う。

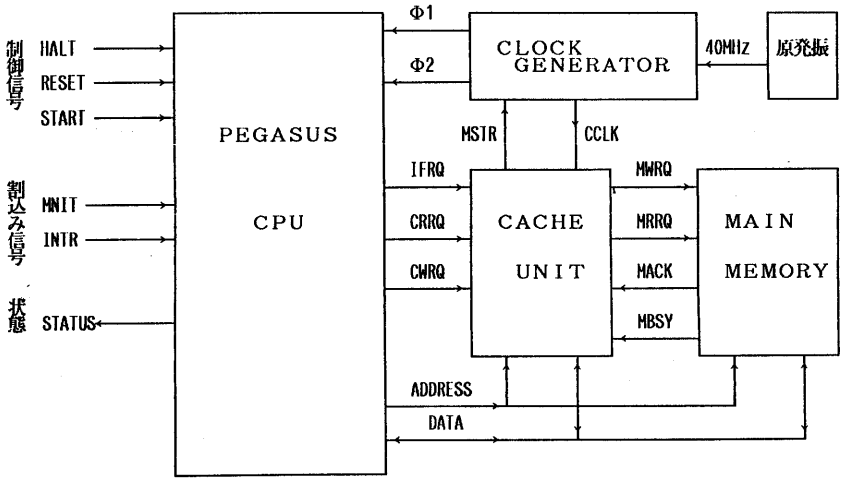


図４．システム構成

従って、命令フェッチやデータ・アクセスの際には、Pegasusプロセッサはキャッシュ・ユニットに対してアクセス要求を出す。キャッシュがヒットしていればそのまま処理が継続されるが、ミスヒットした場合には主記憶のアクセスが完了するまで処理を中断しなければならない。この為の同期操作は、Pegasusプロセッサに供給されているクロックを伸ばすことにより行われる（ストレッチャブル・クロック）。

5．処理性能

Pegasusの処理性能を評価する為に、命令レベルのシミュレータを構築し、簡単なPrologプログラムのシミュレーションを行った。プログラムは図５に示す“Append”と“Quicksort”の２つであり、それぞれのシミュレーション結果を表２にまとめる。

処理性能は、“Append”が239KLIPS、“Quicksort”が149KLIPSといずれも高性能を達成している。処理時間の改善に寄与した項目として、最適化、Shadow操作、バイパス・レジスタの３つが示されている。この内の最適化については、ディレイド・ジャンプによってNo-op命令を他の命令に置き換えたことによるものが大半を占め、２つのプログラムともに８割から９割のNo-op命令を排除することができた。また、非決定的な処理を含む“Quicksort”の場合には、Shadow操作の導入による改善が最も大きく１７％となっている。さらに、Pegasusのようなパイプライン処理ではバイパス・レジスタが不可欠であり、いずれの場合にも高い使用頻度を示している。

今回のシミュレーション結果にはキャッシュのミスヒットが考慮されていないが、現在命令レベルのシミュレーション結果を入力としキャッシュ・シミュレーションを行うプログラムを作成し、評価を行っている。その結果はキャッシュの制御方式に依存するが、今回用いた“Quicksort”の場合最も簡単なライト・スルー方式のキャッシュでも１０％強のペナルティですむことが確認されている。

表２．シミュレーション結果

		Ａｐｐｅｎｄ	Ｑｓｏｒｔ
命令数		６６	Qsort ８９ Split ２５８
実行時間		４１．８μsec （２３９KLIPS）	４．０３msec （１４９KLIPS）
処理時間改善	最適化	１７．９％	１１．２％
	Shadow操作	― ― ―	１７．２％
	バイパスレジスタ	１７．９％	１２．７％

Append Program

```
append([],L,L).
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).

:- append([1,2,3,4,5,6,7,8,9],[10],X).
```

Qsort Program

```
qsort([X|L],R,R0) :- split(L,X,L1,L2), qsort(L2,R1,R0), qsort(L1,R,[X|R1]).
qsort([],R,R).

split([X|L],Y,[X|L1],L2) :- X =< Y, !, split(L,Y,L1,L2).
split([X|L],Y,L1,[X|L2]) :- split(L,Y,L1,L2).
split([],_,[],[]).

list50 = [27,74,17,33,94,18,46,83,65, 2,
          32,53,28,85,99,47,28,82, 6,11,
          55,29,39,81,90,37,10, 0,66,51,
           7,21,85,27,31,63,75, 4,95,99,
          11,28,61,74,18,92,40,53,59, 8]

:- qsort(list50,X,[]).
```

図５．シミュレーションに用いたプログラム

6. おわりに

本報告では、現在開発を行っているProlog指向のRiscプロセッサPegasusについて述べた。シミュレーション結果からも解るように、Prologの実行形態にあわせ命令を絞り込むことによりかなりの高性能を実現することが可能である。

今後は、さらにシミュレーションによる効率測定を行うと共に、Pegasusチップおよびシステムの構築を行ってゆく。

[謝辞]

本報告をまとめるにあたり、日頃ご指導いただいている三菱電機㈱中央研究所房岡璋グループ・マネージャ、平山正治主事、ならびにご討論いただいたシステム第一グループの諸兄に感謝いたします。

[参考文献]

[1] S.Uchida, et al. : "Outline of the Personal Sequential Inference Machine: PSI," New Generation Computing, Vol.1, No.1, 1983.

[2] M.Yokota, et al. : "The Design and Implementation of the Personal Sequential Inference Machine: PSI," New Generation Computing, Vol.1, No.2, 1983.

[3] R.Nakazaki, et al. : "Design of a High-Speed Prolog Machine (HPM)," Proc. of the 12th International Annual Symposium on Computer Architecture, pp191-197, 1985.

[4] T.P.Dobly, et al. : "Performance Studies of a Prolog Machine Architecture," ibid., pp180-190, 1985.

[5] E.Tick and D.H.D.Warren : "Towards a Pipelined Prolog Processor," Proc. of the International Symposium on Logic Programming, pp29-40, 1984.

[6] D.H.D.Warren : "An Abstract Prolog Instruction Set," Technical Note 309, Artificial Intelligence Center, SRI International,1983.

[7] R.P.Colwell, et al. : "Computers, Complexity, and Controversy," Computer, Vol.18, No.9, pp8-19, September 1985.

[8] D.A.Patterson : "Reduced Instruction Set Computers," Communications of the ACM, Vol.28, No.1, pp8-21, 1985.

[9] J.L.Hennessy : "Design of a High Performance VLSI Processor," 3rd Caltech Conference on VLSI, pp33-54, 1983.

[10] P.Chow : "MIPS-X Instruction Set and Programmer's Manual," Technical Report No.CSL-86-289, Computer Systems Laboratory, Stanford University, 1986.

[11] G.S.Taylor, et al. : "Evaluation of the SPUR Lisp Architecture," Proc. of the 13th International Annual Symposium on Computer Architecture, pp444-452, 1986.

[12] D.Ungar, et al. : "Architecture of SOAR : Smalltalk on a RISC," Proc. of the 11th International Annual Symposium on Computer Architecture, pp188-197, 1984.