# Prologマシン PEK における中間コードとその実行方式†

和田耕一†† 宮本昌也†††
金田悠紀夫†††† 前川禎男††††

逐次実行型 Prolog マシン PEK 上に開発した Prolog 中間コードである PEK コードと，その PEK 上での実行方式について述べている．PEK は，内部に複数の専用メモリ，ユニフィケーション専用回路等を持ち，書換え可能な制御記憶 (WCS: Writable Control Storage) 中の水平型マイクロプログラムによって駆動される．PEK コードの設計に当たっては，①実行速度の向上を第一目標とする，②Prolog 原始プログラムをまず PEK コードに翻訳し，さらにマイクロ目的コードに展開する，③PEK のアーキテクチャの特徴を十分に考慮する，等の点を基本方針としている．また，PEK コードの構成は，D. H. D. Warren の提案した WAM (Warren Abstract Machine) 命令を参考にしている．append プログラムのユニフィケーション部に注目し，PEK 内でのデータの動きを詳しく述べた．ベンチマークテストを用いて，本システムの評価を行っている．その結果，最高約 430 KLIPS (Logical Inference Per Second) の実行速度が得られたことを示した．また，テストプログラムの実行時における動的特性を分析し，PEK コードは，PEK の低レベル並列動作機能やユニフィケーション専用回路を有効に用いていることを明らかにしている．

## 1. はじめに

AI システム開発用のプログラミング言語として Prolog が注目されるようになって以来，Prolog プログラムの高速実行を目標とした Prolog 専用マシンの研究・開発が各所で進められている[1)~3)]．筆者らが開発した逐次実行型 Prolog マシン PEK[4)~7)] の場合，インタプリタ下で約 60 KLIPS の性能となっており，十分に高速であることが実証されている．しかしながら，コンパイル技術を導入することによってどの程度の性能が引き出せるかは，今後の研究に対する参照値を得る意味で大変に興味のある問題である．本研究では，PEK システム上にコンパイラを開発し，実行性能の測定を行った．コンパイルの際には，PEK コードと呼ぶ Prolog 中間コードを設定し，最終的な目的コードとしてマイクロプログラムを得る方式をとった．

本論文では，まず PEK のアーキテクチャについて概説する．次に，Prolog プログラムのコンパイルの際に用いる中間コードとして新たに設定した PEK コードについて述べる．さらに，ユニフィケーション実行時の PEK マシン内でのデータの動きについて詳述する．最後に，テストプログラムを用いて PEK コードの評価を行う．

† Intermediate Code and Its Execution Method for Prolog Machine PEK by KOICHI WADA (Institute of Information Sciences and Electronics, University of Tsukuba), MASAYA MIYAMOTO (IBM Japan, Ltd.), YUKIO KANEDA and SADAO MAEKAWA (Department of Systems Engineering, Faculty of Engineering, Kobe University).
†† 筑波大学電子情報工学系
††† 日本アイ・ビー・エム(株)
†††† 神戸大学工学部システム工学科

## 2. Prologマシン PEK

PEK システムは，Prolog プログラムの高速実行に必要となるハードウェア機能を明らかにするための実験機である．ホストプロセッサである MC 68000 システムに共有メモリ (CM: Common Memory) を介して，Prolog プログラム高速実行用の専用ハードウェアである PEK プロセッサが接続されている．PEK プロセッサの特徴を挙げると，

- タグアーキテクチャの採用．
- ストラクチャシェアリング方式の採用．
- メモリの専用化，分散化．
- ユニフィケーション専用回路の付加．
- 自動トレイル，自動アンドゥ回路の付加．
- モレキュール上のタグ部とバリュー部を用いたマイクロプログラムでの多方向分岐を実現するマッチング回路の付加．

などである．

PEK プロセッサの構成図を図1に示す．内部バスは，ソースバスとして R-bus, S-bus, デスティネーションバスとして Y-bus の3本からなる．それぞれ 34 ビット幅で，14 ビットのフレーム部，4 ビットのタグ

