高速内蔵型 Prolog プロセッサ IPP の設計†

山口 伸一朗†† 坂東 忠秋†† 黒沢 憲一††

知識処理を中心とした AI 技術の実用化が着実に進展しつつあるが，実用的知識処理システムでは，従来のソフトウェアと論理型言語 Prolog 等で記述された知識処理用ソフトウェアが，ともに高速実行でき，かつ協調して処理を進める必要がある．そこで，汎用の 32 ビットスーパミニコンに Warren の Prolog 命令を高速実行可能な Prolog 専用機構を内蔵した内蔵型 Prolog プロセッサ IPP を開発した．IPP では，Prolog 命令で使用するタグ付きデータがベースマシンの語長と同じ 32 ビットであり，Prolog 命令がベースマシンの拡張命令となっているため，従来ソフトウェアと Prolog が命令語レベルでリンケージが取れる．また Prolog 命令を高速化するために，タグ処理やスタックアクセス用の専用ハードウェア等を追加した．これらのハードウェアは，全体の 2.7% に過ぎないが，これにより 24% の性能向上となった．IPP は，ECL ゲートアレイでインプリメントされており，最適化コンパイラとの相乗効果により，append で 1 MLIPS を実現した．

1. まえがき

最近，自然言語処理・画像理解・計算制御の分野で，AI 技術を用いたシステムの開発がさかんになっている．このような中で Prolog は，事物間の関係を記述するだけでプログラミングできるので，アルゴリズム（手続き）の開発が難しい AI 分野の記述言語として有効と考えられている．しかしながら Prolog を用いて実用的 AI システムを構築するには，

- ユニフィケーション/バックトラックという高機能で負荷の重い処理のため実行速度が遅い．
- 実用システムでは，記号処理だけでなく数値処理も不可欠だが，Prolog は数値処理の記述力が弱い．

という問題点がある．

前者は，従来の手続き型言語がいわゆる汎用マシンに合った形で設計され，汎用マシンもこれを効率よく実行するように改良されてきたのに対して，Prolog は，汎用マシンのアーキテクチャと無関係に設計されていることに起因しており，専用ハードウェア[1)-4)]なしには，実行速度の大幅な向上は望めない．

また後者に対しては，たとえばプラント制御に AI 技術を応用する場合，プラントの状態をシミュレーションして，この結果と各センサからのデータより推論を行って，最適な制御あるいは指示を出すことが考えられ，シミュレーションでの数値処理と推論での記号処理がともに高速かつ協調しなければならない．

したがって Prolog の実用化には，少なくともこれら二つの問題を解決する計算機システム，すなわち Prolog と手続き型言語を高速かつ協調して実行可能な計算機が必要となる．本論文では一つの解として，汎用マシンに Prolog 専用機構を統合した内蔵型アーキテクチャを提案し，これに基づいた内蔵型 Prolog プロセッサ（以下 IPP と呼ぶ）のハードウェア構成について述べる．更に，IPP に設けた専用ハードウェアの効果について評価し，内蔵型アーキテクチャの有効性を明らかにする．

2. 内蔵型アーキテクチャの提案と課題

2.1 内蔵型アーキテクチャの提案

Prolog と従来の手続き型言語がともに実行可能な計算機システムとして次の四つのアプローチが考えられる．

①汎用マシン上で Prolog を実行させる．
②Prolog 専用マシン上で手続き型言語を実行させる．
③汎用マシンと Prolog 専用マシンのマルチプロセッサ構成で各々の言語を実行させる．
④汎用マシンに Prolog 用機構を追加した Prolog 内蔵型マシンで Prolog を実行させる．

①は最終的に同一マシン上で Prolog と手続き型言語を実行することになるので，言語間のリンケージは取りやすい．また，Prolog の実行速度についても最適化コンパイラを用いることにより速度を向上させることができる．しかし元来汎用マシンにおける Prolog の実行効率が悪いために，専用マシン並みの性能を得るには，大型汎用マシンが必要となる．

† Design of High Performance Integrated Prolog Processor IPP by SHINICHIRO YAMAGUCHI, TADAAKI BANDOH and KENICHI KUROSAWA(Hitachi Research Laboratory, Hitachi, Ltd.).

