マイクロコンピュータ 48－11
計算機アーキテクチャ 69－11
（１９８８．１．２１）

Ｐｒｏｌｏｇ指向ＲＩＳＣプロセッサ“Ｐｅｇａｓｕｓ”—プロトタイプ開発とその評価—

瀬尾和男　横田隆史

三菱電機株式会社　中央研究所

我々の研究所において開発を進めているRISC方式μプロセッサ“Pegasus”は、カスタムVLSI技術に基づくProlog処理の高速化を目指したものであり、Prologの実行形態に即したスタック操作、タグ操作、Backtrackに伴う状態の退避・復旧等を効率良く実行できる命令セットを備えている。特に、Backtrackに伴う状態の退避・復旧に関しては、互いにコピー可能なレジスタ対によって構成されるレジスタ・ファイルをカスタムVLSI設計によって実現し、高速化を図っている。

本報告では、Pegasusアーキテクチャを検証する目的で行ったプロトタイプ・チップの開発について述べる。このチップは、プロトタイプ開発に要する時間の短縮化を目標にフルカスタム／スタンダード・セル方式によって設計されている。テスト・ボードに組み込んだ試験の結果、マシンサイクル200nsで動作可能であり、Append：239KLIPS、Quicksort：149KLIPSの推論性能を達成している。

A PROLOG-ORIENTED RISC PROCESSOR “PEGASUS” —PROTOTYPING AND ITS EVALUATION—

Kazuo SEO and Takashi YOKOTA

Central Research Laboratory
Mitsubishi Electric Corporation
8-1-1, Tsukaguchi-Honmachi, Amagasaki, JAPAN 661

Pegasus is a Prolog-oriented RISC processor. By making the best use of custom VLSI technology, it provides an efficient way of performing Prolog-oriented operations. The cost of backtracking especially is reduced by the use of a register-file partially composed of register pairs which can be copied into each other.

This report presents the development of a prototype Pegasus chip which has a processing speed of 239*KLIPS* (Kilo Logical Inferences Per Second) for a deterministic append program and 149*KLIPS* for a non-deterministic quicksort program.

1．はじめに

第５世代プロジェクトを中心としたProlog言語の普及に伴い、Prologを高速に実行できるマシン・アーキテクチャの研究もさかんになってきている[1-7]。現在開発が進められているPrologマシンのほとんどは、D.H.D.Warrenによって提案されたPrologの抽象命令セット、すなわち、WAM(Warren Abstract Machine)モデル[8]に基づくものであり、第５世代プロジェクトのPSI[1,2]やＵＣバークレイのPLM[4]に代表されるようにμプログラミング方式による実現を行っているものが多い。これに対して、Prologの比較的単純な実行形態に着目したRICS（Reduced Instruction Set Computers）方式の実現も注目されてきている。

我々の研究所において現在開発を進めているμプロセッサ“Pegasus”もPrologをターゲットとしたRISC方式に基づくVLSIアーキテクチャの構築を目指しており、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縮小化し、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（２ステージごとに命令フェッチを行う）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処理に伴うレジスタ群の退避・復旧等を効率良く処理できる命令セットを備えている。

本報告では、Pegasusのアーキテクチャを検証する目的で試作したプロトタイプ・チップについて述べてゆく。プロトタイプ開発に要する時間の短縮化という方針から、このチップはフルカスタム／スタンダード・セル手法を用いて設計されており、VLSI化による処理速度の向上という観点からはまだまだ改善の余地を残したものである。しかしながら、テスト・ボードに組み込んで行った動作試験において200nsのマシンサイクルで動くことが確かめられており、先に報告したシミュレーションによる推論性能（Appendプログラム：239KLIPS, Qsortプログラム：149KLIPS）[7]を達成可能であることが確認されている。

2．Prolog指向アーキテクチャ

Prologの処理効率を上げる為に、Pegasusは以下のようなアーキテクチャ上の特長を持つ。

＊スタック操作命令の導入

＊Backtrackに伴う状態の退避・復旧の効率化

＊データ型によるタグ分岐命令の導入

以下では、これらの特長について、関連するWAMモデルの説明もまじえて簡単に説明してゆく。

2.1 スタック操作命令の導入

WAMモデルでは、プログラム領域に加えて、以下の４つのデータ領域が定義されている。

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

これらのデータ領域へのアクセス形態を調べた結果、Local_Stackでは制御フレームへの先頭ポインタからの変位でアクセスされるのに対し、その他のデータ領域では通常のスタック・アクセスが行われることがわかった。従って、これらのデータ領域へのアクセスは，次の２種類のロード／ストア命令を用意することによって効率的にサポートできる。

