逐次実行型Prolog専用計算機に関する研究

田村，直之

(Degree)
博士（学術）

(Date of Degree)
1985-03-31

(Date of Publication)
2007-10-11

(Resource Type)
doctoral thesis

(Report Number)
甲0524

(URL)
https://hdl.handle.net/20.500.14094/D1000524

博　士　論　文

逐次実行型Prolog専用計算機に関する研究

昭和59年12月

神戸大学大学院自然科学研究科

田　村　直　之

目　次

第1章　緒論

計算機が開発された当初は、計算機がハードウェアで直接実行できる機械語によってプログラミングが行われていたが、機械語によるプログラミング作業は多くの労力を必要とし、正しいプログラムを作成するのが困難なため、より人間の思考水準に近い高レベルなプログラミング言語が開発され、広く使用されるようになった。これらの高級言語（あるいは高水準言語）には、FORTRAN、ALGOL、COBOL、PL/I、Ada、LISP などがあり、それぞれの特徴に応じて使い分けられている。しかし、高級言語での記述のレベルと、計算機が実際に実行できる命令のレベルには大きな差があり、このセマンテックギャップの解消が計算機科学での大きな課題の一つとなっている。このため、高級言語で記述したプログラムを直接ハードウェアあるいはファームウェアによって実行できる高級言語専用計算機（以下、高級言語マシンと呼ぶ）への期待が高まり、各種の言語について研究が進められている。

高級言語のうちLISPやSNOBOLに代表される記号処理言語は、リスト（記号の列）や文字列などの非数値データ処理用の言語であり、人工知能や知識工学などの分野で使用され、その応用分野は急速に広がっている。ところが、記号処理言語では記号列や文字列などの非数値データを取り扱うため、数値データ処理用に設計された従来の計算機とのセマンテックギャップが大きく、記号処理専用マシンの必要性の認識が深まってきている。

本論文で述べるPrologも記号処理言語の一種であり、エキスパートシステムなどの人工知能や知識工学への応用が期待されている比較的新しい言語である。Prologはまた、論理型プログラング言語でもあり、ユニフィケーション機能や

バックトラック機能などのこれまでの言語にはない数々の特徴を持っている。これらの特徴は、人工知能や知識工学での応用プログラムを書く上で、非常に強力な記述能力を与えてくれるが、一方でProlog処理系の効率が低くなる原因にもなっている。したがって、Prologのプログラムを効率よく実行するためのアーキテクチャの研究が重要な課題である。

そこで、本論文ではProlog専用計算機（以下、Prologマシンと呼ぶ）の備えるべき特徴について検討し、実際にそのような特徴を備えたPrologマシンの設計・開発を行う。このPrologマシンＰＥＫ（Prolog Engine of Kobe University）は、Prologのプログラムを高速に実行するためのアーキテクチャの研究を目的としたマシンで、

- 逐次実行
- マイクロプログラム制御
- インタプリタ方式
- ストラクチャシェアリング方式

などを、基本設計方針とし、

- タグアーキクチャ
- メモリの分散化と低レベル並列処理
- マッチング回路
- 自動トレイル回路
- 自動アンドゥ回路

など、各種のProlog向きのハードウェア機能を備えている。

さらに、このマシン上にインタプリタ形式のProlog処理系を開発し、ベンチマークプログラムによるシステムの性能評価を行う。

以下に本論文の構成を示す。

第２章では、総論として、高級言語を効率よく実行するための高級言語マシンの必要性について、そのうち特に、人工知能・知識工学の分野での記号処理専用マシンへの要求について述べる。

第３章では、Prolog言語の特徴と応用分野について概括したあと、Prologを高速に実行するためのアーキテクチャについて検討する。さらに、この検討結果をもとにして、逐次実行型Prologマシンシステムの設計を行う。

第４章では、第３章の設計方針に基づいて作成した、逐次実行型Prologマシンシステムの全体構成について述べる。

第５章では、逐次実行型Prologマシンシステムのハードウェア構成およびアーキテクチャ上の特徴と実現方法について述べる。

第６章では、逐次実行型Prologマシンシステムのソフトウェア構成と開発支援用システム、Prologインタプリタシステムについて述べる。

第７章では、ベンチマークプログラムの実行結果をもとにして、逐次実行型Prologマシンシステムの評価を行う。特に、Prolog専用のハードウェア機能が実行速度の改善に寄与した割合を検討し、評価結果の考察を行う。

第２章　高級言語と高級言語マシン

Ⅱ．１　緒言

本章では、高レベルの記述能力を持つ高級言語を効率よく実行するための高級言語マシンの必要性について述べる。

緒論で述べたように、現在ではFORTRAN やALGOL などの高級言語によるプログラミングが一般的である。しかし、これらの高級言語による記述のレベルと、計算機が実際に実行できる命令のレベルには大きな差があり、このセマンテックギャップの解消が計算機科学での大きな課題の一つとなっている。このため、高級言語によるプログラムを直接ハードウェアあるいはファームウェアで実行できる高級言語マシンへの期待が高まり、各種の言語について研究が進められている。

特に、人工知能や知識工学の分野で広く使用されている記号処理言語は非数値データを取り扱うため、従来の計算機のアーキテクチャとの整合性が悪く、記号処理専用マシンの必要性への認識が深まってきている。

Ⅱ．２　高級言語と高級言語マシン

Ⅱ．２．１　高級言語

計算機が開発された当初は、計算機がハードウェアで直接実行できる機械語によってプログラミングが行われていた。しかし、機械語によるプログラミング作業は多くの労力を必要とし、正しいプログラムを作成するのが困難なため、より人間の思考水準に近い高レベルなプログラミング言語が開発され、広く使用されている。これらの高級言語（あるいは高水準言語）には、FORTRAN,ALGO

L.COBOL.PL/I.Ada.LISP などがあり、それぞれの特徴に応じて使い分けられている。

しかしながら、通常の計算機においては、これらの高級言語で記述されたプログラムを直接ハードウェアで実行することはできないため、ソフトウェアによる処理が行われている。その処理方式には大きく

- コンパイラ方式
- インタプリタ方式
- コンパイラ・インタプリタ方式

の３つの方式がある。

コンパイラ方式は、高級言語で記述されたソースプログラムを機械語に翻訳したのち実行する方式であり、インタプリタ方式は、ソースプログラムそのもの、あるいはそれと一対一に対応する中間言語に変換したプログラムを解釈実行する方式である。コンパイラ・インタプリタ方式は、前記の２方式の中間的な方式で、ソースプログラムをソース言語と機械語の中間のレベルの中間言語のプログラムに翻訳したのち、そのプログラムを中間言語インタプリタで解釈実行する。

Ⅱ．２．２　セマティックギャップ

これらの３方式のすべてについて言えることは、これらの方式は現実に存在する計算機の命令レベルと高級言語の命令レベルの差（セマンティックギャップと呼ばれる）を埋めるための手段である。ところが、高級言語の記述レベルが上がり、人間の思考レベルに近づくにしたがって、セマンティックギャップは広がり、コンパイラやインタプリタの負担が増大してきた。

また、ソフトウェアの信頼性の面からも、セマンティックギャップは大きな問題となっている。たとえば、実行時に配列の添字の範囲チェックやデータタ

イプのチェックを行うことは、ソフトウェアの信頼性を著しく向上させるが、通常の計算機で行うには負担が大きすぎる。

Ⅱ．２．３　高級言語マシン

このようなセマンティックギャップを解消することを目的として、各種の高級言語専用計算機（高級言語マシン）が提案・開発されてきた。

1961年に開発されたバロース社のB-5000およびB-5500,B-6500 は商用マシンではあるが、ALGOL のプログラムの効率的な実行を目的として、いくつかの革新的なアーキテクチャを採用している[Organic 73]。その一つは、ディスクリプタの採用であり、他の一つはスタックの採用である。

ディスクリプタはデータのタイプや性質を示すためにデーター一つ一つやデータのかたまりに対して付けられたもので、Iliffeによって提案されたタグと同様のものである。タグやディスクリプタはハードウェアによる実行時のタイプチェックを可能にし、データタイプに応じた処理の使い分けを容易にする。

B-5000のもう一つの特徴であるスタックも、セマンティックギャップを埋めるための非常に有効なアーキテクチャである。スタックを用いた制御構造はALGOL やPL/Iなどのブロック構造、再帰呼び出し機構、あるいは割り込み処理などの効率的な管理を可能にする。スタックの考え方は、マイクロコンピュータのアーキテクチャにも大きな影響を与えている。

B-5000以降、FORTRAN,COBOL,PASCAL,APL,LISP,SNOBOLなどの高級言語に対するマシンが発表されている。これらの高級言語マシンのほとんどは1951年Wilkesによって提案されたマイクロプログラム制御方式を採用している。マイクロプログラム制御方式は、プロセッサの制御部を組織的に設計する技法として提案されたが、IBM 360 シリーズで採用されて以来、コンピュータシステムの基本的な構成技法となった。高級言語マシンの場合は、前述のセマンティックギ

ャップをマイクロプログラム、すなわちファームウェアのレベルで吸収することをねらいとしている。

これらの高級言語マシンの開発が進められるようになってきた要因をまとめると、まず第一に、セマンティックギャップの解消への要求が高まってきたこと、第二に半導体集積回路技術の進歩が、高級言語マシンの製作をしだいに容易にしてきていること、第三に計算機の会話的利用形体が広まり、ワークステーションに代表されるスタンドアロンでパーソナルな使用の可能な計算機への要求が増してきていること、の三点が上げれる。

Ⅱ．３　人工知能と記号処理専用マシン

前節では高級言語と高級言語マシンについて述べたが、本節では人工知能と、LISPマシンに代表される記号処理専用マシンについて述べる。

Ⅱ．３．１　人工知能と記号処理言語

人工知能と呼ばれる分野には大きく分けて２つの立場がある。一つは、人間の知能のメカニズムを解明することを目的とした科学的立場で、認知科学あるいは認知心理学と呼ばれることが多い。もう一つの立場は、人間の知的能力を計算機に与えることを目的とした工学的立場である。現時点では明らかに、計算機の能力と人間の能力には大きなへだたりがある。人間が推論や言語・音声・画像などの認識を効率よく行っているのに対し、現在の計算機は数値を計算する能力には優れているが、これらの処理を効率よく実行しているとは言えない。

このような知的な能力が必要とされている応用分野は数多くあるが、最近特に注目されているのは、エキスパートシステムと呼ばれるシステムである。たとえば、ある種の病気診断では、診断手順は定式化されていないが、熟練した

医者は各種の経験的な知識や法則を用いて診断を下している。エキスパートシステムはこのような専門的な知識を計算機内に蓄えておき、未知な問題に対する答えを自動的に作りだすための一種のコンサルタントシステムである。医学以外にも化学、分子生物学、鉱業などへの応用が研究されている。

このようなエキスパートシステムを実現するためには、知識の簡潔な表現が可能なデータ構造を取り扱え、効率のよい推論システムが作成可能な言語、すなわち記号処理能力や柔軟な制御構造を持った言語が必要になる。このような機能を備えた言語として、人工知能の研究では古くからLISPが用いられてきた。実際には、LISPは人工知能だけを対象とした言語ではなく、汎用の記号処理言語であるが、上記のような特徴をもつ効率的な言語として、各種の人工知能の分野で利用されている。

Ⅱ．３．２　記号処理専用マシン

LISP[McCarthy 62] やSNOBOLに代表される記号処理言語は、リスト（記号の列）や文字列などの非数値データを取り扱うための言語であり、人工知能やテキスト処理などの分野で使用され、その応用分野は急速に広がっている。ところが、現在の一般の計算機は数値計算を主目的として設計されているため、記号処理言語とのセマンティックギャップは大きく記号処理専用マシンへの期待は年々高まってきている。

Purdue大学のShapiro はSNOBOL 4を高速に処理するためのSNOBOLマシンを提案した。このマシンには、動的に変化するデータを表現するためのディスクリプタや、文字列のパターンマッチングにおけるバックトラック処理のためのスタック機構などが備えられている[Yamamoto 82] 。

また、LISPマシンとしては、Xerox PARCのAlto、MIT のCONSを始めとして、国内外で数多くのマシンが開発されている[Yasui 82]。LISPマシンの多くも、

タグアーキテクチャ（tagged architecture ）やハードウェアによるスタックを採用しており、記号処理の性能向上にはこの二つの機能が有効に働くことを示している。

Ⅱ．４　結言

本章では、高級言語を効率よく実行するための高級言語マシン、そのうち特に、記号処理言語のための記号処理専用マシンの必要性と、タグやスタックなどの記号処理専用マシンのアーキテクチャ上で重要な機能について述べた。記号処理言語は人工知能や知識工学の分野で利用されており、今後もエキスパートシステムなどの研究が進むにしたがい、利用分野がますます広がっていくと思われる。したがって、記号処理専用マシンのアーキテクチャの研究が重要な課題であるといえる。

第３章　逐次型Ｐｒｏｌｏｇマシンシステムの設計

Ⅲ．１　緒言

本章では、逐次実行型Prologマシンシステムの開発目的と設計方針について述べる。

Prologは前章で述べた記号処理言語の一種であり、人工知能や知識工学への応用が期待されている比較的新しい言語である。Prologはまた、論理型プログラング言語でもあり、これまでの言語にはない数々の特徴を持っている。これらの特徴は、人工知能や知識工学での応用プログラムを書く上で非常に強力な記述能力を与えてくれるが、一方でProlog処理系の効率が低くなる原因にもなっている。

そこで、本章ではProlog言語の特徴と応用分野について概括したあと、Prologを高速に実行するためのアーキテクチャについて検討する。さらに、この検討結果をもとにして、逐次実行型PrologマシンＰＥＫの設計を行う。

Ⅲ．２　Ｐｒｏｌｏｇ言語とその特徴

前章では、非数値データを取り扱う記号処理言語の応用が広まるにともない記号処理専用マシンへの要求が高まってきていることを述べた。

本節で述べるPrologも記号処理言語の一種であり、リストや文字列などの非数値データを処理する能力を備えている。最近では、LISPにかわる、人工知能や知識工学での応用プログラム開発用の言語として注目を集めている。

これまではエキスパートシステムの分野でも、LISPが広く利用されてきた。ところが、1970年フランスのマルセイユ大学で開発されたPrologは、一階の述

語論理を基礎とし、低レベルではあるが推論機構を備えているため、エキスパートシステム開発用の言語として適していることが報告され[Clark 80][Mizoguchi 83]、すでにいくつかのシステムが開発されている[Kimura 82]。

また、知的コンピュータの開発を目指している新世代コンピュータ技術開発機構（略称ＩＣＯＴ）で、第五世代コンピュータ[Fuchi 82]の核言語第０版にPrologを選んだこともあり[Chikayama 83a]、Prologによる応用プログラムの開発が世界中で進められている。

Ⅲ．２．１　Ｐｒｏｌｏｇの歴史

Prologの最初の処理系は15年ほど前の1970年にフランスのマルセイユ大学で開発され、フランス語理解システム、数式処理システム、プラン作成システム、定理証明システムなどに使用された。この処理系は、インタプリタシステムであり、最初はALGOL-W 、 2年後にFORTRAN 上に作成されたが、速度が遅く、応用範囲が限られてくるため、1977年イギリスのエジンバラ大学で、最初のPrologコンパイラシステムが開発された[Pereira 78][Warren 77a]。この処理系の速度はLISP処理系の速度に匹敵すると伝えられており[Warren 77b]、現在で最も広く使用されている処理系の一つである。

そのほか、代表的な処理系としてウォータールー大学版、IC-PROLOG などがあり、日本でもDURAL [Goto 83] 、Prolog/KR などが開発されている。＜付表－１＞に初期のProlog処理系を、＜付表－２＞に代表的な処理系の速度を示しておく。

Prologの機能拡張についても、研究が進められており、LISPとの結合[Sato 83] 、文字列処理の導入[Umemura 83]、並列動作制御機構の導入[Shapiro 82][Clark 83]、オブジェクト指向プログラミングの導入[Chikayama 83b][Takeuchi 83]などが検討されている。

Ⅲ．２．２　Ｐｒｏｌｏｇの特徴

PrologはFORTRAN,ALGOL,COBOL,PL/Iなどの手続き型言語とは異なった実行制御を行う非手続き型言語の一種である。非手続き型の言語には、関数型、論理型、項書き換え型などの種類があるが、Prologはそのうちの論理型プログラミング言語であり、その名前もProgramming in Logicからきている。また、非数値データを取り扱うことから、記号処理言語の一つでもある。

Prologは一階の述語論理に対する推論方法である導出原理（resolution principle）を基礎としている。導出原理は1965年にRobinsonが考案し[Robinson 65]、定理の自動証明[Chang 73]あるいは人工知能の分野で広く応用されてきた[Nilsson 80]。

Prologの実行メカニズムはこの導出原理そのものであり、プログラムを一階の述語論理式として静的に解釈でき、プログラムの実行過程も論理式に対する推論（logical inference ）過程としてとらえることができる。したがって、Prolog処理系の速度を推論の速度（単位LIPS、logical inference per second）によってあらわすことが多い。また、導出原理を用いた推論を実行するために、パターン照合の一般化であるユニフィケーション（unification ）の機能や、推論が間違っていた場合の後戻り処理であるバックトラック（backtracking）の機能を有している。

Prologの理論的基礎は、エジンバラ大学のKowalskiらが与えた[Kowalski 74][Emden 76]。このように、プログラムのセマンティックス（意味論）が明確になっているため、プログラムの自動変換や自動合成あるいは正当性の証明などが容易になる。

Ⅲ．２．３　Ｐｒｏｌｏｇのアプリケーション

これらのProlog言語の特徴を生かしたアプリケーションについても、各種の分野で研究が進められており、Prologの記述力の高さが認識されつつある。

Prologを論理型言語としての側面から見ると、Prologをプログラムの仕様記述、正当性の検証、自動変換、自動合成などに応用することが考えられる。また、プログラムではなく論理回路に対する仕様記述、検証、変換、合成についても研究が進められている。

Prologとデータベース、特に関係データベースシステムとの関連も重要なアプリケーションの一つである。これをさらに進めて、知識工学における知識ベースとの結合についても活発に研究されている。

Prologの人工知能あるいは知識工学への応用も、すでに述べたエキスパートシステム[Mizoguchi 83][Kimura 82] を始めとして、自然言語での質問応答システム、自然言語の構文解析システム[Matsumoto 83]、知識獲得システム[Kitakami 84] 等が発表されている。

その他、数式処理、コンパイラ作成、ＣＡＤ、ＣＡＩなどへの応用も研究が進められている。

Ⅲ．３　Ｐｒｏｌｏｇマシン

Ⅲ．３．１　Ｐｒｏｌｏｇマシンへの期待

前節で述べたように、Prolog言語はこれまでの言語にない特徴を持っており、特に知識工学や人工知能の分野での利用が広まりつつある。

ところが、これらの特徴のため、Prologの実行制御方式はこれまでの言語の実行制御方式と大きく異なっている。たとえば、Prologのユニフィケーションは通常の言語での引数の結合に相当するが、引数の結合の場合、仮引数は常に

変数であり、引数結合時の代入の方向は一方向に固定されている。しかし、ユニフィケーションの場合は、仮引数の位置に構造体などのパターンを記述することができ、代入の方向は実行時でないと決定できない。

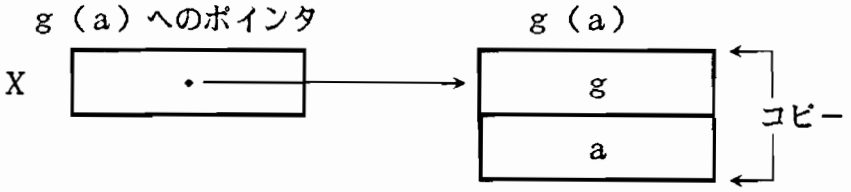
また、通常の言語でのサブルーチン呼び出しあるいは関数呼び出しに相当する述語呼び出しの処理も複雑である。サブルーチンや関数の場合は、呼び出された時に実行管理情報（リターンアドレスや局所変数の値など）を格納するためのコントロールフレームを作成し、呼び出し元へ戻るときにコントロールフレームを消去すればよい。他方、Prologの述語呼び出しの場合は、バックトラック時に前回の続きから実行を再開する必要があるため、直ちにコントロールフレームを消去することはできない。さらに、バックトラック時には変数の値も元の値に戻す必要があり、そのための特別な処理が必要になる。

そのほか、リストなどの構造をもったデータ（構造体、structure ）の取り扱い方法も問題である。現在、構造体の表現方法には、ストラクチャシェアリング（structure sharing ）方式[Boyer 72][Warren 77a]とストラクチャコピー（structure copying ）方式[Bruynooghe 82][Mellish 82] がある。たとえば、次の二つの構造体、

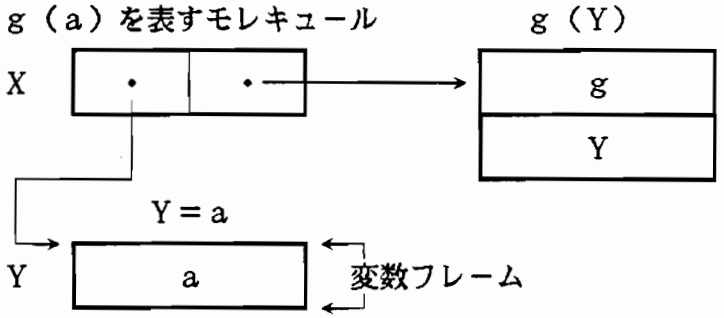
f(X, a)

f(g(Y), Y)

のユニフィケーションを行う場合（ここで大文字は変数を表す）、X にg(a)を Y にa を代入することによりユニファイ（unify ）できる。ここで、X に代入されたg(a)はもとの二つの構造体中には現れていない新しい構造体である。ストラクチャコピー方式では、新しい構造体そのものをコピーにより作成する。



ストラクチャシェアリング方式では、もともとあった構造体g(Y)を利用し、新しい構造体を間接的に表現する。



変数X中の左側のポインタは、構造体中に現れる変数の値を保持している変数フレームの先頭アドレスを示す。このように、構造体を変数フレームと構造体の骨組み（skeleton）の組で表現したものはモレキュール（molecule）と呼ばれる。

ストラクチャコピー方式では、変数への代入時に構造体をコピーする必要があるが、要素を直接に参照できる。一方、ストラクチャシェアリング方式の場合は、代入時にはモレキュールを作るだけでよいが、要素の参照が間接的になる。したがって、代入時のオーバーヘッドと参照時のオーバーヘッドを比較すればよい。ところが、これは実行するPrologのプログラムの性質に大きく依存しており、大きな差は認められないことが報告されている[Shimazu 83]。しかし、実際の処理系、特に初期のものではストラクチャシェアリング方式が多い。

このように、これまでのプログラミング言語の制御構造とPrologの制御構造では異なった部分が多いため、FORTRAN などの既存の言語を効率よく実行する計算機上にProlog処理系を作成しても、Prologのプログラムを効率よく実行できるは限らない。

エジンバラ大学で開発されたDEC-10 Prolog [Bowen 81]は、現在で最も効率のよい処理系であり、同一計算機上のLISP処理系と同程度の性能を持つことが報告されている[Warren 77b]。しかしながら、LISPと同程度の性能が出せるいうことは、逆にいえば、DEC の計算機を含めた一般の計算機上でのProlog処理系の性能の限界と考えることができる。すなわち、DEC-10 Prolog 以上の性能を得るためには、アーキテクチャの改善が必要だと言える。