†† (株)日立製作所日立研究所

②は①と同様に言語間のリンケージは取りやすい．しかし既存のソフトウェア資産をどこまで活用できるかが問題であり，オブジェクトレベルでの互換性を保つために，エミュレーションを行わなければならない状況も発生して，手続き型言語の実行速度が遅くなる．

③は各マシンが最適に設計できるので，高速性を実現しやすい．しかし Prolog と手続き型言語のリンケージを取るには，マシン間の通信が必要となり，このオーバヘッドがシステム全体の性能を低下させてしまう．またハードウェア物量が，2倍になってしまう問題点がある．

④は汎用マシンのアーキテクチャに，Prolog 用アーキテクチャを拡張したものである．したがって Prolog と従来の手続き型言語は，同一ハードウェア上に実現されるそれぞれのアーキテクチャの上で，実行されることになり①②と同様に言語間のリンケージは取りやすい．また Prolog 用専用機構により Prolog の実行速度の向上も望める．しかし，汎用マシンの枠組があるため，どこまで Prolog に適した構成が取れるかが問題となる．

以上述べた各アプローチは一長一短があり，一概に優劣を決め難い．しかし以下の理由から④の内蔵型マシンのアプローチが優れている．第1にハードウェア面から見てⓐ汎用マシンの演算器やレジスタファイルといった基本要素は，Prolog 専用マシンでも共通に使用できるので兼用することでハードウェア物量を少なくできる，ⓑ命令パイプラインやキャッシュメモリ等の汎用マシンにおいて確立された高速化方式を Prolog でも活用することができる．第2にソフトウェア面から見てⓒシステムソフトウェアから応用ソフトウェアに至るまで従来ソフトウェアの変更を抑えられる．

内蔵型マシンのアプローチでは，汎用マシンと同時に開発することで，最新の高速デバイスや実装技術を活用できる．このことは，高速マシンを実現する上で重要なファクタであり，専用マシンにも優る性能を達成する可能性を生み出す．

2.2 内蔵型アーキテクチャの課題

第1の課題は，どのような Prolog 専用機構を内蔵するかである．Prolog プログラムを逐次解釈実行する方式とプログラムを専用命令にコンパイルして実行する方式があるが，実応用を目指して実行速度を最優先に考えるとコンパイル方式が良い．この場合専用命令セットの設計が重要となる．

第2の課題は，汎用マシンの枠組の中で Prolog 専用機構をいかに実現するかである．その中の一つは，データ形式と命令形式の問題である．Prolog を実行するには，タグ付きデータが必要であり，Prolog 専用マシンでは ICOT の PSI のように，8ビットのタグを持つ 40 ビット/語もあるが[1]，内蔵型マシンの場合ベースとなる汎用マシンの基本語長を越えると，メモリシステムを始め，周辺装置のインタフェイスに大幅な変更を生じさせてしまう．2番目は Prolog 高速化用の専用ハードウェアの問題である．Prolog 専用マシン[1)~4)]では，タグ処理ハードウェア・大容量レジスタファイル・スタックバッファ等を持っているが，内蔵型の特徴を生かすためにも，大幅なハードウェア増加やソフトウェアの変更を生じないように考慮しなければならない．

3～4章では，内蔵型マシンにおけるこれらの課題に対する解決策を述べる．

3. IPP の設計

3.1 ベースマシンの概要

ここでまず IPP のベースマシンについて簡単に述べる．ベースマシンは，図1に示すように5段パイプラインの命令処理ユニット (BPU) と2段パイプラインのメモリ管理ユニット (MCU) から成っており，デバイス技術は ECL ゲートアレイを用いている．表1にベースマシンの諸元を示す．Prolog 専用機構は，このベースマシンに組み込まれて全体として IPP となる[5]．

3.2 Prolog 命令セット

コンパイル方式向きの Prolog マシンとして，Warren が提案した仮想マシン (以下 WAM と略す) がある[7),10)]．WAM の主な特徴は次のとおりである．

①ゴールとヘッドの引数を単位として put 系と get

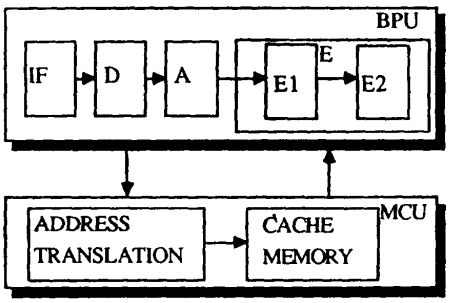


図1 ベースマシンのパイプライン構成
Fig. 1 Pipeline structure of the base machine.

表 1 ベースマシンの諸元
Table 1 Specifications of the base machine.

BPU	
汎用レジスタ	32 bit×16
Floating レジスタ	80 bit×8
パイプライン	5段
MCU	
キャッシュメモリ	16 kB
TLB	64エントリ
パイプライン	2段
性能	
R-R 命令	1 mc
R-M 命令	3 mc
Gibson Mix	8 MIPS

mc: machine cycle

系の命令に展開する.

②ユニフィケーションは，引数レジスタに対して行う.

③述語呼び出しでの生成フレームが，バックトラック用 (choice point) と正常復帰用 (environment) に分かれている.

④構造体コピー方式である．このため構造体に対応する unify 系命令は，コピーが生成される (Write モード) か否 (Read モード) かによって動作が異なる.