＊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）

Pegasusでは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、これらのロード／ストア命令を実現するには、

→メモリ番地生成バス上への加算器の導入

→レジスタ・ファイルの２ポート化

が必要となる。

2.2 シャドウ操作によるBacktrack処理の高速化

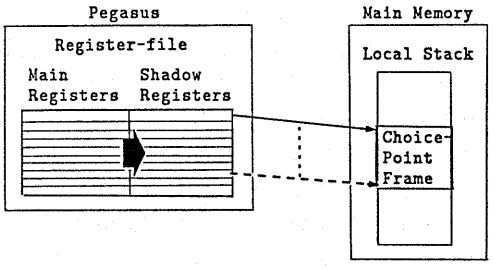
Prologでは、複数のUnification可能な節がある場合、Backtrack処理に備えて節呼出し時の状態を退避しておくことが必要となる。WAMモデルでは、この状態の退避はLocal_Stack上に生成されるChoice_Pointフレームによって実現される。Choice_Pointとして退避されるのは、スタック・ポインタ等の状態レジスタと節の引数分だけの引数レジス

タである。また、実際にBacktrackが起こった場合には、退避されているChoice_Pointを用いて状態の復旧が行われる。これらのChoice_Pointによる状態の退避・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。このことによって、Backtrackの処理負荷は非常に大きくなっている。

PegasusではBacktrack処理の高速化を図る為に、次に説明する２つの“シャドウ”操作を導入している。

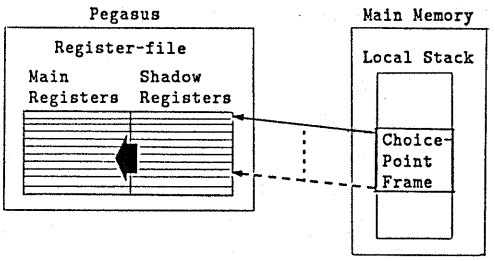
シャドウ・ライト操作（Choice_Pointの生成）

状態レジスタ及び引数レジスタの内容をシャドウ・レジスタと呼ばれるチップ上のもう１組のレジスタに退避する。さらに、連続したChoice_Point生成に備えて、後続の命令の実行と並行してLocal_Stack上へと退避する。



シャドウ・リード操作（Backtrackの実行）

シャドウ・レジスタを用いて状態を復旧する。さらに、連続したBacktrack処理に備えて、後続の命令の実行と並行してもう１つ前のChoice_Pointをシャドウ・レジスタへと読み込んでおく。



これら２つのシャドウ操作におけるレジスタ間のコピー操作を高速に行えるように、Pegasusのレジスタ・ファイルは部分的に互いにコピー可能なワードの対（メイン・レジスタとシャドウ・レジスタ）から構成されている。図１に、Pegasusのレジスタ・ファイルの構成を示す。このレジスタ・ファイルは２ポートのスタティックＲＡＭのレイアウトをもとに設計されており、シャドウ・レジスタを含めた全レジスタが均一にアクセスできるようになっている。

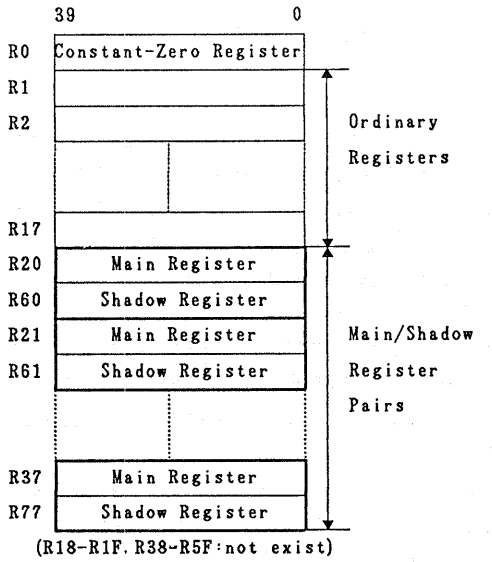


図１．レジスタ・ファイルの構成

２．３ タグ分岐命令

Prologでは４つの基本データ型として変数、アトム、リスト、構造体がサポートされている。Prologの基本演算であるUnificationを行うには、まず２つの項の基本データ型を調べ、それによって異なる処理を行うことが要求される。従って、基本データ型の判別を高速に行うことによって、かなりの処理速度の向上が達成できる。