さらに、DEC-10 Prolog の場合は大型計算機上の処理系であるため、スタンドアロンでパーソナルな使用の可能な低価格のマシンへの要求が高まってきている。

Ⅲ．３．２　ＤＥＣ－１０　Ｐｒｏｌｏｇと　ＤＥＣ－１０／２０のアーキテクチャ

しかし、DEC 2060を含めたDEC-10/20 シリーズのアーキテクチャには興味深い点が多く、汎用計算機ではあるが、いくつかのPrologに適した機能を備えている。すなわち、

① アドレッシング機能が豊富にあり、複雑なデータ構造に対しても高速にアクセスできること

② 一語36ビットの幅を持ち、それを18ビットの２つのフィールドに分割して使用できること

③ 各命令語には、インダイレクトビット（indirect bit）と呼ばれる１ビットが用意されており、そのビットが１になっている限り、自動的に間接ア

ドレス修飾を行う実効アドレス計算（effective address calculation ）

の機能があること

などがPrologの実効速度改善に役立っていると考えられる[Warren 77a]。特に、DEC-10 Prolog で採用しているストラクチャシェアリング方式の場合、構造体をモレキュールと呼ばれる、変数フレームアドレスとスケルトンアドレスの組で表現するが、DEC 2060では上記の②の機能により、モレキュールを一語で表現できる。また、変数に代入された値を参照する時にディリファレンス（dereference ）と呼ばれる、ポインタをたぐる処理が必要になるが、③の機能により機械語一命令で実行できる。したがって、これらの機能はPrologマシンにも必要な機能だと言える。

Ⅲ．３．３　並列実行型Ｐｒｏｌｏｇマシン

Prologマシンには並列実行型と逐次実行型の二つが考えれる。並列実行型Prologマシンは複数台のプロセッサを並列に動作させる方式であり、逐次実行型Prologマシンは基本的には一台のプロセッサにより実行を行う方式である。

Prolog言語は非手続き型の言語であるため、データフロー（data flow ）やリダクション（reduction ）などの並列アーキテクチャとの整合性がよいと考えられており、その方面からの研究が特に盛んに行われている[Ito 84][Motooka 84][Hasegawa 84]。

しかし、実際に使用されているPrologプログラムの並列度は 2～ 3でそれほど大きくないことが報告されており[Onai 84] 、構成要素である一台のプロセッサの性能が高くない限り、全体の性能も上がらないと言える。このことは、D.H.D.Warrenらも、並列Prologマシンでは逐次実行の部分の性能が全体の性能のボトルネックになると指摘している[Tick 84] 。

したがって、逐次実行の性能向上を含めた、逐次実行型Prologマシンの研究

が重要な課題になる。

Ⅲ．３．４　逐次実行型Ｐｒｏｌｏｇマシン

逐次実行型Prologマシンのアーキテクチャに関してもいくつかの研究が進められている。

ＩＣＯＴのＰＳＩ（Personal Sequential Inference ）マシンは、評価用および今後の開発用ツールとして開発されたマシンである[Uchida 83][Taki 84][Yokota 84] 。ＰＳＩマシンでは、DEC-10 Prolog 相当のＫＬＯ（Kernel Language version 0 ）と呼ばれる言語を機械語としてファームウェアにより解釈実行する。マシンアーキテクチャ上の特徴は、タグアーキテクチャ、ハードウェアスタック、マルチプロセス支援用のハードウェア、豊富なアドレス形式、強力なディスパッチ機構などであり、Prolog専用の特殊なハードウェアは備えられていない。しかし、DEC-10 Prolog コンパイラ以上の性能を目標にした場合、たとえば50K LIPSをインタプリタで出そうとすると、マシンサイクルが 200ナノ秒のマシンでは、 100命令の間に一回の推論を行わなければならない。したがって、Prolog専用のハードウェア機能、あるいは単一命令内での複数のハードウェア制御などの機構を取り入れて、一命令の機能を高めない限り、Prologプログラムの高速実行は不可能であろう。

Evan Tick とDavid D.H. Warren が提案したPrologマシンでは、Prologプログラムを低レベルの命令セット（reduced instruction set ）にコンパイルし、パイプライン制御により高速に実行する[Tick 84] 。もちろん、Prologのプログラムを効率のよいコードにコンパイルする方法は重要な研究課題であるが、LISPマシンやPrologマシンの使用形体が会話的利用中心であることを考えると、Prologマシンはコンパイルされたプログラムを高速に実行するだけでなく、インタプリタによる実行も高速に行える能力が必要である。

Ⅲ．３．５　逐次実行型Ｐｒｏｌｏｇマシンのアーキテクチャ

このように、逐次実行型Prologマシンに関する研究はいくつか進められているが、研究の歴史は浅く不十分な点が残されている。したがって、より高い実行効率を目指したPrologマシンのアーキテクチャの研究が望まれる。

これまでに述べた点から、逐次実行型Prologマシンの満足すべき条件をまとめると、

① 高速、高性能であること。少なくともDEC 2060上のDEC-10 Prolog コンパイラと同程度の性能を持つこと

② パーソナルな使用が可能なこと。したがって、マシンの製造コストが低く、ＴＴＬなどの既存のテクノロジーで開発できること

③ インタプリタによる実行形体を主とし、さらにコンパイラも備えていること

などが上げられる。

さらに、アーキテクチャ上で備えるべき特徴は

④ タグやディスクリプタなどのデータの種類を示すための機能を備えていること

⑤ ハードウェアスタックなど、高機能を持ったメモリと豊富なアドレッシング機能を有してること

⑥ 低レベルでの並列処理をできる限り取り入れ、ハードウェアの並列制御を可能にすること

⑦ そのほか、Prolog専用のハードウェア機能を備えており、Prologプログラムの高速実行が可能なこと

などが考えれる。また、構造体の表現方法としてストラクチャシェアリング方式を採用した場合には、DEC-10/20 シリーズのアーキテクチャが参考になる。

すなわち、

⑧　モレキュール単位でのデータ転送、格納が可能であること

⑨　変数の値を取り出すディリファレンス作業が高速に行えること

以上のような点がPrologマシンに必要な機能であると言える。

Ⅲ．４　逐次実行型Ｐｒｏｌｏｇマシンｐｅｋの設計

前節では、逐次実行型Prologマシンの研究が重要な課題であること、また逐次実行型Prologマシンの備えるべき特徴について述べた。

本節では、実際に作成した逐次実行型PrologマシンＰＥＫ（Prolog Engine of Kobe University）の設計方針について述べる。

Ⅲ．４．１　開発の目的と基本設計方針

逐次実行型PrologマシンＰＥＫの開発の第一の目的は、論理型言語Prologのプログラムを高速に実行するためのアーキテクチャについて研究し、実際にどのようなアーキテクチャが適しているのかを明らかにすることである。

したがって、高速実行に必要と思われるハードウェア機構はできる限り採用することにした。しかし一方で、アーキテクチャ研究用の実験機であることを考えると、マシンの規模を小さくし、設計・開発および評価をしやすくする必要がある。特に、仮想記憶、キャッシュ（cache ）、高度のパイプライン処理などは今日の大型計算機では標準的に採用されているハードウェア機構であるが、これらの機構を取り入れるとハードウェアの規模が大きくなりすぎ、そのほかのProlog向きのアーキテクチャの評価が困難になる。そのため、ＰＥＫでは、Prolog専用のハードウェア機構だけを取り入れることにした。さらに、マシンの開発を容易にするために、マイクロプログラム制御方式を採用し、シーケンサ（sequencer ）とＡＬＵ（arithmetic logic unit ）には市販のビット

スライス（bit slice ）ＬＳＩを使用した。また、その他のハードウェアについても標準的なハードウェア技術を用い、特殊な回路技術は使用しないことにした。

そのほか、以下のような基本設計方針をとった。

① Prologの言語仕様は、DEC-10 Prolog と同程度とし、特別な拡張は行わない。DEC-10 Prolog は現在で最も標準的な処理系であり、効率も優れている。また、言語仕様に対して特別な拡張を行うと、言語仕様とアーキテクチャのどちらが効率改善に寄与しているのかが明確でなくなり、アーキテクチャの評価が困難になる。

② マイクロプログラムによるインタプリタ方式を主とする。すでに述べたように、Prologの利用形体は会話的な場合がほとんどであり、インタプリタ方式のほうが適している。Prologコンパイラを導入した場合でも、インタプリタによる実行とコンパイルしたオブジェクトの実行は混在して行えるべきである。したがって、インタプリタによる実行を主とし、コンパイルしたオブジェクトプログラムはインタプリタから呼び出して実行される方式をとる。

③ 構造体の表現法にはストラクチャシェアリング方式を採用する。ストラクチャシェアリング方式とストラクチャコピー方式のどちらが良いかはまだ評価が定まっていない。そこでＰＥＫでは一応ストラクチャシェアリング方式とし、オーバーヘッドとなる部分に関しては特別なハードウェアにより高速化を行うことにした。

④ ホストプロセッサとの結合。ＰＥＫを単体で稼働させることを考えた場合、ディスクの制御回路やＯＳ（オペレーティングシステム）など、ハードウェア・ソフトウェアの両面でマシンの規模が大きくなり、開発が困難にな

る。そこで、市販のコンピュータシステムをＰＥＫに結合し、ハードウェア・ソフトウェアの開発が容易に行えるようにした。

Ⅲ．４．２　ハードウェアの設計方針

ハードウェアに関しては、以上のような基本方針以外に、以下のような設計方針をとった。

① タグアーキテクチャの採用。LISPマシンなどで有効性が示されているタグアーキテクチャを採用する。また、タグの種類を高速に判別するためのハードウェアを付加する。

② 水平型マイクロ命令形式の採用。マイクロ命令の形式には、垂直型と水平型の二種があるが、ハードウェアの並列制御が可能で、マイクロ命令の解釈が高速にできる水平型マイクロ命令形式を採用することにした。

③ メモリの分散化・専用化。一般に高級言語ではいくつかの独立したメモリ領域を使用する。特に、DEC-10 Prolog コンパイラの場合、コード領域、ローカルスタック、グローバルスタック、トレイルなどの領域を使用している。これを通常の計算機と同様の一次元のアドレス空間に割り当てた場合、それぞれの領域に対して各種のアドレス形式が使用されるため、高速にアクセスするためのハードウェアを付加することは困難である。一方、LISPマシンなどではスタックを独立したメモリに取り、高速のアクセスを可能にしている。そこでＰＥＫでは上記のような領域を独立のメモリに割り当て、それぞれに対して特別のアドレス形式で高速にアクセスできるようなハードウェアを付加することにした。しかし、この場合、メモリの容量不足という問題が生じやすくなる恐れがあるが、これは仮想記憶などの付加的なハードウェア技術で解決できると考えた。

④ データ転送能力の増強。ＰＥＫで採用したストラクチャシェアリング方式

では、構造体をモレキュールと呼ばれる変数フレームのアドレスと構造体の骨組み（skeleton）の組で表現する。したがって、実行速度を上げるためには、モレキュールを直接取り扱えるアーキテクチャを採用すべきである。そこで、ＰＥＫではデータをフレーム部、タグ部、バリュー部の三つの部分から構成し、モレキュール単位でのデータ処理を可能にした。さらに、データ高速転送用のバイパス回路を設け、データ転送のスループットを向上させることにした。

⑤ 低レベルの並列処理。ＰＥＫは逐次実行型のPrologマシンであるが、実行速度を上げるため、プログラマに見えないレベルでの並列処理はできる限り取り入れることにした。

⑥ Prolog専用の回路の付加。上に述べたような機能は、特にPrologマシンだけに必要なものではなく、一般の高級言語マシンあるいは記号処理専用マシンの備えるべきアーキテクチャである。しかし、これらの機能だけではPrologのプログラムに対して充分な性能を出すことはむずかしい。なぜなら、これまでにいくつかのLISPマシン上でPrologインタプリタが作成されているが、速度は10～20K LIPS程度であり、DEC-10 Prolog コンパイラに比べて 2～ 4倍の実行時間がかかっている[Takeuchi 84][Otera 84][Hattori 84] 。したがって、DEC-10 Prolog コンパイラと同程度の速度を得るためには、Prolog専用のハードウェアが必要になると考えれる。そこで、ＰＥＫではProlog言語の特徴であり、実行時に占める割合が大きいと思われるユニフィケーション処理とバックトラック処理を中心にハードウェア化を進めることにした。

Ⅲ．４．３　ソフトウェアの設計方針

ソフトウェアの設計方針としては以下のようなものをとった。

① ソフトウェアの開発はホストプロセッサ上で行う。ソフトウェアの開発・保守を容易にするために、ソフトウェアの大部分はホストプロセッサ上で開発することにした。これは、ホストプロセッサ上の既存のソフトウェアを有効に利用し、ソフトウェア作成の労力を大きく軽減することが目的である。

② ハードウェアのデバッグ用のソフトウェア群を用意する。ハードウェアの論理レベルのデバッグ作業を支援するためのソフトウェアを作成する。

③ ユーザフレンドリーなインターフェースを提供する。ユーザあるいは開発者の使用しやすさを考えてソフトウェアを作成する。

④ 拡張の容易な構成をとる。ハードウェアの改良やコンパイラなどの機能の付加に対して柔軟に対応できるような構成にする。

Ⅲ．５　結言

本章では、逐次実行型PrologマシンＰＥＫの開発の目的と設計方針について述べた。

ＰＥＫを設計するにあたっては、Prolog言語の特徴であるユニフィケーション処理とバックトラック処理に注目し、その部分のハードウェア化を目指す。さらに、Prologの実行でボトルネックとなる部分について検討し、その結果に基づいてアーキテクチャの設計を行った。

以下にＰＥＫの基本方針とアーキテクチャ上の設計方針をまとめておく。

① 逐次実行型

② 低コスト

③ マイクロプログラム制御

④ DEC-10 Prolog と同程度の言語仕様

⑤　インタプリタ主体

⑥　ストラクチャシェアリング方式

⑦　ホストプロセッサとの結合

⑧　タグアーキテクチャ

⑨　水平型マイクロ命令

⑩　メモリの分散化・専用化

⑪　データ転送の高速化

⑫　低レベルの並列処理

⑬　Prolog専用のハードウェア

第４章　逐次実行型Ｐｒｏｌｏｇマシンシステムの全体構成

Ⅳ．１　緒言

本章では、開発した逐次実行型Prologマシンシステムの全体構成について述べる。

本システムは、Prologのプログラムの実行を行うＰＥＫマシン部と、ＰＥＫマシンの起動および入出力を担当するホストプロセッサ部から構成されている。ソフトウェアの開発はホストプロセッサ上で行う。

Ⅳ．２　システムの全体構成

逐次実行型Prologマシンシステムの全体構成を＜図Ⅳ－１＞に示す。システムはPrologのプログラムを実行するＰＥＫマシン（あるいはＰＥＫ）の部分と、ＰＥＫマシンの起動やプログラムの読み込み、入出力のサポートなどをおこなうホストプロセッサ（あるいはホスト）の部分から成る。

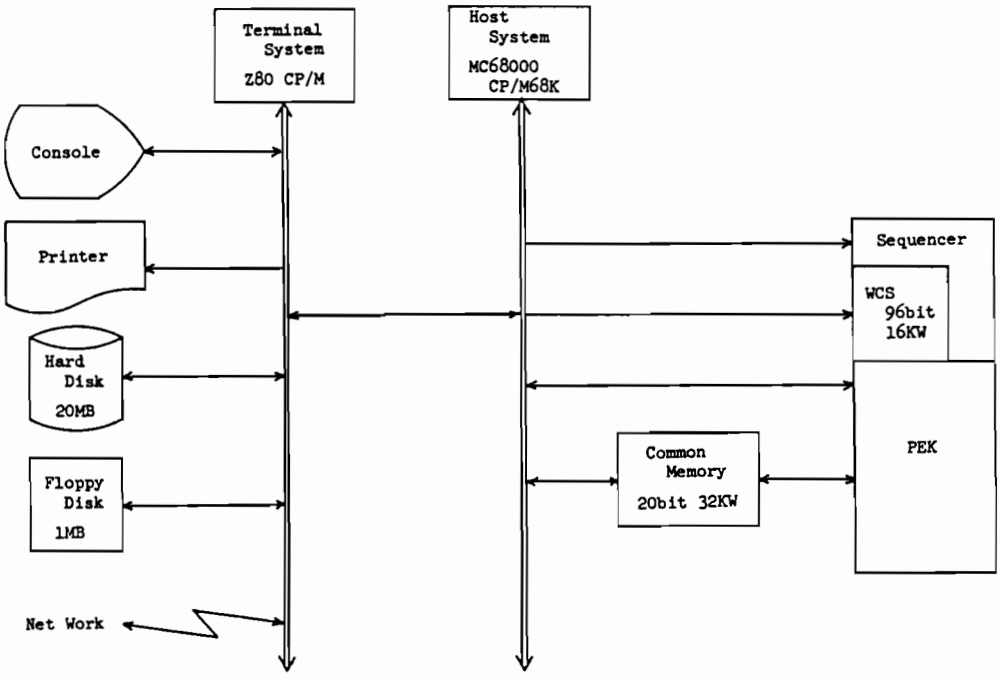
ホストプロセッサには市販のシステムを利用し、ＰＥＫマシンおよびＰＥＫとホストのインターフェース部分について製作を行った。

ホストプロセッサ側には二つのＣＰＵ、モトローラ社の16ビットマイクロプロセッサMC68000 とザイログ社の 8ビットマイクロプロセッサZ80 があるが、ＰＥＫマシンとはMC68000 が結合しており、Z80 は入出力の制御装置としての役割を果たす。ただし、システムの開発段階ではZ80 上のソフトウェアを利用することも可能である。MC68000 上には、CP/M-68Kと呼ばれるシングルユーザ用のオペレーティングシステムがあり、テキストエディタ、Ｃコンパイラなどが使用できる。また、Z80 上のCP/Mには、スクリーンエディタ、Prologインタ

プリタなどのソフトウェアがある。システムの開発はこれらのソフトウェアを利用して行った。

なお、ホストプロセッサには、ディスプレイ、キーボード、プリンタ、フロッピーディスク（容量１Mバイト）、ハードディスク（容量20Mバイト）、ローカルエリアネットワークなどが接続されており、ＰＥＫマシンからもホストプロセッサを介して利用できる。

ホストプロセッサとＰＥＫマシンとは、三つのコマンドレジスタおよび共有メモリによって結合されている。ホストプロセッサはコマンドレジスタを通して、ＰＥＫの実行を制御する。



＜図Ⅳ－１＞　システムの全体構成

さらに、ホストプロセッサ側からは、ＰＥＫのマイクロプログラムを格納するＷＣＳ（Writable Control Storage）の読み書きや、ＰＥＫのマイクロプログラムカウンタ、実行時間計測用のタイマ、実行命令数のカウンタなどの読み出しが可能となっている。

Ⅳ．３　結言

本章では、逐次実行型Prologマシンシステムの全体構成について述べた。

システムはホストプロセッサとＰＥＫマシンから成っており、ホストプロセッサとＰＥＫマシンは三つのコマンドレジスタおよび共有メモリによって結合されている。また、ソフトウェアはホストプロセッサ上で開発することにし、既存のソフトウェアを有効に利用する。

第５章　逐次実行型Ｐｒｏｌｏｇマシンシステムのハードウェア

Ｖ．１　緒言

本章では、逐次実行型Prologマシンシステムのハードウェア構成について述べる。

特に、第３章で述べたPrologマシンの備えるべきアーキテクチャ上の特徴の実現方法について述べる。すなわち、ＰＥＫマシンは、

- タグアーキテクチャ
- 水平型マイクロ命令
- メモリの分散化・専用化
- データ高速転送用のバイパス回路

などの特徴を持ち、さらに

- モレキュール単位でのデータ処理
- 構造体データ読み込みのパイプライン化
- マッチング回路
- 自動トレイル回路
- 自動アンドゥ回路

などのユニフィケーションおよびバックトラック処理用の専用ハードウェア機能を備えている。

Ｖ．２　アーキテクチャ上の特徴

逐次実行型PrologマシンＰＥＫは第３章で述べたような特徴を備えるように設計を行った。そのうち特に、Prolog専用に設けたハードウェア機能は次のよ

うなものである。

1．モレキュール単位でのデータ処理能力。すでに述べたように、データをフレーム部（14ビット）、タグ部（ 4ビット）、バリュー部（16ビット）の３つの部分から構成し、ストラクチャシェアリング方式での構造体の取り扱いを容易にする。

2．構造体データの読み込みのパイプライン化（pipelined reading of structures ）。ユニフィケーション処理では、２つの構造体の要素を連続して読み込む必要がある。そのためＰＥＫではアドレスレジスタとデータレジスタをそれぞれ２組づつ用意し、さらにデータリードをパイプライン化することで、構造体の要素の読み込みを高速に行えるようにした。

3．マッチング回路（matching circuit）。ユニフィケーション処理中で、２つのデータが同一かどうか、あるいはどちらかが変数かどうかなどを判定するマッチング作業が必要になる。この作業を高速に行うためのマッチング回路を付加する。

4．自動トレイル回路（automatic trailing circuit）。Prologでは、変数への代入時にその変数のアドレスをトレイル（trail ）と呼ばれる特別なスタックにプッシュして覚えておく必要がある。この作業（トレイル作業と呼ぶ）を自動化する。

5．自動アンドゥ回路（automatic undoing circuit ）。バックトラック時には、上記のトレイル作業で覚えておいた変数の値を未定義（undefined ）の状態に戻す必要がある。ＰＥＫではこの作業（アンドゥ作業と呼ぶ）をサブシーケンサ（subsequencer）により、メインのシーケンサの動作とは並列に実行する。

Ｖ．３　ハードウェア構成

ＰＥＫマシンのハードウェア構成を＜図Ｖ－１＞に示す。

ハードウェアは大きく分けて、Prologのプログラムを実行するためのハードウェア、ホストプロセッサとのインターフェース用ハードウェア、デバッグおよび評価用のハードウェアから成る。＜図Ｖ－１＞には実行用のハードウェアだけが示されている。

これらのハードウェアは45cm×28cm 300ピンの基板５枚の上に構成されており、全ＩＣ数は約 600石である。５枚の基板には以下のようなハードウェアが実装されている。