⑤前記フレームを格納するローカルスタックと構造体のコピー格納するヒープ領域とバックトラック用のトレイルスタックを用いる.

①，②の特徴から，最適化コンパイラによってユニフィケーションが不要となる引数同士に対して，命令を削除したり，レジスタの割付けを考慮する[6]ことにより，メモリアクセスなしでユニフィケーションを行う命令列を生成することができる．WAM は比較的低レベルの命令セットを持ち，パイプライン処理を前提としているので，IPP と親和性が高い．そこで IPP では，WAM の命令セットを Prolog 命令のベースとし，これに高速リスト処理や cut 等の組込述語命令を追加して，機能・性能を拡張した[14].

3.3 タグ付きデータ形式

IPP では，基本語長がベースマシンと異なると大幅なハード/ソフトウェアの変更が生じてしまう．そこで，4あるいは8ビットのタグと 28 あるいは 24 ビットのデータ部より成る 32 ビットタグ付きデータ形式を採用した．しかしこのデータ形式には，次のような欠点がある.

- 32 ビット以上のデータ (たとえば浮動小数点) を表現できない.

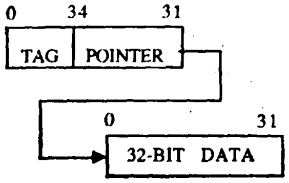


図 2 ポインタ型データ形式
Fig. 2 Pointer type data format.

- アドレス空間が 28 ビット (256 MB) に制限されてしまう.

前者は，たとえばC言語と Prolog の間で，浮動小数点データの受け渡しの時フォーマット変換が必要になったり，精度が合わない等の問題が生じる．そこで 32 ビット以上のデータは，図 2 に示す形式 (以下ポインタ型と呼ぶ) を用いて，ポインタ参照で，任意ビット数のデータを表現するようにした．実行時に生成されるポインタ型のデータ部はヒープ領域に置かれる．またプログラム中に固定データとして出現するものについては，別の問題点があり，これについては後述する.

後者はプログラムが大きくなるにつれいずれ問題となってくる．そこでベースマシンがバイトアドレッシングになっているのに対して，タグ付きデータが 32 ビット (4バイト) を基本としている点を利用して，タグ付きデータではロングワードアドレッシングとした．これにより，アドレス空間を 1 GB (256 MW) まで拡大することができる.

3.4 可変長 Prolog 命令形式

Prolog 命令とベースマシンの汎用命令を一つの命令セットにまとめることによって，マシン語レベルでの高速な言語間リンケージが実現できる．そこで IPP では，Prolog 命令をベースマシンの拡張命令コード

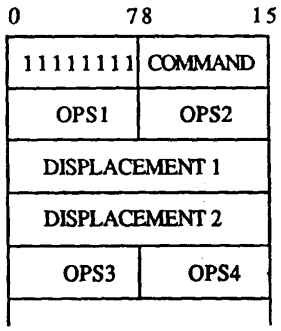


図 3 Prolog 命令フォーマット
Fig. 3 Prolog instruction format.

に割り当てる形で追加した．**図 3** に Prolog 命令形式を示す．命令のオペレーションは，拡張命令を示すオール1に続く COMMAND フィールドで指定され，各オペランドはオペランド指定子 (OPS) によって指定される．COMMAND フィールドと OPS を直交させることにより，命令デコーダ等のハードウェアの負担が軽くなるように考慮した．

Prolog プログラムでは，組込述語が多く使われており，静的出現頻度は約 50% に上っている[8),9)]．したがって組込述語を命令化することは，Prolog 高速化に効果がある．一般に四則演算やデータ型判定の組込述語が重要と考えられるが，今後さまざまな組込述語の命令化が要求されてくると予想される．そこで Prolog 命令は OPS 内のストップビットが立つまでオペランドが続く可変長命令形式として，命令追加時におけるオペランド数の自由度を確保した．

3.5 Prolog 命令用アドレッシングモード

ベースマシンの汎用レジスタは 16 本あり，従来の手続き型言語には十分であるが，Prolog 実行時には 14 本のレジスタを環境設定用に使用するため，引数レジスタ用に2本しか割り当てられず，残りはメモリに割り当てるしかない．しかしメモリ上の引数レジスタでは，Warren のレジスタ指向のアイディアを生かすことができない．そこで8本のフローティング用レジスタを Prolog 命令にも開放して，新たに 32 ビットレジスタを 16 本追加した．これにより，**表 2** に示すレジスタ割り付けが可能となり，11 個の引数レジスタ (AGi) を確保することができた．ここで，GR 0〜15 が，ベースマシンの汎用レジスタであり，PGR 0〜15 がフローティング用レジスタと兼用している Prolog 用のレジスタである．