この為、Pegasusでは、図２に示すようなタグの割当てを行うと共に、Unifyすべき２つの項の基本データ型の組合せによって１６ウェイの分岐を行うタグ分岐命令を導入している。すなわち、２つのオペランド・レジスタの基本データ型を表す２ビットのタグを結合し、それをオフセットとして用いたテーブル・ジャンプを行うものである。

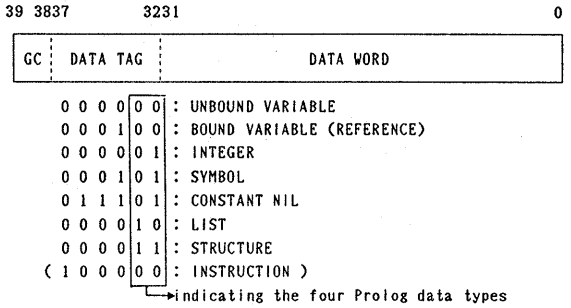


図２．タグによるデータ型の識別

３．命令セットとパイプライン制御

３．１ 命令セット

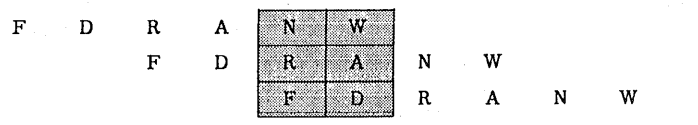
Pegasusの命令は、データと同じくタグ部８ビット、命令部３２ビットの計４０ビット固定長の構成となっている。タグ部には、命令とデータとを区別する為のビットやシャドウ操作に関連して命令の実行を動的に制御する為のフラグ・ビットが保持されている。（シャドウ操作に関する詳しい制御については文献[6]を参照されたい。）命令部は３オペランド構成となっており、各々８ビット長のオペコードと３つのオペランドが置かれている。

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
＊シャドウ操作
＊その他

前節で述べたスタック操作の実現の為にロード／ストア命令には修飾子（Modifier）を持つものがあり、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できるようになっている。

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分岐命令（JTD）が用意されている。この命令は、２つのオペランド・レジスタのタグ部２ビットを連結した４ビットによってこの命令に続いて格納されている分岐アドレス表を引き、そこに置かれた分岐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モデルのIndexing命令等をプログラムするのに活用できる。



F-stage: Instruction Fetch
D-stage: Instruction Decode
R-stage: Register-file Read / Memory-address Calculation
A-stage: ALU Operation / Memory Access / PC Update for Branch
N-stage: No-operation
W-stage: Register-file Write

図３．パイプライン・ステージ

シャドウ・レジスタの存在は命令セットにも反映され、これを扱うシャドウ操作命令がPegasusの命令セットの大きな特徴となっている。シャドウ・リード／ライト命令では、シャドウ・レジスタとメイン・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シャドウ・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、シャドウ・ゲット／プット命令ではシャドウ・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。

シャドウ命令によって起動されたシャドウ・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。あるシャドウ命令の実行から次のシャドウ命令の実行までには通常数十命令が実行される為、シャドウ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。

３．２ パイプライン制御

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。

このようなパイプラインの場合、次の２つに起因するパイプの乱れに対処しなければならない。

(1)レジスタ・ファイルへの書き込みの遅れ
(2)ジャンプ命令

(1)によるパイプの乱れは、ある命令のＲステージが直前の命令のＷステー

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+ *Displ*(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

