ユニバーサル・ホスト計算機 QA-2 による逐次型 Prolog マシンのエミュレーション†

柴山 潔†† 富田眞治†† 萩原 宏††

高速性とシステムの柔軟性を兼備したユニバーサル・ホスト計算機 QA-2 上で，数種類の逐次型 Prolog マシンのエミュレーションを行い，Prolog の高速処理方式についての考察と QA-2 の問題適応能力についての評価を行った．エミュレーションの対象としたマクロ・アーキテクチャの機能レベルは，2種類である．各マクロ・アーキテクチャに対する処理構造として，インタプリト方式とコンパイル方式を適用し，Prolog の処理における ALU 演算機能レベル（低レベル）の並列性について調べた．また，コンパイル方式による処理技法の一つとして，マイクロプログラム・コンパイル方式の効果についても評価した．Prolog プログラムを QA-2 のマイクロプログラムへ翻訳し実行する本方式により，QA-2 上で約 45 K LIPS の性能を発揮する Prolog マシンを実現している．さらに本論文では，対象データに明示的な並列性を持たない応用である逐次型 Prolog マシンのエミュレーションを通じて，QA-2 のユニバーサル・ホスト計算機としての問題適応能力および，低レベル並列処理方式と高機能順序制御方式が有効に機能することを確かめている．

1. まえがき

我々が開発したマイクロプログラム制御計算機 QA-2[2] は，算術論理演算装置（ALU)-レジスタ・レベル（低レベル）での並列処理機能を有するユニバーサル・ホスト計算機である．QA-2 の適用分野は，図形処理や信号処理などのリアルタイム処理，および専用計算機や高級言語計算機のエミュレーションである．QA-2 は計算機アーキテクチャについての研究環境の中心となるホスト計算機として位置付けられている．

現在までに，QA-2 を3次元グラフィックスなどへの応用に適用して，そのアーキテクチャの有効性を確かめている．これらの評価の過程では，各応用における特徴的な処理方式や応用向き計算機アーキテクチャについても考察を加えている．QA-2 の特徴は，アーキテクチャの並列処理機能が，3次元グラフィックスのような処理対象に明示的な並列性を有する問題だけでなく，例えば逐次型高級言語計算機のエミュレーションなどのように明示的な並列性が無い問題に対しても十分適応可能な点にある．

本論文では，本質的な並列性を持たない応用として，逐次型 Prolog マシンのエミュレーション[1]を選び，Prolog の高速処理方式とともに，それらの応用に対する QA-2 の問題適応能力について考察を加える．本研究の目的は，逐次型 Prolog の処理過程において，明示的ではない低レベルの並列処理可能性を明らかにし，それを逐次型 Prolog マシン・アーキテクチャの設計の場で活用することである．

2. QA-2 アーキテクチャの概要

QA-2 アーキテクチャの特徴を次に掲げる．

（1） 処理機能を分散化したシステム構成方式．図1に示したように，QA-2 はレジスタ・ALU 部 (RALU) と順序制御部 (SCU) から成る CPU と，主記憶管理プロセッサ (MMP)，システム管理プロセッサ (SVP) の三つの機能部分を独立に構成し，これらを相異なるクロックと水平型マイクロ命令（1ワード＝256 ビット）の相異なるフィールドによって独立して制御する機能分散方式によって構成されている．

（2） 同一で高機能な4個の ALU による低レベル並列処理方式．QA-2 では，図1に示すように，4個の可変長 ALU がマイクロ命令の相異なるフィールドで独立に制御され，それらが均一構造を持ち大容量のレジスタ・ファイル (6Kバイト) を共有しながら動作する．4個の ALU は互いに独立な四つのオペランド群に対して並列演算を実行できるだけでなく，1個の演算結果を他の ALU 演算の入力オペランドとして使用する ALU 連鎖演算を行うことも可能である．

（3） マイクロプログラムの生産性の向上を指向した順序制御構造．大規模なユーザ・マイクロプログラミングを実現するためには，ユーザが記述する論理レベルのマイクロプログラム構造と物理的なハードウェア構造との対応関係を容易にとり得ることが要求され

† Emulation of Sequential Prolog Machines on a Universal Host Computer QA-2 by KIYOSHI SHIBAYAMA, SHINJI TOMITA and HIROSHI HAGIWARA (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学教室

る．特に，高度な制御構造は構造化マイクロプログラミングを実行する場合に必須であり，“IF-THEN-ELSE” 文や “CASE-OF” 文などの強力な条件判定分岐機能はハードウェアとして実装されていることが望ましい．QA-2 には，低レベル並列処理機能を十分に生かせるように，これらの高度の順序制御機構が装備されている．

3. Prolog の処理方式

3.1 エミュレーションの方式

エミュレーションの対象としたマクロ・アーキテクチャは，その機能レベルに従って**表1**に示したような2種類に大別される．マクロ・アーキテクチャは，ISA (Instruction Set Architecture) とも呼ばれる．いずれも DEC-10 PROLOG 仕様の言語の処理を対象としたアーキテクチャであり，Prolog プログラムをこれらのマクロ命令列に変換した後，それを QA-2 のファームウェアにより実行する．

いずれのマクロ・アーキテクチャ

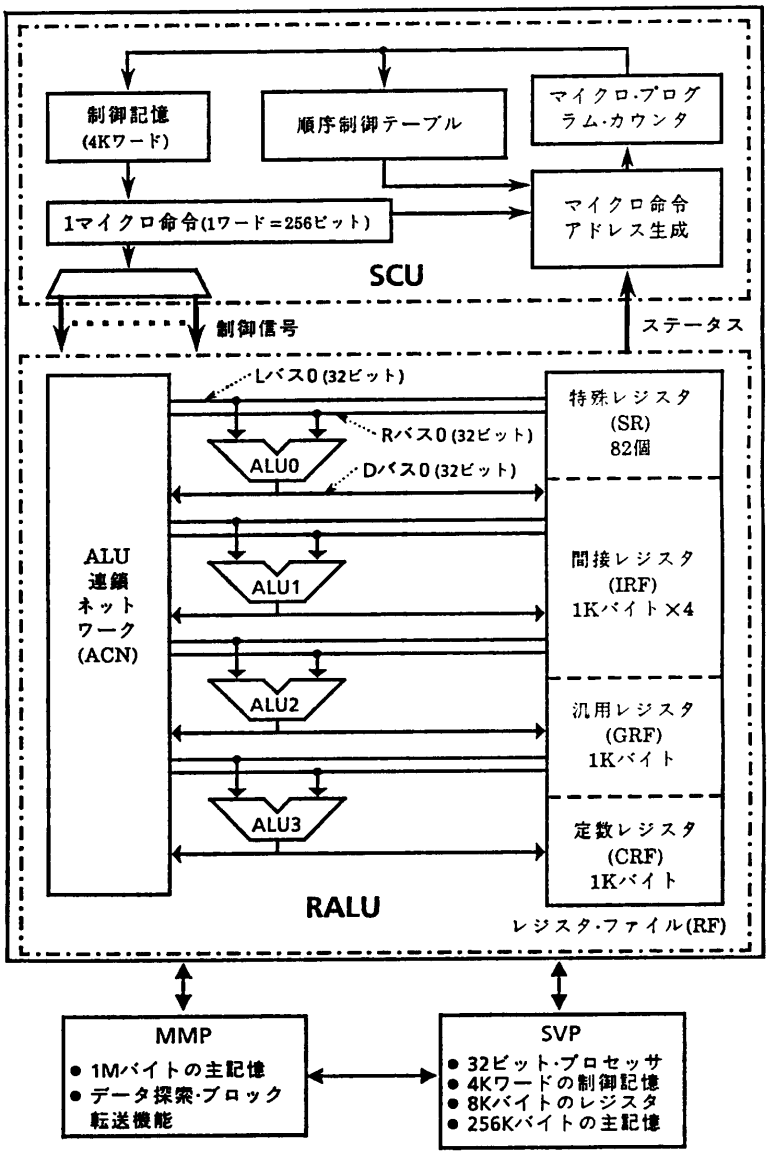

図 1 QA-2 のシステム構成
Fig. 1 System organization of the QA-2.