さて，前述のポインタ型データによって 32 ビット以上のデータを表現することが可能になるが，命令中にポインタ型データが含まれる場合（たとえば定数を引数レジスタにロードする Put-constant 命令の constant が浮動小数点データ），実行時に生成されるポインタ型データと同様にポインタを絶対アドレスにすると，データ部が絶対アドレスに固定されるので，その命令は再配置可能とならない（**図 4**）．この問題を解決する方法として，ポインタ部とデータ本体をすべて命令に含める命令形式が考えられる．しかしポインタ型データへの演算は，ポインタ操作だけで完了することが多く，この時にはデータ部の命令フェッチのために，不要なメモリアクセスが発生してしまう．更にタグによってデータ部の長さを検出する必要があり，命令デコードも複雑になる．

表 2 Prolog 命令でのレジスタ割付け
Table 2 Register assignment on Prolog instructions.

GR		PGR	
0	AGi	0	AGi
1	AGi	1	AGi
2	AGi	2	AGi
3	CP	3	AGi
4	CTR	4	AGi
5	THR	5	AGi
6	TLR	6	AGi
7	TTR	7	AGi
8	—	8	—
9	—	9	—
10	—	10	—
11	B	11	—
12	E	12	S
13	HBR	13	ACR
14	—	14	AGBR
15	—	15	—

TLR: Top of the local stack, THR: Top of the heap stack, TTR: Top of the trail stack, S: Skeleton address (read), E: Base address of the last environment, B: Base address of the last choice point, CP: Next address of the goal, HBR: Release point of the heap stack by backtrack, ACR: Alternative clause address, CTR: Cut operation register, AGBR: Argument base register, —: Reserved for expansion

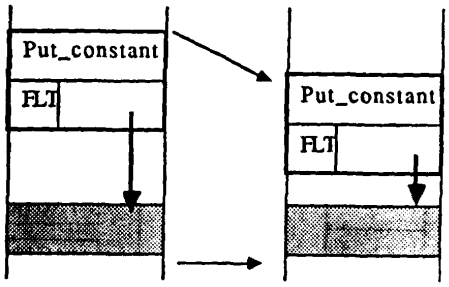


図 4 ポインタ型データの問題点
Fig. 4 Problem on pointer type data.

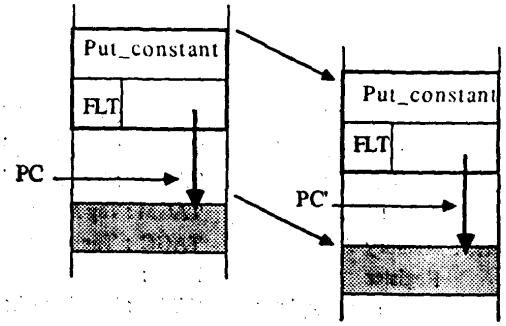


図 5 ポインタ型アドレッシングモード
Fig. 5 Pointer type addressing mode.

そこで，**図5**に示すように命令内のポインタを相対アドレスとし，命令コードとデータ部の位置関係が，命令コードのアドレスによらず一定に保たれるようにした．そして，命令実行時のアドレス計算で絶対アドレスを算出するポインタ型アドレッシングモードを設けた．これによってポインタ型データを含む命令も再配置可能となり，分割コンパイルしたものを結合する時にもポインタの付け替えが不要になる．

4. Prolog 命令高速化方式

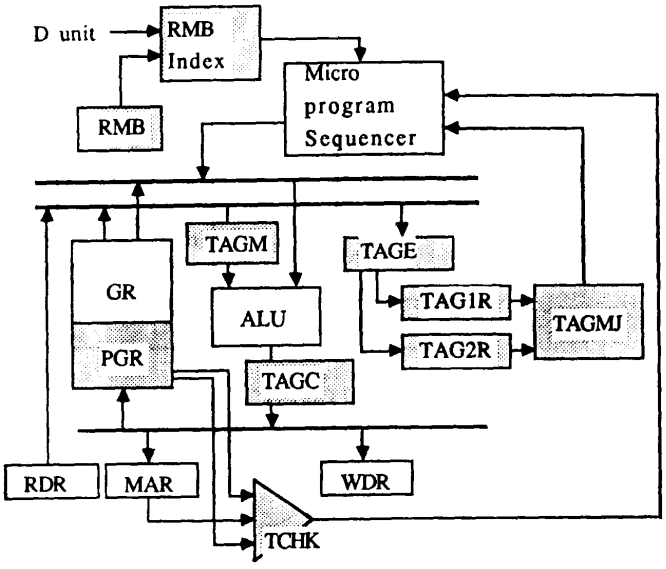
マシンサイクルの短縮やパイプラインによる命令の並列処理等の汎用命令高速化方式は，Prolog 命令においても共通の技術として効果が大きい．しかし，タグによる実行時のデータ型判定など Prolog 特有の処理は，汎用マシンでの効率が悪く，専用ハードウェアを用いて高速化する必要がある．また，汎用マシンでもボトルネックと言われているメモリアクセスについては，Prolog 実行時の特徴として，従来言語よりも高い頻度で発生し，かつライトアクセスが集中的に発生する傾向があり，これに適したメモリ管理が必要である[11]．