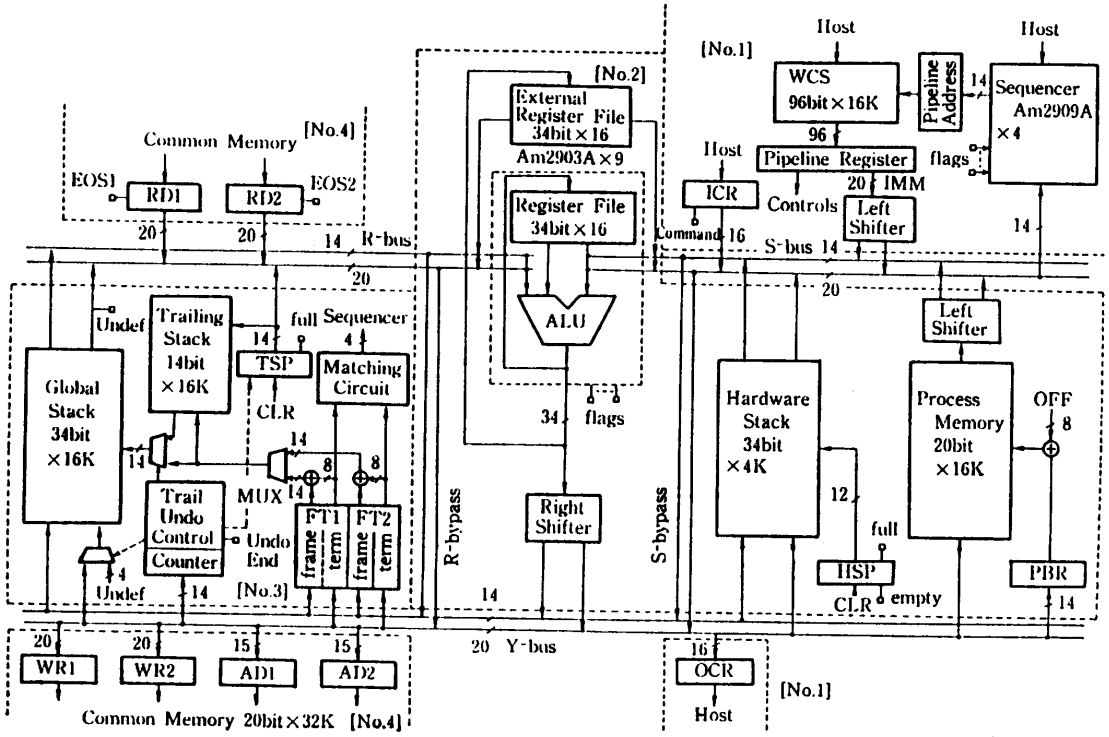



**図 1** PEK のハードウェア構成
Fig. 1 Hardware configuration of PEK.

部，16 ビットのバリュー部を一時に転送できる．

また，PEK プロセッサは，内部に次に示す 3 種類の専用メモリを持っている．

① 実行時のコントロールフレームを格納するプロセスメモリ．

② 変数を格納するグローバルスタック．

③ 代入を行った変数のセルアドレスを保存するトレイルスタック．

さらに，ユニフィケーション中の一時的なデータの退避などに用いるハードウェアスタックを有する．また，ALU は 32 ワードのレジスタファイルを持っている．

マイクロプログラムは書き込み可能な制御記憶 (WCS: Writable Control Storage) に格納され，PEK プロセッサのシーケンサの制御下で実行される．マイクロ命令のビット幅は 96 ビットで，複数のハードウェアモジュールの並列動作を可能にしている．マイクロサイクル長は可変で，マイクロ命令中のサイクル長指定フィールドによって，120 nsec から 400 nsec まで 40 nsec ごとに指定できる．

CM は，PEK プロセッサからみると 2 ポートメモリになっており，2 つのアドレスレジスタ AD 1, AD 2, 読み込み用レジスタ RD 1, RD 2, 書き込み用レジスタ WR 1, WR 2, を用いてアクセスする．特に，連続したアドレスのデータを高速に読み込むため，RD 1 (または RD 2) を読み込むと，自動的に AD 1 (または AD 2) が次のアドレスへと増加し，かつそのアドレス中の内容が RD 1 (または RD 2) にセットされるパイプライン読み出し機能を持っている．

## 3. PEK コード

### 3.1 PEK コード設計の基本方針

PEK コードの設計に当たっては，以下の基本方針をとった[8),9)]．

① Prolog 原始プログラムをマイクロプログラムの目的コードにコンパイルし，PEK マシンで直接実行する．

② コンパイル時の最適化は実行時間短縮に力点を置いて行う．

③ PEK マシンのハードウェア上の特徴を極力生かす目的コードを生成する．

①に関して，コンパイラは Prolog プログラムを PEK コードと呼ぶ中間コードにいったん変換し，さらにこの中間コードをマクロ展開することによってマ

イクロ目的コードを得る方式をとっている.

原始プログラムをマイクロ命令にまで翻訳し，限られた容量の制御記憶に格納する方針は，一般の応用プログラムを考慮すると明らかに現実的とは言えない．しかし，本研究は，PEK 上における今後の処理系の研究に備え，PEK の有するハードウェアを極力有効に利用して，参照値となり得る最高実行速度を確認することを目的としているため，本方針をとった.

また②，③ に関して，PEK コードの構成は D. H. Warren の提案した WAM (Warren Abstract Machine) 命令[8]を参考にしているが，ユニフィケーション専用ハードウェアや CM のパイプライン読み出し機能など，PEK マシンの持つ機能を有効に利用できるよう修正を加えている．最適化については，実行速度の向上を第一目標とした.