表 1 マクロ・アーキテクチャの比較
Table 1 Comparison between the macro-architectures.

比較項目	ISA-h (高レベル)	ISA-l (低レベル)
対象処理方式	インタプリト指向 (QPM-Ih)	コンパイル指向 (QPM-Il, 各 QPM-C)
構造体の表現法	構造共有	構造コピー
基本データ形式	8ビット・タグ，24ビット・データ	8ビット・タグ，24ビット・データ
データ領域 (QA-2 ファシリティ)	● 制御用スタック (GRF+MM) ● ローカル環境格納用スタック (IRF+MM) ● グローバル環境格納用スタック (MM) ● トレイル・スタック (IRF+MM)	● 引数レジスタ (GRF) ● バックトラック制御用スタック (IRF+MM) ● 環境格納用スタック (MM) ● ヒープ (MM) ● トレイル・スタック (MM)
マクロ命令コード領域 (QA-2 ファシリティ)	● 節ヒープ (MM) ● 構造ヒープ (MM)	● コード領域 (MM)

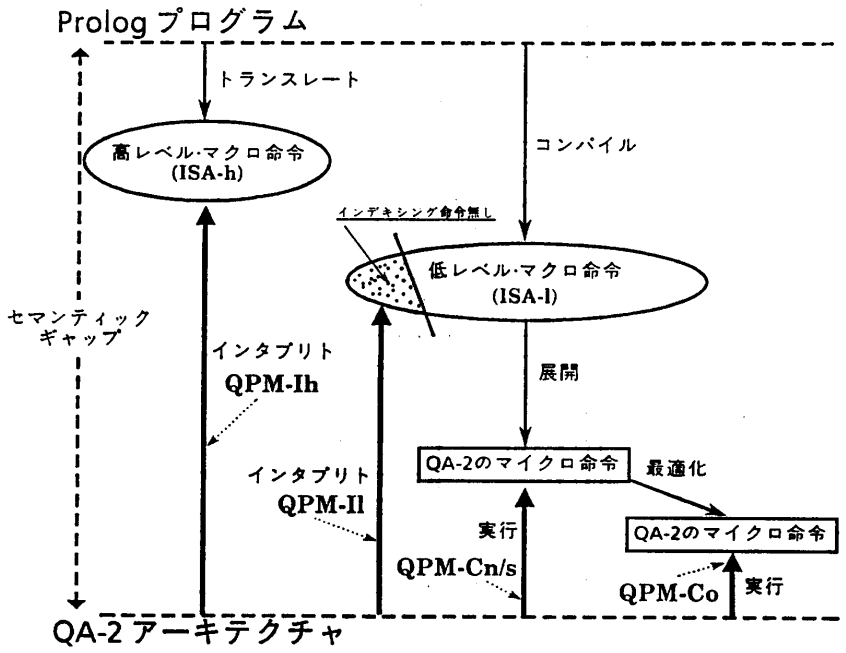

図 2 Prolog マシンのエミュレーションの概念図
Fig. 2 Schematic diagram of emulation of Prolog machines.

も基本データ形式として，8ビットのタグ・フィールドと24ビットのデータ・フィールドを持つタグ・アーキテクチャである．静的に識別可能な変数属性（例えば，グローバル/ローカルの別や，値の既/未束縛の別など）をタグ化して，変数の処理（値の束縛やデレファレンス）過程での最適化を図る．QA-2 の低レベル並列処理機能を利用したタグ切り出し・埋めこみ操作や，高機能順序制御方式を用いたタグ判定・分岐操作により，Prolog の処理に特有なユニフィケーション時の変数のデレファレンスや束縛操作で必要となるデータ属性検査を高速に行うことが可能である．

そのほか，いずれのマクロ・アーキテクチャのエミュレーションにおいても採用した処理技法として，次のようなものがある．

（1） 構造体データは CDR コーディング法によってヒープ内に格納し，冗長なデータ領域を無くすことと，ALU-レジスタ・レベル並列処理機能の活用（マイクロ命令の複数フィールドで並列制御する）を図る．

（2） TRO (Tail Recursion Optimization) をサポートすることによって，述語呼び出しなどの高速化を図る．

3.2 インタプリト指向マクロ・アーキテクチャ

高レベルのマクロ・アーキテクチャ（ISA-h と呼ぶ）は，Prolog で書かれたソース・プログラムの構造をそのまま反映した（1対1対応した）命令機能レベルにあり，ファームウェアで直接インタプリトする処理方式に適している．ISA-h のハードウェアないしはファームウェアによるインタプリタは，直接実行型高級言語計算機とみなしても良い．

ISA-h の機能レベルは ICOT で開発された PSI[5] とほぼ同等である．PSI の機能のかなりの部分はハードウェア化（専用の ALU・レジスタやバス構造）されているが，QA-2 ではこの命令機能のインタプリタを ALU-レジスタ・レベルの並列処理機能を活用して実現する．QA-2 上のこのインタプリタを，**QPM-Ih** と呼ぶ（**図2**参照）．**QPM-Ih** の実現においては，QA-2 の並列処理機能の活用を図るために，ISA-h と QA-2 のハードウェア・ファシリティとの対応付けが重要である．

QPM-Ih の処理方式においては，**図3**に示すように，Prolog プログラムは節単位にまとめられた1次元配列形式のマクロ命令コード列に変換され，節ヒープ領域に格納される．また，このマクロ・アーキテクチャでは，構造体の表現法として構造共有法を採用しており，構造体データのスケルトンは構造ヒープ領域に格納され，そのインスタンスはグローバル変数の値としてグローバル・スタック上のモレキュールとヒー

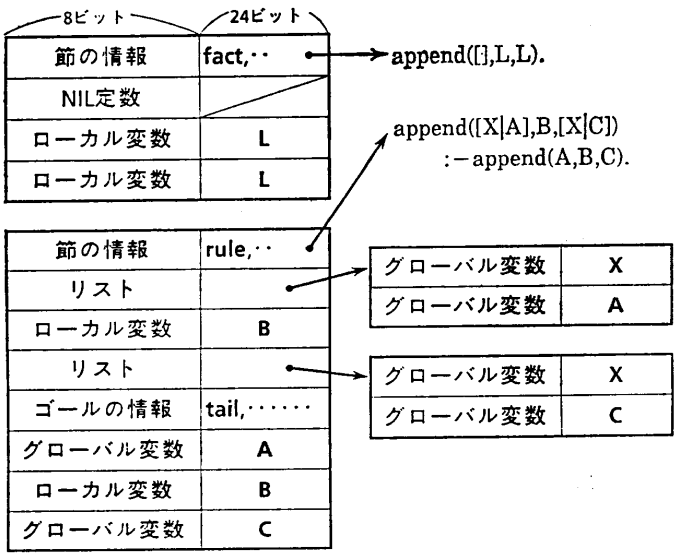

図 3 ISA-h の命令コード
Fig. 3 Instruction codes of the ISA-h.