以下本章では，**図6**に網がけで示す BPU の命令実行ユニットにおける専用ハードウェアの内タグ処理とトレイルスタック制御および unify 命令のモード判定について述べる．

① タ グ 処 理

タグなしの汎用アーキテクチャの計算機で Prolog 命令を実行する時，最も効率が悪いのが，タグの切出し・合成・判定といったタグ処理である．タグ処理は通常の演算器を用いても可能であるが，マイクロプログラムで 3〜18 ステップを要し，append プログラムでは，実行時間の 50% 以上をタグ処理が占めている[5]．このためタグ処理のハードウェア化は，高速化ばかりでなく，マイクロプログラム数削減の効果も大きい．

そこで，ALU にタグをマスクしてデータ部を取り出す回路 (TAGM) とタグとデータ部からタグ付きデータを生成する回路 (TAGC) を設けた．またタグ判定のために，タグだけを取り出す回路 (TAGE) とこれを格納するレジスタ (TAG 1R, 2R) を設け，レジスタ内のタグに従ってマイクロプログラムを制御する回路 (TAGMJ) を設けた．この中で，TAGM・TAGE・TAGC は，単純な回路で実現できるが，TAGMJ の実現には工夫が必要である．TAGMJ はマイクロプログラムの多分岐用オフセットを生成する回路であり，メモリを用いると柔軟な回路が比較的容易に実現できる．しかし，1組の4ビットタグに対して，4ビットのオフセットを生成するには，$2^4*2^4*4=1,024$ ビットのメモリが必要となり，生成パタンが複数個存在すると必要なメモリは，更に増加する．また一般にマイクロシーケンサの分岐回路は，クリティカルパスになる可能性が高く，結果的に非常に高速なメモリが必要となる．そこで，メモリに比べて柔軟性は失われるものの，高速性とハードウェア量の観点から見て有利なランダム論理によって TAGMJ を構成した．Prolog 命令の仕様からオフセット生成の 17 パタンを抽出して，300 ゲートで TAGMJ を実現した．



RMB : Read Mode Bit
RMB Index : RMB Indexing
GR : General Register
PGR : GR for Prolog
MAR : Memory Address Register
WDR : Write Data Register
TAGMJ : Tag Multiple Way Jump
TAG1R : Tag 1 Register
TAG2R : Tag 2 Register
TAGM : Tag Mask
TAGC : Tag Concatenation
RDR : Read Data Register
TCHK : Trailing Checker

図6 Eユニットのブロック図
Fig. 6 Block diagram of Eunit.

② トレイル・チェッカ

Prolog ではバックトラック時に変数を未定義状態に戻す (undo 処理) のために，ユ

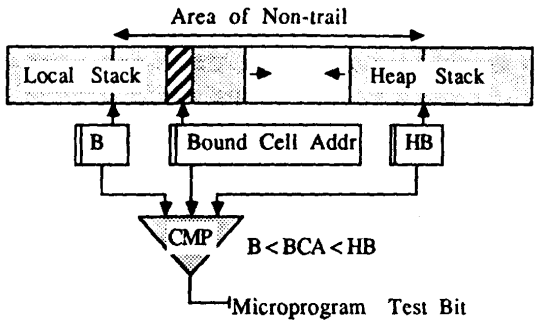


図 7 トレイルチェッカ
Fig. 7 Trailing checker.

ニフィケーションでバインドされた変数のアドレスをトレイルスタックに保存するが，この変数がバックトラックによって解放されてしまう時には undo 処理が不要となるので，アドレスの保存は行わなくてもよい．たとえば図7に示すようにローカルスタックとヒープを割り付けると各々のバックトラックポイントに挟まれた所にある変数のアドレス保存は不要となる．したがってこの条件判定を行えば，ユニフィケーション時のアドレス保存とバックトラック時の undo 処理における余分なメモリアクセスを省くことができ，Prolog の高速化につながる．しかしユニフィケーション時にマイクロプログラムだけで条件判定を行うには，2回の減算と演算結果の判定が必要となり，5ステップを要する．そこで，図7に示すように比較器を設けることにより，条件判定の高速化を図った．