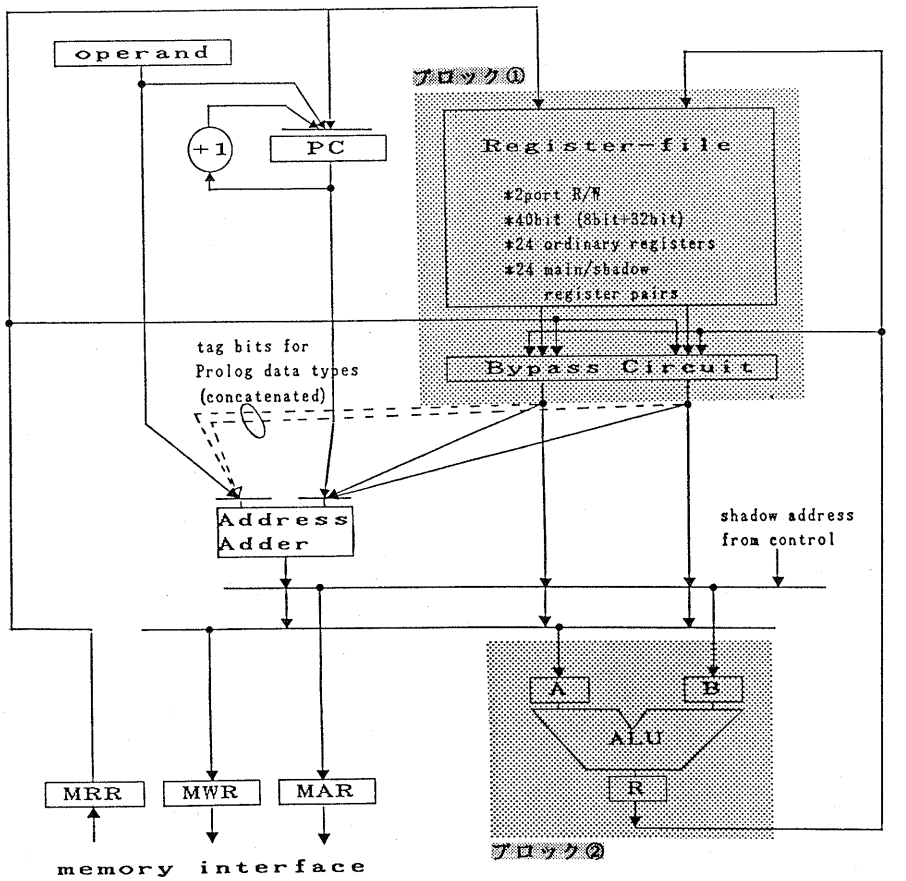


図４．データ・パスの構成

ジよりも先に実行される為に起こるもので、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。Pegasusでは、レジスタ・ファイルの入出力ポートにバイパス回路を付けることにより、この乱れを解決している。また、(2)によるパイプの乱れに対しては、ジャンプ命令の後にNo-op命令を挿入し、さらにこのNo-op命令を置き替えの可能な命令を探し出すことによって最適化を図るという遅延ジャンプ手法[9]により対処している。

４．プロトタイプ・チップの開発

VLSIアーキテクチャの研究を行う上で、いかに短期間でプロトタイピングを行えるかは大きな課題となっている。Pegasusの場合も、RISC方式のチップとはいえ、そのレイアウト設計にはかなりの時間と労力を要する。従って、今回の試作では、真にフルカスタム設計が必要な部分を除いてスタンダード・セル方式による設計を行い、開発期間の短縮化を図った。

図４にプロトタイプ・チップのデータパスの構成を示す。最も大きなモジュールであるレジスタ・ファイルに関しては、基本的にはレイアウト設計においてその占有面積を考慮すべきメモリ・モジュールであり、また、シャドウ操作に伴うコピー機能を実現することが必要であるといった理由から、フルカスタム・レイアウトを行った。その他のモジュールについてはスタンダード・セルを用いて設計されているが、モジュール間の結合度等を考慮してＡＬＵだけを独立したブロックとし、その他は制御ロジックも含め１つのブロックにまとめた。なお、今回の実装では制御ロジックに含まれる命令デコード用のＲＯＭをデバッグ可能なようにチップの外付けとした。