プ上のスケルトンとで表される．節ヒープ領域も構造ヒープ領域も主記憶上に割り付けられている．1次元配列形式の命令コードや CDR コーディングされた構造体データの表現方式は，主記憶とレジスタ・ファイル間の大量（128 ビット）同時転送機能に良く適合している．

質問ゴールの実行は，節ヒープから命令コードを汎用レジスタ・ファイル (GRF) 内にフェッチし，それらが表1に示した4本のスタックを用いてインタプリトされる．変数の値は，述語呼び出し(ゴールの実行)ごとの環境として，グローバル/ローカル変数別に各各スタック上に積まれる．述語呼び出し時やバックトラック時の情報は制御用スタックに格納され，実行制御に利用される．トレイル・スタックには，バックトラック時に必要とされる変数束縛情報が積まれる．

これらのスタックのうち，ローカル環境格納用とトレイル・スタックは，間接レジスタ・ファイル (IRF) に割り付けられ，処理の高速化が図られている．IRF では多様なアクセス・モードを実現可能であり，かつ IRF 上に割り付けられたスタック領域は主記憶をバックアップ領域に使用して仮想化されている．制御用スタックとグローバル環境格納用スタックは主記憶に割り付けられている．さらに，制御用スタックでは，1回の述語呼び出しに関する情報をフレーム化して GRF に積んでいる．最新の述語呼び出しに関する情報は，このフレーム・バッファで管理し，主記憶アクセスをできるだけ減らすことによって，QA-2 の低レベル並列処理機能の活用を図っている．

ユニフィケーションにおいては，タグによる判定操作とデータ演算とを，1マイクロ命令で行うことができる．8ビット・タグによる順序制御においては，QA-2 の CASE-OF 分岐機構が有効に機能し，**QPM-Ih** インタプリタの核部分（組み込み述語用マイクロプログラムなどを除く）はわずか 161 マイクロ命令で記述されている．したがって，**QPM-Ih** インタプリタは，組み込み述語用マイクロプログラムを併せても，QA-2 の物理的な制御記憶容量（4Kワード）内に納まる．

3.3 コンパイル指向マクロ・アーキテクチャ

ISA-h をインタプリトする QPM-Ih に対して，Prolog プログラムの静的な解析をできるかぎり行い，よりハードウェアに近い機能レベル（低レベル）の命令に展開した後，これを高速に実行する方式がある．この展開されたマクロ命令の機能を実現するアーキテクチャを本論文では，ISA-l と呼ぶ．ISA-l マクロ命令列へのコンパイルでは，Prolog プログラムの構文解析やセマンティック解析などの手間がかかる．また，コンパイル後のマクロ命令列のソース・プログラムへの復元は不可能である．しかし，いったんコンパイルされた後の ISA-l マクロ命令列の実行は，高速に行い得る．

この ISA-l マクロ命令セットは，Warren によって最初に提案され[10]，その後各種のコンパイル指向型 Prolog マシン[13),14)]の命令セットとして採用されたものとほぼ同じである．

まず，Prolog プログラムを節ごとにノイマン型（1個のオペレーション・コードと数個のオペランドを持つ）の命令列にコンパイルし，その際に静的に決定できる情報をできるかぎり命令に反映させ，マクロ命令列の最適化を図る．そして，この低レベル・マクロ命令列をファームウェアで解釈・実行する．Warren らが提案したマシン[11)]においては，引数評価レベルのノイマン型命令のフェッチと実行とをパイプライン制御する方式を前提にしているが，QA-2 上では ALU-レジスタ・レベル並列処理機能を活用しつつ，このマクロ命令列を逐次的に処理する．

Prolog プログラムから ISA-l マクロ命令列へコンパイルする際に適用した最適化戦略としては，(1)節頭部と節本体に現れる引数に対する命令をそれぞれ区別 (get/put 命令) する，(2)バックトラック制御を

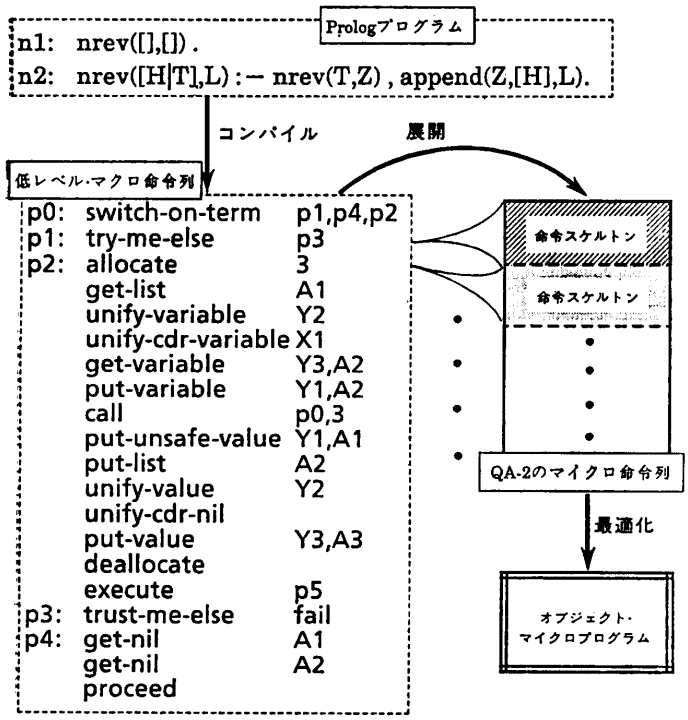

図 4 低レベル・マクロ命令を介したマイクロ命令列へのコンパイル
Fig. 4 Compiling into micro-instructions through the low-level macro-instructions.

命令化して（try/retry/trust 命令など）しかも不必要なバックトラック情報は生成しない，(3)節形式の違い（単位節や唯一ゴールの無単位節など）により述語呼び出し時の不必要な環境や選択点の割り付けを省略する，(4)第1引数によるインデキシング技法を適用する，(5) TRO により不必要な環境を畳み込むための情報を（call 命令などに）付加する，(6)変数の有効範囲によりその属性（一時/永久/非安全）を識別する，(7)節における変数の出現順により変数への値の束縛についての情報（variable/value 命令）を決定する，(8)ユニフィケーション時の冗長な引数レジスタ間転送命令を除去する，(9)構造体の CDR 部を識別して CDR コーディング法を適用する，などがある．また，コンパイルされた命令コード列（図4に例を示した）は，主記憶に割り付けられたコード領域内に格納される．なお，このコンパイラは Prolog で記述してある．

ゴールの実行においては，述語呼び出し側の引数が順々に評価されて（put 命令），その値が引数レジスタに格納される．その後，呼び出された側の述語の引数とのユニフィケーション（get 命令）が行われる．また，ユニフィケーション（unify 命令）によりインスタンスとなった構造体データはヒープ中にコピーされる(構造コピー法)．バックトラック情報と変数環境とは分離されて，各々バックトラック制御用スタックと環境格納用スタックとで管理される．また，構造体データのユニフィケーションは，変数の処理（値の束縛やデレファレンス）用のマイクロ・サブルーチンを多重に呼び出す方式により行っている．

ISA-l で採用しているデータ形式は，ISA-h と同じであり，8ビット・タグと 24 ビット・データを基本としている．マクロ命令のオペランドとして指定される引数レジスタは，汎用レジスタ・ファイル (GRF) に割り付けられ，マイクロ命令のオペランドとして直接アドレス指定が可能である．スタックやヒープは主記憶上に割り付けられているが，選択点に関する情報を格納しておくバックトラック制御用スタックのトップ部分は，間接レジスタ・ファイル (IRF) に割り付けられている．この IRF には選択点ごとに固定長のフレームが積まれ，最新のフレームが 64 個まで格納できるバッファとして機能させている．マクロ・アーキテクチャとして必要な特殊レジスタ（例えば，スタック・トップを指すアドレス・レジスタなど）は，すべて GRF に割り付けてある．