③ RMB 修飾

WAM によれば unify 命令は，マシンのステータスが Read モードか Write モードかで処理内容が異なる．このため IPP は，ステータスレジスタとして RMB (Read Mode Bit) を設け，これによって処理を切り分けている．RMB の判定をマイクロプログラムで行うと unify 命令の実行が遅くなるため，専用ハードウェアによって，RMB の判定を行っている（RMB 修飾）．図 8 に RMB 修飾の方法を示す．命令デコーダから unify 命令のマイクロプログラムエントリーアドレスが与えられると，RMB の値に従って Write モード，Read モードいずれかのエントリーアドレスが選択される．これによって unify 命令実行以前にモード判定が完了する．Prolog プログラムではリストや構造体を扱うことが多く，unify 命令の出現頻度が高い．このため，RMB 修飾の効果も大きい．

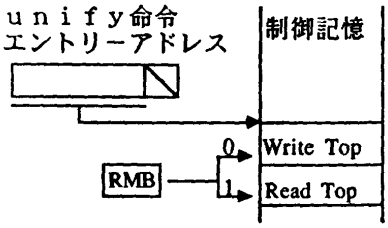


図 8 RMB 修飾の方法
Fig. 8 RMB indexing.

5. 評　価

5.1 ハードウェア量の評価

IPP は，2,000 ゲートおよび 5,000 ゲートの ECL ゲートアレイを使用して，ベースマシンの BPU と MCU を含めて，3枚のプリント板に実装した．

Prolog 専用機構の追加によるハードウェアの増加分を調べるために，ハードウェア量の解析を行った．一つ一つのゲートを専用ハードウェアか否か区別するのは，非常にむつかしい作業であるが，およその解析結果を表3に示す．

専用ハードウェアの内データ系では，ライトデータとアドレスを保持する8段のストアバッファ[11]が最大である．タグ処理ハードウェアの内300ゲートは，マイクロプログラム多分岐用のオフセットを生成する TAGMJ の論理であり，他のハードウェアは 120 ゲートにすぎない．トレイルチェッカのゲート数が比較的多いのは，32 ビットの大小比較器を二つ持っているためである．Prolog 命令は，図3に示すようにオペコードとオペランド指定子が直交しているので，命令デコード回路が簡素化できる．また，アドレス計算についても，アドレッシングモードの多くは，ベースマシンと同一なので，わずかにポインタ型アドレッシングモードの計算にゲートを費やしているだけである．その他の部分には，RMB および RMB 修飾回路，ロングワードアドレッシングのためにデータ部を

表 3 ゲート使用率
Table 3 Contents of hardware.

項目	ゲート使用率 (%)
Base machine	97.30
Prolog instruction decode	0.06
Address calculation	0.07
Tag processing	0.20
Trail checker	0.32
Store buffer	2.00
Miscellaneous	0.06
Total	100.00

2ビット左シフトする制御回路等が含まれている.

表3からわかるように, Prolog 専用ハードウェアが全体に占める割合は, 約 2.7% となった. ただし, 全ハードウェア量には, マイクロプログラム用メモリ・キャッシュメモリ等のメモリは含まれていない. メモリまで含めた時には, Prolog 専用ハードウェアが, LSI 個数ベースで約 5% を占めた.

5.2 性能評価

append プログラムと第1回 Prolog コンテスト課題[12]のいくつかを用いて, 性能評価を行った. append は, 実機で実行時間を測定し, その他は命令シミュレータによって実行時間を予測した.

表 4 にモード宣言のない append のオブジェクトコードと各命令の実行ステップ数を示す. 本コードは IPP と同時に開発した最適化 Prolog コンパイラ[13]によるクローズインデキシング処理後のものであり, 別解のない形となっている. 表からわかるように1推論を 43 マシンサイクルで実行しており, 1マシンサイクルが 23 ns の時, 約 1 MLIPS となる. 表4で命令2, 3の(S)+は, ヒープスタックの読み出しポインタの更新をポストインクレメントのアドレス計算で行うことを示している. これによってEユニットでのポインタ更新時間がAユニットに移り, 見かけ上の命令実行時間が4サイクル短くなる. ただし命令5, 6では, 命令がライトモードで動作するので, (S)+ は余分なアドレス計算となる.

表 4 append の実行結果
Table 4 Execution in append program.

	オブジェクトコード		実行時間 (mc)
1.	Get_lst	AG1	6
2.	U_var	(S)+, AG4	3
3.	U_var	(S)+, AG1	3
4.	Get_lst	AG3	11
5.	U_val	(S)+, AG4	4
6.	U_var	(S)+, AG3	5
7.	Jmpt	L, AG1	11
	Total		43

表 5 推論性能の評価結果
Table 5 Performance evaluation result.