### 3.2 内部メモリの割り当て

WAM では，実行時に変数格納用のヒープ領域と，環境および選択点の情報を格納するスタック領域を用いる．したがって，WAM 命令を PEK 上で実行する場合，これらのメモリ領域を PEK の内部メモリに対応付けねばならない．本システムでは，内部メモリのそれぞれのアドレッシング機能を考慮し，WAM でのヒープ領域をグローバルスタックに，スタック領域をプロセスメモリに対応させた．こうすることにより，グローバルスタックの変数アドレス自動計算機能と，プロセスメモリのプロセスベースレジスタからのオフセット値によるアクセス機能を有効に利用できる.

### 3.3 PEK のアーキテクチャとの関連

#### 3.3.1 ストラクチャシェアリング方式の採用

本マシンはユニフィケーションに関して，ストラクチャシェアリング方式を前提としたハードウェアを備えている．したがって，ユニフィケーション命令は，グローバルスタック上の変数フレームを管理する命令など，ストラクチャシェアリング向きの命令セットを用意した．そのため，ユニフィケーション時のデータの動きは，ストラクチャコピー方式をとる WAM とは異なっている.

#### 3.3.2 プロシージャ呼び出し命令の多様化

ボディ部に複数の述語を有する節のボディ部を，WAM 命令に翻訳すると以下のようになる．ただし，簡単のため，各述語は引数を持っていないとして述べる．この場合，まず最初に環境を確保する allocate 命令が生成され，各述語に対してはプロシージャ呼び出しの call 命令が生成される．そして，最後の述語の呼び出しについては，環境の解放を行う deallocate 命令が生成された後，execute 命令が生成される．たとえば，

*a* :- *b*, *c*, *d*.

という節に対して，下記のような WAM 命令列が生成される.

allocate.
call *b*.
call *c*.
deallocate.
execute *d*.

これに対し，PEK コードでは first_call, call, last_call の 3 種類の call 命令を設定し，次のような PEK コード列を生成することとした.

first_call *b*.
call *c*.
last_call *d*.

first_call 命令では，まず環境の確保を行った後，プロシージャの呼び出しを実行する．すなわち，WAM における allocate, call の命令列に相当する．call 命令は，WAM 命令と同様，プロシージャ呼び出しを行うのみである．また，last_call 命令では，環境を解放した後，プロシージャへ分岐する．これは，WAM における deallocate, execute の命令列に相当する.

このような命令を設定した理由は，次のとおりである．中間コードをマイクロ目的コードにマクロ展開する場合について考える．前述のプロシージャ呼び出しは，マイクロ命令における分岐に対応する．PEK において，分岐命令は，ALU を用いた演算など他のオペレーションと並列に実行できる．そこで，WAM の allocate 命令に対応するマイクロ目的コードに，同 call 命令に対応するマイクロ命令を埋め込み，マイクロステップ数の削減を図り，対応する中間コードとして，first_call 命令を設定した．last_call 命令に関しても同様である.

### 3.4 最 適 化

Prolog ソースプログラムから PEK コードへの翻訳時には，①述語引数のレジスタ割り付けと，②インデキシング，③モード宣言，の最適化手法を導入している．PEK コードからマイクロ目的コードを得る際には，マクロ展開を行うのみで，最適化は行っていない.

## 4. インタプリタとのインタフェース

コンパイル後の目的コードは，インタプリタ[4),5)]から呼び出されることによって実行される．これは，組み込み述語を実行する場合に似ている．インタプリタと目的コードでは，述語引数を異なる場所に保持しているため，呼び出し時に両者間で情報の引き渡しを行わねばならない．インタプリタでは，ALU 内のレジスタ ARGS が CM 上の引数リストを指している．目的コードに制御が移る直前の様子を**図 2** に示す．その後，レジスタ ARGS が指しているアドレスから引数を順に読み出して，ALU 内の ARG 1, ARG 2, ARG 3 と名付けられたレジスタにセットする．

## 5. ユニフィケーションの実行手順

**図 3** に示した 2 つのリストを連結する append プログラムを，PEK コードに翻訳すると**図 4** のようになる．翻訳に当たっては，前述のようにインデキシングとモード宣言の最適化を施している．図 4 には，左の欄から順に行番号，PEK コード，PEK コードを実行するのに必要なマイクロサイクル数が示されている．図において，1 行目の i_fail はユニフィケーション失敗時の処理へのラベルで，x_ と 4 桁の数字もラベルである．PEK コードの引数のうち，$ で始まる 5 桁の数字は，最上位の 1 桁が 4 ビットのタグを，下 4 桁が対応する値を表している．すなわち，4 行目の $30000 は，アトム nil を表している．また，6 行目と 13 行目の引数は，イミーディエイト値を表している．10, 11, 14 行目の $8xxxx は変数番号を表している．ARG 1, ARG 2, ARG 3, WK 3 は，ALU 内のレジスタを意味する．