ISA-l を忠実にインタプリトするエミュレーション方式を **QPM-II** と呼ぶ(図2参照)．ただし，**QPM-II** では，マクロ・アーキテクチャの違いによる QA-2 の適応性を調べるために，最適化による性能向上の大きな要因となる可能性の高いインデキシング命令はサポートしていない．ISA-l のレジスタなどは，QA-2 のハードウェア・ファシリティと直接対応している部分も多い（例えば，引数レジスタは汎用レジスタに割り付けられマイクロ命令で直接アドレス指定される）．したがって，QPM-Ih のマクロ・アーキテクチャである ISA-h とは異なり，マクロ命令のフェッチを実行するマイクロプログラムがオーバヘッドとなる可能性が高い．これについては，4.2節で評価を加える．

QPM-II インタプリタは，ISA-l マクロ命令セットに対するマイクロプログラムをすべて制御記憶内に用意しておく必要があるので，組み込み述語用マイクロプログラムを除いても約500ワードの制御記憶領域を占める．

3.4 マイクロプログラム・コンパイル方式

前節で述べた ISA-I は，そのマクロ命令を直接フェッチ・実行できる専用ハードウェアがあれば，QPM-II で生じるような命令フェッチ・サイクルによるオーバヘッドが無くなり，ISA-h のエミュレーションよりはかなり高速の処理が可能である．その方法としては，Warren が提案したマシンで採られたように，命令フェッチ・サイクルと実行サイクルをオーバラップさせて，命令列をパイプライン処理するアーキテクチャである．しかし，QA-2 のハードウェア構造はパイプライン形処理よりも MIMD 形並列処理向きに設計されており，この形のマシンをそのままエミュレーションする方式には馴染まない．

そこで我々は，Prolog プログラムを直接 QA-2 のマイクロプログラムに変換するマイクロプログラム・コンパイル方式を導入した．Prolog プログラムと QA-2 アーキテクチャのセマンティック・ギャップは大きいので，Prolog プログラムはいったん QPM-II と同じ ISA-I マクロ命令列にコンパイルされ，その後 QA-2 のマイクロプログラムに変換され，実行される．この方式によるエミュレーションを **QPM-Cn** と呼ぶ（図2参照）．QPM-II からインタプリタ方式では必須である命令フェッチ・シーケンスを取り除き，さらに低レベル・マクロ命令としてインデキシング命令をサポートした処理方式が **QPM-Cn** であると言える．

QPM-Cn では，ISA-I マクロ命令の各々に対応するマイクロルーチン（これを単独では不完全なマイクロ命令列という意味で「命令スケルトン」と呼ぶ）を用意しておき，Prolog プログラムからコンパイルされたマクロ命令列を，これらの命令スケルトンを用いてマイクロ命令列に機械的に展開する（図4参照）．この命令スケルトンは，次のような意味であらかじめ QA-2 アーキテクチャに依存した最適化を施されていると言える．(i)定数データは，マイクロ命令のリテラル・フィールドか定数レジスタ・ファイル (CRF) へ埋めこまれる．(ii)引数レジスタやバックトラック制御用スタックは，QPM-II と同様に，各々 GRF や IRF に割り付けられ，間接アドレッシング機能などの多様なアクセス・モードを利用している．(iii)手続き型のマクロ命令 (call や proceed 命令など) は，マイクロ命令の順序制御フィールドに直接埋めこまれる．(iv)マクロ・アーキテクチャのファシリティを QA-2 のハードウェア・ファシリティに直接対応（例えば，プログラム・カウンタをマイクロプログラム・カウンタに，ヒープ・アドレス・レジスタを MAR になど）させる．(v)マイクロ・サブルーチンを，それを呼び出しているマイクロルーチン内でインライン展開し，そこに埋めこむ．

(v)の最適化を行わない方式を **QPM-Cs** と呼び，QPM-Cn とは区別して評価の対象とする．**QPM-Cs** では，QPM-II と同じく，実行頻度の高いユニフィケーション・ルーチンの一部をサブルーチン化している．しかし，マクロ命令列から展開された個々のマクロ命令の機能レベルは低いので，サブルーチンの呼び出しやそれからの復帰に要するマイクロ命令がオーバヘッドとなることが予想される．これについては，4.3 節で評価を加える．

ISA-I の機能レベルは低いので，QA-2 の並列処理機能や順序制御機能を利用すると，各マクロ命令は短く記述できる．特に，ALU 演算機能をほとんど必要としない手続き型のマクロ命令（例えば，call/execute/proceed 命令）のスケルトンは，それぞれ1～2マイクロ命令で実現されている．生成されるマイクロプログラムの長さは，もちろん問題ごとに異なるが，例えば33マクロ命令から構成される「naive-reverse 30」プログラム（以後，「nrev 30」プログラム）では，**QPM-Cn** の場合静的に213マイクロ命令（同じ問題でも，**QPM-Cs** では，191マイクロ命令）を必要とする．

QPM-Cn で生成された問題ごとに独立なマイクロプログラムは，さらにマイクロ命令レベルでの最適化が可能である．この方式によるエミュレーションを **QPM-Co** と呼ぶ（図2・図4参照）．マイクロプログラムの最適化の方式として，まず元の命令スケルトンの継ぎ目にあたるマイクロ命令列に注目して，可能なものは1マイクロ命令に合成する．特に，手続き型のマクロ命令に対応するマイクロプログラムには ALU 演算フィールドの空きが多く，この命令を合成できる可能性が高い．また，命令スケルトンごとでは独立していたフラグ判定用マイクロ命令を，展開されて1本のストリームとなったマイクロプログラム中で何回も実行することは冗長なので，省略する．これらのマイクロプログラムの最適化戦略は，機械的に適用可能なものばかりである．例えば「nrev 30」プログラムの静的なマイクロプログラム容量は，16マイクロ命令だけ減る．

QA-2 の制御記憶は，主記憶をバックアップにして

仮想化されており（仮想制御記憶方式），コンパイルされた大量マイクロプログラムの格納，実行にも耐え得るようになっている．すなわち，制御記憶の現在の実装容量は 4Kワードであるが，制御記憶は1ページ＝512ワードでページ化され，15ビットの論理アドレスによる仮想空間を構成している．この仮想制御記憶方式は，QA-2のシステム管理プロセッサ(SVP)により管理されている．ページ・フォールトはアドレス変換時に QA-2 内のハードウェア回路により検知され，SVP によって物理的な制御記憶1ページが動的に入れ換えられる．現在のページ入れ換えアルゴリズムは，FINUFO (First In Not Used First Out) 法であり，これは SVP のファームウェアを書き換えれば，問題に応じて変更することが可能である．また，主記憶から制御記憶への1ページ分の転送には，約400 マイクロ秒を要する．

なお，マイクロプログラム化されたインタプリタ (QPM-Ih と QPM-Il) や，マイクロプログラム・コンパイラ (QPM-Cn/Cs/Co) によって展開された例題（本論文で引用している）プログラムに対するオブジェクト・マイクロプログラムは，それぞれすべて物理的な制御記憶容量内に納まる．しかし，特に実用的なプログラムをマイクロプログラムにコンパイルすれば，そのオブジェクト・マイクロプログラムの静的な容量はかなり大きなものになることが予想される．したがって，逐次型 Prolog マシンのエミュレーションに適応した，仮想制御記憶方式におけるページ分割法やページ入れ換えアルゴリズムの開発は，今後の課題として残されている．

4. Prolog マシンの評価

4.1 例題プログラムの特性

各マクロ・アーキテクチャ (ISA) やその処理方式 (QPM) の評価およびそれらのエミュレーションへの QA-2 アーキテクチャの適応性を評価するために，「nrev 30」プログラムを各処理方式に従って実行した場合について，種々の定量的な評価データを収集した（**表2・表3**参照）．

「nrev 30」プログラムは，Prolog コンテスト[3]の課題でもある有名なベンチマークであり，比較のために

表 2 各エミュレーション方式の性能比較
Table 2 Performance comparison among the emulations.