［Ｎｏ．１］　CCU ボード

シーケンサ、ＷＣＳ、およびＣＭＲ，ＩＣＲ，ＯＣＲ等のホストプロセッサMC68000 とのインターフェース用のレジスタ

［Ｎｏ．２］　ALU ボード

ＡＬＵ、Ｒバイパス、Ｓバイパス、プロセスメモリ、ハードウェアスタックなど

［Ｎｏ．３］　UNIFICATION ボード

グローバルスタック、トレイルスタック、マッチング回路、アンドゥ回路などのユニフィケーションに関するハードウェア

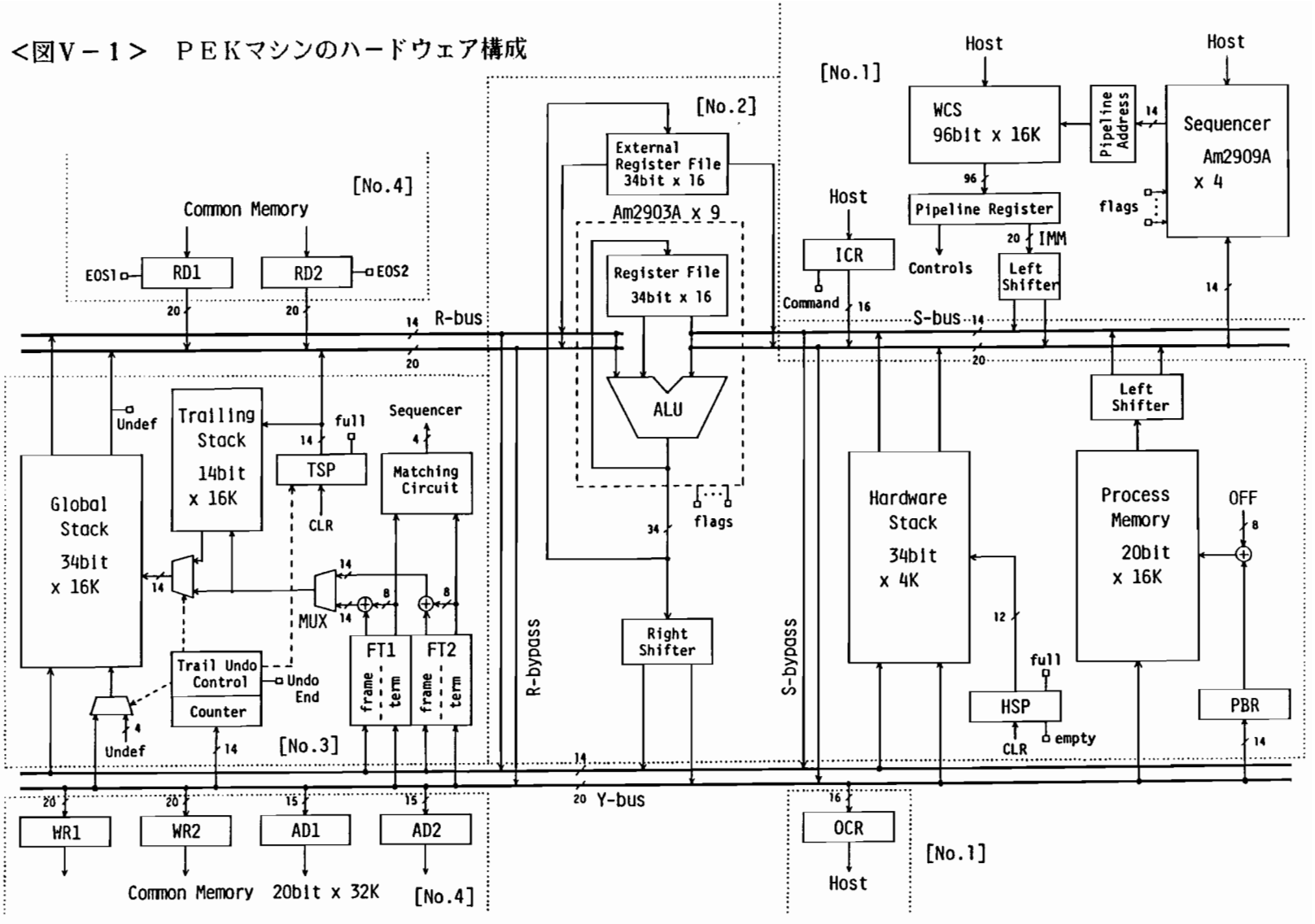
［Ｎｏ．４］　COMMON MEMORY ボード

共有メモリおよびＡＤ１，ＡＤ２，ＲＤ１，ＲＤ２，ＷＲ１，ＷＲ２などのアドレスレジスタとデータレジスタ

［Ｎｏ．５］　評価用ボード

実行時間計測用のタイマ、実行命令数計測用のカウンタなど

<図V－1>　PEKマシンのハードウェア構成



ＰＥＫはProlog向きのアーキテクチャの研究を目的とした実験機であるためマシンの規模も大きくなく、比較的短期間で設計・製作できた。

Ｖ．４　実行用ハードウェアの機能

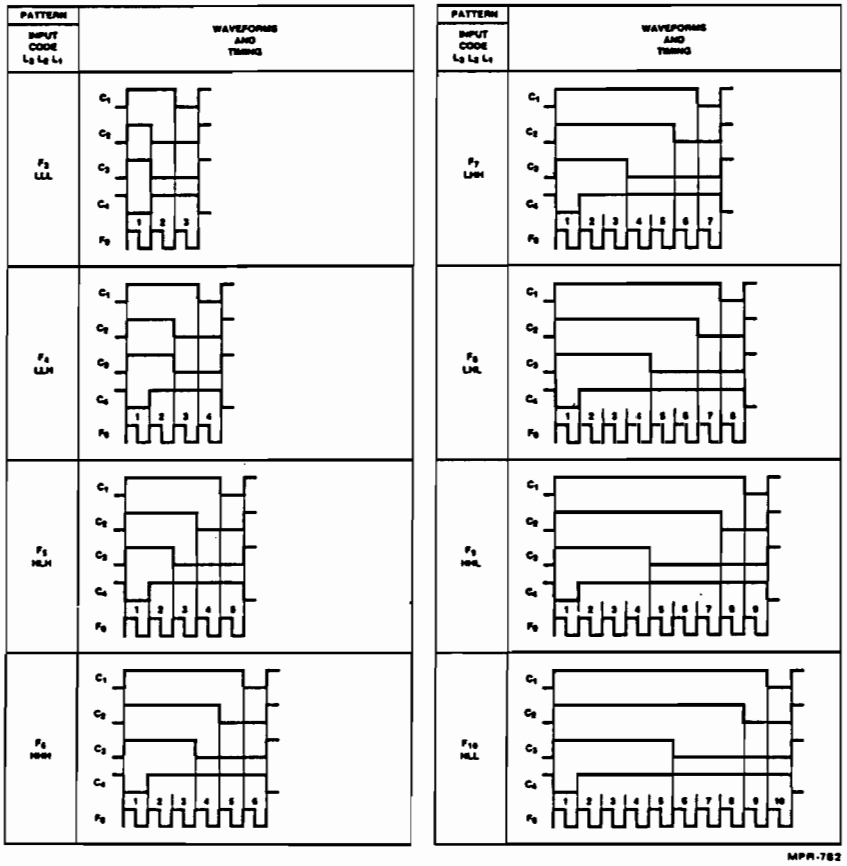
本節では実行用ハードウェアの機能とその特徴について述べる。

Ｖ．４．１　マイクロプログラム制御

ＰＥＫではハードウェアのコストを下げ、開発を容易にするためにマイクロプログラム制御（microprogram control）方式を採用した。シーケンサにはＡＭＤ（Advanced Micro Devices）社の４ビットスライスマイクロプロセッサAm2909A [AMD 83]を４個接続したものを使用している。したがってマイクロプログラムのアドレス空間は 4× 4ビットの16ビット（64K語）が可能であるが、そのうちの14ビット（16K語）を使用している。また、マイクロプグラム格納用のメモリは書き込みの可能なＷＣＳ（Writable Control Storage）とし、マイクロプログラムの開発・変更を容易にできるようにした。ＷＣＳへのプログラムのロードと起動はホストプロセッサが行う。

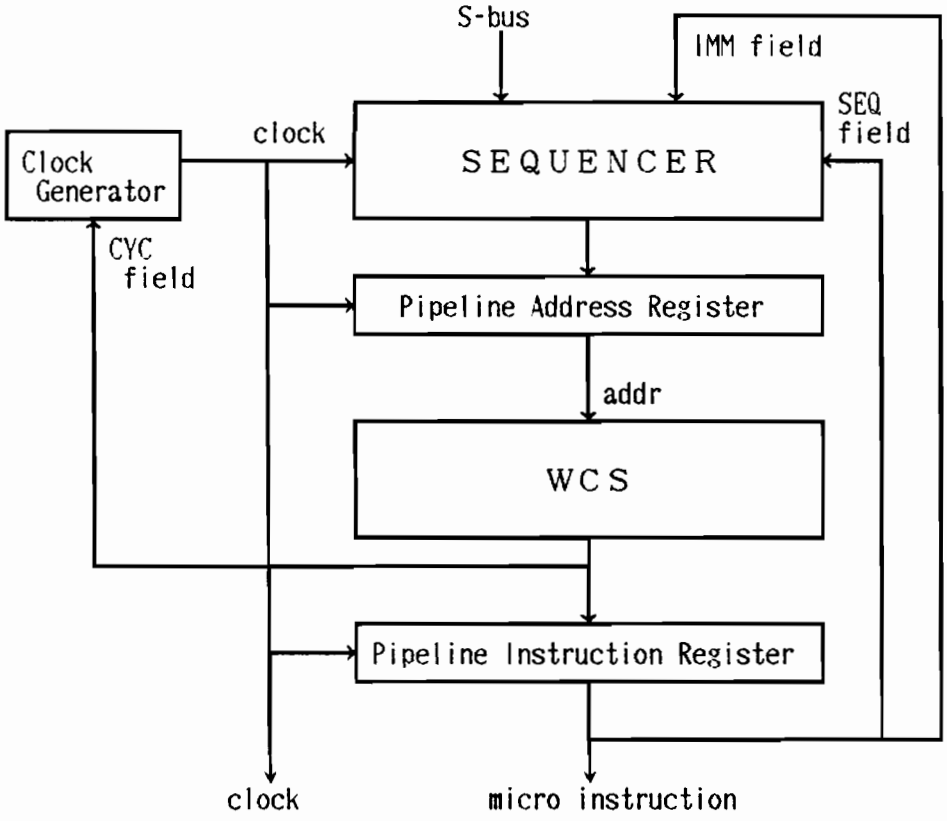
マイクロ命令は96ビット幅の水平型で、24個のフィールドから構成されており、複数のハードウェアの並列制御が可能である。

クロックジェネレータにはＡＭＤ社のAm2925を使用し、クロック長を可変にした。クロック長の制御はマイクロ命令中の３ビット幅のフィールドを使用する。これにより、命令ごとに最適なクロック長で実行することが可能になった。クロックは４相で 120ナノ秒～ 400ナノ秒のうち40ナノ秒単位の 8種が利用できる<図Ｖ－２>。



<図Ｖ－２>　Ａｍ２９２５のクロック

また、マイクロ命令の実行と次のマイクロ命令の読み込みとをオーバーラップさせた、１レベルパイプライン制御（1 level pipeline control）を行っている<図Ｖ－３>。



<図Ⅴ－３>　シーケンサでのパイプライン制御

さらに、パイプラインアドレスレジスタ（pipeline address register ）とパイプライン命令レジスタ（pipeline instruction register ）とを設けることにより、継続命令（分岐を行わない命令）の実行速度を向上させた。すなわち、分岐命令の場合は、サイクルの前半で分岐先アドレスの生成を行い、サイクルの後半で命令のフェッチ（fetch ）を行っているが、継続命令に対してはサイクルの前半で次の命令のフェッチを行っている。このようなパイプライン処理により、継続実行で最短 120ナノ秒の実行速度が可能になった。

＜表Ｖ－１＞　分岐条件

FMX	分岐条件
0x0000	バリュー部　≦　signed
0x0001	バリュー部　＞　signed
0x0010	バリュー部　＜　signed
0x0011	バリュー部　≧　signed
0x0100	バリュー部　＝
0x0101	バリュー部　≠
0x0110	バリュー部　overflow
0x0111	バリュー部　not overflow
0x1000	使用しない
0x1001	使用しない
0x1010	バリュー部　≧　unsigned
0x1011	バリュー部　＜　unsigned
0x1100	バリュー部　≦　unsigned
0x1101	バリュー部　＞　unsigned
0x1110	バリュー部　minus
0x1111	バリュー部　plus
100000	フレーム部　＝
100001	フレーム部　≠
100010	フレーム部　≧　unsigned
100011	フレーム部　＜　unsigned
10010n	Hardware Stack Full
10011n	Hardware Stack Empty
10100n	使用しない
10101n	Undo End
10110n	Trailing Stack Full
10111n	Undefined FT1
11000n	End of Structure RD1
11001n	End of Structure RD2
11010n	ICR command
11011n	Undefined FT2
11100n	True
11101n	True
11110n	True
11111n	True

x：don't care

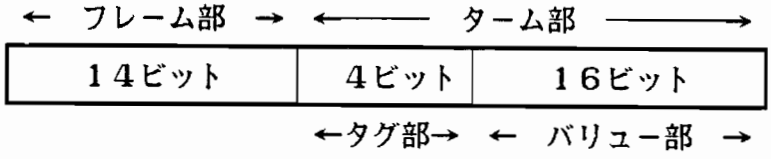
n：0　positive
　　1　negative

分岐命令としては、シーケンサ内のマイクロスタック（４語、マイクロサブルーチンなどで使用）、Ｓバス上のデータ（レジスタ、プロセスメモリ、ハードウェアスタックなどの内容による間接分岐）、あるいはマイクロ命令中のイミディエイトフィールドによる分岐が可能である。特に、イミディエイトフィールドによる分岐は、Ｓバスを使用せずに行うことが可能であり、命令中の並列度を高めるのに役立っている。

条件分岐は、ＡＭＤ社のAm2904の使用により36種の条件判断が可能になっている＜表Ｖ－１＞。さらに、Am2909A のＯＲ入力端子を利用して、マッチング回路からの出力結果に応じた16通り２語おきの多方向分岐（multiway jump ）機能を付加した。

Ｖ．４．２　語形式

ＰＥＫで取り扱うデータは＜図Ｖ－４＞に示すように、14ビットのフレーム部と20ビットのターム部から成り、さらにターム部は 4ビットのタグ部と16ビットのバリュー部から構成される。



＜図Ｖ－４＞　ＰＥＫの語形式

タグ部はデータの種類を示すための部分で、タグとデータタイプは＜表Ｖ－２＞のように割り当てている。バリュー部の値はタグの種類によって異なる意味を持つ。

<表Ⅴ－２>　タグとデータの種類

タグ値	タグ名	データの種類	バリュー部の意味
0	EOS	end of structure	－
1	UNDEF	undefined	－
2	CODE	micro code	マイクロプログラムのアドレス
3	LIT	literal atom	アトムヘッダのアドレス
4	INT	integer	数値
5	－	－	－
6	－	－	－
7	－	－	－
8	GVAR	global variable	グローバル変数の番号
9	LVAR	local variable	ローカル変数の番号
A	VVAR	void variable	－
B	CTERM	compound term	ポインタ
C	LIST	list	ポインタ
D	CLAUSE	clause	ポインタ
E	－	－	－
F	NONE	－	－

DEC-10 Prolog ではリストは・（ドット）を関数名（functor ）とするコンパウンドターム（compound term ）で表現されている。しかし、リストはPrologで最も頻繁に取り扱われるデータ構造であり、このようにポインタの連鎖による表現方法は好ましくない。そこで、ＰＥＫではリストを別扱いとし、特別のタグを設けて、連続アドレスにリストの要素を格納するようにした。リストの終端は、ＥＯＳ（End Of Structure）と呼ぶタグを付けたセルを余分に付け加えることで表現する。この場合、リストの終端かどうかを調べるには、一つ先のセルを読みだす必要があるが、その検出はハードウェアで行う。

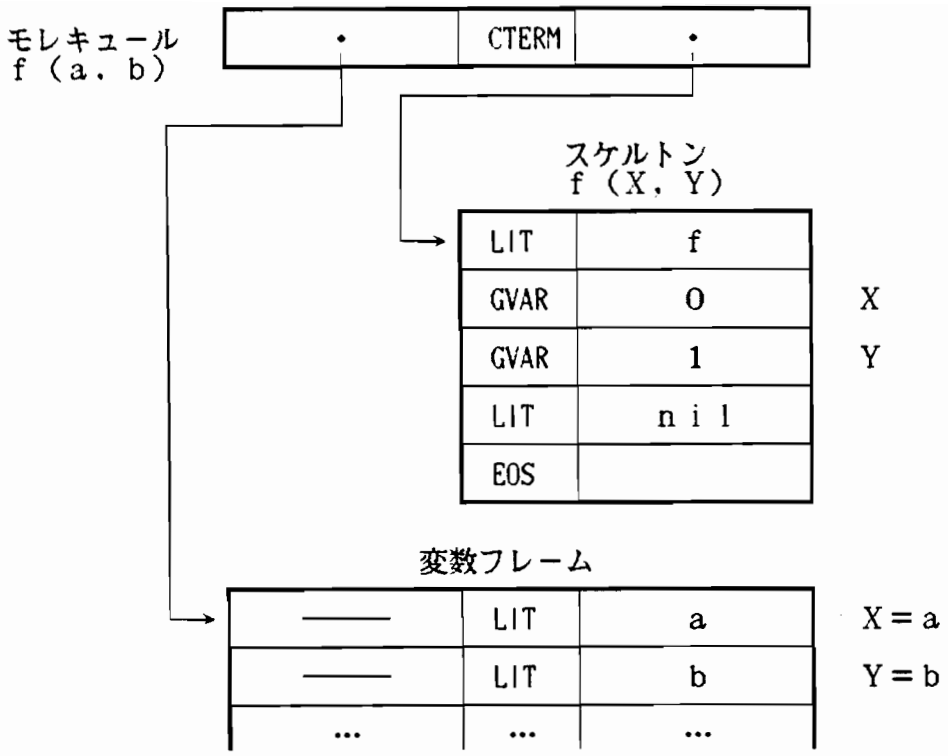
コンパウンドタームについてもリストと同様に不定長の表現を許し、構造体の表現方法を統一する。

フレーム部はターム部がグローバル変数かローカル変数、あるいはそれらの変数を含む構造体（コンパウンドタームまたはリスト）の時に意味があり、変数の値を格納している変数フレームの先頭アドレスを示している。したがって、構造体をモレキュールとして一語（34ビット）で表現できる。たとえば＜図Ｖ－５＞中のモレキュールはf(X,Y)というスケルトンと {X=a, Y=b} という変数フレームの組であり、f(a,b)なる構造体を表現している。

Ｖ．４．３　データパスとバス構成

ＰＥＫのデータパスを＜図Ｖ－６＞に示す。

内部バスはすべて34ビット幅で、ソースバスとしてＲバスとＳバス、デスティネーションバスとしてＹバスの合計３つのバスがある。通常は、ＲバスとＳバスに出力されたデータはＡＬＵ（arithmetic logic unit ）により処理され、Ｙバスを経由して各種のメモリやレジスタに書き込まれるが、ＰＥＫではさらに、バイパス（bypass）用の回路を設け、ＲバスからＹバスあるいはＳバスからＹバスへ高速にデータを転送できるようにした。

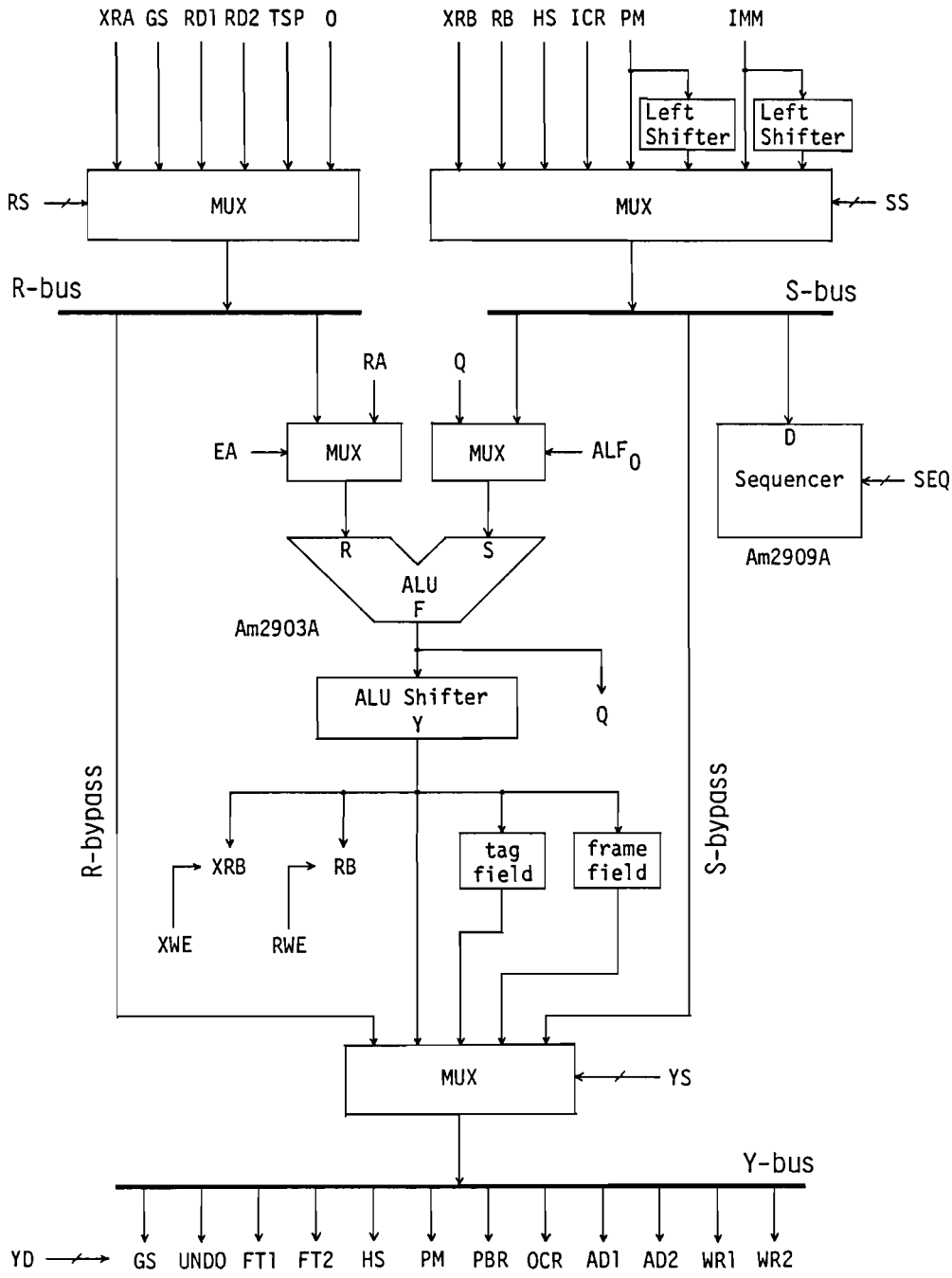


<図Ｖ－５>　モレキュールの例

このように、複数のデータバスが用意されているため、データの並列処理が可能となっている。たとえば、シーケンサによるＳバスを利用した分岐と、ＲバスからＹバスへのデータ転送、およびＡＬＵでの演算を一つのマイクロ命令で同時に実行できる。

Ｖ．４．４　メモリモジュールとアドレス形式

ＰＥＫでは、複数のメモリへの並列アクセスと、複雑なアドレス形式でのアクセスを高速化するためにメモリモジュールを分散化した。共有メモリとレジスタを除いて、すべてアクセスタイムが55ナノ秒のスタティックメモリを使用している。



<図V－6> ＰＥＫのデータバス

1．ＷＣＳ　(Writable Control Storage)

96ビット×16Ｋ語のメモリで、マイクロプログラムが格納される。ホストプロセッサから自由に読み書きできる。

2．共有メモリ　(Common Memory)