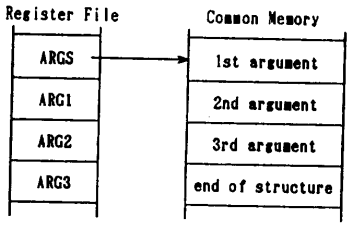


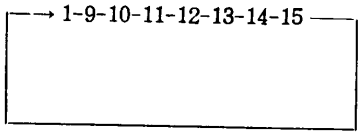

**図 2** レジスタとゴール引数との対応
Fig. 2 Correspondence between registers and goal arguments.

```
:- mode(app(+, +, -)).

append([], Y, Y).
append([H|X], Y, [H|Z]) :- append(X, Y, Z).
```

**図 3** append プログラム
Fig. 3 Append source program.

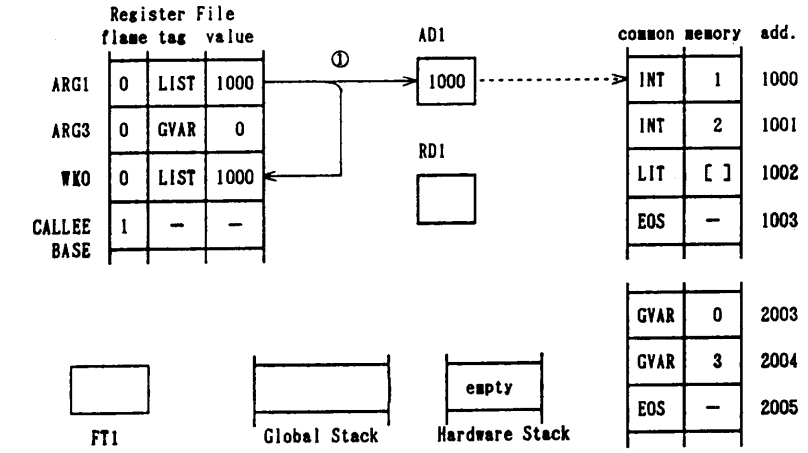
このプログラムの場合，実行の流れは，最後の 1 回の推論を除いてループを形成する．このループを行番号で示すと次のようになる．