評価項目	QPM-Ih	QPM-Il	QPM-Cs	QPM-Cn	QPM-Co
マイクロ命令実行ステップ総数† († の相対比)	50,197 (4.03)	32,128 (2.58)	13,470 (1.08)	12,449 (1)	9,568 (0.77)
1 LI 当たりの平均実行マイクロ命令数	101	65	27	25	19
LIPS	9.2K	14K	35K	37K	45K

注）いずれも「nrev 30」プログラム実行時のデータ．

表 3 エミュレーションによる QA-2 アーキテクチャの動的評価
Table 3 Run-time statistics of the QA-2 architecture on the emulations.

評価項目	QPM-Ih	QPM-Cn	QPM-Co
マイクロ命令実行ステップ総数† († の相対比)	50,197 (4.03)	12,449 (1)	9,568 (0.77)
主記憶アクセス命令ステップ数/†(%)	32.4	37.0	48.2
条件分岐命令ステップ数/†(%)	41.1	50.3	46.6
1マイクロ命令当たりの平均ALU使用個数	2.93	1.64	2.14
レジスタ間転送演算数/実行ALU演算総数 (%)	56.3	61.6	61.6
ALU連鎖演算数/実行ALU演算総数(%)	20.8	36.6	36.8
カレント・モード分岐命令ステップ数/ 条件分岐命令実行ステップ総数(%)	84.1	71.3	100
2方向分岐命令ステップ数/ 条件分岐命令実行ステップ総数(%)	61.8	0.48	0.67

注）いずれも「nrev 30」プログラム実行時のデータ．

本論文で引用している種々の LIPS (Logical Inference/秒) 値は，できるかぎりこのプログラムを基に算出するようにした．このプログラムは，非常に簡単なものであり，これによる評価が Prolog に対する処理機能のすべてを規定するものではない．例えば，組み込み述語の呼び出し，カットの処理，データベースへのアクセス，入出力機能などについては，別のベンチマークが必要である．しかし，Prolog マシンの処理機能としての基本部分（特に，ユニフィケーション演算，述語呼び出しの制御，構造体データの処理など）の特性は，このベンチマークで測定できるものと考え，他のマシンにおける資料もそろっていて比較しやすいことも配慮して，本論文における動的評価のベンチマークは「nrev 30」プログラムに統一した．

表2では，全処理方式 (QPM) についての相対的な性能比較を行った．また表3には，QPM-Ih/Cn/Co に関する，さらに詳しい評価（特に QA-2 アーキテクチャについて）データを掲げてある．本章の以降の議論は，主として表2や表3に示した評価データに基づいて行う．

4.2 インタプリト方式の評価

◎**QPM-Ih** の評価

高レベル・インタプリタである **QPM-Ih** では，実行マイクロ命令総数に占めるユニフィケーション（引数フェッチ，デレファレンス，変数束縛など）用ルーチン実行の割合が約70%である．したがって，実行頻度の高いこれらの処理シーケンスの専用ルーチン化（節や変数の属性を前提とした処理の最適化）の効果は著しいと言える．また，1マイクロ命令当たりの平均 ALU 使用個数（以降では，単に「平均 ALU 使用個数」と呼ぶ）は，約3であり，高レベル・インタプリト方式においては QA-2 の低レベル並列処理性を，十分に引き出せることが分かった．

具体的には，実行頻度の高いユニフィケーション用ルーチンにおいて，2個のオペランドを別々の ALU で並列にフェッチ，デレファレンスする技法を用いている．またこのルーチンでは，タグ・フィールドの抽出や埋め込みなども余った ALU を用いて行い，変数束縛ルーチンへの場合分けについても，タグに従った CASE-OF 分岐マイクロ命令1個で実行している．

しかし，1LI (Logical Inference) の実行に必要なマイクロ命令数は101命令であり，プログラムを静的に解析すれば得られる情報についてもインタプリト方式として実行時に判断が持ち越されるためのオーバヘッドが相当にあることが推定される．

◎**QPM-Il** の評価

ISA-h をインタプリトする QPM-Ih に対して，より低レベルのマクロ・アーキテクチャである ISA-l をインタプリトする **QPM-Il** では，性能が1.6倍になる．**QPM-Il** はインデキシング命令をサポートしていないので，コンパイラがマクロ命令レベルでの最適化を十分に行っているとは言えないが，この性能差が ISA-h と ISA-l の機能レベル差であり，Prolog プログラムを低レベル・マクロ命令列にコンパイルする際に種々のセマンティック解析を行う効果を表していると判断して良い．また，**QPM-Il** において，インデキシング命令をサポートした場合，さらに約40% 実行マイクロ命令数が減少することが見込まれる．

しかし，**QPM-Il** の平均 ALU 使用個数は，QPM-Ih のそれよりも1程度低下する．ISA-l はもともとパイプライン形アーキテクチャ向きに設計されたものであり，その処理においては本質的に逐次性が顕在化してくる．QA-2 では，これに ALU 連鎖演算機能を適用しており，また同一オペランドによる異種演算も並列処理可能なので，最適化を十分に行えば，ISA-l 向きの専用マシンに近い性能を発揮できる．

しかし，**QPM-Il** による実行マイクロ命令総数のうち約25% は低レベル・マクロ命令をフェッチするために費やされたものである．**QPM-Il** では，命令解釈（1マクロ命令当たり2マイクロ命令を必要とする）のオーバヘッドが，個々のマクロ命令機能の単純化（1マクロ命令は平均6マイクロ命令で実行される）により，顕在化したと言える．

4.3 コンパイル方式の評価

◎**QPM-Cs** の評価

QPM-Cs は，QPM-Ih の3.7倍，QPM-Il の2.4倍の性能を持っている．また「nrev 30」プログラムでは，35 K LIPS を発揮し，QA-2 を ISA-l に適応させる方式で，PSI などのインタプリト指向専用マシンとほぼ同程度の性能が出ることが分かった．

◎**QPM-Cn** の評価

QPM-Cn では，マイクロ・サブルーチンが呼び出し箇所でインライン展開されており，サブルーチン呼び出しなどによるオーバヘッドが無いので，QPM-Cs よりも約8% 実行マイクロ命令数が減少する．

例題プログラムでは，**QPM-Cn** の1マクロ命令は平均3マイクロ命令で実行されており（1LI は25マイクロ命令で実行される），この低レベル・マクロ命

令の機能が QA-2 のハードウェア機能にほぼ適応化していることが分かる．**QPM-Cn** の性能は QPM-Ih の約4倍であるが，逆に平均 ALU 使用個数は1.6程度にとどまっている．ISA-I では，Prolog プログラムのコンパイルの際に十分なセマンティック解析を行い，その結果を **QPM-Cn** でも最大限に利用している．しかし，QPM-II の評価でも述べたように，ISA-I はもともとパイプライン形マシンによる逐次処理を指向しており，広域的な最適化などを行わないならば，ハードウェアとして用意すべき ALU 演算レベルの並列処理機構としては2ALU 程度で十分である．