ホストプロセッサとの共有メモリで、20ビット×32Ｋ語の容量を持つ。アトムヘッダ（atom header ）、構造体、Prologプログラムなどが格納される。

3．プロセスメモリ　(Process Memory)

20ビット×16Ｋ語のメモリで、Prologプログラム実行中の管理情報の格納に用いる。述語の呼び出しごとに、管理情報格納用のコントロールフレーム（control frame ）が一つ作られる。

4．グローバルスタック　(Global Stack)

グローバル変数とローカル変数の格納に利用する34ビット×16Ｋ語のメモリである。グローバル変数領域とローカル変数領域はグローバルスタックを２つの部分に分割して使用する。

5．トレイルスタック　(Trailing Stack)

バックトラック時に未定義の状態に戻すべき変数のアドレスを格納するためのメモリで14ビット×16Ｋ語の容量を持つ。

6．ハードウェアスタック　(Hardware Stack)

34ビット×　4Ｋ語の容量を持ち、ユニフィケーション処理で使用する。

7．レジスタファイル　(Register File)

34ビット×16語の内部レジスタと同数の外部レジスタがあり、大域的な情報の確保や、局所的なデータの保存に用いる。

これらのメモリモジュールには、それぞれに適したアドレス形式があり、Ｐ

ＥＫではそれをハードウェアでサポートしている。すなわち、

１．共有メモリのアドレスレジスタに対する自動インクリメント。

２．プロセスメモリに対するベースアドレス形式。

３．グローバルスタックに対する変数セルアドレスの自動計算。

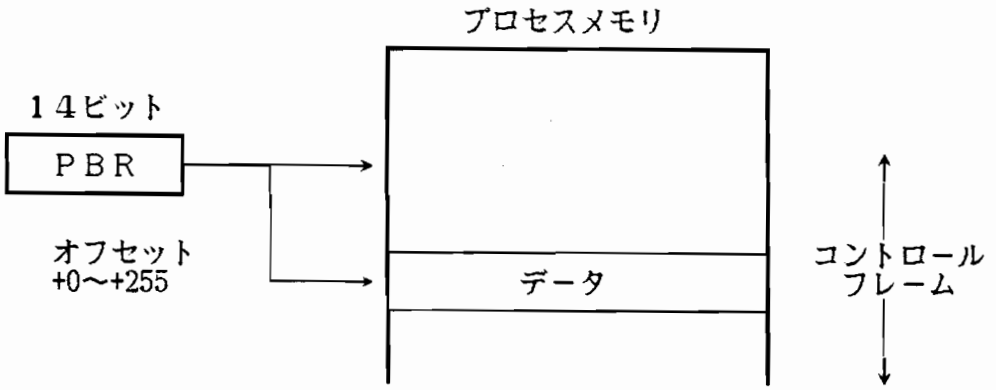
４．ハードウェアスタックに対するスタックアドレス形式。

などである。これらの機能は、メモリモジュールを独立させたため、比較的小規模のハードウェアの付加で可能になった。

Ｖ．４．５　実行制御用ハードウェア

Prologにおける実行メカニズムは通常の言語のメカニズムとは大きく異なっている。これはPrologに、バックトラック機能が備わっているためで、一度実行に成功して制御を呼び出し元に戻したあとも、再呼び出しによる継続実行が必要になる場合がある。そのため、実行管理情報を格納するコントロールフレームは再呼び出しに備えるため、簡単に消去することができない。さらに、バックトラックのたびに、使用するコントロールフレームを切り替える必要がある。そこでＰＥＫでは、コントロールフレーム格納用のメモリ（プロセスメモリ）を特別に用意し、バックトラック制御の高速化を図った。

プロセスメモリは、専用のベースレジスタＰＢＲ（Process Base Register ）を持ち、ＰＢＲに対する+0～+255までのオフセット値をマイクロ命令中で指定できる。したがって、ＰＢＲ＋0 からＰＢＲ＋255 の範囲は１マイクロ命令でアクセスでき、コントロールフレームの切り替えもＰＢＲの書き換えだけでよい<図Ｖ－７>。



＜図Ｖ－７＞　プロセスメモリのアクセス形式

Ｖ．４．６　ユニフィケーション用ハードウェア

ＰＥＫでは、ユニフィケーションの高速化のために

- 構造体データの読み込みのパイプライン化
- マッチング回路
- 変数セルのアドレスの自動計算機構

などの機能を備えている。

Ｖ．４．６．１　構造体データ読み込みのパイプライン化

Prologのユニフィケーション処理は２つのデータに対して行われる。そこでＰＥＫではアドレスレジスタ、読み込みレジスタ、書き込みレジスタをそれぞれ２組づつ設けた（ＡＤ１とＡＤ２、ＲＤ１とＲＤ２、ＷＲ１とＷＲ２）。また、構造体の要素を連続アドレスに置くことにし、連続アドレスのデータリードをパイプライン化した。すなわち、読み込みレジスタからのデータリードによりアドレスレジスタを自動インクリメントし、次アドレスのデータの読み込みを開始する。データは約 250ナノ秒後に読み込みレジスタに準備される。

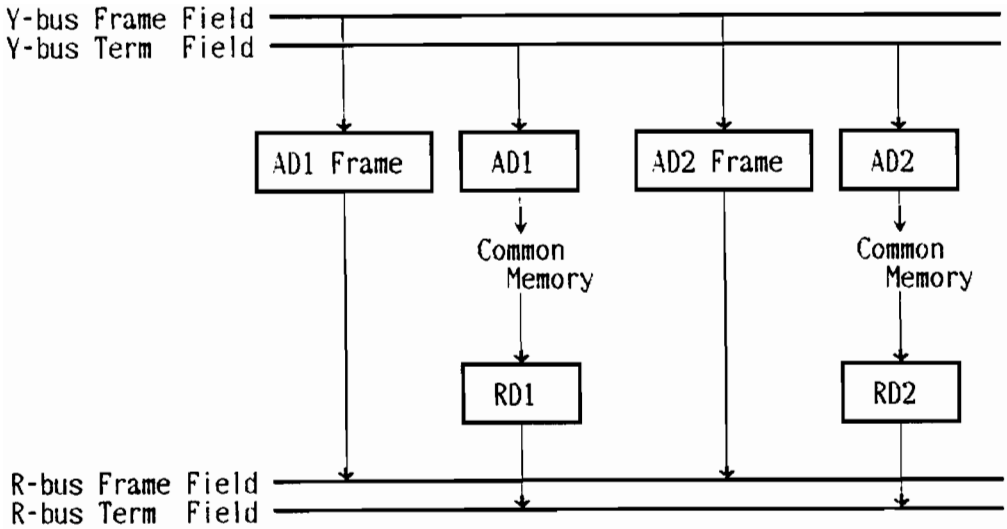
また、共有メモリを疑似的に２ポート化することにより、アドレスの設定、データの読み込み、データの書き込みを２つのポート（ＡＤ１側とＡＤ２側）でオーバーラップできるようにした。

連続アドレスのデータリードが高速になったため、リストに関してもDEC-10 Prolog 流のポインタの連鎖による表現ではなく、要素を連続アドレスに置くことにし、アクセス回数の減少をはかった。リストの最後にはＥＯＳ（End Of Structure）と言う特別のタグを持つセルを付け加えておく。また、読み込みレジスタ内の値がＥＯＳかどうかを判別するためのフラグ（EOS1，EOS2）を用意した。

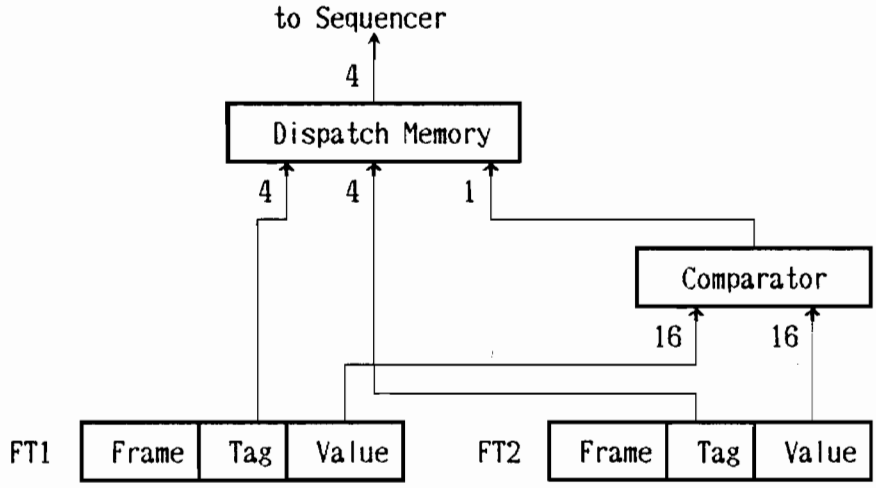
さらに、構造体の要素の読み込み時に、グローバル変数のフレームアドレスをフレーム部に自動的に付け加えるための、フレームレジスタを用意した。すなわち、共有メモリへのアドレスレジスタＡＤ１、ＡＤ２に構造体の先頭アドレスを書き込む時に、フレーム部の値をグローバル変数のフレームアドレスにしておくと、データレジスタＲＤ１、ＲＤ２からの読み込み時に、設定したグローバル変数のフレームアドレスがフレーム部に自動的に付け加えられる＜図Ｖ－８＞。

Ｖ．４．６．２　マッチング回路

Prologではユニフィケーション時に、２つのデータの種類に応じて異なる処理をする必要がある。そこで、ＰＥＫでは２つの専用レジスタＦＴ１とＦＴ２（おのおの34ビット幅）を設け、タグ部 4ビットづつと、バリュー部の比較結果 1ビットの計 9ビットにより、16通り２語おきのマルチウェイジャンプを行えるようにした。すなわち、同一アトムどうしかどうかを１命令で判別できる。またＦＴ１、ＦＴ２のどちらか一方に特別のNONEと呼ばれるタグを持つデータを書き込むと、反対側のタグの値に応じてマルチウェイジャンプする。



＜図Ｖ－８＞　フレームレジスタ



＜図Ｖ－９＞　マッチング回路

このマッチング回路は、上記の９ビットをアドレスとする４ビット幅のＲＯＭである＜図Ｖ－９＞。ＲＯＭの内容を＜表Ｖ－３＞に示す。

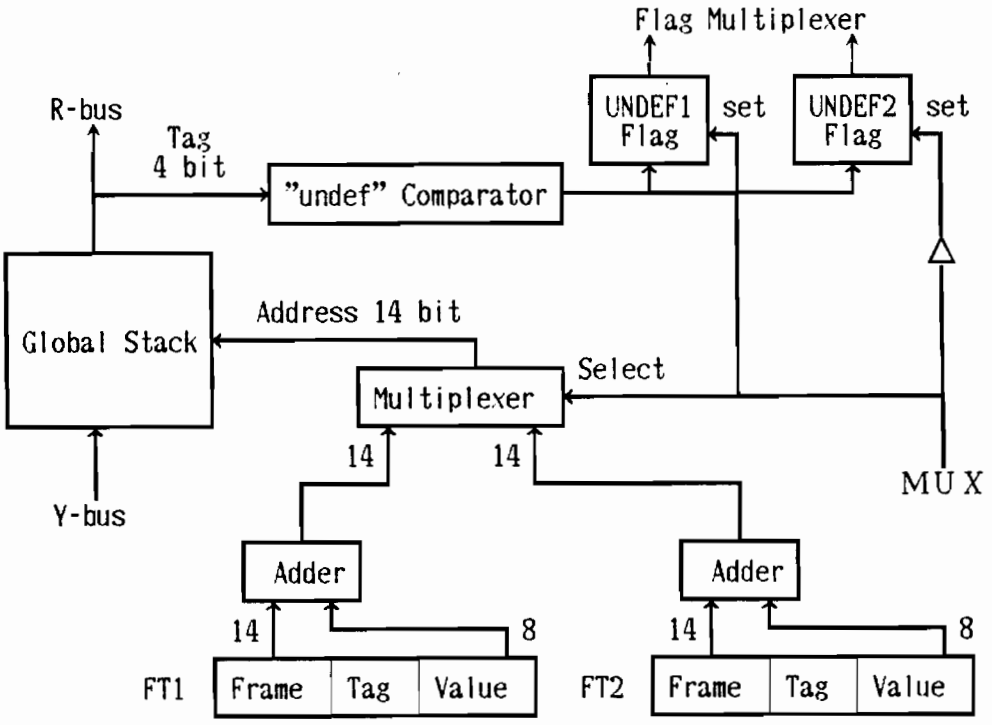
＜表Ｖ－３＞　マッチング用ＲＯＭの内容

FT2 \ FT1	0 EOS	1 UNDEF	2 CODE	3 LIT	4 INT	5	6	7	8 GVAR	9 LVAR	A VVAR	B CTERM	C LIST	D CLAUSE	E	F NONE
0 EOS																0
1 UNDEF																1
2 CODE																2
3 LIT				F／1	F				6	8	4	F	F			3
4 INT				F	F／1				6	8	4	F	F			4
5																5
6																6
7																7
8 GVAR				5	5				9	B	4	5	5			8
9 LVAR				7	7				C	A	4	7	7			9
A VVAR				4	4				4	4	4	4	4			A
B CTERM				F	F				6	8	4	2	F			B
C LIST				F	F				6	8	4	F	3			C
D CLAUSE																D
E																E
F NONE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

未記入は０　　欄内は　≠／＝

Ｖ．４．６．３　変数セルのアドレスの自動計算

上記のレジスタＦＴ１、ＦＴ２を利用して、変数セルのアドレスの自動計算を行う。すなわち、ＦＴ１のフレーム部14ビットとバリュー部の下位８ビット（変数のインデックス値、 0～ 255）の合計、あるいはＦＴ２についての同様の値のどちらかをグローバルスタックへのアドレスとして使用できる<図Ｖ－１０>。



<図Ｖ－１０>　変数セルアドレスの自動計算

ＦＴ１およびＦＴ２のフレーム部には通常はグローバル変数領域のフレームアドレスをセットしておく。したがってローカル変数の場合にはＦＴ１またはＦＴ２のフレーム部の値を書き替える必要がある。ＦＴ１、ＦＴ２のどちらを選ぶかはマイクロ命令中の１ビット（ＭＵＸフィールド、０のときＦＴ１側で１のときＦＴ２側となる）で指定する。

また、変数に代入されている値が未定義かどうかは実際にグローバルスタックから値を読み出さなくてもフラグ（ＦＴ１側はUNDEF1、ＦＴ２側はUNDEF2）で判断できるようになっている。

Ｖ．４．７　バックトラック用ハードウェア

Ｖ．４．７．１　自動トレイル回路

Prologではバックトラック時に変数の値を未定義値に戻すためのアンドゥ作業を行うため、変数へ値を代入する時に変数セルのアドレスをトレイルスタックにプッシュしておく必要がある。ＰＥＫではこの作業をハードウェアにより自動的に行う。すなわちグローバルスタックへの書き込み時に変数セルのアドレスをトレイルスタックへ自動的にプッシュする。プッシュの必要ない場合のために、プッシュするか否かはマイクロ命令中のＴＳＣフィールドで指定できるようにしてある。

Ｖ．４．７．２　自動アンドゥ回路

ＰＥＫではアンドゥ作業も自動化している。この作業はサブシーケンサにより行われ、メインのシーケンサの動作とは並列に実行できる。サブシーケンサの動作はアンドゥカウンタにアンドゥの回数を書き込むことによって起動される。また、トレイルスタックからの読み出しとグローバルスタックへの書き込みをパイプライン化することにより、高速のアンドゥ作業（一回80ナノ秒）を可能にしている。また、アンドゥ作業の終了はフラグによって調べることが

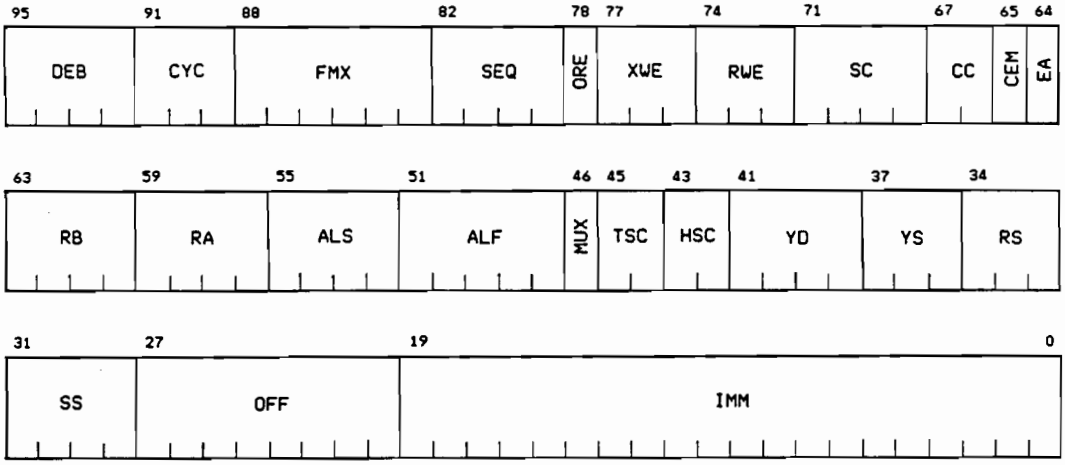
できる。

Ｖ．５　マイクロ命令形式

ＰＥＫのマイクロ命令のフォーマットと各フィールドの名称を＜図Ｖ－１１＞に示す。

ＰＥＫのマイクロ命令は１語96ビットの水平型であり、24のフィールドから構成される。

- ＤＥＢフィールドはデバッグおよびシステム評価用のもので、Haltビット、タイマ／カウンタのStart ビット、Stopビット、命令数カウント用のビットの計４ビットから成っている。ブレークポイントの設定や、実行時間計測用のタイマのストップ／スタート、実行命令数のカウンタのストップ／スタート、カウントの指定などに用いる。
- ＣＹＣフィールドはクロックジェネレータAm2925コントロール用のフィールドで、８種（120 ～400 nsec）のクロックの選択を行う＜図Ｖ－２＞。
- ＦＭＸフィールドはフラグマルチプレクサのコントロール用で、分岐条件の選択に用いる。条件には次のようなものがある＜表Ｖ－１＞。
  - ①　ＡＬＵの演算結果のフラグ（ステータスレジスタ）
  - ②　ハードウェアスタックのEmpty フラグ、Fullフラグ
  - ③　トレイルスタックのFullフラグ
  - ④　アンドゥ作業の終了検出フラグ
  - ⑤　UNDEF1フラグ、 UNDEF2 フラグ
  - ⑥　EOS1フラグ、EOS2フラグ
  - ⑦　ＩＣＲへのコマンド到着フラグ



position	length	name	
95 - 92	4	DEB	Debug
91 - 89	3	CYC	Cycle Control (Am2925)
88 - 83	6	FMX	Flag Multiplexer Control
82 - 79	4	SEQ	Sequencer (Am2909A) Control
78	1	ORE	Or Enable Address Source
77 - 75	3	XWE	External Register Write Enable
74 - 72	3	RWE	Register Write Enable
71 - 68	4	SC	Shift Control (Am2904)
67 - 66	2	CC	Carry Control (Am2904)
65	1	CEM	Condition Enable (Am2904)
64	1	EA	Enable A (Am2903A)
63 - 60	4	RB	Register B (Am2903A)
59 - 56	4	RA	Register A (Am2903A)
55 - 52	4	ALS	ALU (Am2903A) Shift Operation
51 - 47	5	ALF	ALU (Am2903A) Function
46	1	MUX	Multiplex Global Stack Address Source
45 - 44	2	TSC	Trailing Stack Control
43 - 42	2	HSC	Hardware Stack Control
41 - 38	4	YD	Y-bus Destination
37 - 35	3	YS	Y-bus Source
34 - 32	3	RS	R-bus Source
31 - 28	4	SS	S-bus Source
27 - 20	8	OFF	Offset to Process Base Register
19 - 0	20	IMM	Immediate Data

<図Ｖ－１１>　マイクロ命令形式とフィールド名

- ＳＥＱフィールドはシーケンサAm2909A への命令を指定する。シーケンサは４段のマイクロスタックを持っているので、サブルーチンのコールやループなどが可能である。また、Ｓバス上のデータをアドレスとする分岐やＩMMフィールドで指定されたアドレスへの分岐が行える。
- ＯＲＥフィールドはマルチウェイジャンプ用のフィールドで、このビットを１にセットすることにより、マッチング回路の出力結果に基づいて多方向分岐が実行される。
- ＸＷＥ、ＲＷＥフィールドは外部および内部レジスタへの書き込み指定用のフィールドで、フレーム部、タグ部、バリュー部それぞれ独立に指定できる。書き込むレジスタの番号はＲＢフィールドで指定する。
- ＳＣ、ＣＣフィールドはＡＬＵへのシフト入力、キャリー入力コントロール用のフィールドで、ＡＬＳ、ＡＬＦフィールドと組み合わせて用いる。
- ＣＥＭフィールドはＡＬＵの演算結果のフラグをステータスレジスタにセットするために用いる。
- ＥＡフィールドはＡＬＵのＲ側ソースの選択を行う。０のときＲ側ソースは内部レジスタとなり、１のときＲバスとなる。
- ＲＢ、ＲＡフィールドはＡＬＵのＳ側（Ｂ側）あるいはＲ側（Ａ側）のソースレジスタ（内部および外部レジスタ）番号の指定に用いる。ＲＢフィールドはデスティネーションレジスタ番号の指定も兼ねている。
- ＡＬＳ、ＡＬＦフィールドはＡＬＵであるAm2903A のシフト演算および算術・論理演算命令用のフィールドである。演算はフレーム部、タグ部、バリュー部ごと独立に行われる。
- ＭＵＸフィールドはすでに述べたようにグローバルスタックのアドレス

ソースの選択を行うためのもので、０の時ＦＴ１側、１の時ＦＴ２側である。また、UNDEF フラグのリフレッシュも行われる。

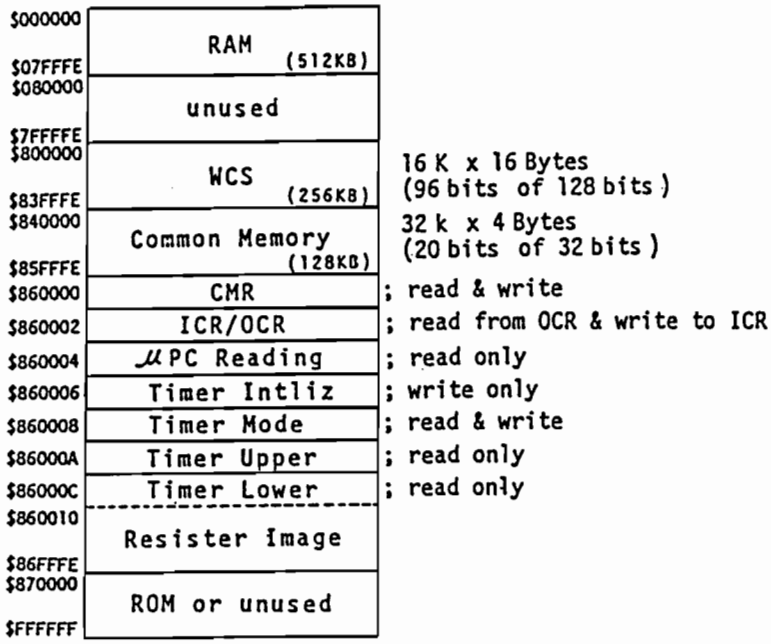
- ＴＳＣ、ＨＳＣフィールドはそれぞれトレイルスタック、ハードウェアスタックのコントロール用のフィールドで、スタックのクリアあるいは自動プッシュ／ポップの指定を行う。
- ＹＤフィールドはＹバスのデスティネーション指定用に使用される。
- ＹＳフィールドはＹバスソースの指定およびライトシフタの制御を行うためのフィールドである。
- ＲＳフィールドはＲバスのソース指定用のフィールドである。
- ＳＳフィールドはＳバスのソース指定およびレフトシフタの制御に用いる。
- ＯＦＦフィールドはプロセスメモリのベースレジスタに対するオフセット（+0～+255）指定用のフィールドである。
- ＩMMフィールドはＳバスへ出力する定数値設定用のフィールドで、20ビット幅である。通常はＳバスのターム部に出力され、レフトシフタを利用した場合はＳバスのフレーム部に出力される。