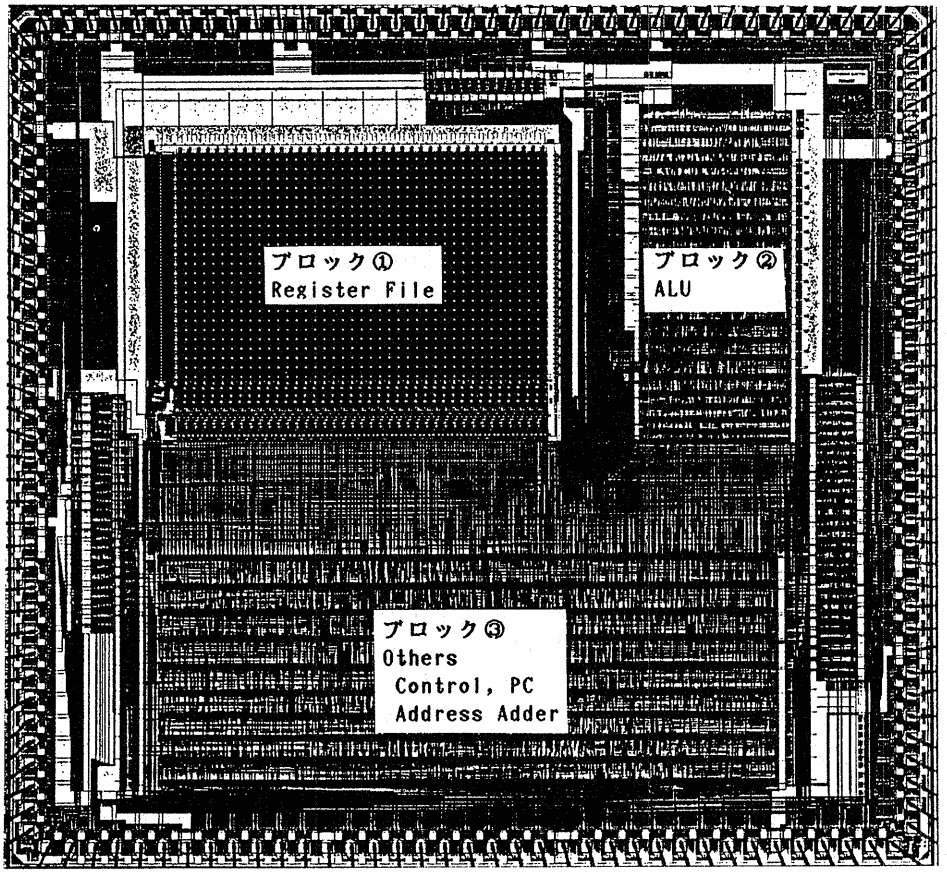


図５．プロトタイプ・チップ

表２．プロトタイプ・チップの概要

設計方式	フルカスタム／スタンダード・セル方式
設計人月	約８人月
製造プロセス	２メタル・２μｍ ＣＭＯＳ
チップサイズ	１０.３㎜×１０.９㎜
パッケージ	１５０ピンＰＧＡ
マシンサイクル	２００ｎｓ
推論性能	２３９KLIPS（ａｐｐｅｎｄ） １４９KLIPS（ｑｕｉｃｋｓｏｒｔ）

表３．各モジュールのゲート／トランジスタ数

レジスタ・ファイル（含バイパス回路）	約４万トランジスタ
ＡＬＵ	２，１５３ゲート
制御ロジック	２，６２１ゲート
ＰＣ	７４６ゲート
アドレス加算器	４５１ゲート
マルチプレクサ	６７８ゲート
バッド＆ドライバ	６６３ゲート

図５に実装されたプロトタイプ・チップを示す。このチップは、パッド及びそのドライバを除けば上で述べた３つの大きなブロックから構成されている。表２にチップの概要を、表３に各モジュールのゲート／トランジスタ数を示す。フルカスタム／スタンダード・セル方式による設計の結果、レイアウト設計に要した期間は８人月と非常に短縮化されている。

レジスタ・ファイルはトランジスタ数が約４万個（nチャネルのパス・トランジスタが多く使われている為にゲート数に換算すれば約２万５千ゲート）と多くなっているが、繰り返しが多い為に実際にレイアウトされたのは約２百トランジスタである。制御ロジックに関してはRISC方式を採ったことにより非常に縮小化されており、外付けの命令デコード用のＲＯＭ（32bit×128words）を考慮に入れても全体の２割程度に収まっている。

このチップのクリティカル・パスは、レジスタ・ファイルからアドレス加算器を経由してＡＬＵもしくはメモリ・インタフェースの入力側のラッチへとつながる経路である。現在のモジュール構成では100nsのパイプライン・ピッチ以下で動作させるのは難しいが、アドレス加算器の構成を工夫することによってかなりの性能向上を達成することが可能である。

テスト・ボードに組み込んで行った動作試験の結果、シャドウ操作を含めたすべての命令が正常に動作しており、先に報告したシミュレーションによる推論性能（Appendプログラム：239KLIPS, Qsortプログラム：149KLIPS）[7]を達成可能であることが確認できた。

５．おわりに

本報告では、現在開発を進めているProlog指向RISCプロセッサPegasusについて、今回試作したプロトタイプ・チップを中心に述べた。このチップ自体はVLSIとしてまだまだ改善の余地を残したものであるが、Prologの実行形態に即したアーキテクチャを持つことによりかなりの高性能を実現している。

RISC方式を採る場合の問題点としてコード量の増加が挙げられるが、タグ操作等を高度にすることによって分岐を少なくし、コード量を削減すると共に、さらに性能向上を図ってゆきたい。

［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研究部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] K.Seo and T.Yokota : “PEGASUS: A RISC Processor for High-Performance Execution of Prolog Programs,” Proc. of the International Conference VLSI’87, pp.221-234, 1987.

[7] 瀬尾、横田 : “Prolog指向RICSプロセッサ Pegasus,” 情報処理学会アーキテクチャ研究会資料63-5, 1986.

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

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