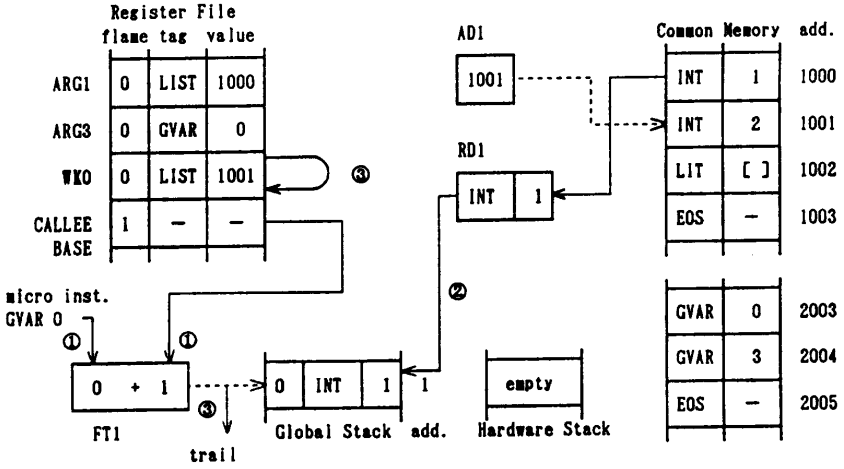

このうち，行番号 9 から 12 までの合計 9 マイクロステップは，ユニフィケーションのための命令である．このユニフィケーション部分に注目し，PEK 内でどのようにデータが移動するかを**図 5** に示す．図では，ゴールとして append ([1, 2], [], Q) を与えたと仮定し，最初の 1 回の推論に対するユニフィケーションを示している．まず，インタプリタは，ゴールの引数に対する構造体データを CM 上に生成する．ここでは，リスト [1, 2] が 1000 番地から生成されたとする．同様に，ソースプログラムの第 2 節の [H|Z] に対する構造が 2003 番地以降に生成されている．ゴールの変数フレームベースアドレスは 0，呼ばれたプロシージャのそれは 1 としている．図中のレジスタおよび内部メモリは，説明に必要な部分のみ示してある．マイクロ命令のイミーディエイトフィールドから与えられる値も，必要な場合のみ示した．実線はデータの動きを示し，点線はアドレスレジスタなどが，メモリアドレスを指している様子を示している．①，②，③は，マイクロ命令のステップを表している．INT, GVAR, LIT, EOS は，それぞれ整数，グローバル変数，リテラル，構造体の終わりを表すタグである．以下に，動作の概略を述べる．

```
line#                                                    # of microcycles
  1   append: switch_on_term(i_fail, x_0001, x_0002,                  3
                  x_0004, l_fail).
  2
  3   x_0001: try_me_else(x_0003).                                   17
  4   x_0002: uin_atom0_1('$30000').                                  3
  5           uout_gref0(ARG3, ARG2).                                 2
  6           proceed('$00001').                                      3
  7
  8   x_0003: trust_me_else_fail.                                     2
  9   x_0004: uin_skel0_1.                                            1
 10             uin_gvar1('$80000', WK3).                            3
 11             uin_gvar1_1('$80001',ARG1).                          3
 12           uout_skel0('$C2003', ARG3).                            2
 13           neck('$00004').                                        1
 14           put_var('$80003', ARG3).                               1
 15           execute(append).                                       1
 16
 17           backtrack.                                            15
```

**図 4** append プログラムに対する PEK コード
Fig. 4 PEK code for append program.




(a) "uin_skel0_1"命令に対するデータの流れ
(a) Data flow for "uin_skel0_1".




(b) "uin_gvar1('$80000',WK3)"命令に対するデータの流れ
(b) Data flow for "uin_gvar1('$80000', WK3)".

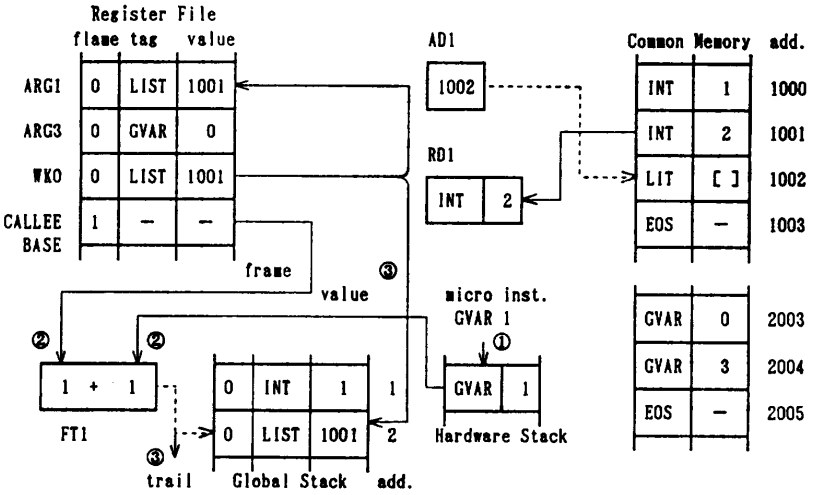
図 5 (その1)

(a) "uin_skel0_1" 命令 (1命令, 160 nsec)

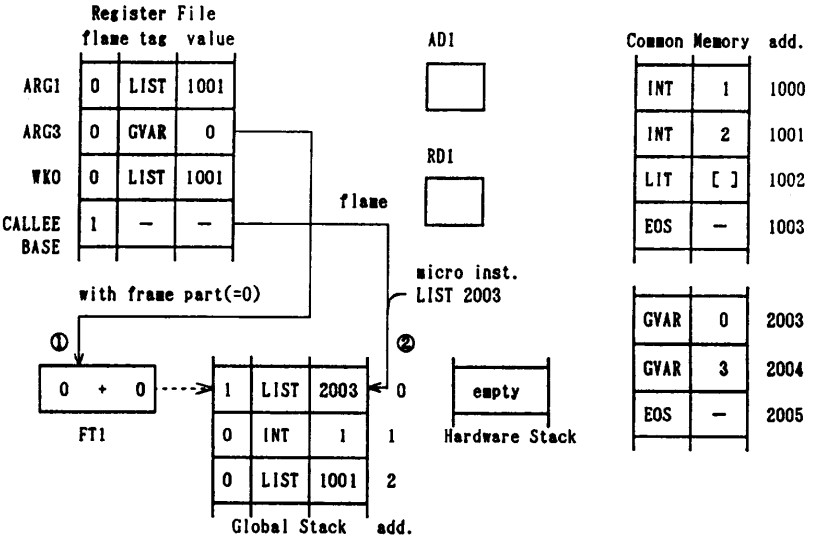
CM のアドレスレジスタ AD1 にゴールの第1引数へのポインタが格納されているレジスタ ARG1 をセットし, リスト [1, 2] の読み出しの準備を行っている.