Ｖ．６　ホストプロセッサとのインターフェース

ＰＥＫマシンは、MC68000 をホストプロセッサとするマイクロコンピュータシステムに、制御用のレジスタと二種類の通信用レジスタおよび共有メモリを介して接続されている。二種類の通信用レジスタは、インプットコマンドレジスタＩＣＲとアウトプットコマンドレジスタＯＣＲであり、ＩＣＲはホストプロセッサからＰＥＫへ、ＯＣＲはＰＥＫからホストへ通信する時に使用する。データ幅はともに16ビットであり、データの到着はそれぞれのフラグによって

示される。さらに、大量のデータの通信には共有メモリを利用する。

ホストプロセッサから制御用レジスタや通信用レジスタおよび共有メモリへは、MC68000 のメモリ空間を通して参照する。ＷＣＳやタイマ／カウンタなどに対しても同様の方法で参照できる。MC68000 のメモリ空間のマップを＜図Ｖ－１２＞に示す。



＜図Ｖ－１２＞　ＭＣ６８０００のメモリマップ

Ｖ．７　デバッグ・評価用ハードウェア

ＰＥＫマシンのデバッグ・評価用にいくつかのハードウェアが設けられている。

１．ＣＭＲ　（コマンドレジスタ）

ＣＭＲはＰＥＫマシンの制御および状態監視用のレジスタで、ＰＥＫマシンの実行／停止の切り替えや状態表示、ステップ実行、ブレークポイントの有効／無効の切り替え、ＯＣＲへのコマンド到着のフラグ表示などの機能を含んでいる。

２．タイマ／カウンタ

マイクロプログラムの実行時間や実行命令数をカウントするためのハードウェアで、マイクロ秒単位の時間計測、全マイクロ命令のカウント、特定マイクロ命令のカウントの三通りのモードがあり、32ビット幅のカウンタである。

これらのハードウェアをマイクロ命令中のＤＥＢフィールドおよびマイクロプログラムカウンタの読み出し用ハードウェアと組合わせて使用することにより、各種の評価データを収集できる。

Ｖ．８　結言

本章では、逐次実行型Prologマシンシステムのハードウェアとアーキテクチャ上の特徴について述べた。

本システムでは、第３章で述べたPrologマシンの備えるべき特徴

- タグアーキテクチャ
- 高機能のメモリと豊富なアドレッシング機能

- 低レベルの並列処理
- Prolog専用のハードウェア機能

を備えており、さらに、

- 水平型マイクロ命令
- メモリの分散化・専用化
- データ高速転送用のバイパス回路

などの機能を取り入れた。

Prolog専用の機能に関しては

- モレキュール単位でのデータ処理
- 構造体データ読み込みのパイプライン化
- マッチング回路
- 自動トレイル回路
- 自動アンドゥ回路

など、ユニフィケーションおよびバックトラック処理用の専用ハードウェアを備えており、Prologプログラムの実行速度の改善に大きく役立つものと思われる。また、システムのデバッグ・評価用のハードウェアを備えさせ、システムの開発・保守・評価が容易に行えるようにした。

第6章　逐次実行型Ｐｒｏｌｏｇマシンシステムのソフトウェア

Ⅵ．１　緒言

本章では逐次実行型Prologマシンシステムのソフトウェア構成およびPrologインタプリタの構成について述べる。

本システムのソフトウェアは、開発支援用ソフトウェアシステムとPrologインタプリタシステムから成る。開発支援用ソフトウェアシステムは、システムの開発・保守・デバッグを支援する目的で作成されたもので、モニタ／デバッガシステム、マイクロアセンブラシステムなどから成る。Prologインタプリタシステムは、Prologのプログラムを実行するためのシステムであり、ＰＥＫマシンの性能を充分に発揮できるように設計されている。

Ⅵ．２　ソフトウェア構成

ＰＥＫのソフトウェアシステムは大きく

- 開発支援用ソフトウェアシステム
- Prologインタプリタシステム

の二つから成っている。開発支援用ソフトウェアシステムはＰＥＫのハードウェアおよびソフトウェアの開発を支援する目的でホストプロセッサ上に作成されたものである。PrologインタプリタシステムはＰＥＫ上でPrologプログラムを実行する時に使用するシステムで、ホストプロセッサ上のプログラムとＰＥＫマシン上のプログラムから成る。

開発したソフトウェアは約一万行である。

Ⅵ．３　開発支援用ソフトウェア

高級言語マシンなどのシステムを開発するときに大きな問題となる点は、ハードウェアのデバッグや保守である。さらにマイクロプログラム制御方式を採用している場合は、マイクロプログラムの開発やデバッグも問題になる。

開発支援用ソフトウェアはこれらの作業を能率的に行えるように開発されたソフトウェア群で、

- モニタ／デバッガシステム
- マイクロアセンブラシステム

などから成っている。

Ⅵ．３．１　モニタ／デバッガシステム

モニタ／デバッガシステムはハードウェアおよびマイクロプログラムのデバッグを主目的としてホストプロセッサ上に開発されたシステムで、

- ＭＯＰ　(Monitor Of PEK)
- ＭＭＡＰ　(Mini Micro Assembler of PEK)
- ＰＢＵＧ　(PEK debugger)

の三つのプログラムから成る。プログラムは大部分がＣ言語で、一部アセンブラで記述されており、CP/M-68K上で稼働する。

ＭＯＰシステムは上記の三つを統合するシステムで、

- ＰＥＫマシンの状態表示と制御
- ＷＣＳのメモリテスト、内容の表示、書き換え、逆アセンブラ、簡易アセンブラ、ファイルへのセイブとロード
- 共有メモリのテスト、内容の表示、書き換え、ファイルへのセイブとロード
- タイマ／カウンタの制御、実行のプロフィール

- ファイルのディレクトリや内容の表示、CP/M-68K中の他のプログラムの実行
- スクリーンエディタ
- MMAP、PBUGシステムの呼び出し

などを行う機能を持っている。<図Ⅵ－１>にＭＯＰシステムの全体構成を示す。

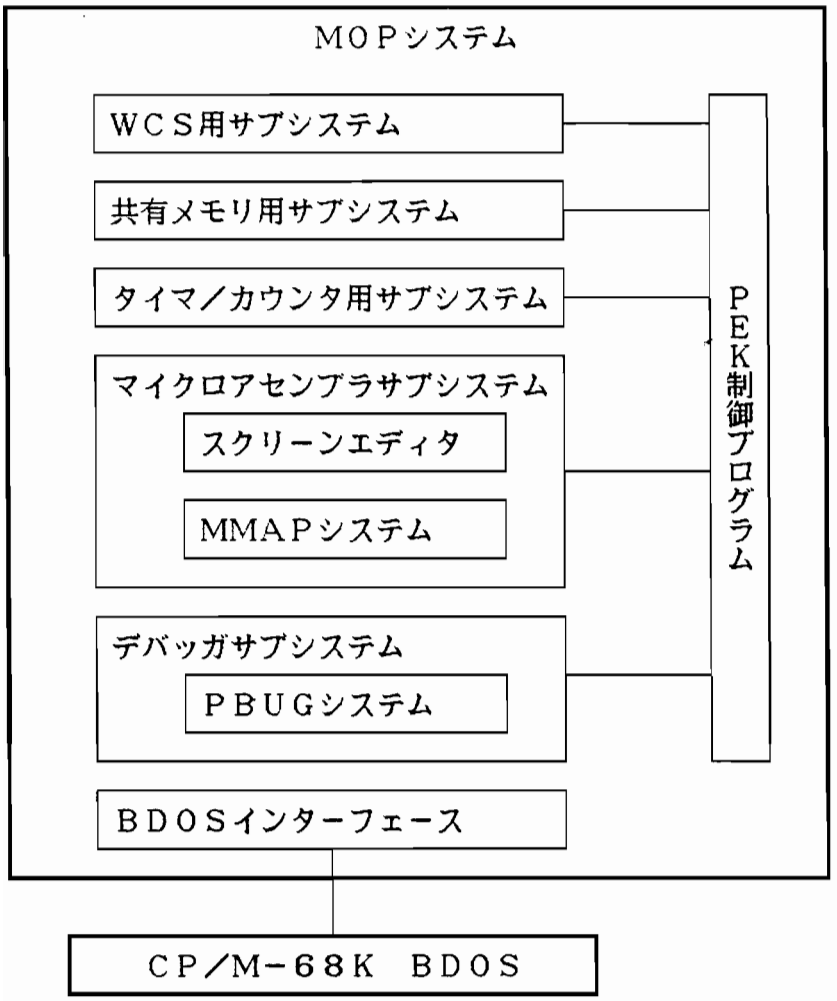
コマンドのメニューやＰＥＫマシンの状態、タイマ／カウンタの値などは、常にディスプレイ上の一定箇所に表示されており、上記の機能は一文字のコマンドで実行できる。また、コマンドのマクロ機能も有しており、一連のキー入力シーケンスを登録できる。さらに、メモリの内容の書き換えなどは、スクリーンエディタ風に任意の場所にカーソルを移動して行うことが可能である。画面のハードコピーもキー入力の任意の時点で行える。

MMAPはＰＥＫのマイクロプログラム開発用のミニアセンブラで、後述のマイクロアセンブラシステムの一部である。このMMAPをＭＯＰ中に組みこんであるスクリーンエディタと結合させることにより、ハードウェアデバッグ用のマイクロプログラムの開発が効率よく行えるようにした。

ＰＢＵＧはマイクロプログラムのデバッグを支援する目的で開発されたプログラムで、ＭＯＰに組みこまれている。ＰＢＵＧには、

- ＰＥＫマシン内のレジスタ、フラグ、メモリなどの内容の表示、書き換え、クリア
- ブレークポイントの設定

等の機能がある。ＰＥＫマシン内のレジスタの値などは、直接ホストプロセッサからアクセスすることはできない。そのため、ＰＢＵＧでは、ホストプロセッサからの制御機能を使って、ＰＥＫマシンのＷＣＳにホストへのデータ転送


ＭＯＰシステム
ＷＣＳ用サブシステム
共有メモリ用サブシステム
タイマ／カウンタ用サブシステム
マイクロアセンブラサブシステム
スクリーンエディタ
ＭＭＡＰシステム
デバッガサブシステム
ＰＢＵＧシステム
ＢＤＯＳインターフェース
ＰＥＫ制御プログラム
ＣＰ／Ｍ－６８Ｋ　ＢＤＯＳ

＜図Ⅵ－１＞　ＭＯＰシステム

を行う命令を書き込み、それをステップ実行させることで、通信を実現している。したがって、ＰＥＫマシン上にプログラムを常駐させる必要はなく、任意の時点でＰＢＵＧを使用できる。

Ⅵ．３．２　マイクロアセンブラシステム

ＰＥＫのマイクロ命令は96ビット幅の水平型であり、24ものフィールドから構成されているため、マイクロプログラムの開発を効率よく行うには、高度な機能をもつマイクロアセンブラが必要である。

今回、開発したマイクロアセンブラシステムＭＡＰ（Micro Assembler of PEK）は、パターンマッチによるマクロ機能を持つプリプロセッサ部（ＰＲＥＭＡＰプログラム）と、その結果をアセンブルするアセンブラ部（ＭＭＡＰプログラム）から成る。アセンブラ部はモニタ／デバッガシステムのＭＯＰ内に組みこまれているものと同一のプログラムである。

＜図Ⅵ－２＞にマイクロアセンブラシステムの構造を示す。プリプロセッサＰＲＥＭＡＰはZ80 上のProlog-1で記述されており、ＭＭＡＰプログラムはMC68000 上のＣ言語で記述されている。

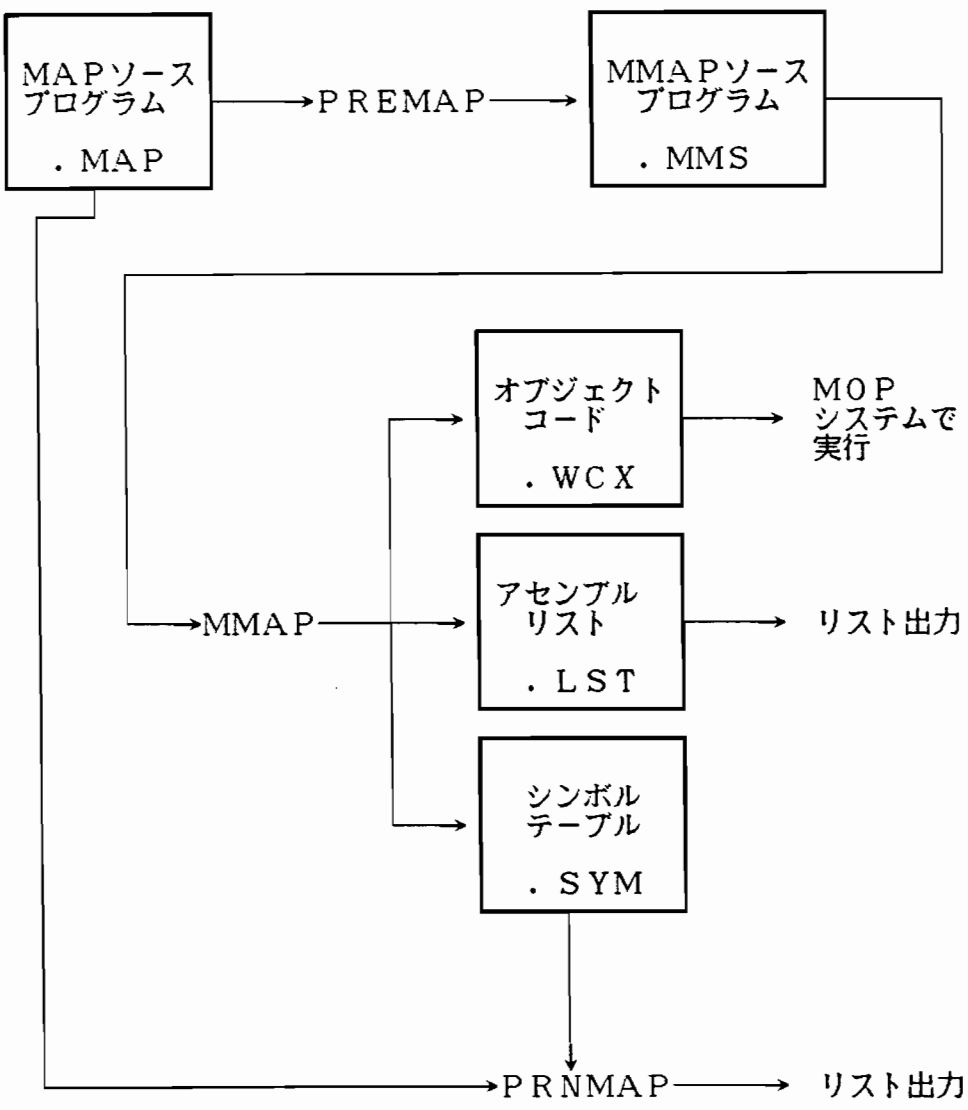
プリプロセッサではProlog風の挿入記法を許し、さらにパターンマッチによる高度なマクロ展開機能を持っている。たとえば、入力コマンドレジスタＩＣＲの内容を、出力コマンドレジスタＯＣＲにＡＬＵを通して転送する命令は

ｏｃｒ　＝　ａｌｕ（ｉｃｒ）．

と記述する。ここで、ａｌｕ（）はＩＣＲのデータがＡＬＵとＳバイパスのどちらを通して転送されるかを区別するために必要である。この命令はＭＭＡＰで処理可能なフィールドごとの命令の列に展開される。

＞ＯＣＲ　ＡＬＵ　ＳＰＣ　ＣＣ．０　＜ＩＣＲ

＞ＯＣＲはＹバスデスティネーションがＯＣＲであることの指定、ＡＬＵはＹ



<図Ⅵ－２>　マイクロアセンブラシステム

バスソースがＡＬＵの出力であることの指定、ＳＰＣはＡＬＵの演算命令（S plus carry）でＳバス上のデータとキャリー（carry ）入力の和を計算して出力するための指定、ＣＣ．０はキャリー入力が０であることの指定、＜ＩＣＲはＳバスソースがＩＣＲであることの指定である。

このように、プリプロセッサは記述を簡潔にし、プログラムの開発を容易にしてくれる。これには、Prologのユニフィケーションすなわちパターンマッチの機能が大きく貢献している。たとえば、上記の例ではＩＣＲがＳバスに接続されていることを判断し、さらにＡＬＵからＹバスへの出力を可能にするため、ＳＰＣという命令を生成している。すなわち、ａｌｕ（ｉｃｒ）のカッコ内をｉｃｒ以外に変更した場合、それぞれの場合に応じた命令列に展開する。例を上げると、

ｏｃｒ　＝　ａｌｕ（ｇｓ（ｆｔ１））．

という命令は、

＞ＯＣＲ　ＡＬＵ　ＲＰＣ　ＣＣ．０　＜ＧＳ　ＭＵＸ．０

という命令列に展開される。これは、アドレスソースをＦＴ１レジスタとして、グローバルスタックを読み出し、ＯＣＲに書き込むための命令である。グローバルスタックがＲバス側に接続されているため、ＡＬＵの演算命令はＲＰＣ（R plus carry）に変更されている。また、ＭＵＸ．０はグローバルスタックのアドレスソースがＦＴ１レジスタであることの指定である。

さらに、ＰＲＥＭＡＰでは複数行への命令の展開も可能で、＜図Ⅵ－３（ａ）＞のようなマクロを定義しておくと、高級言語風のif-then-else構文などの記述が可能になる。すなわち、＜図Ⅵ－３（ｂ）＞のプログラムは＜図Ⅵ－３（ｃ）＞を経過して＜図Ⅵ－３（ｄ）＞に展開される。

```
?- op(100,fx,if), op(99,yfx,then), op(99,yfx,else).
define( if COND then THEN else ELSE, [
          gensym(L1), gensym(L2),
          jmp(COND, #L1),
          ELSE,
          jmp(#L2),
    L1:   THEN,
    L2:   []
]).
```

(a)

```
if eos1 then  ocr = sbyp(#1)
        else  [ alu(rb(0) = #1),
                ocr = alu(rb(0))
              ].
```

(b)

```
          jmp(eos1, #L00000).
          alu(rb(0) = #1).
          ocr = alu(rb(0)).
          jmp(#L00001).
  L00000: ocr = sbyp(#1).
  L00001: [].
```

(c)

```
  JMP ?EOS1 <IMM #.L00000
  RWE.7 RB.0 SPC CC.0 <IMM #.1
  >OCR ALU SPC CC.0 <RB RB.0
  JMP TRUE <IMM #.L00001
L00000
  >OCR SBYP <IMM #.1
L00001
```

(d)

<図VI－３>　マイクロアセンブラのマクロ機能

MMAPプログラムは、PREMAPプリプロセッサによってフィールドごとの命令に変換されたプログラムをアセンブルし、オブジェクトコード、リスト、シンボルテーブルをファイルに出力する。アセンブル機能には、

- #include　　他のファイルのインクルード
- #field　　マイクロ命令中のフィールド位置の指定
- #macro　　フィールドのニーモニック指定
  　　およびマクロ定義
- #label　　定数ラベル定義
- #default　　フィールドのデフォールト値定義
- #org　　ロケーションアドレス指定
- #adjust　　ロケーションアドレスを指定された
  　　ビットパターンになる場所まで進める

などがあり、小規模のプログラム開発は、PREMAPを使用しなくても充分可能である。

マイクロアセンブラのもう一つの機能として、命令のサイクル長の決定がある。PEKで使用しているクロックジェネレータAm2925は 8種類（40ナノ秒単位 120～ 400ナノ秒）、 4相（C1～C4）のクロックを生成するが<図V－2>、その選択はマイクロ命令中の 3ビット幅のCYCフィールドで指定する。したがって、アセンブル時にマイクロ命令から各種デバイスの動作時間を求めて、サイクル長（F3～F10 のどれか）を決定する必要がある。この時、問題になる点は

① 複数のデータバス、大きくわけてALU、Rバイパス、Sバイパス、シーケンサの四つの経路がある。

② デバイスごとの動作時間が大きく異なる。たとえばTSPからRバスへの

読み出しは42ナノ秒であるが、グローバルスタックからRバスへの読み出しは 137ナノ秒かかる（ＦＴ１あるいはＦＴ２の二つのアドレスソースのどちらかをＭＵＸフィールドで選択するため）。

③ ＡＬＵやシーケンサなどは命令の種類や分岐条件によって、動作時間が異なる。特にＡＬＵはその組み合わせが膨大になる。

④ 複数命令にわたる最適化が可能である。たとえば、条件分岐命令の場合、一つ前の命令でも同一条件の選択を行っておくと（分岐命令は書かない）39ナノ秒速くなる。また、グローバルスタックの読み出しの場合も、一つ前の命令からアドレスソース（ＦＴ１またはＦＴ２）の選択を行っておくと97ナノ秒速くなる。

今回作成したプログラムでは以上の点について次のように対処した。

① 四つの経路それぞれについて動作時間および条件を求めてから最大のものを選び出す。

② それぞれのデバイスについて動作時間と15種（ 4相のクロックのエッジからエッジまでの15通り）のクロック条件の表を与えておく。

③ シーケンサについては 6通り、ＡＬＵについても大きく 6通りに分類するにとどめた。

④ アセンブラが複数命令を参照してサイクル長を定めるのは困難なため、特別な条件に関してはプログラマが命令中に記述することにした。

サイクル長決定のプログラムは、二通り作成し、比較検討を行った。

一つ目のプログラムでは、各デバイスの動作時間を40ナノ秒ごとに区切って合計の時間を求めた。この方法の場合、ほとんどの条件を考慮する必要がなくなり、簡単にサイクル長を決定できた。ハードウェアでサイクル長を定めたとすると、この方法によると思われる。

二つ目のプログラムでは、40ナノ秒ごとに区切ることはせず、同一バス上のデバイス動作時間の合計を求めた。

＜表Ⅵ－１＞に二つのプログラムで求めたサイクル長と、第２のプログラムで特別な条件（グローバルスタックのアドレスソースを一つ前の命令でも指定しておく）がある場合のサイクル長を示す。（Ａ）が第１のプログラム、（Ｂ）が第２のプログラム、（Ｃ）が特別な条件を与えた場合である。