4.4 マイクロプログラム・コンパイルの効果

◎**QPM-Co** の評価

QPM-Cn で生成されたマイクロプログラムをさらに最適化すると（**QPM-Co**），約20% 実行マイクロ命令数が減少し（実行速度は1.3倍になる），最終的な実行速度は 45 K LIPS となった．また，1 LI は平均19マイクロ命令で（1マクロ命令当たり2.4マイクロ命令で）実行され，Warren が提案したパイプライン形 Prolog マシンの必要実行命令ステップ数（1 LI を22サイクルで実行する）をしのいでいる．

マイクロプログラム・レベルの最適化により，平均 ALU 使用個数も2を超え，水平型マイクロ命令フィールドへの ALU 演算の合成効果が現れている．また **QPM-Co** では，主記憶アクセスがあい路になることもなく，冗長な順序制御命令も最適化により省略しているので，問題を直接マイクロプログラムで記述した（ただし Prolog の構文やセマンティクスは生かしたままで）場合とほぼ同程度の性能を発揮しているものと考えて良い．

QA-2 における問題適応化方式として残されているものは，マイクロプログラムの広域的な最適化手法であるが，これを各々が数マイクロ命令にしか展開されない ISA-I の命令スケルトンを前提にして機械的に

表 4 種々の Prolog マシンの性能比較
Table 4 Performance comparison among several Prolog machines.

評価項目	専用 Prolog マシン					
	インタプリト指向			コンパイル指向		
	PSI	PEK	連想メモリ	Pipelined	PLM-1	CHI
開発者・機関	ICOT	神戸大学	NTT 厚木研	Tick and Warren	Univ. of California, Berkeley	ICOT and 日本電気
命令長（ビット）	64	96	—	80～100	144	80
命令サイクル時間（ナノ秒）	200	120～400	200	100	100	100
LIPS	36K	67K	100K*	450K*	187K¶	280K*
平均命令ステップ数/1LI	139	124†	50*	22*	53¶	36*

評価項目	専用 Lisp マシンやユニバーサル・ホスト計算機による Prolog マシンのエミュレーション					
	インタプリト指向			コンパイル指向		
	FACOMα	QPM-Ih	ELIS	FACOMα	QPM-Co	3600
開発者・機関	富士通	京都大学	NTT 武蔵野研	富士通	京都大学	Symbolics Inc.
命令長（ビット）	48	256	64	48	256	17
命令サイクル時間（ナノ秒）	152	1,076‡	180	152	1,150‡	200
LIPS	12K	9K	11K	32K	45K	53K§
平均命令ステップ数/1 LI	548	101	505	206	19	94§

† 命令サイクル時間が可変長なので，最小サイクル時間(120 ナノ秒) で計算した．
‡ 動的に測定した平均サイクル時間である．
¶ 人間がコーディング（ハンド・コンパイル）する方法によるデータ．
§ IFU (Instruction Fetch Unit) 付きマシンによるデータ．
* 「append」プログラムによるデータ．他は「nrev 30」プログラムによるデータ．

行うのは困難である．また，発見的手法に頼ったとしても劇的な性能向上の可能性は少なく，QA-2 上での最高速 Prolog マシンは **QPM-Co** としても良い．

4.5 他の Prolog マシンとの比較

表 4 に示したように，**QPM-Ih** と **QPM-Co** について，他の Prolog マシンとの性能比較を行った．この比較で取り上げた対象[3),4),12)]は，専用の Prolog マシンあるいは Lisp マシン上でのエミュレーションによる場合である．また，提案の段階にとどまっているものでも性能予測を行っているものは掲げた．基本的なマシン性能の尺度を示すものとして，マイクロプログラム制御方式のものはマイクロ命令サイクルを，それ以外のものはマシン命令サイクルを掲げてある．また，ALU-レジスタ・レベルの機能を評価する尺度として参考にするために，命令（マイクロ命令あるいはマシン命令）長も掲げた．なお，表 4 に掲げた LIPS 値は比較のためにできる限り「nrev 30」プログラムによるものを引用しているが，一部(表 4 において * を付けたもの）は「append」プログラムによる．

PSI[5),6)] と PEK[7),8)] は，いずれも ISA-h と同程度の機能レベルのマクロ命令をインタプリトする専用マシンであり，1 LI 当たりの平均命令ステップ数が 100 を超えている．

QPM-Ih は，これらの専用 Prolog マシンよりもさらに少ない命令ステップ数で 1 LI を実行しており，Prolog の言語機能に近い高級言語のインタプリトにおいては，低レベル並列処理方式が効果的であると判断できる．

NTT・厚木研が提案している Prolog マシン[9)]は，高レベル命令の解釈・実行機能を連想メモリを用いて実現しようというものであり，高レベル・インタプリタの専用化の効果を高機能メモリの導入により得ようとしている．

PLM-1[12),13)] は，Warren らが提案したパイプライン形コンパイル指向専用 Prolog マシン[10),11)] の実現を図ったものと考えられ，両者のマイクロ・アーキテクチャはほぼ同一である．ISA-l はこれらのマシンのマクロ・アーキテクチャとほぼ同じである．PLM-1 のマイクロ命令長は水平型で 144 ビットと比較的長いが，マイクロ・アーキテクチャは非常に簡潔なパイプライン構造になっており，QA-2 のように ALU 演算機能レベルでの並列処理方式は採用されていない．

CHI[14),15)] のマクロ・アーキテクチャも ISA-l と同じ機能レベルにあり，PSI のバックエンド・プロセッサとして位置付けられている．

Prolog 専用ではなく，Lisp マシンを Prolog の処理に適用する試みは，インタプリタ方式を採るもの[16),18),19)]が，1 LI 当たり約 500 命令程度の実行ステップを必要とし，一般的にインタプリト指向専用 Prolog マシンの 1/3 位の性能を示している．これらの Prolog インタプリタはいずれもマイクロプログラムで記述されているが，マイクロ・アーキテクチャが Lisp 処理あるいは一般的なリスト処理向きとなっている点に，専用 Prolog マシンとの Prolog の処理における性能差が現れていると言える．

リスト処理という点からみると，Lisp マシンというよりむしろ汎用的なマシン（例えば，Symbolics 3600[20)]）やユニバーサル・ホスト計算機上では，ISA-l の機能レベルのマクロ命令に最適なマイクロプログラムを実現しさえすれば，相当の（インタプリト指向専用 Prolog マシンを凌駕するくらいの）性能が発揮されることが予想される．

例えば，Lisp マシンである FACOMα におけるインタプリト方式[18)]とコンパイル方式[17)]との差は 2.7 倍であるのに対して，**QPM-Ih** と **QPM-Co** の差は約 5 倍もある．これは，QA-2 のハードウェア機能がユニバーサル・ホスト計算機として汎用化されているために，マイクロプログラム・コンパイル方式による問題適応化が有効に機能したことを示している．QA-2 の実装方式を改良して，マイクロ命令の高速化を図れば，ユニバーサル・ホスト計算機でもコンパイル指向専用 Prolog マシンとほぼ同程度の性能が出るものと考えている．

5. QA-2 アーキテクチャの評価

5.1 低レベル並列処理方式の評価

本節では，QA-2 を相異なる機能レベルを持つエミュレーションに適用した結果を基に，QA-2 で実現している低レベル並列処理方式の有効性について検討を加える．そのために，評価の対象を各マクロ・アーキテクチャの代表的な処理方式である **QPM-Ih** と **QPM-Co** の 2 種類に絞り，各々について，現在のハードウェア構成である 4 ALU を基に，ALU 個数を 1, 2, 8 と変えた場合の性能比を調べた（**表 5** に，この結果を示す）．ただし，この評価においては，主記憶へのアクセス幅も ALU 個数に合わせて変えるものと仮定している．

QA-2 の低レベル並列処理機構を 8 ALU 構成にし

表 5 ALU 構成や順序制御方式を変えた場合の性能比較
Table 5 Evaluation of the QA-2 architecture on a variety of its ALU organization and sequence control mechanism.