(b) "uin_gvar 1 ('$80000', WK 3)" 命令 (3命令, 480 nsec)

ここでは, リストの第一要素を番号0の変数へ書き込んでいる. すなわち, CM 読み出し用レジスタ RD 1の値をグローバルスタックの変数セルに書き込んでいる. 変数セルへのアドレッシングは FT 1 (Frame Term register 1) に変数フレームと変数番号を書き込むことによって行われる. FT 1 のフレーム部とバリュー部は自動的に加算され, 変数セルアドレスとしてグローバルスタックに与えられる. ここで, PEK コードでは, 引数を与えることによって, グローバルスタックへ書き込むと同時に任意のレジスタへも同じ値を書き込むことができる. この機能を使うことによって, 節のボディ側に同じ変数が現れている場合, ゴールの呼び出しに備えてレジスタに引数をセットする put 処理をあらかじめ行うことができ, 高速化できる. このプログラムの場合は本機能を有効に使えないので, データはレジスタ WK 3 に捨てられている. ステップ②において, RD 1 から読み出したデータのフレーム部には, 前回に AD 1 にアドレスをセットした時のフレームが自動的に結合される. ステップ③に




(c) "uin_gvar1_1('$80001',ARG1)"命令に対するデータの流れ
(c) Data flow for "uin_gvar1_1('$80001', ARG1)".




(d) "uout_skel0('$C2003',ARG3)"命令に対するデータの流れ
(d) Data flow for "uout_skel0('$C2003', ARG3)".

図 5 ユニフィケーション時のデータの流れ
Fig. 5 Data flow in unification.

おいて，レジスタ WK 0 は AD 1 と常に同じ値を保持するように制御されている．すなわち，レジスタの RD 1 (または RD 2) からの読み出しが行われるごとに WK 0 を増加させている．これは，RD 1 (RD 2) を読み出すごとに自動増加する AD 1 (AD 2) の値が，構造上読み出せないためである．

(c) "uin_gvar 1_1 ('$80001', ARG 1)" 命令 (3 命令，480 nsec)

リストの残りの部分へのポインタを，番号 1 の変数へ書き込むと同時にレジスタ ARG 1 へも書き込み，次の呼び出しに備えている．ステップ①では，マイクロ命令のイミーディエイトフィールドからあらかじめハードウェアスタックに GVAR 1 をプッシュしている．これは，後続する命令で，構造体の終わりかどうかによる条件分岐が行われ，そのときにイミーディエイトフィールドが使われるためである．さらに，図には明示していないが，構造体の終わりに対する処理に備えて RD 1 を ALU 内のワーキングレジスタに読

表 1 ベンチマークテストの実行結果
Table 1 Result of benchmark test.

| プログラム | 生成マイクロ命令数 | 推論回数 | 実行時間 (msec) | 総ステップ数 | 速度 (KLIPS) | 推論当りのステップ数 |
|---|---|---|---|---|---|---|
| append 30 | 57 | 31 | 0.071 | 462 | 434.4 | 15 |
| reverse 30 | 111 | 496 | 1.39 | 8955 | 353.4 | 18 |

み込んでいる．ステップ②で変数セルへのアドレッシングが行われ，ステップ③でセルへの書き込みが実行されている．

(d) "uout_skel 0 ('$C 2003', ARG 3)" 命令（2命令，280 nsec）

ここでは，レジスタ ARG 3 が指している変数へ，現在のフレームの [H|Z] を表すモレキュールをセットしている．

このように，後に用いられる可能性のあるデータは，極力レジスタにもコピー可能にするなど，PEK コードは PEK の並列動作機能を最大限に活用できるように設計されている．

## 6. 評　価

2つのテストプログラムを用いて，本システムの評価を行う．2つのプログラムを，コンパイル後実行し，実行時間とマイクロ命令数を計測する．さらに，実行されたマイクロ命令を分析し，その動的特性をもとに PEK コードの妥当性を評価する．

### 6.1 実行時間の測定

評価に用いたテストプログラムは，図3の append プログラムと**図6**のリストの反転を行う reverse プログラムである．ゴールの引数として 30 個の要素からなるリストを与え，それぞれを append 30, reverse 30 と呼ぶことにする．

コンパイルの際には，両プログラムともインデキシング，モード宣言，および冗長な get, put 命令の省略などの最適化を施している．**表1**は，テストプログラムを翻訳したときの生成マイクロ命令数と実行結果を示している．既に述べたように，目的コードは，インタプリタから呼び出されて実行される．したがって実行時間は，インタプリタから目的コードに制御が移ってから，再びインタプリタに戻るまでの時間を測定した．なお，各マイクロ命令のサイクルタイムは，120 nsec と 160 nsec の2種類からなっている．

```
:- mode(append(+, +, -)).
:- mode(reverse(+, -)).

append([], X, X).
append([H|X], Y, [H|Z]) :- append(X, Y, Z).

reverse([], []).
reverse([H|X], R) :- reverse(X, Y), append(Y, [H], R).
```

図 6 reverse プログラム
Fig. 6 Reverse program.