＜表Ⅵ－１＞　マイクロ命令のサイクル長

マイクロ命令の内容	（Ａ）	（Ｂ）	（Ｃ）
グローバルスタックから読み出して、ＡＬＵで論理演算とシフト演算を行う	２８０	２４０	１６０
グローバルスタックから読み出して、ＡＬＵで割り算を行う	３２０	２８０	２４０

Ⅵ．４　Ｐｒｏｌｏｇインタプリタ

本節では、ＰＥＫ上に開発したPrologインタプリタの言語仕様、設計方針、構成について述べる。

Ⅵ．４．１　Ｐｒｏｌｏｇ言語の仕様

すでに述べたように、Prologの言語仕様はDEC-10 Prolog と同程度とし、特別な拡張は行わない。DEC-10 Prolog は現在で最も標準的な処理系であり、効率もすぐれている。そこで、処理系の実現方法についてもDEC-10 Prolog コンパイラ[Warren 77a]の方法を参考にした。

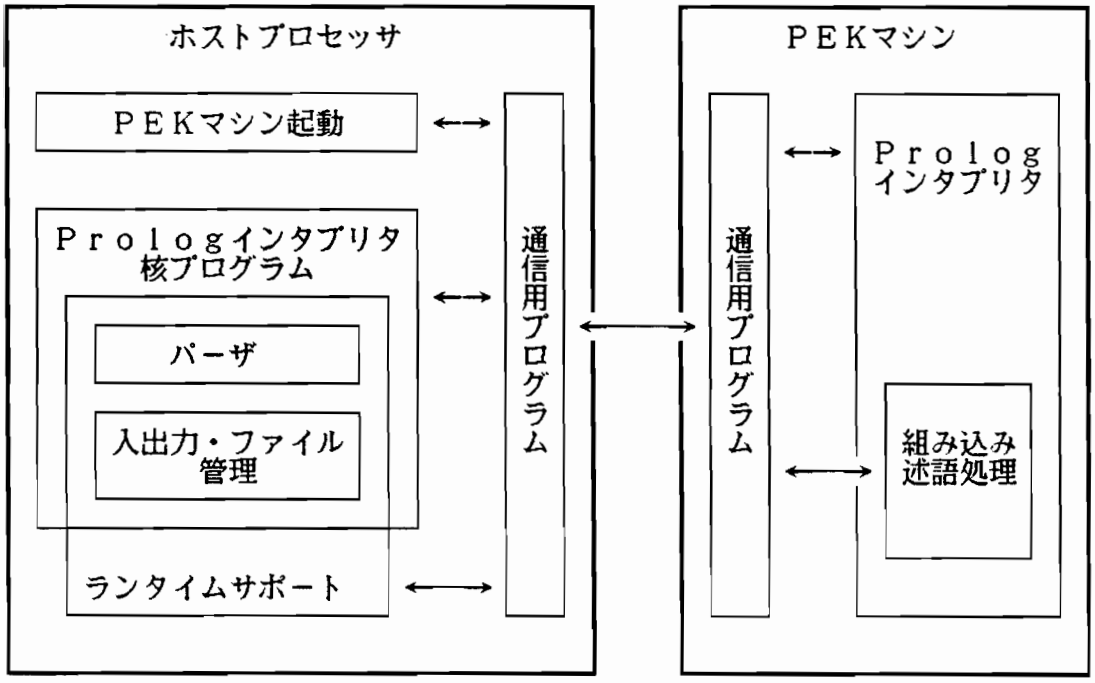
Ⅵ．４．２　Ｐｒｏｌｏｇインタプリタシステムの構成

PrologマシンＰＥＫには、Prologプログラムの効率的な実行をサポートする

ための各種のハードウェアが備えられている。したがって、ソフトウェアでは、これらのハードウェア機能を有効に利用する必要がある。

Ⅵ．４．２．１　ホストプロセッサとの役割分担とメモリ割り当て

まず、ホストプロセッサとＰＥＫマシンの役割分担は以下のようにする<図Ⅵ－４>。



<図Ⅵ－４>　ホストプロセッサとＰＥＫマシンの
役割分担

- ホストプロセッサMC68000 がＷＣＳへのプログラムのロードおよびＰＥＫの起動を行う。ＰＥＫはホストからのコマンド待ちになる。
- ホストプロセッサはキーボードあるいはファイルから読み込んだPrologプログラムを内部形式に変換して、共有メモリに書き込む。また、それがゴール文であれば、ＰＥＫにゴール文の実行を要求し、実行が終了するまで待つ。
- ＰＥＫがゴール文を実行するが、その途中で入出力をともなう組み込み述語などを実行する場合には、ホストプロセッサに要求を出し、要求が実行されるまで待ってからゴール文の実行を再開する。実行の終了あるいはエラーが生じた場合は、ホストにその旨を伝えて、再びコマンド待ちになる。

また、以下のようにメモリを利用した。

- ＷＣＳ

  Prologインタプリタの格納。
- 共有メモリ

  Prologプログラム、アトムヘッダなどの格納。
- プロセスメモリ

  コントロールフレームの格納。
- グローバルスタック

  グローバル変数、ローカル変数の格納。
- トレイルスタック

  アンドゥ作業用のトレイル情報の格納。
- ハードウェアスタック

  ユニフィケーションで使用。

・ レジスタファイル

大域的な情報の保存などに使用。

Ⅵ．４．２．２ プログラムの内部形式

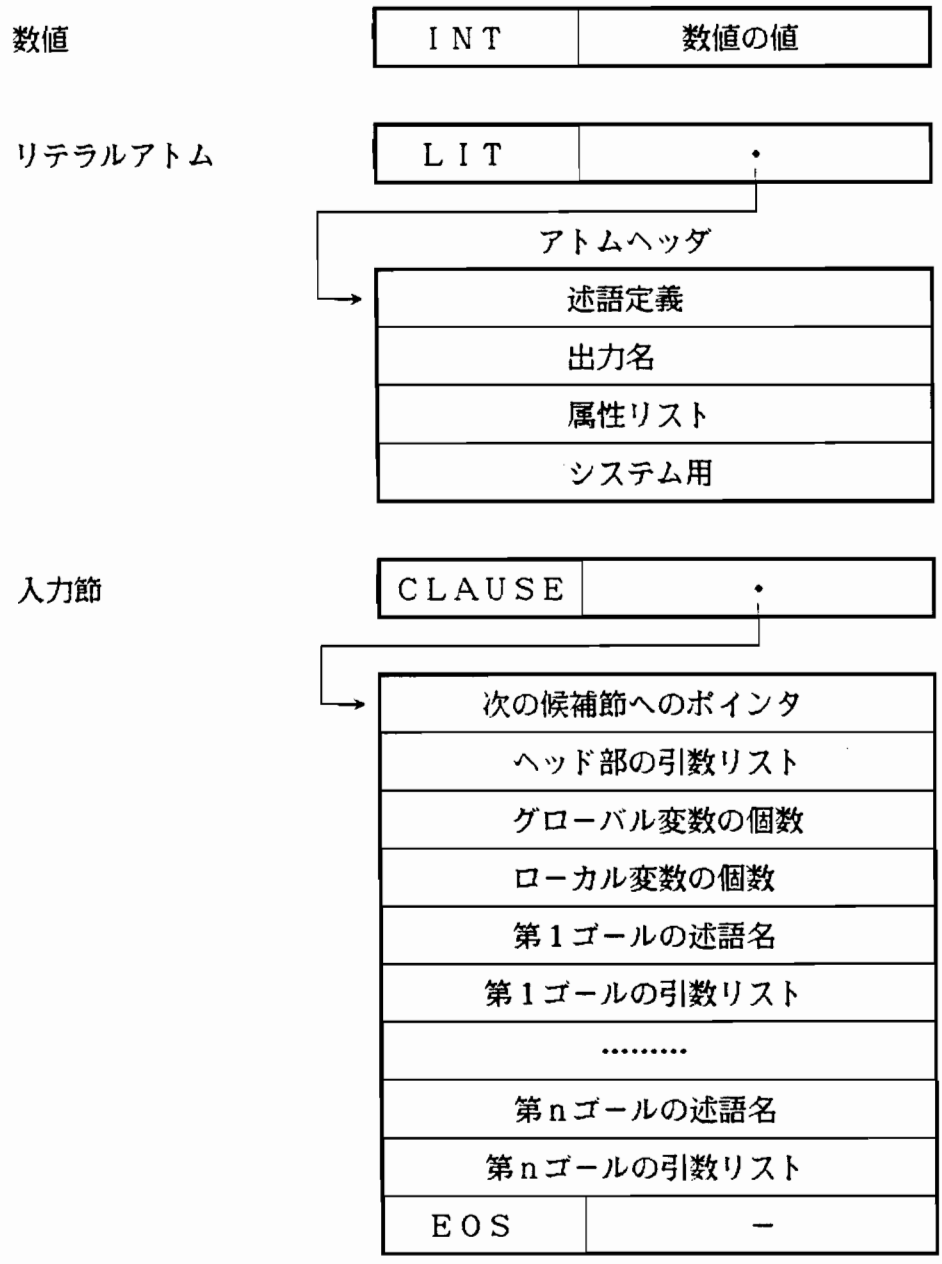
ＰＥＫ側のインタプリタは内部形式に変換されたPrologプログラムを解釈実行する。内部表現形式を＜図Ⅵ－５＞に示す。

数値（-32768～+32767の整数）は、INT タグの付いた一語で表現される。バリュー部はその数値の値そのものである。

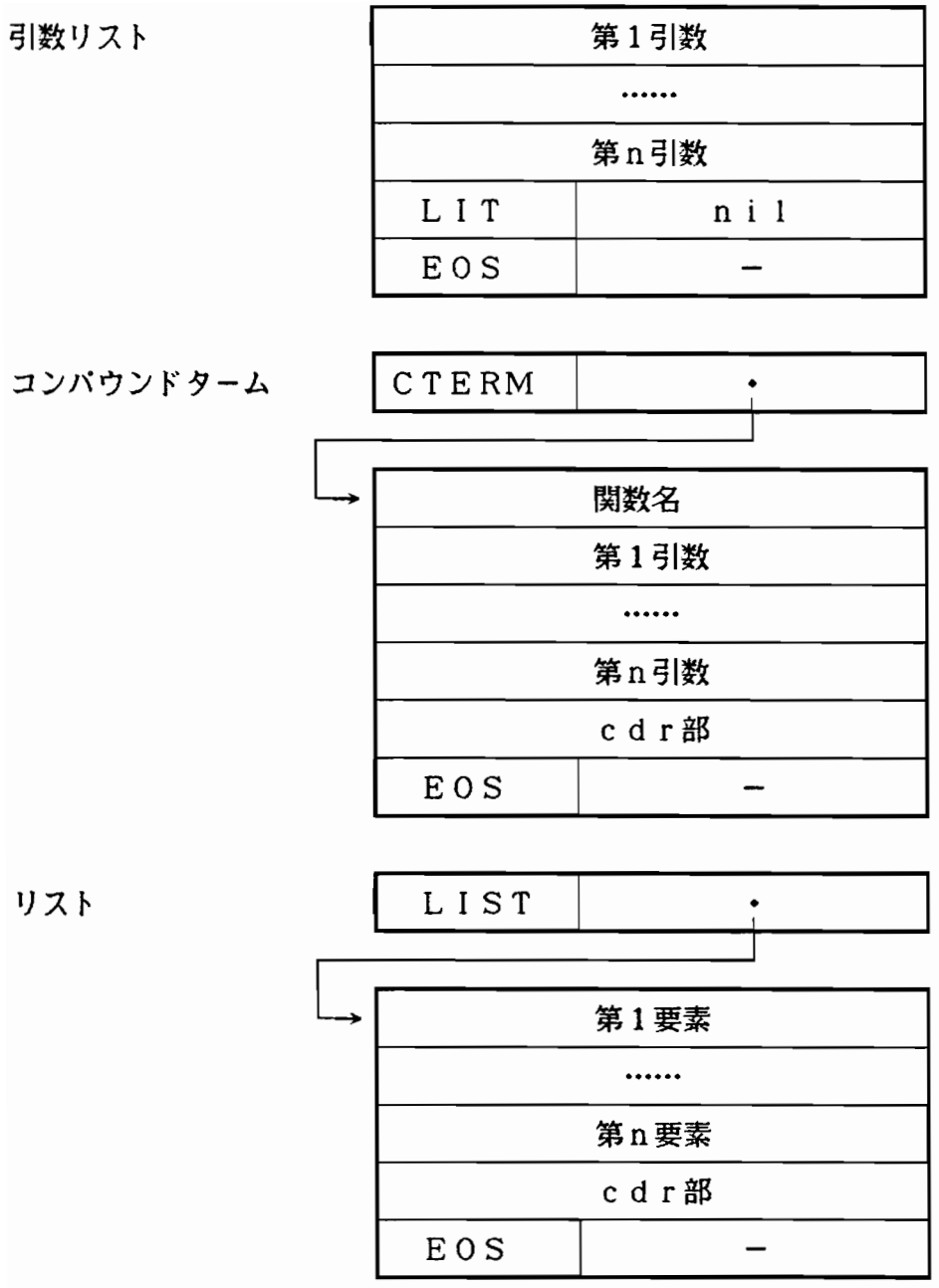
リテラルアトムにはLIT タグが付けられ、バリュー部はそのリテラルアトムのアトムヘッダへのポインタである。アトムヘッダは 4語長で、そのリテラルアトムを述語名とする述語の定義、リテラルアトムの出力名（print name）、属性リスト（property list ）などが格納されている。

述語定義には、組み込み述語とユーザ定義述語の二種類がある。組み込み述語の場合はCODEタグが付き、バリュー部はその組み込み述語を実行するためのマイクロプログラムのアドレスを示す。ユーザ定義述語の場合は、CLAUSEタグが付き、バリュー部は入力節（input clause）へのポインタである。入力節の内部形式には、次の候補節へのポインタ、ヘッド部の引数リスト、グローバル変数の個数、ローカル変数の個数、ボディ部の各ゴールの述語名と引数リストなどが含まれている。

引数リストは、第１引数から第ｎ引数までを連続アドレスに並べたもので、コンパウンドタームやリストと同一の構造である。各引数は、数値、リテラルアトム、グローバル変数、ローカル変数、ボイド変数、コンパウンドターム、リストのいずれかである。


数値
INT
数値の値
リテラルアトム
LIT
アトムヘッダ
述語定義
出力名
属性リスト
システム用
入力節
CLAUSE
次の候補節へのポインタ
ヘッド部の引数リスト
グローバル変数の個数
ローカル変数の個数
第1ゴールの述語名
第1ゴールの引数リスト
.....
第nゴールの述語名
第nゴールの引数リスト
EOS
－

＜図Ⅵ－５＞　内部表現形式



<図Ⅵ−5>　内部表現形式　（続き）

グローバル変数、ローカル変数、ボイド変数にはそれぞれ、GVAR，LVAR，VVARのタグが付き、グローバル変数とローカル変数のバリュー部はそれぞれの変数の番号を示すためのインデックス値（ 0～ 255）である。

コンパウンドタームはCTERM タグが付き、バリュー部は関数名（functor ）と引数を格納してある領域の先頭アドレスを示す。

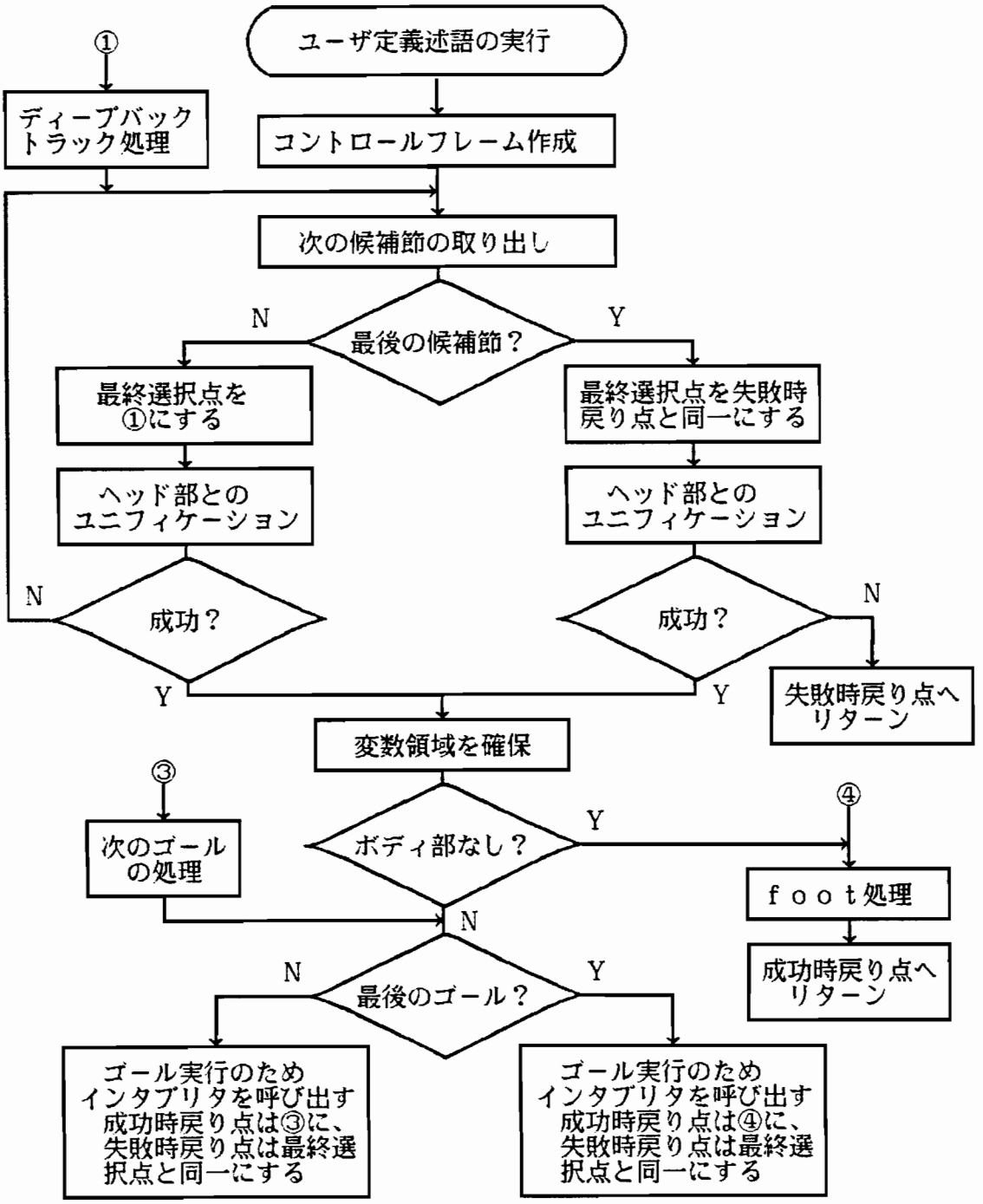
リストにはLISTタグが付き、バリュー部はリストの要素を格納してある領域の先頭アドレスである。

コンパウンドタームとリストの要素は、連続アドレスに格納され、最後の要素の次にはEOS タグを持った要素が付け加えられる。また、最後の要素は全体のcdr 部である。すなわち、a，b，c，EOSなる要素が格納されている時、このリストは［a，b｜c］を意味し、［a｜［b｜c］］と同じ構造を表す。

これらの内部形式への変換は、ホストプロセッサ上のパーザ（parser）が行い、変換結果は共有メモリに格納される。すなわち、ゴール文以外の入力節に関しては、ホストプロセッサだけで処理が行われる。

Ⅵ．４．２．３　インタプリタの処理手順

ＰＥＫ上のインタプリタは各述語の実行ごとに再帰的に呼び出され、必要ならば実行の管理情報を保存するためのコントロールフレームを作成する。インタプリタの処理の流れのうちユーザ定義述語の処理手順を＜図Ⅵ－６＞に示す。



＜図Ⅵ－６＞　ユーザ定義述語の処理手順

また、インタプリタの呼び出し手順は、コンパイラ導入などの拡張を考慮に入れて、マイクロプログラムレベルでの呼び出し手順を設計した。すなわち、インタプリタの呼び出しには以下のようなパラメータを使用する。

- 成功時戻り点 (success return point)
  述語の実行が成功した場合に制御を戻すマイクロアドレスとコントロールフレームのベースアドレスを示す。
- 失敗時戻り点 (failure return point)
  述語の実行が失敗した場合に制御を戻すマイクロアドレスとコントロールフレームのベースアドレスを示す。
- 呼び出し側グローバルフレーム (caller global frame)
  呼び出し側のグローバル変数のフレームアドレスを示す。
- 呼び出し側ローカルフレーム (caller local frame)
  呼び出し側のローカル変数のフレームアドレスを示す。
- 述語名 (predicate name)
  インタプリタが実行する述語名を示す。
- 引数リスト (argument list)
  実行する述語の引数のリストを示す。

成功時戻り点と失敗時戻り点は、復帰先のマイクロプログラムのアドレスとコントロールフレームの先頭アドレスの２語からなっている。２つの戻り点を用意することにより、むだな戻り作業の連鎖をさけることができ、実行効率の改善に役立つ。また、復帰先の指定が、マイクロプログラムレベルで行えるため、マイクロコードにコンパイルした述語などからインタプリタを呼び出すことが容易になる。

インタプリタは述語実行が成功した場合には、リターン情報として

・ 最終選択点(latest choice point)

をレジスタに設定する。最終選択点は、バックトラックの際に実行を再開する点を示すもので、実行を再開する場所のマイクロアドレスとコントロールフレームアドレスの２語からなっている。したがって、インタプリタは実行している述語の候補(alternative) が残っている場合は、最終選択点のコントロールフレームを自分自身のコントロールフレームと同一にし、候補が残っていない場合は、最終選択点を失敗時戻り点と同一にする。

述語名としては、リテラルアトム（タグはLIT ）か、組み込み述語プログラムの先頭マイクロアドレス（タグはCODE）が可能である。リテラルアトムの場合は、アトムヘッダ内に述語定義へのポインタが書き込まれている。述語定義としては、ユーザ定義述語へのポインタ（タグはCLAUSE）か組み込み述語プログラムの先頭マイクロアドレス（タグはCODE）が可能である。

コントロールフレームには、以上のパラメータのほかに

・ 実行中のclauseへのポインタ
・ alternative のclauseへのポインタ
・ 使用しているグローバルフレームのアドレス
・ 使用しているローカルフレームのアドレス
・ 述語の実行開始時のトレイルスタックポインタ
・ 実行中のclauseのグローバル変数の個数
・ 実行中のclauseのローカル変数の個数