	実行時間 (msec)	推論性能* (kLIPS)
nrev 30	0.414	1,197
qsort 50	1.049	573
cons 1000	0.366	330
trav 1000	5.942	357
8_queenl	3.979	1,490

* 組込述語も推論としてカウント

表 5 に Prolog コンテストプログラムの実行性能を示す. これは, 命令シミュレータを用いて算出したものであり, 次のような条件下で測定した.

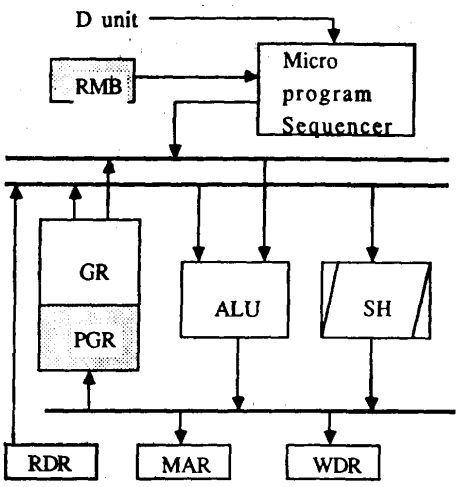
①キャッシュメモリのヒット率は 100% である.
②パイプラインの空きは生じない.
③命令フェッチとオペランドアクセス間でのメモリアクセス競合は発生しない.

キャッシュメモリのヒット率は性能に大きく影響するが, 評価用プログラムは小規模なもので, プログラム・データともにキャッシュメモリに納まるので, 条件①を設けた. 命令シミュレータは, 分岐命令によるパイプラインの空きも実行時間に加えるように作られている. ここでの空きは, レジスタ競合による待ち, 命令フェッチ不足による命令実行開始待ちによって生じるものを指しており, 発生率は少ない. また IPP のように命令とデータのキャッシュメモリが分離されていない構成では, 命令フェッチとオペランドアクセスの競合は生じる可能性がある. しかし, Prolog 命令は実行時間が長いものが多く, これらの命令では命令実行時間中に十分な命令フェッチ時間を確保できるので③による影響は少ない.

またコンパイラは append で用いたのと同じ最適化コンパイラを用いており, LIPS 値の算出には組込述

表 6 専用ハードウェアの効果
Table 6 Effect of acceleration hardware.

	TAGE		TAGC		TAGMJ		TRAIL		RMB	
	出現回数	向上率 (%)	出現回数	向上率 (%)	出現回数	向上率 (%)	出現回数	向上率 (%)	出現回数	向上率 (%)
append 30	65	4.3	31	4.1	66	13.2	31	4.1	90	6.0
nrev 30	1,084	4.2	496	3.8	1,083	12.6	466	3.6	1,425	5.5
qsort 50	1,799	3.2	670	2.4	1,750	9.4	498	1.8	1,119	2.0
cons 1000	284	1.4	1,081	10.6	283	4.2	41	0.4	1,040	5.1
trav 1000	7,008	2.2	6,008	3.9	9,009	8.7	4,003	2.6	3,002	1.0
8_queen	11,809	5.1	2,473	2.1	11,236	14.6	1,010	0.9	6,156	2.7
平均	—	3.4	—	4.5	—	10.5	—	3.7	—	2.2



SH ; Shifter その他は図6と同じ

図 9 性能評価用のEユニットブロック図
Fig. 9 Block diagram for performance evaluation.

語の実行も1推論とカウントしている．前述のように，表5は命令シミュレータによる評価結果であり，実機との誤差がある．この誤差は append において約3%なので，表5のプログラムにおいても同程度と考えられる．

表 6 に専用ハードウェアの効果を示す．TAGE, TAGC, TAGMJ, TRAIL, RMB の出現回数は，前章で述べた各々の専用ハードウェアを使用したマイクロプログラムのステップ数である．また向上率は，専用ハードウェアを用いずに一般的なハードウェアだけで各処理を行った時に対する推論性能の高速化の度合を示している．ここで，一般的ハードウェアとして，**図 9** を想定している．演算はレジスタ読出しと演算/書込みの2サイクルをパイプラインで行い，シフタは，左右32ビット可能，リテラルデータ（即値）は，ソースバスに対して下位16ビットのみ使用できる．そしてマイクロプログラムは次のものを同時に制御できる．①二つの読出しレジスタまたはリテラルデータ，②ALU またはシフタのファンクション，③書込みレジスタ．

専用ハードウェアを用いない場合には，タグ分離に1ステップ，タグ合成に2ステップ，タグ判定に平均3ステップ，トレイルスタックにアドレスを保存するか否かのチェックに2ステップ，RMB の判定に1ステップずつマイクロプログラムのステップ数が増える．向上率は append が最も高く，32% 向上した．また最低は trav であり 18% であった．この結果より2.7% のハードウェア追加で，平均 24% の性能向上が実現されたことになる．