処理方式	ALU 個数				連鎖演算機能なし	単純分岐形式のみ
	1	2	4 (現状)	8		
QPM-Ih†	2.94	1.65	1	0.79	1.38	1.40
	1/1	1.78/2	2.93/4	3.70/8	2.12/4	2.09/4
QPM-Co†	2.14	1.28	1	0.99	1.48	1.42
	1/1	1.67/2	2.14/4	2.16/8	1.45/4	1.51/4

† 上の行…マイクロ命令実行ステップ数の相対比.
下の行…1マイクロ命令当たりの平均 ALU 使用個数.
いずれも「nrev 30」プログラム実行時のデータ.

た場合，QPM-Ih では平均 ALU 使用個数が2.93から3.70へ約26%も上がるのに対して，QPM-Co ではこれが約9%しか向上しない．QPM-Co では現状の4ALU 構成で，既にマイクロプログラムの最適化など，高度な問題適応化が進められ，低レベル・マクロ命令の処理における本質的な逐次性の壁にぶつかっているものと考えられる．

ユニバーサル・ホスト計算機はもともと高レベル・インタプリト方式向きに設計されている．QPM-Ih での評価では，2ALU から 4ALU 構成へ変えた場合に性能（実行速度）が1.65倍になるのに対して，4ALU を8ALU 構成へ変えた場合には，それが1.27倍にしか向上しない．さらに QPM-Co の場合には，現状から 8ALU 構成にしても，その性能向上はほとんど無い．8ALU 構成の場合，ハードウェア規模は現状の2倍以上になることと，平均 ALU 使用個数の評価結果を併せて考えると，現状の 4ALU 構成は逐次型 Prolog の処理に対して適当であると判断できる．

また，ALU 連鎖演算機能の効果を調べるために，この機能を無くした場合の性能について定量的に調べた（表5参照）．実行 ALU 演算総数に対する ALU 連鎖演算数の割合は，既に表3に示したように，20～37% もある．特に，QPM-Co の場合は QPM-Ih よりも逐次処理が多いので，ALU 連鎖演算機能を取り外すと，実行マイクロ命令数が約50%も増加し，平均 ALU 使用個数は 2ALU 構成にした場合よりも低く（1.45 個）なってしまう．QPM-Ih の場合でも，この機能の効果は大きいことも考慮すると，本エミュレーションのように明示的な並列性が無い応用に対しては，ALU 連鎖演算は必須の機能であると判断して良い．

主記憶アクセスの割合は，3次元グラフィックスのように大量で一様な構造のデータへの並列アクセスを伴う問題とは異なり，Prolog マシンのエミュレーションでは，どの処理方式においても余り高くない（表3参照）．主記憶に割り付けられているのは，スタックやヒープ領域であり，いずれもマクロ命令の処理時に逐次アクセスされる性質をもつものである．したがって本応用では，RALU と MMP とのアクセス幅も4ALU 構成に合わせておけば（32×4 ビット）十分である．

表3によると，ALU 演算を何も行わずに単にレジスタ間でデータを授受するだけという演算の割合が，いずれの処理方式の場合にも50%を越えている．これは，各マシンがタグ・アーキテクチャを採用し，タグとデータ・フィールドとを別の ALU によって並列処理しているために，ユニフィケーションにおけるデレファレンスや変数束縛の操作が単なるデータ転送のみで済む場合が多いことによる．しかし，MAR や MDR などの特殊レジスタ上のデータを汎用レジスタ（GR）へ退避したりする処理も多い．したがって，特殊レジスタのハードウェア構成も GR と同様に一様化（どの ALU からもアクセス可能）して，その個数も増せば，例えばマクロ・アーキテクチャの論理的レジスタ（スタック・アドレス・レジスタなど）を直接 MAR などに対応付けることができるので，レジスタ間転送のオーバヘッドを若干軽減することが可能となる．

5.2 高機能順序制御方式の評価

いずれのマクロ・アーキテクチャもタグ・アーキテクチャを基本としているので，タグの操作機能が処理性能を左右する要因となる．特に，QA-2 の高機能順序制御方式が，これらのタグ・アーキテクチャに適合しているかどうかを調べる必要がある．

QA-2 の順序制御方式は，IF-THEN-ELSE 分岐と CASE-OF 分岐のいずれの形式においても，最大8個のステータスを自由に組み合わせた条件判定機能や，GOTO 文以外にマイクロ・サブルーチン呼び出しなどを自由に組み合わせることのできる条件分岐機能を指定できる，非常に高い機能を持つものである．この機能について定量的に評価するために，現状の順序制御機能を "IF X THEN GOTO A ELSE NEXT;" の単純な分岐形式（1個のステータスによる2方向分岐）しか持たない場合に，例題プログラムでどれだけの性能低下があるのかを調べた．表5に示したように，いずれの処理方式の場合にも約40%も実行マイクロ命令数が増加する．また平均 ALU 使用個数も，ステータスを生成する演算部とステータスを使用する順序制御部との機能バランスが崩れて，相当に（特に QPM-Ih の場合には約1個も）低下する．

またいずれの処理方式においても，カレント・モード（そのマイクロ命令実行によって生成されたステータスを直ちに使用して順序制御を行うモード）による条件分岐の割合が高い．このモードの分岐では，RALU による ALU 演算と SCU による順序制御がオーバラップされない．しかし，このモードによる分岐が多い理由は，QA-2 の並列処理機能が高くて，生成されたステータスをセーブする必要がなく，直ちに使用するからであり，RALU と SCU のいずれもが処理のあい路になることなく，機能バランスはとれていると判断できる．実際に，いずれの処理方式においても，おおむね2マイクロ命令（4×2 個の ALU 演算能力）につき1個の割合で高機能条件分岐機能が働いていることを確かめている（表3参照）．

低レベルのマクロ・アーキテクチャを用いる QPM-Cn/o では，2方向分岐をほとんど使わず，条件分岐の大部分を多方向分岐機能に頼っている．しかし，これらの分岐で使用する条件変数（ステータス）の個数は4個までが大部分であり，QPM-Ih におけるユニフィケーション時のようにハードウェア機能の上限である8個を使用する多方向分岐は無い．その理由としては，QPM-Ih はインタプリト指向の処理方式であり，タグの検査が実行時に持ち越されることが多いことが挙げられる．

QA-2 における IF 分岐機能の利点は，ステータスを単に論理値として並べるだけの CASE 分岐機能と異なり，最大8個のステータス変数を論理関数として複雑に組み合わせることができる点にある．しかし，実際に QPM-Ih で使用された論理変数は単純なものばかりであり（各タグ・ビットは論理値としてあらかじめデコードされている情報とみなせる），2方向分岐機能のハードウェア規模が大きいことを考慮すると，マイクロプログラム・コンパイラに論理関数の展開（場合分け）を行わせて，ハードウェアとしては CASE 分岐だけをサポートする構成方式も考えられる．例えば，2方向分岐命令 "IF X & Y THEN GOTO A ELSE NEXT;" を，多方向分岐命令 "CASE (X, Y) OF /00/ NEXT /01/ NEXT /10/ NEXT /11/ GOTO A;" に変換してしまう．この方式の評価については，さらに考察を加える必要がある．