が格納されている（計15語）。

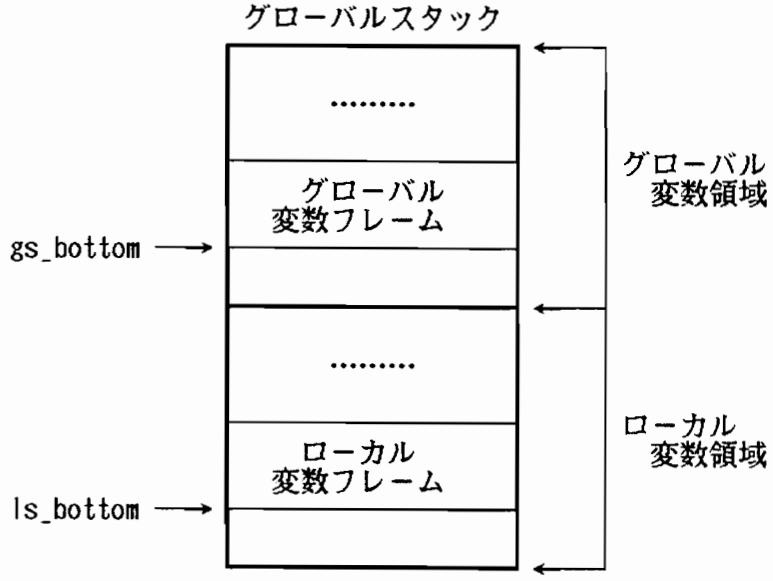
コントロールフレームの格納にはプロセスメモリを使用する。コントロールフレームの先頭アドレスを、プロセスメモリのベースレジスタＰＢＲに書き込んでおけば、コントロールフレーム内のデータは１マイクロ命令でアクセスで

きる。述語実行の成功あるいは失敗時のリターンは、ベースレジスタの切り替えを含めて、２マイクロ命令で実行できる。

Ⅵ．４．２．４　変数フレームの管理

変数はDEC-10 Prolog と同様に、グローバル変数、ローカル変数、ボイド変数の三種に分類して管理している。グローバル変数は、構造体のなかに現れている変数であり、バックトラック時にしか消去できない。ローカル変数は、構造体中に現れていない変数で、最終候補節で成功した時に消去できる。ボイド変数は、出現が一度だけのローカル変数であり、変数セルを必要としない。

グローバル変数領域とローカル変数領域の格納には、グローバルスタックを二分して使用する<図Ⅵ－７>。グローバル変数フレームとローカル変数フレームは入力節のヘッドとのユニフィケーションが成功した時点で確保される。



<図Ⅵ－７>　変数領域の割り当て

変数セルのアクセスは、変数フレームの先頭アドレスと変数の組をレジスタＦＴ１またはＦＴ２に書き込んでおけば、１マイクロ命令でアクセスできる。また、変数の値が未定義かどうかは、UNDEF1またはUNDEF2のフラグにより示される。

Ⅵ．４．２．５　ユニフィケーションの手順

ＰＥＫには、構造体データのパイプラインリード、マッチング回路、自動トレイル回路などのユニフィケーション用のハードウェアが備わっており、ユニフィケーションを高速に実行できる。

二つの構造体のユニフィケーションは以下のようにして行う。

１．一つ目の構造体のアドレスセット．．．．．．．．．．．．．．　２命令

一つ目の構造体の先頭アドレスを、共有メモリのアドレスレジスタＡＤ１に書き込む。このとき、フレーム部はグローバル変数フレームの先頭アドレスに一致させておき、フレームレジスタの設定を同時に行う。リード待ちのための１命令が必要である。

２．二つ目の構造体のアドレスセット．．．．．．．．．．．．．．　２命令

二つ目の構造体の先頭アドレスを、共有メモリのアドレスレジスタＡＤ２に書き込む。このとき、フレーム部はグローバル変数フレームの先頭アドレスに一致させておき、フレームレジスタの設定を同時に行う。リード待ちのための１命令が必要である。

３．一つ目の構造体の第一要素のリード．．．．．．．．．．．．．　２命令

共有メモリのデータレジスタＲＤ１から各構造体の最初のデータを読み込み、その値をレジスタＦＴ１に書き込む。フレームレジスタの働きにより、フレーム部の値はＡＤ１側の構造体のグローバル変数フレームアドレスになっている。また、リード待ちを１命令行う。

4．二つ目の構造体の第一要素のリード．．．．．．．．．．．．．　２命令

共有メモリのデータレジスタＲＤ２から各構造体の最初のデータを読み込み、その値をレジスタＦＴ２に書き込む。フレームレジスタの働きにより、フレーム部の値はＡＤ２側の構造体のグローバル変数フレームアドレスになっている。また、マイクロ命令中のＭＵＸフィールドの値を０にすることにより、ＦＴ１で指される変数セルの値が未定義かどうかのフラグ（UNDEF1フラグ）をセットする。リード待ちを１命令行う。

5．マッチング．．．．．．．．．．．．．．．．．．．．．．．．　１命令

ＦＴ１とＦＴ２に格納されたデータのタグとバリューの値にしたがって、多方向分岐を行う。また、マイクロ命令中のＭＵＸフィールドの値を１にすることにより、ＦＴ２で指される変数セルの値が未定義かどうかのフラグ（UNDEF2フラグ）をセットする。

このあと、２つのデータのタグの種類とバリュー部の値の一致・不一致により、16通りに分岐することが可能であるが、実際に使用しているのは以下の14種類である。

- エラー　　ありえないタグ
- マッチ　　同一のリテラルアトムまたは数値
- cterm-cterm　　ともにコンパウンドターム
- list-list　　ともにリスト
- vvar　　一方がボイド変数
- -gvar　　ＦＴ２側がグローバル変数で
ＦＴ１側は変数以外
- gvar-　　ＦＴ１側がグローバル変数で
ＦＴ２側は変数以外

- -lvar　　ＦＴ２側がローカル変数で
  ＦＴ１側は変数以外
- lvar-　　ＦＴ１側がローカル変数で
  ＦＴ２側は変数以外
- gvar-gvar　　ＦＴ１，ＦＴ２ともにグローバル変数
- lvar-lvar　　ＦＴ１，ＦＴ２ともにローカル変数
- lvar-gvar　　ＦＴ１側がローカル変数で
  ＦＴ２側はグローバル変数
- gvar-lvar　　ＦＴ１側がグローバル変数で
  ＦＴ２側はローカル変数
- ノットマッチ　どちらも変数でなくタグも一致しない
  あるいは同一でないリテラルアトムか数値

グローバル変数またはローカル変数の場合は、値が未定義かどうかを調べ、未定義でなければ、その値を取り出すディリファレンス作業が必要になる。この作業はＰＥＫでは以下のようにして行う（簡単のため、ＦＴ１側がグローバル変数の場合について説明する）。

６．値が未定義かどうかチェック．．．．．．．．．．．．．．．．　１命令

UNDEF1フラグはすでにセットされているので、そのフラグによる条件分岐命令を実行する。

７．未定義でなければディリファレンスを行う．．．．．．．．．．　１命令

マイクロ命令中のＭＵＸフィールドを０にして、グローバルスタックのアドレスソースをＦＴ１側に切り替え、グローバルスタックから値を読みだす。読みだした値はそのままＦＴ１へ書き込む。

８．再度マッチングを行う．．．．．．．．．．．．．．．．．．．．　１命令

ＦＴ１とＦＴ２に格納されたデータのタグとバリューの値にしたがって、再度多方向分岐を行う。また、マイクロ命令中のＭＵＸフィールドの値を０にすることにより、ＦＴ１で指される変数セルの値が未定義かどうかのフラグ（UNDEF1フラグ）をセットする。

変数の値が未定義であり、もう一方のデータが変数でない場合には、変数にそのデータを代入する。このとき、バックトラックにそなえてトレイル作業（代入のあった変数のアドレスをトレイルスタックにプッシュして覚えておく）が必要になる。ＰＥＫでは、このトレイル作業はマイクロ命令中の１ビットをセットするだけで自動的に行える（下の例は、ＦＴ１がグローバル変数の場合である）。

９．変数に代入する．．．．．．．．．．．．．．．．．．．．．．．　１命令

マイクロ命令中のＭＵＸフィールドを０にして、グローバルスタックのアドレスソースをＦＴ１側に切り替え、ＦＴ２に格納したデータと同一の値をグローバルスタックに書き込む。このとき、マイクロ命令中のＴＳＣフィールドの値を１にして、トレイル作業を自動的に行う。

このように、ＰＥＫではハードウェアの機能を有効に利用すれば、ユニフィケーション処理の効率的な実行が可能である。

Ⅵ．５　結言

本章では、逐次実行型Prologマシンシステムのソフトウェア構成について述べた。

モニタ／デバッガシステムはハードウェアの開発・保守・デバッグを支援するためのシステムであり、マイクロプログラムのステップ実行、ブレークポイントの設定、レジスタの内容表示・書き換えなどを行える。

マイクロアセンブラシステムは、パターンマッチによるマクロ展開機能を有しており、高級言語風の記述も可能になっている。Prologインタプリタシステムは、ホストプロセッサ上のプログラムとＰＥＫマシン上のプログラムから成っている。インタプリタはＰＥＫマシンの性能を充分発揮できるように設計されている。

第７章　逐次実行型Ｐｒｏｌｏｇマシンシステムの評価

Ⅶ．１　緒言

本章では逐次実行型Prologマシンシステムの評価を行う。二種類のテストプログラムを実行し、実行命令数からＰＥＫマシン上のインタプリタの性能を求める。また、速度向上における、各ハードウェアの貢献度を調べるために、標準的なLISPマシンと同様のアーキテクチャを持つ仮想マシンを設定し、その上のPrologインタプリタの処理と比較を行う。

Ⅶ．２　評価プログラムと実行結果

作成したシステムの性能を評価する目的で、二種類のPrologプログラムについて、実行した命令をトレースし、実行速度を求めた。

インタプリタは第１版を使用した。このインタプリタでは、変数はすべてグローバル変数とし、インデクシング[Warren 77a]、テイルリカージョンの最適化[Warren 80] は行っていない。また、ハードウェア機能のうち、共有メモリの２ポート化による高速リード機能と、Ｓバスを使用しないイミディエイトジャンプの機能は用いていない。

テストプログラムとしては、長さｎのリストを空リストに連結するappendプログラムと、長さｎのリストを逆転させるreverse プログラムの２つを実行した。

・　appendプログラム

```
append([],Z,Z).
append([W|X],Y,[W|Z]) :- append(X,Y,Z).
?- append([1,2,3,..,n],[],Z).
```

・ reverse プログラム

```
reverse([P|Q],R) :- !, reverse(Q,S), append(S,[P],R).
reverse([],[]).
append([W|X],Y,[W|Z]) :- !, append(X,Y,Z).
append([],Z,Z).
?- reverse([1,2,3,..,n],R).
```

＜表Ⅶ－１＞にそれぞれのゴール文を実行した時の、実行命令数を示す。また＜表Ⅶ－２＞には、特にｎ＝30の場合の実行命令数と、実行速度を示す（ただし、１マイクロ命令の実行速度を平均 170ナノ秒としている）。

＜表Ⅶ－１＞ append, reverse での実行命令数

	推論の回数	実行した命令の総数	コントロール関係の命令数	ユニフィケーション関係の命令数
append([1,2,...,n],[],Z)	n + 1	145n + 149	49n + 94	96n + 55
reverse([1,2,...,n],R)	$\frac{1}{2}n^2+\frac{3}{2}n+1$	$\frac{145}{2}n^2+\frac{357}{2}n+178$	$33n^2+85n+101$	$\frac{79}{2}n^2+\frac{187}{2}n+77$

＜表Ⅶ－２＞ append, reverse での実行命令数（n=30、一命令＝170nsec.）

	推論の回数	実行した命令の総数	コントロール関係の命令数	ユニフィケーション関係の命令数	一秒当たりの推論の回数
append([1,2,..,30],[],Z)	31	4499	1564	2935	40.5K LIPS
reverse([1,2,..,30],R)	496	70783	32351	38432	41.2K LIPS

また、appendプログラムとreverse プログラムの処理単位ごとの命令数を＜表Ⅶ－３＞と＜表Ⅶ－４＞に示す。ここで、それぞれの処理単位名は

init ：goal実行前の初期化

pred ：述語のタイプ判断

ent ：enter 処理

try ：try 処理

unif ：unification 処理

tryl ：try last処理

neck ：neck処理

cut ：cut の呼び出しと実行

call ：述語の呼び出し

calll ：再右端の述語の呼び出し

foot ：foot処理

ret ：goal実行の終了

を意味する。

＜表Ⅶ－１＞と＜表Ⅶ－２＞から、どちらのプログラムについても、一回の推論当たり 145マイクロ命令で実行でき、速度も約40K LIPS(Logical Inference Per Second)であることが分かる。これは、現在実用的な処理系としては最も速いとされているDEC 2060上のDEC-10 Prolog コンパイラとほぼ同等の速度である。

これは、ＰＥＫと同様の回路技術を用いているLISPマシン上のPrologインタプリタが、10～20K LIPSであることを考えると[Takeuchi 84][Otera 84][Hattori 84]、ＰＥＫのアーキテクチャがPrologプログラムの実行速度を 2～ 4倍程度改善したと言える。

＜表Ⅶ－３＞　appendプログラムの実行命令数の内訳

	init	pred	ent	try	unif	tryl	unif	neck	calll	foot	ret
g(n)	12	5	14			5	13	8	8+a(n)	4	1
a(0)		5	14	5	44			8		5	
a(n)		5	14	5	21	5	65	8	8+a'(n-1)	4	
a'(0)		5	14	5	52			8		5	
a'(n)		5	14	5	24	5	72	8	8+a'(n-1)	4	

g(n)　：goalを実行した時の総命令数
a(n)　：append([1,2,...,n],[],Z)を実行した時の総命令数
a'(n)：append(X,Y,Z) を実行した時の総命令数

＜表Ⅶ－４＞　reverse プログラムの実行命令数の内訳

	init	pred	ent	try	unif	tryl	unif	neck	cut	call	calll	foot	ret
g(n)	12	5	14			5	13	8			8+r(n)	4	1
r(0)		5	14	5	21	5	36	8				7	
r(n)		5	14	5	58			8	7+12	8+r'(n-1)	9+a(n-1)	6	
r'(0)		5	14	5	24	5	39	8				7	
r'(n)		5	14	5	61			8	7+12	8+r'(n-1)	9+a(n-1)	6	
a(0)		5	14	5	24	5	52	8				7	
a(1)		5	14	5	74			8	7+12		9+a'(0)	6	
a(n)		5	14	5	78			8	7+12		9+a'(n-1)	6	
a'(0)		5	14	5	24	5	53	8				7	
a'(1)		5	14	5	75			8	7+12		9+a'(0)	6	
a'(n)		5	14	5	79			8	7+12		9+a'(n-1)	6	

g(n)　：goalを実行した時の総命令数
r(n)　：reverse([1,2,...,n],Z)を実行した時の総命令数
r'(n)：reverse(Q,S)を実行した時の総命令数
a(n)　：append(S,[P],R) を実行した時の総命令数
a'(n)：append(X,Y,Z) を実行した時の総命令数

Ⅶ．３　システムの評価

Ⅶ．３．１　各ハードウェアモジュールの評価

Prologプログラムの実行における、各ハードウェアの貢献度を調べるために、ＰＥＫからProlog用のハードウェア機能を取りのぞいた仮想マシン（以下ではＶＭと呼ぶ）を設定し、その上でのappendプログラムの実行過程とＰＥＫでの実行過程の比較を行った。

ＶＭのアーキテクチャとしては、

- 垂直型マイクロ命令。マイクロ命令には分岐命令と演算命令があり、演算命令はＰＥＫと同様の２レジスタ演算である。
- データ幅はタグ部 4ビット、バリュー部16ビットの計20ビットであり、フレーム部は存在しない。
- タグによるディスパッチ機構。１つのタグによるディスパッチは多方向分岐命令を含めて２命令で行えるが、２つのタグによるディスパッチ機構は備えていない。
- 専用のハードウェアスタックを１つ備えている。スタックポインタ設定後、スタックトップのリード／ライト、プッシュ／ポップが可能である。
- 主メモリに対しては、アドレスレジスタ設定後、リード／ライトが可能になる。ただしリード／ライトは２サイクル必要とする。

を想定した。さらにインタプリタの構造に関しては、

- ＰＥＫと同一のデータ構造をとる。
- 全体的な処理の流れは、ＰＥＫのインタプリタと同じにするが、局所的な部分は最適化する。
- ＰＥＫのプロセスメモリとハードウェアスタックをＶＭのハードウェアス

タックに割り当て、他のスタック類は主メモリを利用する。

などを仮定した。

上記のような仮定のもとで、appendプログラムのループの部分（<表Ⅶ－３>でのa'(n)）をＶＭ上でトレースしたところ 300命令となった（ＰＥＫでは 145命令）。ＰＥＫとＶＭの比較結果を<表Ⅶ－５>に示す。ただし、ＶＭ側のプログラムでは、無条件分岐などのむだな命令は除去し、主メモリのリード／ライトは２命令として数えている。

ＶＭの１命令を 170ナノ秒とすると、約19.6K LIPSであり、LISPマシンでの結果に比較的良く一致する。

<表Ⅶ－５>　ＰＥＫとＶＭの命令数の比較

処理名	ＰＥＫ	ＶＭ	比率
pred	５	６	1.20
ent	１４	１５	1.07
try	５	８	1.60
unif	２４	４１	1.71
tryl	５	９	1.80
unif	７２	１８４	2.56
neck	８	１４	1.75
call	８	１７	2.13
foot	４	６	1.50
合計	１４５	３００	2.07

ＰＥＫの命令数とＶＭの命令数の差は

- 水平型マイクロ命令
- 34ビット転送
- プロセスメモリのアドレス形式
- 構造体データのパイプライン読み出し機能
- フレームレジスタ
- マッチング回路
- グローバルスタックの専用化
- 自動トレイル回路
- 自動アンドゥ回路

などが要因となっている。ＰＥＫ側で減少した命令を、これらの要因別にまとめると、＜表Ⅶ－６＞のようになる。

水平型マイクロ命令の採用により、減少した命令数は24命令で、ほとんどがＡＬＵ演算と条件分岐を同時に行っている命令である。

34ビット転送によるものは10命令であり、その内訳はX=[m,...,n]のディリファレンスで一命令、Z := [W|Z]とX := [m+1...,n] の代入で二命令、[m,...,n]と[W|Z] のpushとpop で四命令、そのほかEOS 処理で三命令となっている。

プロセスメモリに専用のベースレジスタを設け、ベースレジスタ＋オフセット（+0～+255）のアドレス形式を使用可能にしたことにより 6命令減少している。これは、ＶＭ側では連続したpush/popでアクセスできない場合、スタックポインタの再設定が必要になるためである。

共有メモリからの読み出しのパイプライン化により、減少した命令数は55命令となっている。これは、ＶＭ側では常にアドレスのセットが必要なこと（19命令増加）、次のデータがEOS かどうか判断するための先読み処理を行ってい

ること（14命令増加）、EOS フラグの判断（13命令増加）などがＶＭでの増加の原因である。

フレームレジスタの利用による減少は 6命令となっている。これは、ＶＭのプログラムで変数フレームアドレスの退避（caller側とsource側）を三回行っているためである。

マッチング回路による減少は 9命令となっている。ＰＥＫでは、二つのレジスタFT1 とFT2 に格納されている二つのタグの値とバリュー部の比較結果を使用して16通りの多方向分岐を行う。したがって、単一のタグを使用したディスパッチ機構を二回行う場合に比較して一命令減少し、双方がliteral atomなどの場合は、さらに二命令減少する。ただし、ＰＥＫでは一方のタグのみでディ

＜表Ⅶ－６＞　ＰＥＫで減少した命令の内訳

ハードウェア機能	減少命令数	割合	順位
水平型マイクロ命令	２４	15.5	③
34ビット転送	１０	6.5	⑤
プロセスメモリ	６	3.9	⑦
パイプライン	５５	35.4	①
フレームレジスタ	６	3.9	⑦
マッチング回路	９	5.8	⑥
グローバルスタック	３０	19.3	②
自動トレイル回路	１６	10.3	④
自動アンドゥ回路	－１	-0.6	⑨
合計	１５５	100.0	

スパッチする機能を省いたため、その場合には反対側に特別のタグを持つタームを書き込む必要があり、逆に一命令増加する。

グローバルスタックの専用化により、減少した命令数は30命令となっている。そのうち、アドレスの設定が10命令、リード／ライト待ちが14命令、UNDEF フラグによるものが 6命令である。ＰＥＫでは、変数の値がUNDEF かどうかは、実際にグローバルスタックから読み出さなくても、フラグで判断できる。

自動トレイル回路による命令数の減少は16命令となっている。これは、ＰＥＫではトレイル処理はマイクロ命令中の１ビットをセットするだけで良いが、ＶＭでは四回のトレイル作業ごとに、主メモリのアドレスセットとライト、トレイルスタックポインタのインクリメントを行っていることによる。

自動アンドゥ回路では、逆に１命令増加している。これは、実行したプログラムが決定的であるため、アンドゥの必要がないこと、また自動アンドゥ作業の終了を調べるための命令が増加しているためである。

Ⅶ．３．２　システムの総合評価

＜表Ⅶ－５＞のＶＭとＰＥＫの比較結果からみると、ＶＭでのユニフィケーション処理はＰＥＫの約2.3 倍かかっており（ＶＭで41+184の 225命令、ＰＥＫで24+72 の96命令）、ＰＥＫのアーキテクチャがユニフィケーションの高速化に大きく貢献したと言える。

また、＜表Ⅶ－６＞をみると、構造体データのパイプライン読み出し機能や、グローバルタック、プロセスメモリ、自動トレイル回路など、メモリやスタックの分散化・専用化に関する効果が大きい（合計で全体の69%）。また、ＶＭではアドレスレジスタへの書き込みが44命令あるが、ＰＥＫでは共有メモリのアドレスレジスタAD1,AD2 への書き込みが11命令、プロセスメモリのベースレジスタPBR への書き込みが 2命令であり、分散化させたメモリごとに特別のア

ドレス形式を設けたことが性能向上に貢献したと考えれる。

Ⅶ. 4　結言

本章では、逐次実行型Prologマシンシステムの評価について述べた。

二種類のテストプログラムを実行し、実行命令数からインタプリタの性能を測定した。どちらのテストプログラムに対しても、命令数は推論一回当たり 145命令であり、速度も約40K LIPSとなっている。これは、実用的なProlog処理系として最も高速であるDEC-10 Prolog コンパイラとほぼ同等の速度である。ＰＥＫと同様の回路技術を用いているLISPマシン上のインタプリタの性能が、10〜20K LIPSであることを考えると、ＰＥＫのアーキテクチャが 2〜 4倍の性能向上を果たしたと言える。

また、システムの評価を行うために、標準的なLISPマシンと同様のアーキテクチャを持つ仮想マシンを設定し、その上のインタプリタの実行過程との比較を行った。その比較結果によれば、一回の推論がＰＥＫでは 145命令で実行できたが、仮想マシンでは二倍以上の 300命令かかった。これは、LISPマシンでの結果に良く一致している。また、ＰＥＫはユニフィケーション処理の速度を約2.3 倍向上させており、ＰＥＫのユニフィケーション用ハードウェアが有効に働くことが確かめられた。さらに、性能向上に対する各ハードウェア機能の貢献度を調べ、以下の結果を得た。

① 水平型マイクロ命令形式の採用により、24命令減少できた。これは、減少した 155命令のうち、15.5% を占める。特に、終了条件などの判断を行う時に有効に働いている。

② 34ビット幅の転送能力により、10命令（6.5%）減少できた。これは、変数のディリファレンスや代入処理の高速化に有効だった。

③ プロセスメモリの採用による減少命令数は 6命令（3.9%）であり、実行制御の効率化に役立つことが分かった。

④ 構造体データ読み込みのパイプライン化により、55命令（35.4% ）減少した。連続アドレスのデータリードの高速化、および先読みによるフラグ判断機能が、有効であることが明らかになった。