表 2 インタプリタとの比較
Table 2 Comparison with interpreter.

| プログラム | インタプリタ (msec) | コンパイラ (msec) | 比率 |
|---|---|---|---|
| append 30 | 0.44 | 0.071 | 6.2 |
| reverse 30 | 7.39 | 1.39 | 5.3 |

表 3 ハードウェアモジュールの利用率
Table 3 Utilization ratio of hardware modules.
(reverse 30 program)

| ハードウェアモジュール | 利用率 (%) |
|---|---|
| common memory | 15.6 |
| process memory | 2.4 |
| global stack | 37.7 |
| hardware stack | 10.4 |
| trail stack | 21.1 |
| register file | 79.5 |
| bypass | 17.6 |
| ALU | 61.8 |
| jump | 26.0 |
| multiway jump | 20.8 |

表より，append 30 プログラムで約 430 KLIPS, reverse 30 プログラムで約 350 KLIPS の速度が得られていることがわかる．1回の推論に必要なマイクロ命令数は，append 30 プログラムで 15 ステップ，reverse 30 プログラムで 18 ステップである．

また，**表2**にインタプリタとの速度比を示す．すなわち，インタプリタによる実行に比べて，append 30 プログラムで約6倍，reverse 30 プログラムで約5倍の速度が得られた．

### 6.2 ハードウェアモジュールの利用率

**表3**に，reverse 30 プログラムを実行した時，各ハードウェアがどの程度の比率で利用されたかを示す．

まず，内部メモリとレジスタファイルに関しては，レジスタファイルの利用率がもっとも高く，約 80%

表 4 マイクロ命令中の並列度
Table 4 Ratio of parallelism in microinstructions.

(reverse 30 program)

| 並列度 | 全命令における比率 (%) (8955 steps) | ユニフィケーションにおける比率 (%) (4340 steps) | ユニフィケーション以外の命令における比率 (%) (4615 steps) |
|---|---|---|---|
| 2 | 18.0 | 24.2 | 12.1 |
| 3 | 26.0 | 52.2 | 1.3 |
| 4 | 0.3 | 0.6 | 0.0 |
| 合計 | 44.3 | 77.1 | 13.4 |

に達している．これは，レジスタファイルに実行管理情報が格納されている上に引数のレジスタ割り付けを行っているためである．また，グローバルスタックも約 40% の命令で利用されている．これは，reverse 30 プログラム中に出現する引数のほとんどが変数であるためと考えられる．

分岐命令についてみると，多方向分岐を含めた分岐命令は全体の 47% を占めている．このことは，分岐とそれ以外のオペレーションとを並列に実行できる機能は重要であることを示していると考えられる．

### 6.3 マイクロ命令中の並列度

PEK は，水平型のマイクロ命令で駆動されるため，複数の機能を並列に実行できる能力を持つ．そこで，実行されたマイクロ命令を分析し，PEK コードがマイクロ命令レベルの並列動作機能をどの程度利用できているかについて考察する．ここで，PEK の有するさまざまな機能のうち，マイクロ命令で明示的に指定できる動作に特に注目して分析する．PEK は，次の各機能のうち複数を並列に実行できる．

- 内部メモリへのアクセス．
- レジスタファイルへのアクセス．
- 分岐．
- トレイル動作．

ここで，上記の動作のうち 2 つを 1 マイクロ命令で行った場合，並列度 2 とする．同様に 3 つ，または 4 つを行った場合，それぞれ並列度 3，または 4 とする．この時の，全命令，ユニフィケーション命令，put 命令，実行制御命令中の並列度をそれぞれ表 4 に示す．この結果，全命令中 44% で 2 つ以上の機能が動作していることが確かめられた．特に，ユニフィケーション命令での並列度が，77% と非常に高いことがわかる．これは，PEK コードでは，PEK のユニフィケーション専用のハードウェアが，他の機能と並行して，有効に使われていることを示している．

図 5 に示したように，並列度が 2 以上の場合，メモリとレジスタへのアクセスが同時に実行されている場合が多い．PEK コードでは，ゴール引数をレジスタに割り付ける方法をとったためレジスタの利用頻度が高くなっているが，PEK の並列動作の能力がこの負担を吸収できていると考えられる．また，以上の結果より，PEK の構造を考慮した PEK コードの妥当性を確認できた．

## 7. 結　論

本論文では，逐次実行型 Prolog マシン PEK における，PEK コードと呼ばれる Prolog 中間コードとその実行方式について述べた．PEK コードの設計に当たっては，実行速度の向上を第一目的とし，PEK のアーキテクチャとの整合性を特に考慮した．PEK コードは，以下の特徴を持つ．