6. むすび

ユニバーサル・ホスト計算機 QA-2 上で，逐次型 Prolog マシンのエミュレーションを行い，Prolog の処理方式について考察するとともに，QA-2 アーキテクチャについて定量的かつ実際的に評価した．エミュレーションの対象マクロ・アーキテクチャとしては，機能の相異なる2種を考え，各々に対して QA-2 アーキテクチャの特徴を活用した処理方式を開発した．

特に，Prolog プログラムを低レベル・マクロ命令を介して QA-2 のマイクロプログラムに変換する，マイクロプログラム・コンパイル方式によって，ユニバーサル・ホスト計算機である QA-2 がインタプリト指向 Prolog マシンと同程度の性能を発揮できるアーキテクチャを持つことを確かめた．しかし，QA-2 ハードウェアの実装規模は大きく，本応用における平均マイクロ命令実行サイクルは，約1,000ナノ秒となっている．現在では，AMD 社の Am 293 XX シリーズや Weitek 社の浮動小数点演算器など，新しいビルディング・ブロックが出そろってきたので，これらを利用してよりコンパクトでしかも高性能な低レベル並列処理計算機を実現できるものと考えている．その場合のマイクロ命令実行サイクル時間としては，他の専用 Prolog マシンが実現している 100〜200 ナノ秒は十分可能な値であり，本論文で述べたマイクロプログラム・コンパイル方式による処理を行うと，現在提案されている最高速のコンパイル指向 Prolog マシン[10),11)]とほぼ同程度の処理性能を得られるものと判断している．

本研究により，ユニバーサル・ホスト計算機 QA-2 が高級言語処理などの明示的な並列性を含まない応用

の高速化も十分達成し得る問題適応性を持つことを確認した．広域的かつ一般的なマイクロプログラムの最適化方式の開発や，マン－マシン・インタフェースを考慮した応用システムの実用化が，今後の課題である．

謝辞 Prologプログラムを低レベル・マクロ命令列へ変換するコンパイラの基本部分を作成してくださった，本学大学院・修士課程2年島田陽一君に感謝いたします．また，本学・萩原研究室の QA-2 開発・保守チームの諸氏に深謝いたします．本研究の一部は文部省・科学研究費補助金によっている．

参考文献

1) 柴山，富田，萩原：ユニバーサル・ホスト計算機 QA-2 による逐次型 Prolog マシンのエミュレーション，電子通信学会技術研究報告，Vol. EC-85, No. 52, pp. 1-12 (1985).
2) Tomita, S., Shibayama, K., Kitamura, T., Nakata, T. and Hagiwara, H.: A User-Microprogrammable Local Host Computer with Low-Level Parallelism, *IEEE Conf. Proc. of the 10th Annual Int. Symp. on Comput. Architecture*, pp. 151-157 (1983).
3) Okuno, H. G.: 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，Vol. CA-33, No. 4, p. 24 (1985).
4) 金田：Prologマシン，情報処理，Vol. 25, No. 12, pp. 1345-1352 (1984).
5) 瀧ほか：パーソナル逐次型推論マシン PSI のハードウェア設計，*Proc. of the Logic Programming Conf. '84*, No. 8.1, p. 15 (1984).
6) 西川ほか：PSI の性能評価(1)，情報処理学会第30回全国大会講演論文集，No. 1C-2, pp. 153-154 (1985).
7) 田村ほか：シーケンシャル実行型 Prolog マシン PEK—ハードウェア構成—，情報処理学会論文誌，Vol. 26, No. 5, pp. 855-861 (1985).
8) 和田ほか：逐次実行型 Prolog マシン PEK の性能評価，情報処理学会第31回全国大会講演論文集，No. 2C-3, pp. 39-40 (1985).
9) 長沼，小倉，山田：連想メモリを用いた Prolog マシンの構成と処理アルゴリズム，情報処理学会記号処理研究会資料，Vol. CA-32, No. 3, p. 8 (1985).
10) Warren, D. H. D.: An Abstract Prolog Instruction Set, *SRI Technical Note*, No. 309, p. 30 (1983).
11) Tick, E. and Warren, D. H. D.: Towards a Pipelined Prolog Processor, *New Generation Computing*, Vol. 2, No. 4, pp. 323-345 (1984).
12) Dobry, T. P., Despain, A. M. and Patt, Y. N.: Performance Studies of a Prolog Machine Architecture, *IEEE Conf. Proc. of the 12th Annual Int. Symp. on Comput. Architecture*, pp. 180-190 (1985).
13) Dobry, T. P., Patt, Y. N. and Despain, A. M.: Design Decisions Influencing the Microarchitecture for a Prolog Machine, *Proc. of the 17th Annual Microprogramming Workshop*, pp. 217-231 (1984).
14) Nakazaki, R. et al.: Design of a High-Speed Prolog Machine (HPM), *IEEE Conf. Proc. of the 12th Annual Int. Symp. on Comput. Architecture*, pp. 191-197 (1985).
15) 小長谷，阪野，梅村：高性能 Prolog マシン CHI における Prolog プログラムのコンパイル方式，情報処理学会第 31 回全国大会講演論文集，No. 3C-5, pp. 61-62 (1985).
16) 大寺ほか：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，Vol. CA-27, No. 4, p. 6 (1984).
17) 岸本，篠木，木村，服部：Prolog コンパイラの設計と評価，*Proc. of the Logic Programming Conf. '85*, No. 10.2, pp. 247-258 (1985).
18) 木村ほか：PROLOG インタプリタの性能評価，情報処理学会第31回全国大会講演論文集，No. 2C-4, pp. 41-42 (1985).
19) 日比野，渡辺，大里：Lisp マシン ELIS のアーキテクチャーメモリレジスタの汎用化とその効果—，情報処理学会記号処理研究会資料，Vol. CA-24, No. 3, p. 8 (1983).
20) Moon, D. A.: Architecture of the Symbolics 3600, *IEEE Conf. Proc. of the 12th Annual Int. Symp. on Comput. Architecture*, pp. 76-83 (1985).

(昭和61年1月17日受付)
(昭和61年5月15日採録)

柴山　潔（正会員）

昭和26年生．昭和49年京都大学工学部情報工学科卒業．昭和54年同大学院博士課程単位修得退学．同年同大学工学部情報工学教室助手，現在に至る．京都大学工学博士．計算機システム，計算機アーキテクチャの研究に従事．電子通信学会，IEEE, ACM 各会員，ICOT・WG 委員．

富田 眞治（正会員）

昭和20年生．昭和43年京都大学工学部電子工学科卒業．昭和48年同大学院博士課程修了．この間，零交さ波による音声合成の研究に従事．京都大学工学博士．同年京都大学工学部情報工学教室助手．昭和53年同助教授，現在に至る．計算機アーキテクチャ，並列処理システムなどに興味をもつ．著書（分担執筆）「計算機ハードウェア実験」「VLSI コンピュータⅠ」．電子通信学会，ACM, IEEE 各会員．ICOT・WG 委員．

萩原 宏（正会員）

大正15年生．昭和25年京都大学工学部電気工学科卒業．NHK を経て，昭和32年京都大学工学部助教授，昭和36年同教授，現在に至る．工学博士．情報理論，パルス通信，電子計算機などの研究に従事．昭和31年度稲田賞受賞．昭和50年本学会論文賞受賞．昭和56～58年度本学会副会長．著書「電子計算機通論 1～3」「マイクロプログラミング」など．電子通信学会，ACM, IEEE 各会員．