6. あとがき

知識処理に代表される AI 技術の実用化に耐えうる計算機は，Prolog 等の AI 向き言語だけでなく，従来の手続き型言語も高速に実行できることが不可欠であり，またこれらが協調して動作できなければならない．この考えに立って，汎用マシンに WAM ベースの Prolog 専用機構を内蔵した IPP を開発した．

IPP では，タグ付きデータをベースマシンと同じ32ビットとし，Prolog 命令をベースマシンの拡張命令とすることで，機械語レベルで両者を混在可能とし，これによって Prolog と従来言語が同一計算機上で実行可能となった．

また Prolog 命令高速化のためにタグ処理等の専用ハードウェアを設け，これによって append で1 MLIPS の性能が達成できることを確認した．IPP の専用ハードウェアは，全体の 2.7% にすぎず，いずれも単純な論理回路である．しかし，これらによって 24% の性能向上が達成されており，この点から見て，内蔵型アーキテクチャは，マイクロプロセッサから大型機まで，比較的容易に AI 機能を強化できるアプローチと考える．

参考文献

1) Taki, K. et al.: Hardware Design and Implementation of the Personal Sequential Inference Machine (PSI), *International Conference on Fifth Generation Computer System*, pp. 398-409 (Nov. 1984).
2) Nakazaki, R. et al.: Design of a High-speed Prolog Machine (HPM), *Proceedings of the 12 th International Symposium on Computer Architecture*, pp. 191-197 (June 1985).
3) Despain, A. M.: A High Performance Prolog Co-processor, Proceedings of WESCON 85, no 18/2(1985).
4) 斉藤光男ほか：AI ワークステーションの開発思想，第1回人工知能学会全国大会論文集，pp. 237-244 (1987).
5) Yamaguchi, S. et al.: Architecture of High Performance Integrated Prolog Processor IPP, *Proceedings of 1987 Fall Joint Computer Conference*, pp. 175-182 (Oct. 1987).
6) Abe, S. et al.: High Performance Integrated Prolog Processor IPP, *Proceedings of the 14th International Symposium on Computer Archi-*

tecture, pp. 100-107 (June 1987).
7) Warren, D. H.: An Abstract Prolog Instruction Set, Technical Note 309, Artificial Intelligence Center, SRI International (Oct. 1983).
8) Onai, R. et al.: Static Analysis of Prolog Programs, *Proceedings of the Logic Programming Conference 84*, Tokyo, March, pp. 19-21 (1984).
9) Matsumoto, H.: A Static Analysis of Prolog Programs, *SIGPLAN*, Vol. 20, No. 10, pp. 48-59 (1985).
10) Tick, E. and Warren, D. H.: Towards a Pipelined Prolog Processor, *1984 International Symposium on Logic Programming*, pp. 29-40 (Feb. 1984).
11) 森岡道雄ほか：内蔵型高速 Prolog プロセッサ IPP(III)，第34回情報処理学会全国大会論文集，6P-6, pp. 193-194 (1987).
12) Okuno, H.: The Report of the Third Lisp Contest and the First Prolog Contest, 情報処理学会記号処理研究会資料，33-4 (1985).
13) 桐山 薫ほか：内蔵型 Prolog プロセッサ IPP の最適化コンパイル方式の提案と性能評価，情報処理学会論文誌，Vol. 29, No. 6, pp. 589-595 (1988).
14) 黒沢憲一ほか：Prolog 最適化コンパイラの開発 (VI)，第 33 回情報処理学会全国大会論文集，3 E-4, pp. 493-494 (1986).

(昭和63年 1 月 5 日受付)
(昭和63年11月14日採録)

山口伸一郎 (正会員)

昭和32年生．昭和55年九州大学工学部電子工学科卒業．昭和57年九州大学総合理工学研究科情報システム学修士課程修了．同年(株)日立製作所日立研究所入社．制御用計算機，Prolog プロセッサのハードウェアアーキテクチャの研究に従事．計算機アーキテクチャ，並列処理，分散処理システムに興味を持つ．

坂東 忠秋 (正会員)

昭和43年東京大学工学部電気工学科卒業．同年(株)日立製作所入社．日立研究所にて，制御用計算機，マルチプロセッサ，画像処理，ワークステーション等の研究に従事．昭和51年スタンフォード大学電気工学科修士．昭和60年東京大学より工学博士受理．現在，日立研究所主任研究員．

黒沢 憲一 (正会員)

昭和28年生．昭和55年東北大学大学院工学研究科修士課程情報工学修了．同年(株)日立製作所入社．知識処理計算機の命令アーキテクチャ，PROLOG 言語の最適コンパイラの研究に従事．人工知能マシンアーキテクチャ，並列処理に興味をもつ．現在，同社日立研究所第8部研究員．電子情報通信学会会員．