⑤ フレームレジスタによる減少命令数は、 6命令（3.9%）だった。ストラクチャシェアリング方式の場合、34ビット転送機能と合わせて使用することにより、有効に機能することが分かった。

⑥ マッチング回路により、 9命令（5.8%）減少した。割合は低いが、小規模のハードウェアで実現可能なことを考えると、効果は大きい。

⑦ グローバルスタックの専用化による減少命令数は、30命令（19.3% ）であった。グローバルスタックに対する高速のアクセスを可能にし、さらに特別のフラグを設けて変数が未定義かどうかの判断を高速に行える機能が有効であることが分かった。

⑧ 自動トレイル回路により、16命令（10.3% ）減少した。変数への代入ごとに、確実に 4命令づつ減少しており、バックトラックのために必要なトレイル作業を実質的に無くすことができた。

⑨ 自動アンドゥによる減少はなかったが、これは実行したテストプログラムが決定的であるためであり、バックトラックを頻繁に行うプログラムに対しては、非常に強力な機能であると思われる。

以上のうち、構造体の要素のパイプライン読み出し機能や、グローバルタック、プロセスメモリ、自動トレイル回路など、メモリやスタックの分散化・専用化による効果が最も大きく、性能向上の69% を占めることが分かった。

第8章　結論

本論文では、論理型言語および記号処理言語であるPrologのプログラムを高速に実行するためのアーキテクチャの研究を目的として、逐次実行型Prologマシンシステムの設計と製作および評価を行った。

すなわち、本マシンは

- 逐次実行
- マイクロプログラム制御
- インタプリタ方式
- ストラクチャシェアリング方式

などを、基本設計方針とし、

- タグアーキテクチャ
- メモリの分散化と低レベル並列処理
- マッチング回路
- 自動トレイル回路
- 自動アンドゥ回路

など、各種のProlog向きのハードウェア機能を備えている。

さらに、このマシン上にインタプリタ形式のProlog処理系を開発し、ベンチマークプログラムによりシステムの評価を行った。その結果によれば、一回の推論を 145マイクロ命令で実行でき、速度も約40K LIPSとなっている。これは、実用的なProlog処理系として最も高速であるDEC-10 Prolog コンパイラとほぼ同等の速度である。ＰＥＫと同様の回路技術を用いているLISPマシン上のインタプリタの性能が、10～20K LIPSであることを考えると、ＰＥＫのアーキテク

チャが 2～ 4倍の性能を向上させたと言える。

また、システムの評価を行うために、標準的なLISPマシンと同様のアーキテクチャを持つ仮想マシンを設定し、その上のインタプリタの実行過程との比較を行った。その比較結果によれば、一回の推論がＰＥＫでは 145命令で実行できたが、仮想マシンでは二倍以上の 300命令かかった。これは、LISPマシンでの結果に良く一致している。また、ＰＥＫはユニフィケーション処理の速度を約2.3 倍向上させており、ＰＥＫのユニフィケーション用ハードウェアが有効に働くことが確かめられた。さらに、性能向上に対する各ハードウェア機能の貢献度を調べ、以下の結果を得た。

① 水平型マイクロ命令形式の採用により、24命令減少できた。これは、減少した 155命令のうち、15.5% を占める。特に、終了条件などの判断を行う時に有効に働いている。

② 34ビット幅の転送能力により、10命令（6.5%）減少できた。これは、変数のディリファレンスや代入処理の高速化に有効だった。

③ プロセスメモリの採用による減少命令数は 6命令（3.9%）であり、実行制御の効率化に役立つことが分かった。

④ 構造体データ読み込みのパイプライン化により、55命令（35.4% ）減少した。連続アドレスのデータリードの高速化、および先読みによるフラグ判断機能が、有効であることが明らかになった。

⑤ フレームレジスタによる減少命令数は、 6命令（3.9%）だった。ストラクチャシェアリング方式の場合、34ビット転送機能と合わせて使用することにより、有効に機能することが分かった。

⑥ マッチング回路により、 9命令（5.8%）減少した。割合は低いが、小規模のハードウェアで実現可能なことを考えると、効果は大きい。

⑦ グローバルスタックの専用化による減少命令数は、30命令（19.3% ）であった。グローバルスタックに対する高速のアクセスを可能にし、さらに特別のフラグを設けて変数が未定義かどうかの判断を高速に行える機能が有効であることが分かった。

⑧ 自動トレイル回路により、16命令（10.3% ）減少した。変数への代入ごとに、確実に 4命令づつ減少しており、バックトラックのために必要なトレイル作業を実質的に無くすことができた。

⑨ 自動アンドゥによる減少はなかったが、これは実行したテストプログラムが決定的であるためであり、バックトラックを頻繁に行うプログラムに対しては、非常に強力な機能であると思われる。

以上のうち、構造体の要素のパイプライン読み出し機能や、グローバルタック、プロセスメモリ、自動トレイル回路など、メモリやスタックの分散化・専用化による効果が最も大きく、性能向上の69% を占めることが分かった。

したがって、本マシンのアーキテクチャはPrologプログラムの高速化に充分貢献したと考えることができる。

参考文献

[AMD 83] Advanced Micro Devices, inc.
Bipolar Microprocessor Logic and Interface Data Book
1983.

[Bowen 81] D.L.Bowen
DEC system-10 PROLOG USER'S MANUAL
Department of Artificial Intelligence,
Univ. of Edinburgh, 1981.

[Boyer 72] R.S.Boyer and J.Moore
The sharing of structure in theorem-proving programs
Machine Intelligence 7 (eds. B.Meltzer and D.Michie)
Edinburgh University Press, 1972.

[Bruynooghe 82] M.Bruynooghe
The memory management of PROLOG implementations
Logic Programming (eds. K.L.Clark and S.-A.Taernlund),
pp.83-98, Academic Press, New York, 1982.

[Chang 73] C-L.Chang and R.C-T.Lee
Symbolic Logic and Mechanical Theorem Proving
Academic Press, 1973.

[Chikayama 83a] T.Chikayama, M.Yokota, and T.Hattori
Fifth Generation Kernel Language: Version-0
Proc. of the Logic Programming Conference '83,
7-1, Tokyo, 1983.

[Chikayama 83b] T.Chikayama
ESP -- Extended Self-contained Prolog -- as a Preliminary Kernel Language of Fifth Generation Computers
New Generation Computing, vol.1 no.1 pp.11-24, 1983.

[Clark 80] K.L.Clark and F.G.McCabe
PROLOG: A Language for Implemeting Expert Systems
Machine Intelligence 10 (eds. J.Halyes and D.Michie)
Edinburgh University Press, 1980.

[Clark 83] K.L.Clark and S.Gregory
PARLOG: A Parallel Logic Programming Language
Research Report Doc 83/5
Department of Computing, Imperial College 1983.

[Emden 76] M.van Emden and R.Kowalski
The semantics of predicate logic
as a programming language
J.ACM 23-4 (Oct.1976) pp.733-742, 1976.

[Fuchi 82] 渕一博
第５世代コンピューター
数理科学1982年 4月号 pp.5-6, 1982.

[Goto 83] S.Goto
DURAL: an Extended Prolog Language
Lecture Notes in Computer Science
vol.147 pp.73-87, 1983.

[Hasegawa 84] R.Hasegawa and M.Amamiya
Parallel Execution of Logic Programs
based on Dataflow Concept
Proc. of the International Conf. on FGCS 1984
pp.507-516, Tokyo, 1984.

[Hattori 84] 服部彰、丹羽雅司、篠木剛、木村康則、岸本光弘
ＰＲＯＬＯＧインタプリタの構成
情報処理学会第２９回全国大会　７Ｂ－１、１９８４年

[Ito 84] N.Ito and K.Masuda
Parallel Inference Machine Based on
the Data Flow Model
Proc. of International Workshop on High-Level
Computer Architecture 84, 4.31, Los Angeles, 1984.

[Kaneda 84] Y.Kaneda, N.Tamura, K.Wada, and H.Matsuda
Sequential PROLOG Machine PEK Architecture and
Software System
Proc. of International Workshop on High-Level
Computer Architecture, 4.1, Los Angeles, 1984.

[Kimura 82] 木村通男
抗生剤選択支援システムＡＮＴＩＣＩＰＡＴＯＲへの
Ｐｒｏｌｏｇ／ＫＲの使用
Proc. of Prolog Conference 1982, 1982.

[Kitakami 84] H.Kitakami, S.Kunifuji, T.Miyachi, and K.Furukawa
A Methodology for Implementation of a Knowledge
Acquisition System
Proc. of 1984 International Symposium on Logic
Programming, pp.131-142, Atlantic City, 1984.

[Kowalski 74] R.Kowalski
Predicate logic as programming language
Proc. IFIP Cong. 1974, pp.569-574
North-Holland Pub.Co., Amsterdam, 1974.

[Matsumoto 83] Y.Matsumoto et al.
BUP: A Bottom-Up Parser Embedded in Prolog
New Generation Computing vol.1 no.1 pp.145-158, 1983.

[McCarthy 62] J.McCarthy, P.Abrahams, D.Edwards, T.Hart, and M.Levin
LISP 1.5 Programmers Manual
The MIT Press, 1962.

[Mellish 82] C.S.Mellish
An Alternative to Structure Sharing in the
Implementation of a Prolog Interpreter
Logic Programming (eds. K.L.Clark and S.-A.Taernlund)
pp.99-106, Academic Press, New York, 1982.

[Mizoguchi 83] F.Mizoguchi
PROLOG Based Expert System
New Generation Computing vol.1 no.1 pp.99-104, 1983.

[Mizoguchi 84] 溝口文雄、片山佳則、武田正之
論理型言語Ｐｒｏｌｏｇの比較検討
Proc. of the Logic Programming Conference '84
5-4, Tokyo, 1984.

[Moto-oka 84] T.Moto-oka, H.Tanaka, H.Aida, K.Hirata, and T.Maruyama
The Architecture of a Parallel Inference Engine -PIE-
Proc. of the International Conf. on FGCS 1984
pp.479-488, Tokyo, 1984.

[Nilsson 80] N.J.Nilsson
Principles of Artificial Intelligence
Tioga Pub.Co., Palo Alto, 1980.

[Onai 84] 尾内理紀夫、清水肇、益田嘉直、麻生盛敏
逐次型Ｐｒｏｌｏｇプログラムの解析
Proc. of the Logic Programming Conference '84
9-5, Tokyo, 1984.

[Organic 73] E.I.Organic
Computer System Organization
Academic Press, 1973.

[Otera 84] 大寺信之、斎藤年史、清原良三、西開地秀和、安井裕
ＥＶＬＩＳマシン上のＰｒｏｌｏｇインタプリタと
その動特性
情報処理学会記号処理研究会資料　２７－４、１９８４年

[Pereira 78] L.M.Pereira, F.Pereira, and D.H.D.Warren
User's Guide to DEC system-10 PROLOG
Department of Artificial Intelligence
University of Edinburgh, 1978.

[Robinson 65] J.A.Robinson
A machine-oriented logic based on
the resolution principle
J.ACM vol.12 pp.23-44, 1965.

[Sato 83] M.Sato
Qute: A Prolog/Lisp type Language
for Logic Programming
Proc. the Logic Programming Conference '84
6-3, Tokyo, 1983.

[Shapiro 82] E.Shapiro
Concurrent Prolog
ICOT Technical Report, 1982.

[Shimazu 83] 島津秀雄、小長谷明彦、中崎良成、梅村護
ＳｈａｐｅＵｐにおけるコピイ方式と
シェアリング方式の考察
情報処理学会第２６回全国大会　６Ｄ－９、１９８３年

[Taki 84] K.Taki, M.Yokota, A.Yamamoto, H.Nisikawa,
S.Uchida, H.Nakashima, and A.Mitsuishi
Hardware Design and Implementation of the
Personal Sequential Inference Machine (PSI)
Proc. of the International Conf. on FGCS 1984
pp.398-409, Tokyo, 1984.

[Takeuchi 84] 竹内郁雄、日比野靖、奥乃博、大里延康、渡辺和文
ＥＬＩＳ－ＴＡＯの性能評価
情報処理学会第２８回全国大会　３Ｆ－２、１９８４年

[Takeuchi 83] 竹内彰一、古川康一、E.Y.Shapiro
Concurrent Prolog によるオブジェクト指向プログラミング
Proc. of the Logic Programming Conference '83
5-2, Tokyo, 1983.

[Tamura 84a] 田村直之、和田耕一、松田秀雄、金田悠紀夫、前川禎男
ＰＲＯＬＯＧマシンＰＥＫについて
Proc. of the Logic Programming Conference '84
8-2, Tokyo, 1984.

[Tamura 84b] N.Tamura, K.Wada, H.Matsuda, Y.Kaneda, and S.Maekawa
Sequential Prolog Machine PEK
Proc. of the International Conf. on FGCS 1984,
Tokyo, 1984.

[Tick 84] E.Tick and D.H.D.Warren
Towards a Pipelined Prolog Processor
Proc. of 1984 International Symposium on Logic
Programming, Atlantic City, 1984.

[Uchida 83] S.Uchida, M.Yokota, A.Yamamoto, K.Taki, H.Nishikawa
Outline of the Personal Sequential Inference Machine:
PSI
New Generation Computing vol.1 no.1 pp.75-79, 1983.

[Umemura 83] 梅村譲、中崎良成、小長谷明彦、幅田伸一、島津秀雄、
横田実
文字列処理機能を導入したProlog: ShapeUp について
Proc. of the Logic Programming Conference '83
6-4, Tokyo, 1983.

[Warren 77a] D.H.D.Warren
Implementing PROLOG -- Compiling Predicate Logic
Programs Vol.1,2
D.A.I. Research Report No.39,40, Department of
Artificial Intelligence, Univ. of Edinburgh, 1977.

[Warren 77b] D.H.D.Warren, L.M.Pereira, and F.Pereira
PROLOG -- The Language and its Implementation
Compared with LISP
ACM, SIGART/SIGPLAN Symposium, Rochester, 1977.

[Warren 80] D.H.D.Warren
An Improved PROLOG Implementation which Optimises
Tail Recursion
Proc. Logic Programming Workshop, pp.1-11, 1980.

[Yamamoto 82] 山本昌弘、梅村譲、小長谷明彦、横田実
文字列処理とアーキテクチャ
情報処理　第２３巻　第８号　pp.719-729, １９８２年

[Yasui 82] 安井裕
ＬＩＳＰマシン
情報処理 第２３巻 第８号 pp.757-772, １９８２年

[Yokota 84] M.Yokota, A.Yamamoto, K.Taki, H.Nishikawa,
S.Uchida, K.Nakajima, and M.Mitsui
A Microprogrammed Interpreter for the Personal
Sequential Inference Machine
Proc. of the International Conf. on FGCS 1984
pp.410-418, Tokyo, 1984.

付　録

＜付表－１＞　初期のＰｒｏｌｏｇ処理系一覧

作成年	作成場所	使用コンピュータ	使用言語	作成者　その他
1970	マルセイユ大学	IBM 360	Alogol-W	P.Roussel
1972	マルセイユ大学	IBM 360	FORTRAN	Battani, Melloni ③
1975	ブダペスト	ICL-1903/A	CDL	Szeredi et al.
1976	ウォータールー大学	IBM 370/158 (VM/CMS)	Assembler	Roberts ④
1976	マルセイユ大学	Solar Exorciser	Assembler	P.Roussel
1976	ブダペスト	IBM 370	CDL	Szeredi et al.
1977	エジンバラ大学	DEC-10/20	Macro-10, Prolog	D.Warren et al. ⑤⑥ コンパイラ
1978	エジンバラ大学	PDP-11/60 (UNIX)	Assembler	C.Mellish
1978	マルセイユ大学	Exorciser	Candide	Colmerauer et al.
1979	ロサンジェルス	PDP-11	FORTH, SNOBOL	Dwiggins コンパイラ
1979	ロンドン大学	IBM 370	Pascal	McCabe IC-Prolog⑦
1979	エジンバラ大学	PDP-11 (RT-11)	Assembler	
1979	電電公社	DEC 2020	Lisp	後藤滋樹 DURAL⑧
1980	東京大学	M-280H	UTI-Lisp	中島秀之 PROLOG/KR⑨
1980	ロンドン大学	Z80, 8085 (CP/M)	Assembler	
1981	マルセイユ大学	Apple-II	Candide	PROLOG-II⑩

マルセイユ大学（フランス）：
Groupe d'intelligence artificielle, Univ. d'Aix-Marseille
ブダペスト（ハンガリー）：Institute for coordination of computer technique
ウォータールー大学（カナダ）：University of Waterloo
エジンバラ大学（イギリス）：University of Edinburgh
ロンドン大学（イギリス）：
Imperial College of Science and Technology, Univ. of London
電電公社：電信電話公社武蔵野電気通信研究所

<参考資料>

① Coelho,H., J.C.Cotta, and L.M.Pereira
How to Solve It with Prolog, 2nd edition,
Laboratorio Nacional de Engenharia Civil, Lisboa, 1982

② 横井俊夫、渕一博
推論機構を内蔵した述語論理型言語Ｐｒｏｌｏｇ
日経エレクトロニクス、1982年 9月27日号 (no.300), pp.267-286

③ Battani,G. and H.Meloni
Interpreteur du Language de Programmation PROLOG
Groupe d'intelligence artificielle, Univ. D'Aix-Marseille, 1973

④ Roberts,G.M.
An Implementation of Prolog
Master's Thesis, Dept. Computer Science, Univ. of Waterloo, 1977

⑤ Bowen,D.L.
DEC system-10 PROLOG USER'S MANUAL
Dept. of Artificial Intelligence, Univ. of Edinburgh, 1981

⑥ Warren,D.H.D.
Implementing PROLOG -- Compiling Predicate Logic Programs, Vol.1,2
Dept. of Artificial Intelligence, Research Report No.39,40
Univ. of Edinburgh, 1977

⑦ Clark,K. and F.McCabe
IC-Prolog -- Language Features
Proc. Logic Programming Workshop, pp.45-52, 1980

⑧ Goto,S.
DURAL:an Extended Prolog Language
Lecture Notes in Computer Sciences, Vol.147, pp.73-87, 1983

⑨ Nakashima,H.
Prolog/KR User's Manual. METR 82-4
Dept. of Mathematical Engineering and Instrumentation Physics,
Faculty of Engineering, Univ. of Tokyo, 1980

⑩ Colmerauer,A.
PROLOG II, manual de reference at modele theorique
Groupe d'intelligence artificielle,
ERA CNRS 363 Faculte des Science de Luminy, Marseille, 1982

<付表－２>　Ｐｒｏｌｏｇ処理系の速度

No	処理系名	機種名	時間 sec	LIPS	文献
1	DEC-10 Prolog I	DEC 2060	0.191	2600	①
2	DEC-10 Prolog C	DEC 2060	0.011	45100	①
3	DEC-10 Prolog F	DEC 2060	0.0095	52200	①
4	横浜国立大学	M-280H	0.0058	85500	③
5	ShapeUp	ACOS1000	0.018	27600	②
6	LONLI	M200H	0.07	7090	②
7	Prolog/KR	M200H	0.24	2070	②
8	漢字Prolog　I	MELCOM-COSM0900 II	0.168	2950	④
9	漢字Prolog　C	MELCOM-COSM0900 II	0.045	11000	④
10	C-Prolog	VAX 11/780	0.321	1550	①
11	MProlog	VAX 11/750	0.9	551	②
12	K-Prolog	UX-300	1.068	464	①
13	K-Prolog	U-STATION (68000)	0.93	533	②
14	H-Prolog	UT98　(68000)	2.81	177	①
15	Prolog-KABA	PC9801F (8086)	0.722	687	①
16	Prolog-KABA	PC9801　(8086)	1.58	314	②
17	Prolog-1	IF-800 (CP/M)	16.65	30	①
18	Prolog-1	AS-100 (CP/M-86)	17.3	29	①
19	micro-Prolog	(CP/M-86)	8.334	60	①
20	LM-Prolog　I	CADR	1.17	424	②
21	LM-Prolog　C	CADR	0.13	3815	②
22	電電公社	ELIS	0.0439	11300	⑤
23	富士通	ALPHA	0.0359	13800	⑥

＜注釈＞

結果はすべて、長さ30のリストのリバース（496 LI）
No.2，3，4，9，21 はコンパイラ、No.3はfast code オプション時
その他はインタプリタ
No.12 ～No.19 はパーソナルコンピュータ
No.20 ～No.23 はＬｉｓｐマシン

＜参考資料＞

① 溝口文雄、片山佳則、武田正之
論理型言語Ｐｒｏｌｏｇの比較検討
Proc. of the Logic Programming Conference '84, 5-4
Tokyo, March 1984

② 稲葉則夫
汎用機からパソコン用まで出そろった
商用Ｐｒｏｌｏｇ処理系を比較する
日経エレクトロニクス、１９８４年１１月５日号（ｎｏ．３５５）

③ 小沢年弘、岸川徳幸、中川裕志
高速ＰＲＯＬＯＧ処理系の試作　（その２）
情報処理学会第２８回全国大会、２Ｇ－６、１９８４年

④ 太細孝、向井国昭、鈴木克志、伊草ひとみ
ＭＥＬＣＯＭ－ＰＲＯＬＯＧコンパイラ
情報処理学会第２６回全国大会　６Ｄ－５、１９８３年

⑤ 竹内郁雄、日比野靖、奥乃博、大里延康、渡辺和文
ＥＬＩＳ－ＴＡＯの性能評価
情報処理学会第２８回全国大会、３Ｆ－２、１９８４年

⑥ 服部彰、丹羽雅司、篠木剛、木村康則、岸本光弘
ＰＲＯＬＯＧインタプリタの構成
情報処理学会第２９回全国大会　７Ｂ－１、１９８４年

謝　辞

本研究を行うにあたって、理解ある御指導を賜った神戸大学大学院自然科学研究科システム科学専攻　前川禎男教授（神戸大学工学部システム工学科）に心からの感謝の意を表する。また、熱心な御指導、有益な御助言を賜ったシステム科学専攻　金田悠紀夫助教授（システム工学科）に深く感謝する。本研究をまとめるにあたって、御指導、御助言を賜ったシステム科学専攻　村上温夫教授（応用数学）、松本治彌教授（計測工学科）に厚く御礼申し上げる。また、暖かい御指導を賜ったシステム科学専攻　平井一正教授（システム工学科）、システム科学専攻　田中初一助教授（電気工学科）に深く感謝する。

本研究において、有益な御助言、多大なる御協力をいただいたシステム科学専攻　和田耕一助手に心から感謝する。また、本研究において御協力をいただいた松下電器産業株式会社　野田克彦博士、日根俊治博士、仲辻俊之氏に深く感謝する。さらに、本研究をすすめるにあたって多大なる御協力をいただいた神戸大学大学院自然科学研究科システム科学専攻　小畑正貴氏、松田秀雄氏、神戸大学大学院工学研究科システム工学専攻　小林久和氏、綾部雅之氏、元神戸大学工学部システム工学科　久保康治氏、神戸大学工学部システム工学科　宮本昌也氏に深く感謝する。また、種々の面でお世話になったシステム工学科第４講座の小畑美保子技官、システム工学科の藤井勝宏技官、ならびに第４講座諸氏に深く感謝する。