①ストラクチャシェアリング方式の採用．
②変数のグローバルスタックへの割り付け．
③プロシージャ呼び出し命令の多様化．

また，PEK コードが PEK 上でどのように実行されるかを，append プログラムのユニフィケーション部分を例にとって詳しく述べた．

実際にテストプログラムを実行し，実行速度を測定した．その結果，append 30 プログラムで約 430 KLIPS，reverse 30 プログラムで約 350 KLIPS の実行速度が得られたことを示した．本システムは，PEK のアーキテクチャから引き出し得る性能を見極めることを目的として開発されたが，応用プログラム中のボトルネックとなる述語を組み込み述語化し，実行速度の向上を図る場合にも，有効に利用できると考える．

また，テストプログラムの実行時における動的特性を分析し，PEK コードは，PEK のマイクロ命令レベルの並列動作機能やユニフィケーション専用回路を有効に用いていることを明らかにした．

## 参考文献


1) Tick, E. and Warren, D. H.: Towards a Pipelined Prolog Processor, *1984 International Symposium on Logic Programming*, pp. 29-40 (1984).
2) Clocksin, W. F.: Design and Simulation of a Sequential Prolog Machine, *New Generation Computing*, Vol. 3, No. 1, pp. 101-120 (1985).
3) Yokota, M. et al.: The Design and Implementation of a Personal Sequential Inference Machine PSI, *New Generation Computing*, Vol. 1, No. 2, pp. 125-144 (1983).
4) 田村直之，和田耕一，小畑正貴，金田悠紀夫，前川禎男，日根俊治：シーケンシャル実行型 Prolog マシン PEK，情報処理学会論文誌，Vol. 26, No. 5, pp. 855-861 (1986).
5) Kaneda, Y., Tamura, N., Wada, K. and Matsuda, H.: Sequential Prolog Machine PEK, *New Generation Computing*, Vol. 4, No. 1, pp. 51-66 (1986).
6) Tamura, N., Wada, K., Matsuda, H., Kaneda, Y. and Maekawa, S.: Sequential Prolog Machine PEK, *Proc. of the International Conf. on FGCS 1984*, Tokyo, pp. 542-550 (1984).
7) 田村直之，和田耕一，松田秀雄，小林久和，綾部雅之，金田悠紀夫，前川禎男：Prolog マシン PEK 上のインタプリタの動特性，第 30 回情報処理学会全国大会論文集，7 C-5, pp. 211-212 (1985).
8) Warren, D. H. D.: An Abstract Prolog Instruction Set, *SRI International Technical Note*, No. 309 (1983).
9) Warren, D. H. D.: Implementing Prolog-Compiling Predicate Logic Programs, *D. A. I. Research Report.*, Vols. 1-2, Nos. 39-40, Department of Artificial Intelligence, Univ. of Edinburgh (1977).


(昭和 62 年 7 月 15 日受付)
(平成 元 年 7 月 18 日採録)

**和田 耕一**（正会員）

昭和 31 年生．昭和 53 年神戸大学電気工学科卒業．昭和 59 年同大学院博士課程修了．同年神戸大学自然科学研究科助手．昭和 62 年筑波大学電子・情報工学系講師，現在に至る．学術博士．計算機アーキテクチャ，並列処理システムに関する研究に従事．ソフトウェア科学会会員．

**宮本 昌也**（正会員）

昭和 37 年生．昭和 60 年神戸大学工学部システム工学科卒業．昭和 62 年同大学大学院工学研究科システム工学専門課程修士課程卒業．同年日本アイ・ビー・エム(株)入社．以来，オフィスシステム・ソフトウェアの開発に従事．現在，大和研究所東京プログラミングセンター勤務．

**金田悠紀夫**（正会員）

昭和 15 年生．昭和 39 年神戸大学工学部電気工学科卒業．昭和 41 年神戸大学大学院電気工学専攻修士課程修了．昭和 41 年電気試験所(現電総研)入所．電子計算機研究に従事．昭和 51 年神戸大学工学部システム工学科，現教授．工学博士．コンピュータシステムのハードウェア，ソフトウェアの研究に従事．高級言語マシン，並列マシン，AI に興味を持っている．

**前川 禎男**（正会員）

昭和 6 年生．昭和 29 年大阪大学工学部通信工学科卒業．昭和 34 年同大学院工学研究科博士課程修了．大阪大学助手を経て，昭和 36 年神戸大学助教授（電気工学科）．昭和 47 年同大学教授（電子工学科）．現在，同大学システム工学科勤務．システム情報講座担当．この間，システム理論，高級言語マシン，人工知能などの研究に従事．工学博士．電子情報通信学会，電気学会，計測自動制御学会，人工知能学会などの正会員．